Web개발가이드

version 2.3.0

작성자	DT혁신연구팀
작성일자	2022.04.06

롯데정보통신

제·개정이력

버전	제·개정 페이지 및 내용	제·개정 일자
v2.3.0	기존 문서에서 분리 및 병합	2022-04-06

목 차 (Table of Content)

개요

개발표준프레임워크란?

개발표준프레임워크는 개발표준을 확립하고 효율적이고 안정적인 개발과 운영환경을 제공한다.

- 자바기반 웹/배치 어플리케이션 개발을 위한 플랫폼
- 전사비즈니스 요구사항 신속 대응을 위한 L.Cloud 기반 플랫폼

특장점

개발표준 프레임워크는 아래와 같은 특장점을 제공한다.

- **표준 개발 환경 (개발 생산성 제고)**
  - 단순 업무를 위한 자동화된 개발도구
  - Eclipse, 프로젝트/소스코드/배치JOB 생성도구
- **클라우드 환경 및 MSA 대응**
  - 캐모마일 기반의 어플리케이션을 쉽게 만들기 위한 독립형(standalone) 캐모마일 Boot
  - 멀티 인스턴스(어플리케이션) 관리
- **상용 솔루션 연계 (손쉬운 시스템 통합)**
  - 다양한 형태의 연계모듈 제공
  - UI 어댑터, 카톡, 텔레그램, 이노룰스 BRMS, SAP JCO
- **그룹 보안성 심의 준수**
  - 프레임워크 차원의 그룹 보안성 심의 사전 대응
  - 인증/인가, 암호화(SHA, AES, ARIA), 로그 마스킹
- **시스템 공통 컴포넌트/어드민**
  - 검증된 시스템 컴포넌트
  - 자원관리, 권한관리, 시스템관리, 통계, 모니터링 등
- **모바일 앱 실행/운영 환경**
  - iOS, Android Hybrid 모바일 앱 개발, 실행 환경 제공
  - 모바일 앱 관리, 배포를 위한 운영 환경 제공

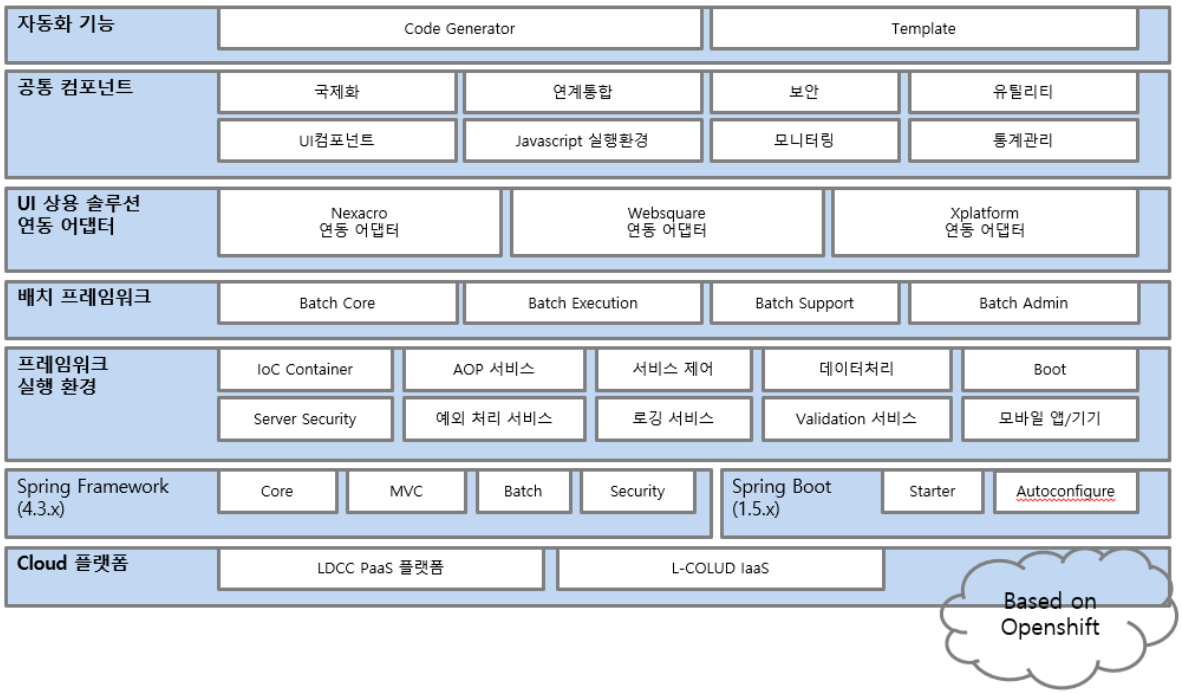
- **배치 어플리케이션 개발 및 관리**
  - 독립형, 내장 배치 컴포넌트, 워크플로우, 실행(재시작) 및 중지
- **오픈소스 APM을 활용한 모니터링**
  - Xlog, Active Service, CPU, Memory, Network, TPS 등.
- **설치/교육/운영 지원**
  - 전담 지원조직을 통한 지원

아키텍처

Java 기반의 Spring Framework로 구성되어져 있으며, 그룹사의 다양한 환경을 고려해 Spring 버전 5.2으로 구성되었다.

v1.2 이후부터는 모바일 및 스프링 부트를 지원한다. (Spring Boot 버전 2.3)

자사의 클라우드 플랫폼인 L.Cloud와 Clepass에서의 동작을 지원하고, 아래와 같은 다양한 기능들을 제공한다.



[그림] 개발 표준 프레임워크 아키텍처

구성요소

개발표준 프레임워크의 구성요소는 아래와 같다.

구 분	설 명
온라인 프레임워크	• 웹 어플리케이션 실행에 필요한 기능 제공 (온라인 프레임워크 Core 기능) • 웹 어플리케이션 실행에 필요한 어드민 기능
배치 프레임워크	• 배치 어플리케이션 실행에 필요한 기능 제공 (배치 프레임워크 Core 기능) • 배치 어플리케이션 실행에 필요한 어드민 기능
공통 컴포넌트/ 어드민	• 재사용 가능한 시스템 컴포넌트 / 유틸리티를 제공 • FW 운용 및 유지보수를 지원하는 기능 제공 (배치운영, 모니터링 등)
개발환경	• App. 개발환경 지원 도구 제공 (프로젝트 템플릿 생성, 코드 자동 생성) • 모바일 앱 개발을 위한 기본 프로젝트 제공 (Android Studio, Xcode 기반 프로젝트)

[그림] 개발 표준 프레임워크 구성요소

프레임워크 차별성

- 상용솔루션 서비스 연계
- 글로벌 다국어 지원 (화폐표현 및 로케일정보 포함)
- 오픈소스 그리드 컴포넌트 선정 및 연동
- 캐모마일 Boot(클라우드 및 MSA 대응), 모바일 실행/운영 환경 제공
- 테스트(시나리오)관리, 배치관리, 모니터링 지원기능
- 보안서비스 강화(그룹 보안성 심의 항목 대응)

Web 프레임워크

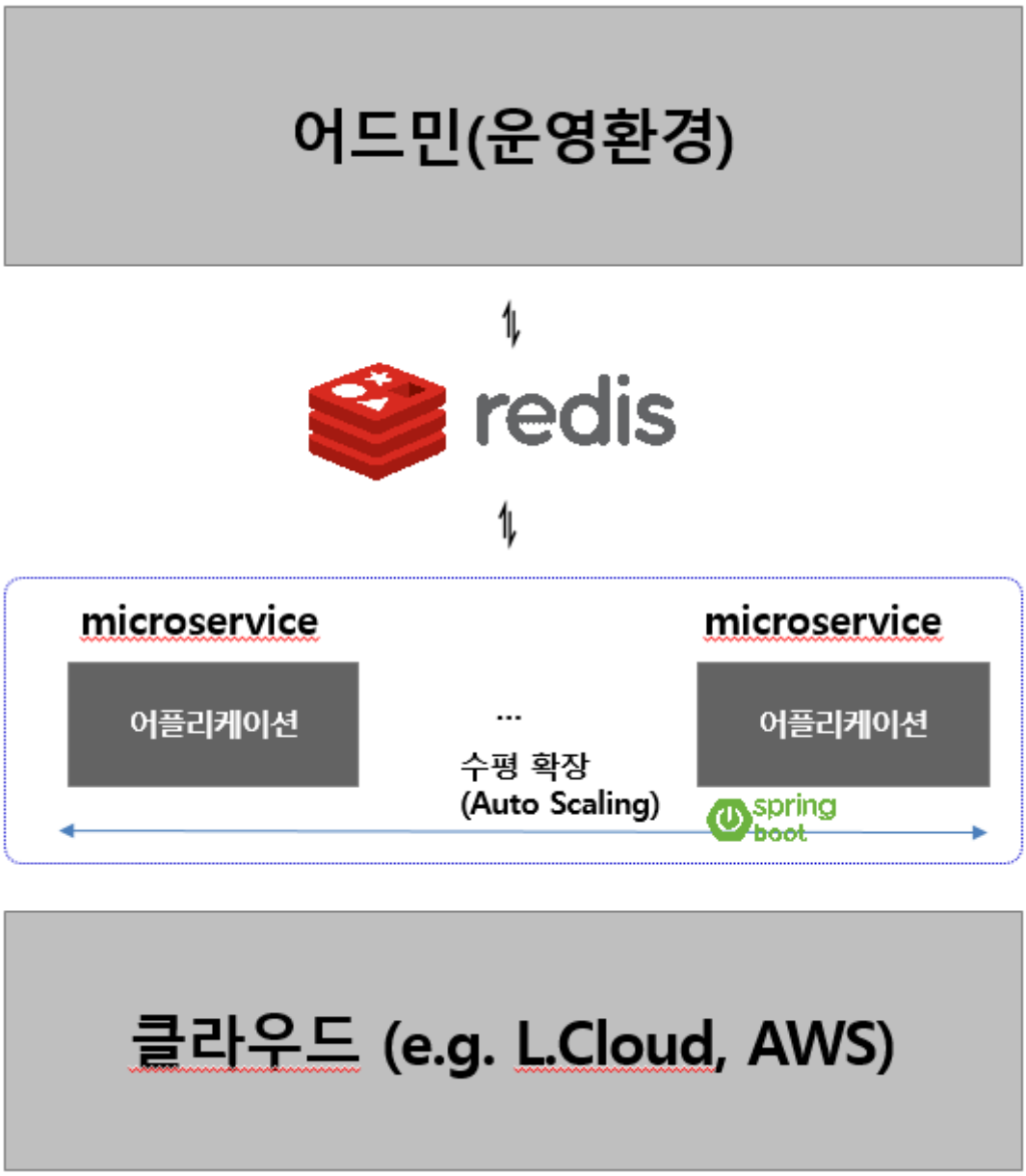
온라인 프레임워크는 일반적인 웹 어플리케이션 개발에 필요한 다양한 기능들로 구성되어져 있다.

온라인 프레임워크는 아래와 같이 크게 10개의 기능으로 구분되어져 제공된다.

대분류	중분류	소분류
실행 환경	MVC 서비스	• 서비스 관리 및 서비스제어 정보 • 서비스 실행 및 테스트 (트랜잭션)
	로깅 서비스	• 로그마스킹, 동적로그변경 • 거래추적 ID별로 로깅 및 조회 • 사용자별 로그 활성화
	데이터처리	• 동적 SQL 반영 • 페이징/배치 처리 모듈
	Server Security	• DB 기반 인증 및 접근제어 • Multi-factor 인증
	예외처리	• 비즈니스 예외처리 • 예외처리 서비스
	UI 연동 어댑터	• 연동 어댑터 추상화 • Websquare, Nexacro, Xplatform 연동 어댑터
	Validation	• 서버 단 Validation
	AOP 서비스	• AOP 활성화/비활성화 기능 (개인이력관리, 거래추적) • 메소드 별 처리속도 측정 및 리포팅 • 사용자 사용성 분석 (이력 및 도식화)
실행 환경 Boot	Boot	• Starter 모듈 • 관례에 따른 자동 환경설정
실행 환경 모바일	모바일 앱/기기	• 앱 /기기 제어 및 관리 • 앱, 웹 source 버전 관리

클라우드 환경을 위해 아래와 같은 기능을 제공한다.

- 멀티인스턴스 관리 (로그변경, 어플리케이션 테스트, 사용자별 로그레벨 변경)
- 클라우드 환경에서 autoscaling 된 서버 자동감지 및 helthcheck



요구사항

시스템 요구사항은 아래와 같다.

- 서버

항목	설명
운영체제 (OS)	Windows, Unix, Linux, MacOS
JVM	Java 1.8
WAS	Java 1.8을 지원하는 WAS는 기본적으로 지원
Database	Oracle 12c, SQL Server 2012, Mysql 5.7+, Mariadb 10.0+, Tibero 6.0

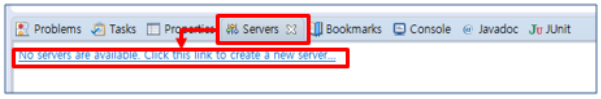
- 개발환경 (eclipse oxygen 4.7)

항목	설명
운영체제 (OS)	Windows, Linux, MacOS
JVM	Java 1.8+ (32bit, 64bit)
CPU	2GHz 이상, dual/quad core processor
메모리	1GB, (Recommend 2GB)
디스크 공간	2.1GB 이상

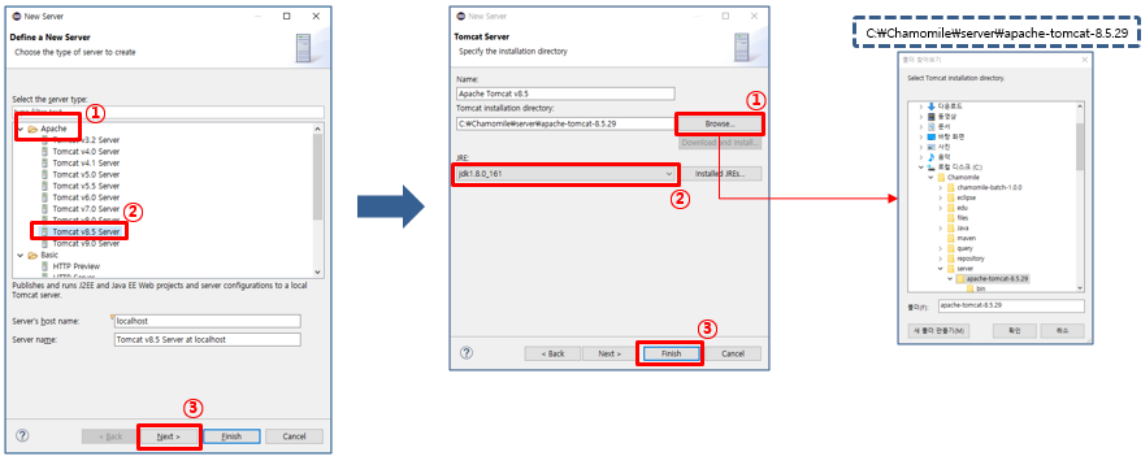
개발 도구

서버 생성

1. 서버 생성



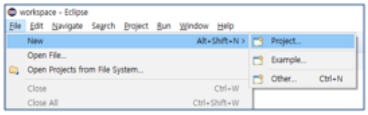
2. 서버 정보 입력



[그림] 서버생성

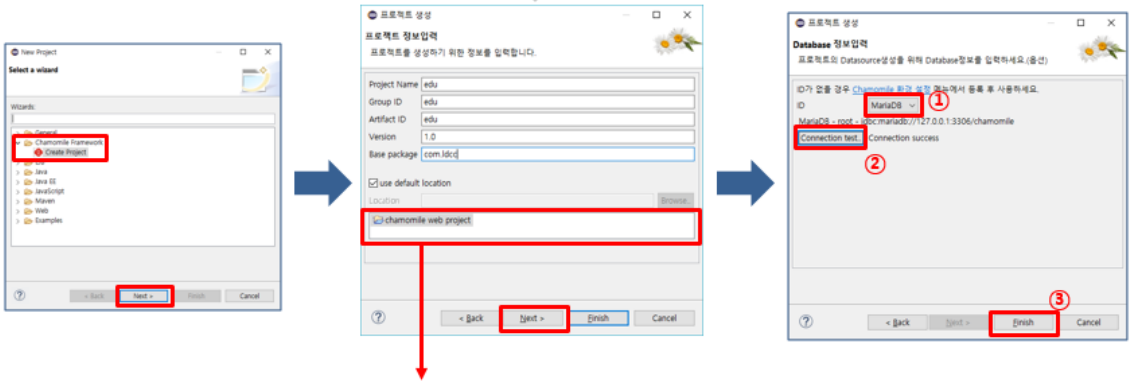
프로젝트 생성

1. 프로젝트 생성



2. 프로젝트 정보 입력

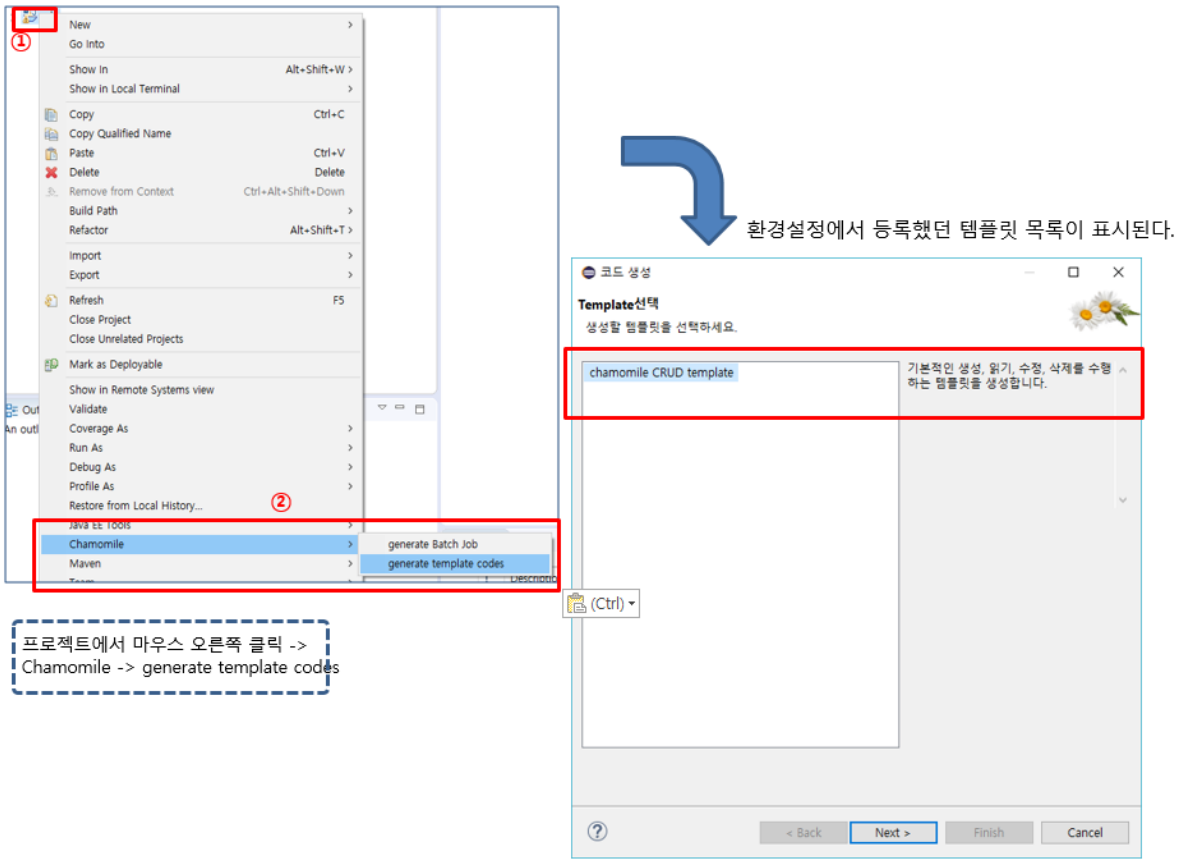
Project Name : edu
Group ID : edu
Artifact ID : edu
Version : 1.0
Base package : com.ldcc



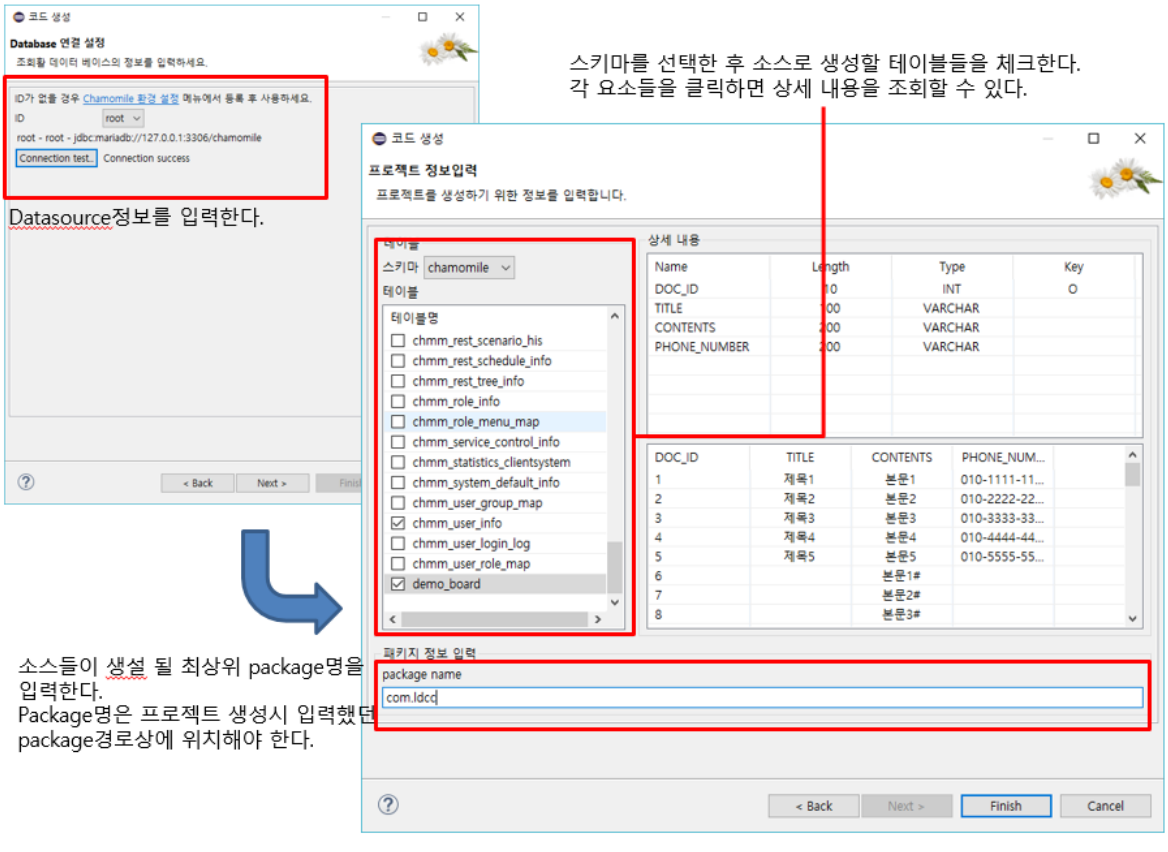
환경설정에서 등록했던 프로젝트 샘플목록이 표시된다.

[그림] 프로젝트 생성

코드 생성

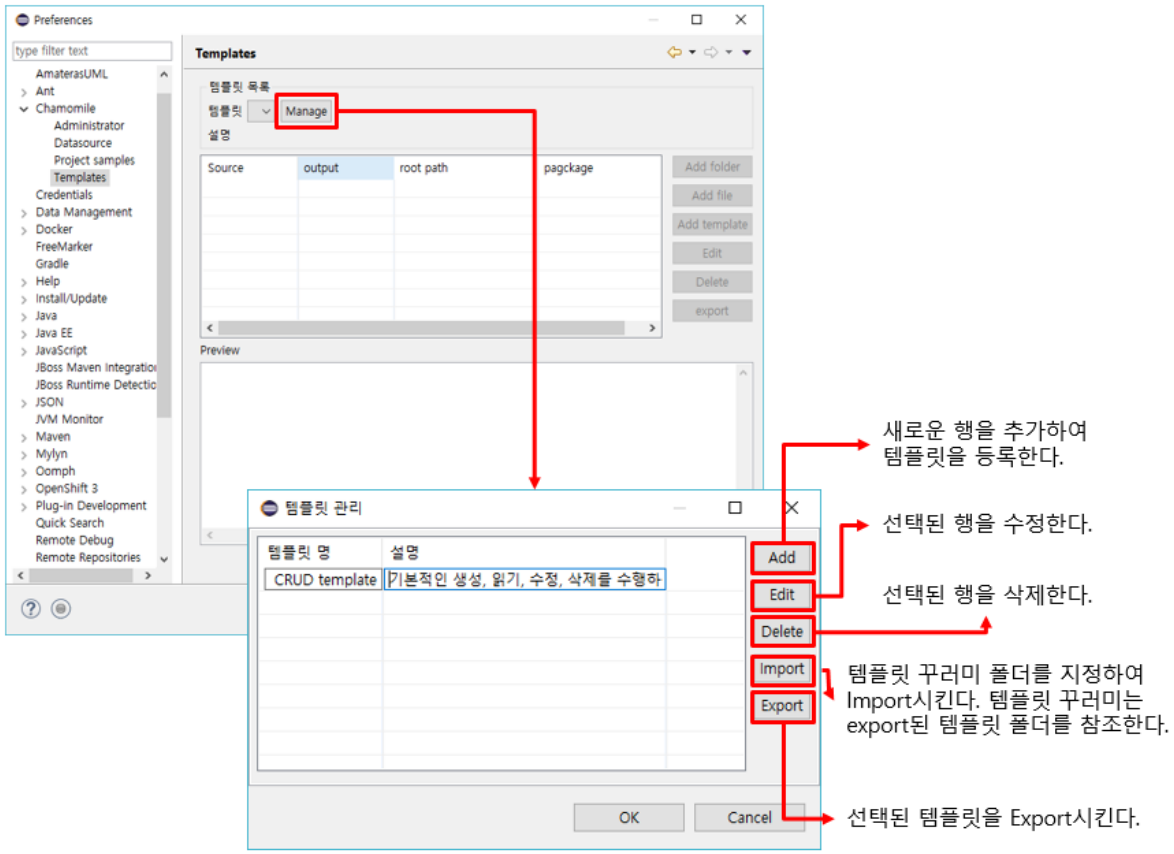


[그림] 코드 자동생성1



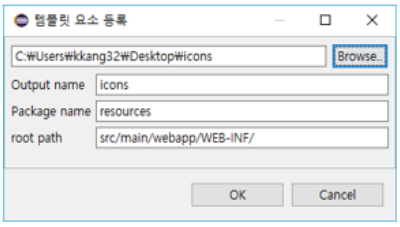
[그림]코드 자동생성2

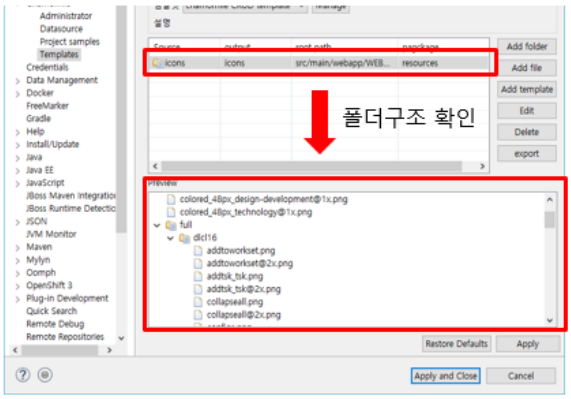
Template



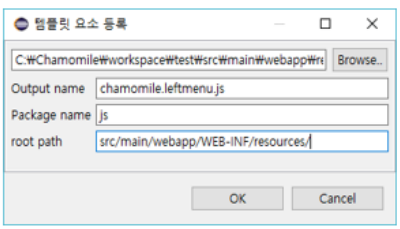
[그림] 템플릿 등록

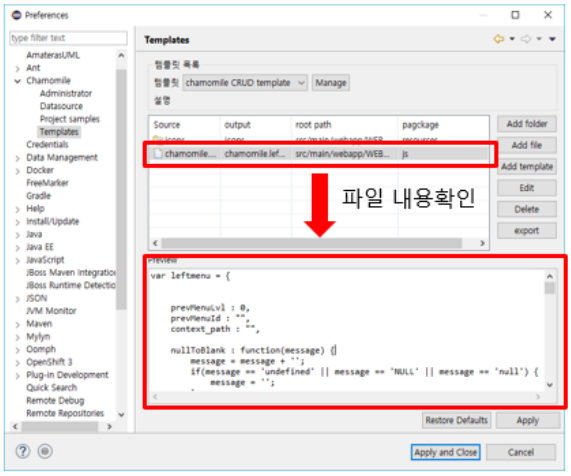
Add folder를 통하여 템플릿으로 사용할 폴더를 선택한다.



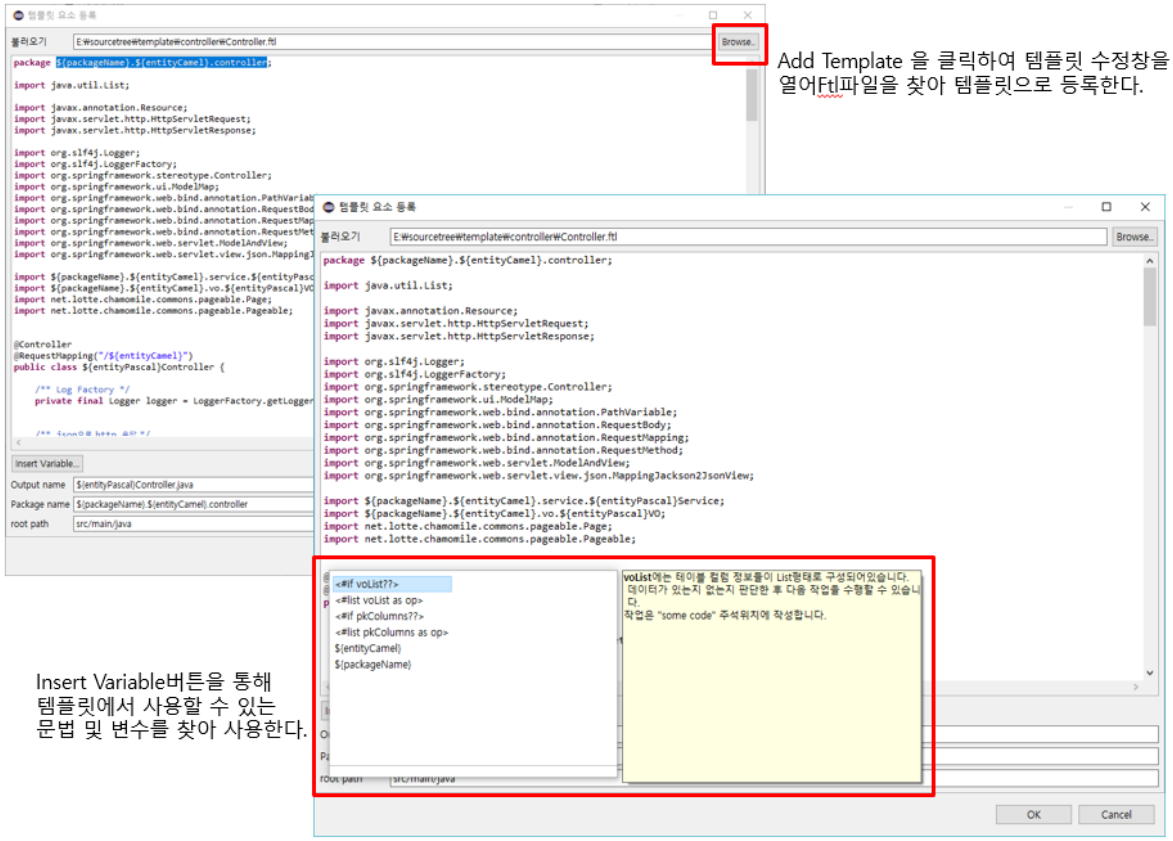


Add file을 통하여 템플릿으로 사용할 파일을 선택한다.





[그림] 템플릿구성요소 등록1



[그림] 템플릿구성요소 등록2

- 템플릿파일 등록시 각 항목 설명
  - Output name : 템플릿 파일이 생성될 최종 이름
  - package name : 템플릿 파일이 생성될 경로. java파일의 경우 root path아래로 생성되며 그 외에는 폴더 형식으로 경로를 입력하여 생성 될 수 있도록 한다.
  - root path : 파일이 생성될 최상위 폴더를 지정해 준다. 해당 폴더 아래의 package name을 참조하여 폴더를 만들어준다.
- 기타 추가 버튼 설명
  - Edit : 선택된 템플릿 구성요소를 수정한다. 폴더, 파일, 템플릿 파일에 맞게 수정창을 띄워준다.
- Delete : 선택된 템플릿 구성요소를 삭제한다.
- Export : 선택된 템플릿 구성요소를 별도로 저장한다.

Template 파일 변수목록

> - `pagakeName`
> - `entityCamel`
> - `entityPascal`
> - `entityOrigin`
> - `projectName`
> - `voList`
> - `pkColumns`

packageName

설명

- 사용자가 입력한 package명

사용법

```
${packageName}
```

예시

```
package ${packageName}.${entityCamel}.controller;

import java.util.List;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

entityCamel

설명

- 선택한 테이블명을 camel case로 변환한 문자열

사용법

```
${entityCamel}
```

예시

- package명

```
package ${packageName}.${entityCamel}.controller;

import java.util.List;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

entityPascal

설명

- 선택한 테이블명을 Pascal case로 변환한 문자열

사용법

```
${entityPascal}
```

예시

- 클래스명

```
@Controller
public class ${entityPascal}Controller {

  /** Log Factory */
  private final Logger logger = LoggerFactory.getLogger(this.getClass());
  //...
}
```

entityOrigin

설명

- 조회된 테이블의 원본이름

사용법

```
${entityOrigin}
```

예시

- SQL쿼리

```
SELECT * from ${entityOrigin} A
```

projectName

설명

- 프로젝트 명

사용법

```
${projectName}
```

예시

```
${projectName}
```

voList

설명

- 테이블에서 제공하는 컬럼 목록을 변수 화 한 목록
- 하위요소

변수명	타입	설명

변수명	타입	설명
dataType	String	java형 데이터 타입으로 변환한 값(Deprecated) - TINYINT, SMALLINT, BOOLEAN, INTEGER, BIGINT => **Integer** - REAL, FLOAT, DOUBLE, DECIMAL => **Double** - VARCHAR, LONGVARCHAR, CLOB, NVARCHAR, CHAR, DATE, TIME, TIMESTAMP, 기타 => **String**
colNameOrg	String	테이블에서 조회된 컬럼명
colNameCamel	String	테이블에서 조회된 컬럼명을 camel case로 변환시킨 값
colNamePascal	String	테이블에서 조회된 컬럼명을 pascal case로 변환시킨 값
colBindValue	String	colNameCamel 값을 mybatis 에서 사용할 수 있도록 "#{변수명}" 형태로 처리한 값(deprecated)
colBindValueItemPrefix	String	colNameCamel 값을 mybatis 에서 사용할 수 있도록 "#{item.변수명}" 형태로 처리한 값(deprecated)
length	integer	컬럼의 길이
pk	boolean	Primary key여부 (pk일 경우 true)
nullable	boolean	null 허용 여부(null허용일 경우 true)
comment	String	컬럼 comment값(SQL Server는 지원하지 않음)
dataTypeClassName	String	java형 데이터 타입으로 변환한 값 NUMERIC, DECIMAL => **java.math.BigDecimal** BIT => **java.lang.Boolean** TINYINT => **java.lang.Byte** SMALLINT => **java.lang.Short** INTEGER => **java.lang.Integer** BIGINT => **java.lang.Long** REAL => **java.lang.Float** FLOAT, DOUBLE => **java.lang.Double** BINARY, VARBINARY, LONGVARBINARY => **byte[]** DATE => **java.sql.Date** TIME => **java.sql.Time** TIMESTAMP => **java.sql.Timestamp** BLOB => **java.sql.Blob** CLOB => **java.sql.Clob** 나머지 -> **java.lang.String**

사용법

```
<#if voList??>
  <#list voList as op>
```

```
    ${op.colNameOrg}
    ${op.colNameCamel}
    ${op.colNamePascal}
    ${op.colBindValue}
    ${op.colBindValueItemPrefix}
    ${op.length}
    ${op.pk}
    ${op.nullable}
    ${op.comment}
    ${op.dataTypeClassName}
 </#list>
</#if>
```

예시

- *dataType*

```
<#if voList??>
    <#list voList as op>
        private ${op.dataType} ${op.colNameCamel};
    </#list>
</#if>
```

- *colNameOrg*

```
<#if voList??>
        <#list voList as op>
            ${op.colNameOrg} ${op.colNameCamel} <#if op_has_next>,</#if>
        </#list>
</#if>
```

- *colNameCamel*

```
<#if voList??>
    <#list voList as op>
    private ${op.dataType} ${op.colNameCamel};
    </#list>
</#if>
```

- *colNamePascal*

```
<#if voList??>
    <#list voList as op>
    public ${op.dataType} get${op.colNamePascal}(){
        return this.${op.colNameCamel};
    }
    public void set${op.colNamePascal}(${op.dataType} ${op.colNameCamel}){
        this.${op.colNameCamel} = ${op.colNameCamel};
    }

    </#list>
</#if>
```

- *colBindValue*

```
SELECT
*
            FROM
            table A

            WHERE
            <#if voList??>
                    <#list voList as op>
                        ${op.colNameOrg} = ${op.colBindValue} <#if op_has_next>
and </#if>
                    </#list>
            </#if>
```

`#` 문자가 freemarker에서 예약된 문자이기 때문에 바로 사용할 수 없다. 이 때문에 사용의 편의를 위해 제공하는 변수로 좀더 확장성있게 사용하고자 할 경우 `#`문자를 `${}`으로 감싸서 표현한다. 아래와 같이 작성할 경우 `colBindValue` 에서 제공하는 값과 동일하게 사용할 수 있다.

```
${"#"}{${op.colNameCamel}}
```

- *length*

```
<#if voList??>
    <#list voList as op>
    @Size(max=${op.length})
        private ${op.dataType} ${op.colNameCamel};
    </#list>
</#if>
```

- *pk* (boolean값으로 조건문에서만 사용해야 한다.)

```
<#if voList??>
    <#list voList as op>
        <#if op.pk>
            This is pk!
        </#if>
    </#list>
</#if>
```

- *nullable* (boolean값으로 조건문에서만 사용해야 한다.)

```
<#if voList??>
    <#list voList as op>
        <#if !op.nullable>
            @NotNull
        </#if>
  </#list>
</#if>
```

- *comment*

```
<#if voList??>
    <#list voList as op>
    //${op.comment}
        private ${op.dataType} ${op.colNameCamel};
    </#list>
</#if>
```

- *dataTypeClassName*

```
<#if voList??>
    <#list voList as op>
        private ${op.dataTypeClassName} ${op.colNameCamel};
    </#list>
</#if>
```

pkColumns

- 테이블의 PK컬럼들만 모아놓은 변수로 사용법은 `voList`와 같다.

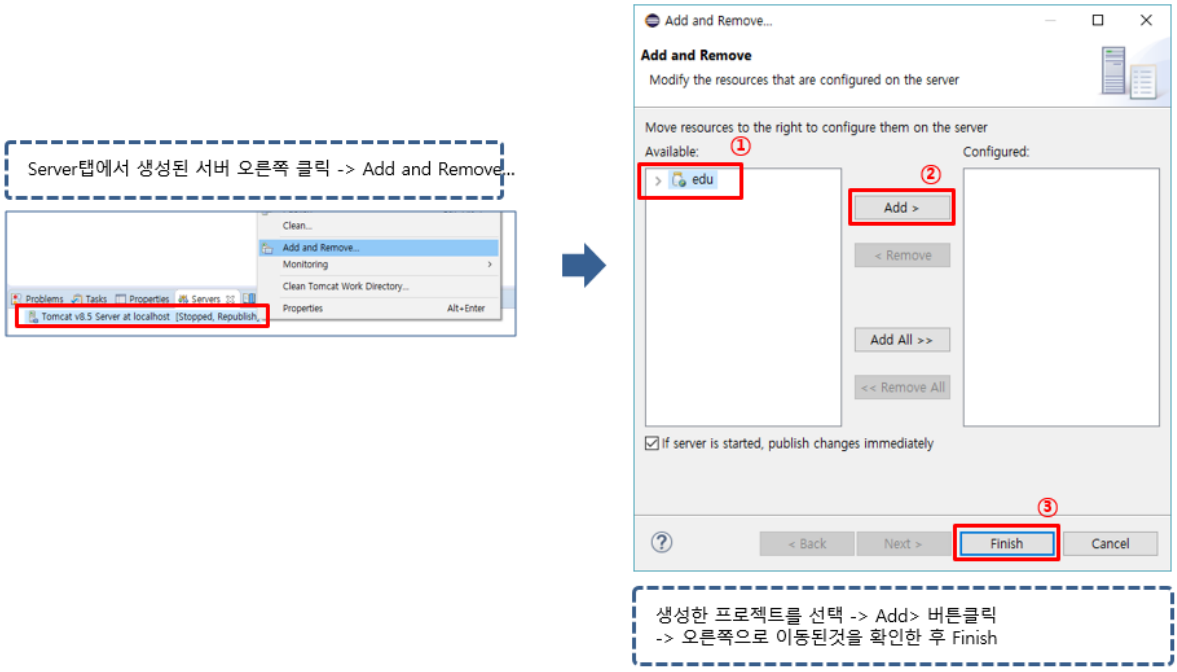
```
<#if pkColumns??>
    <#list pkColumns as op>

    ${op.colNameOrg}
    ${op.colNameCamel}
    ${op.colNamePascal}
    ${op.colBindValue}
    ${op.colBindValueItemPrefix}
    ${op.length}
    ${op.pk}
    ${op.nullable}
    ${op.comment}
    ${op.dataTypeClassName}

    </#list>
</#if>
```

> **freemarker의 자세한 문법은 아래 사이트를 참고 한다.**
>
> **https://freemarker.apache.org/docs/index.html**

로컬서버에 배포

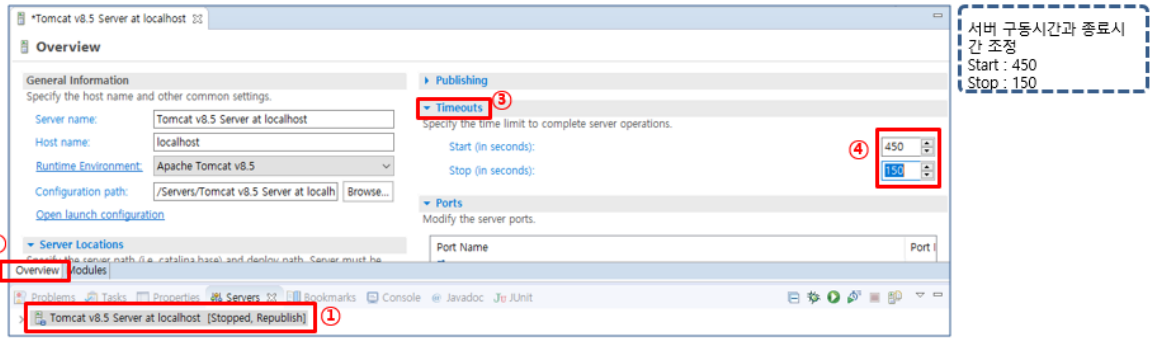
서비스 모듈 설정

1. 서버에 프로젝트 Add

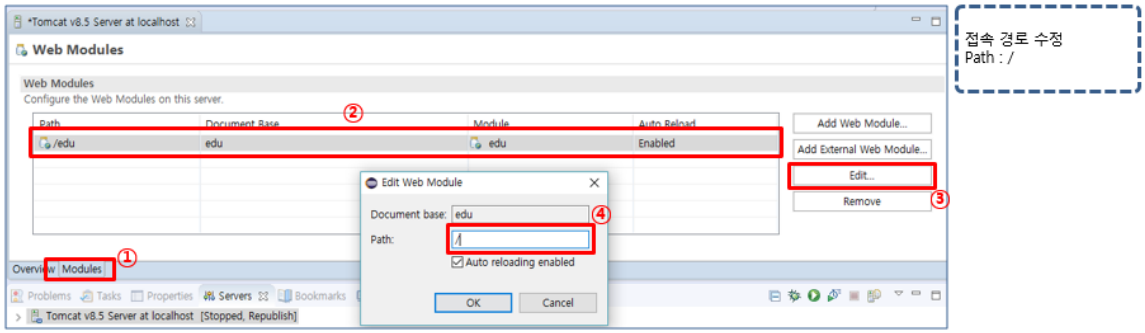


[그림] 로컬 서버 배포1

2. 타임아웃설정



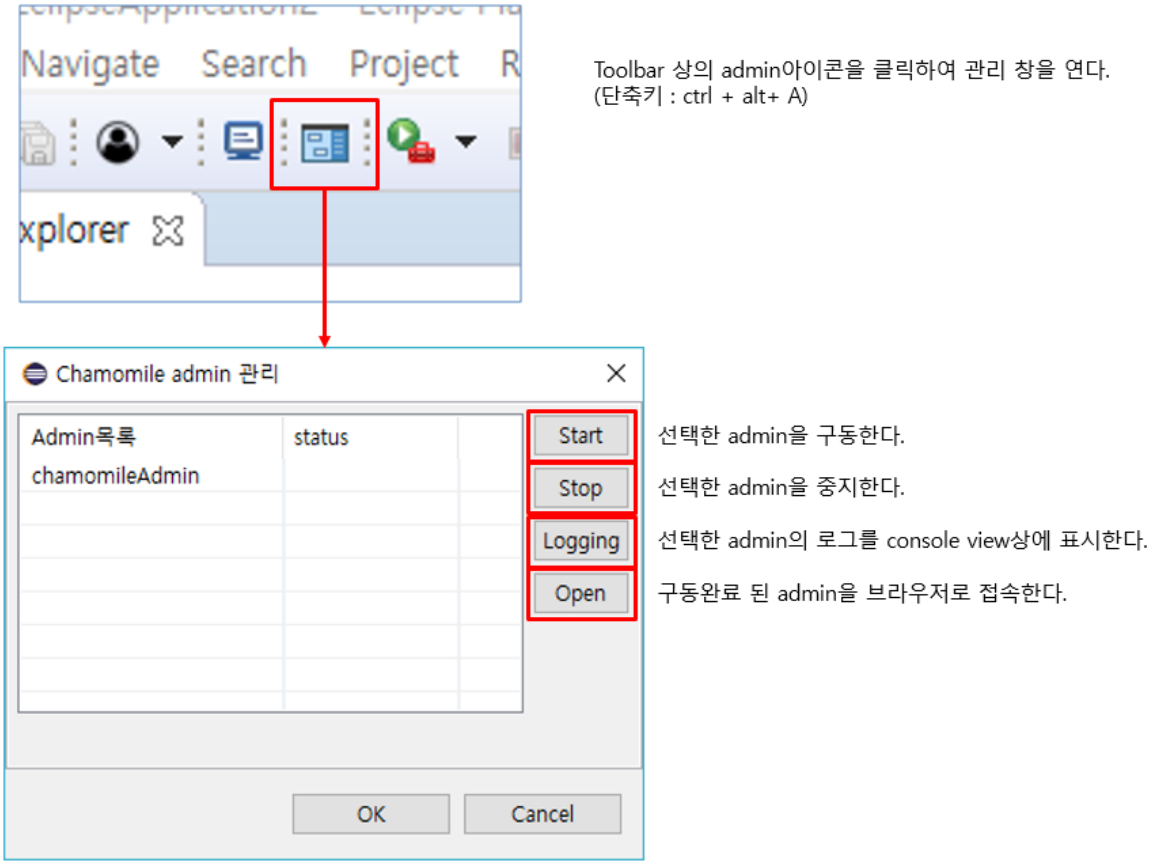
3. Path 설정



[그림] 로컬 서버 배포 2

admin서버 구동

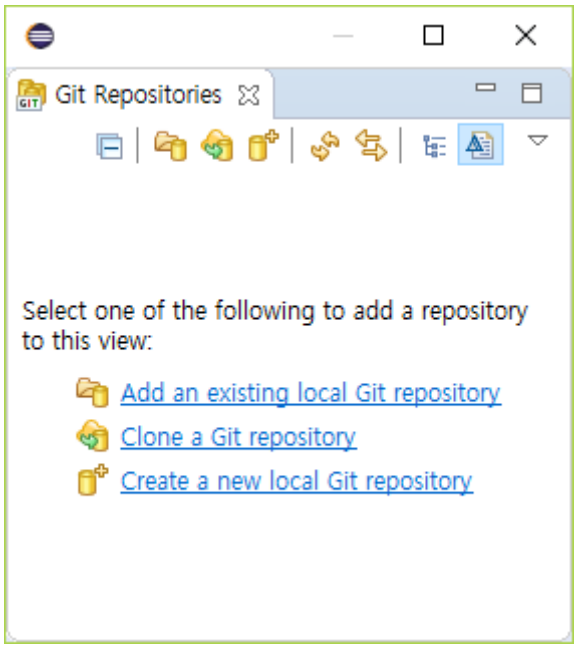
환경설정의 administrator에 등록했던 서버를 실행하면 admin에서 서비스 application을 관리 할 수 있다.



[그림] admin서버 관리

원격 저장소 사용

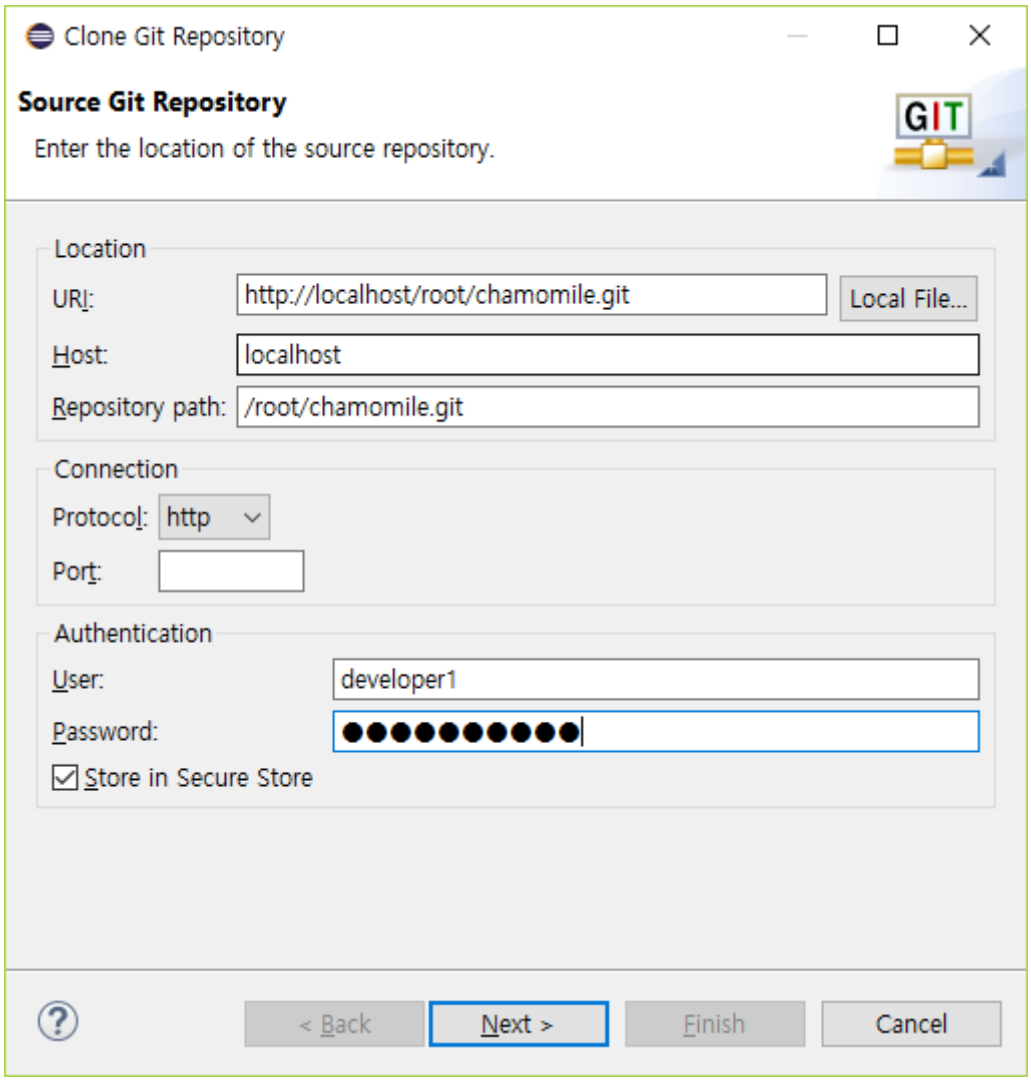
1) 왼쪽 영역에서 Git Repositories를 선택한다. 없을 경우 Window -> show view -> other -> Git Repositories 검색 후 Open 한다.



[그림] Git Repositories View화면

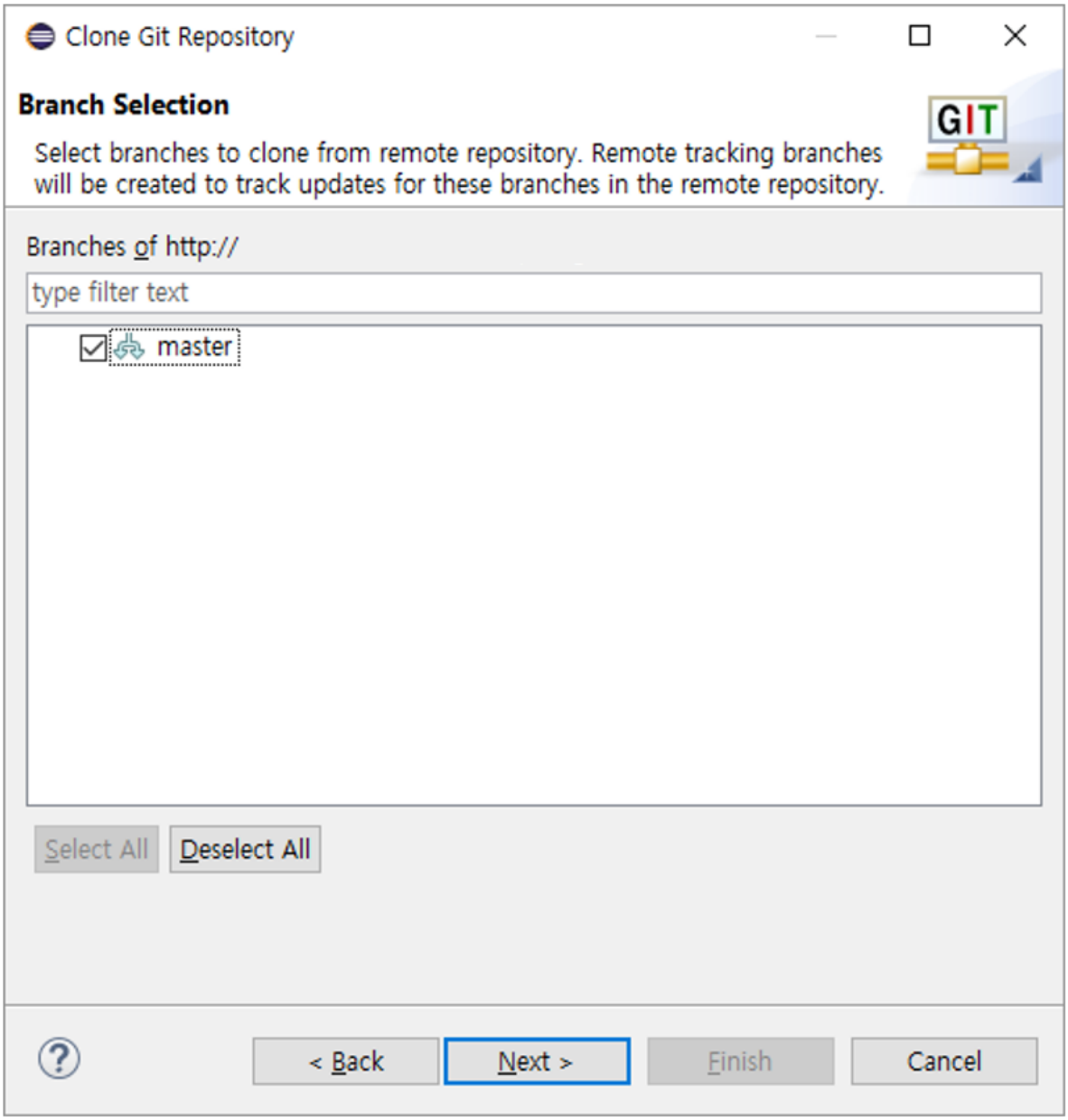
2) Clone a Git repository항목을 클릭하여 저장소 정보와 계정을 입력한다.

(Location URI항목에 주소만 입력하면 Authentication외 나머지 항목들은 자동으로 채워진다.)



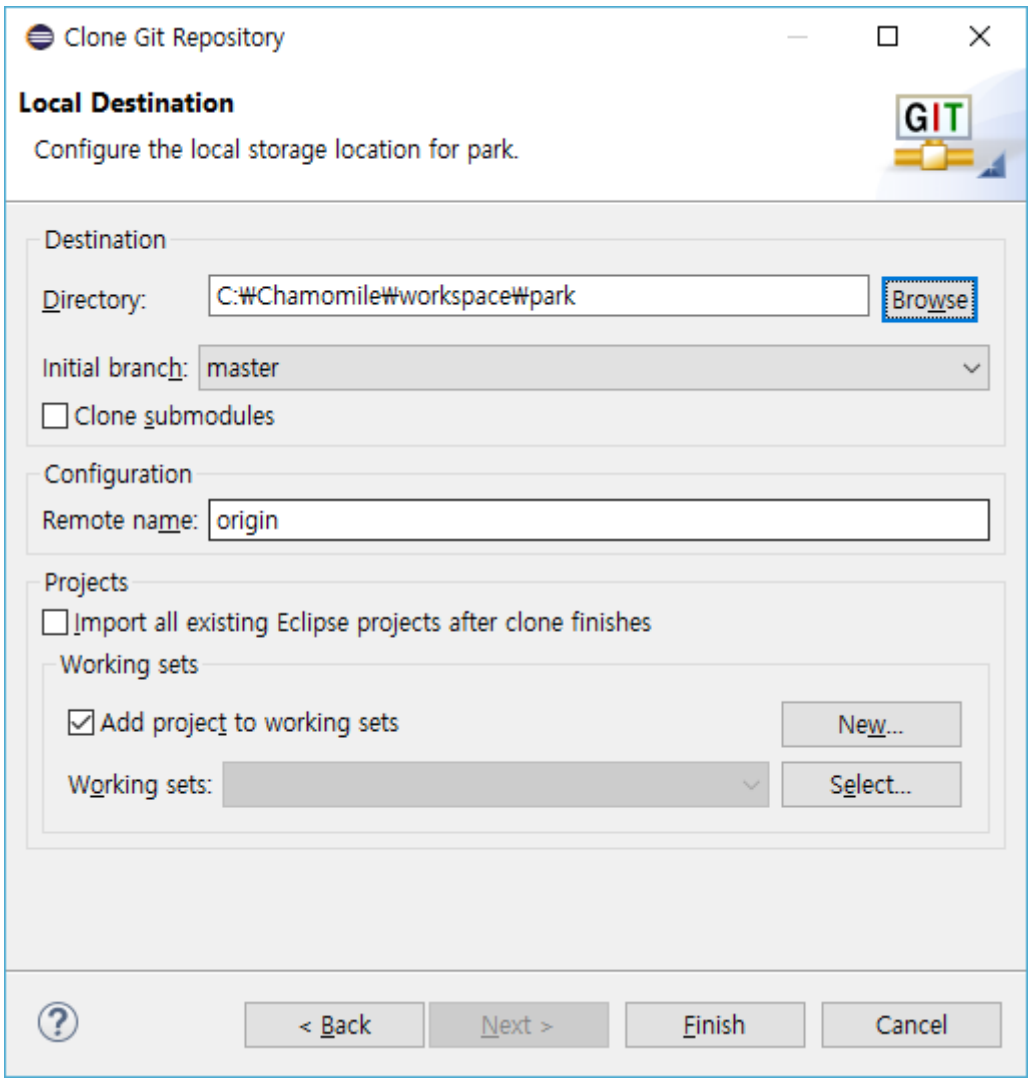
[그림] 저장소 정보 입력

3) Next를 클릭하면 clone시킬 Branch를 선택하게 되고 알맞은 Branch를 선택한후에 Next버튼을 클릭한다.



[그림] Branch선택

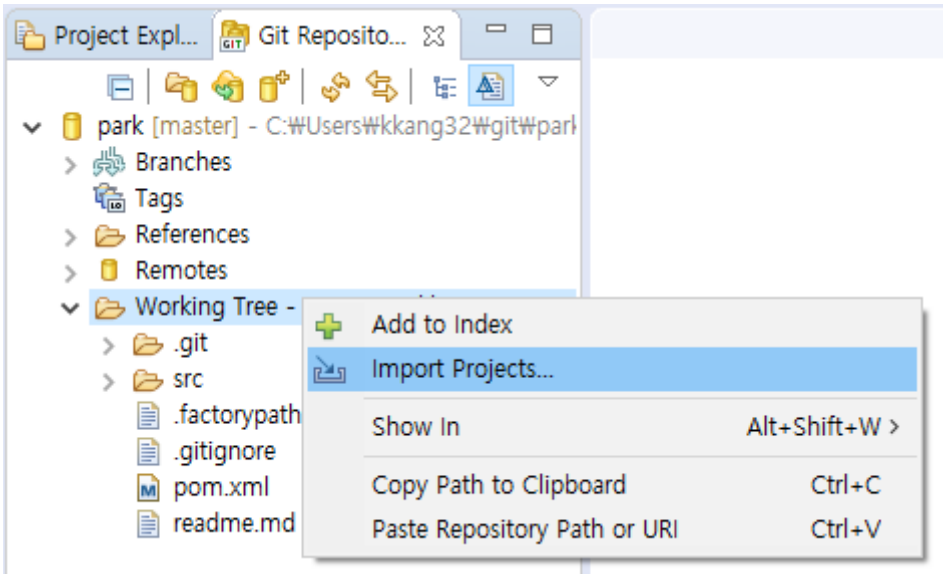
4) Directory항목에서 프로젝트가 위치할 경로를 지정하고Finish를 진행한다.



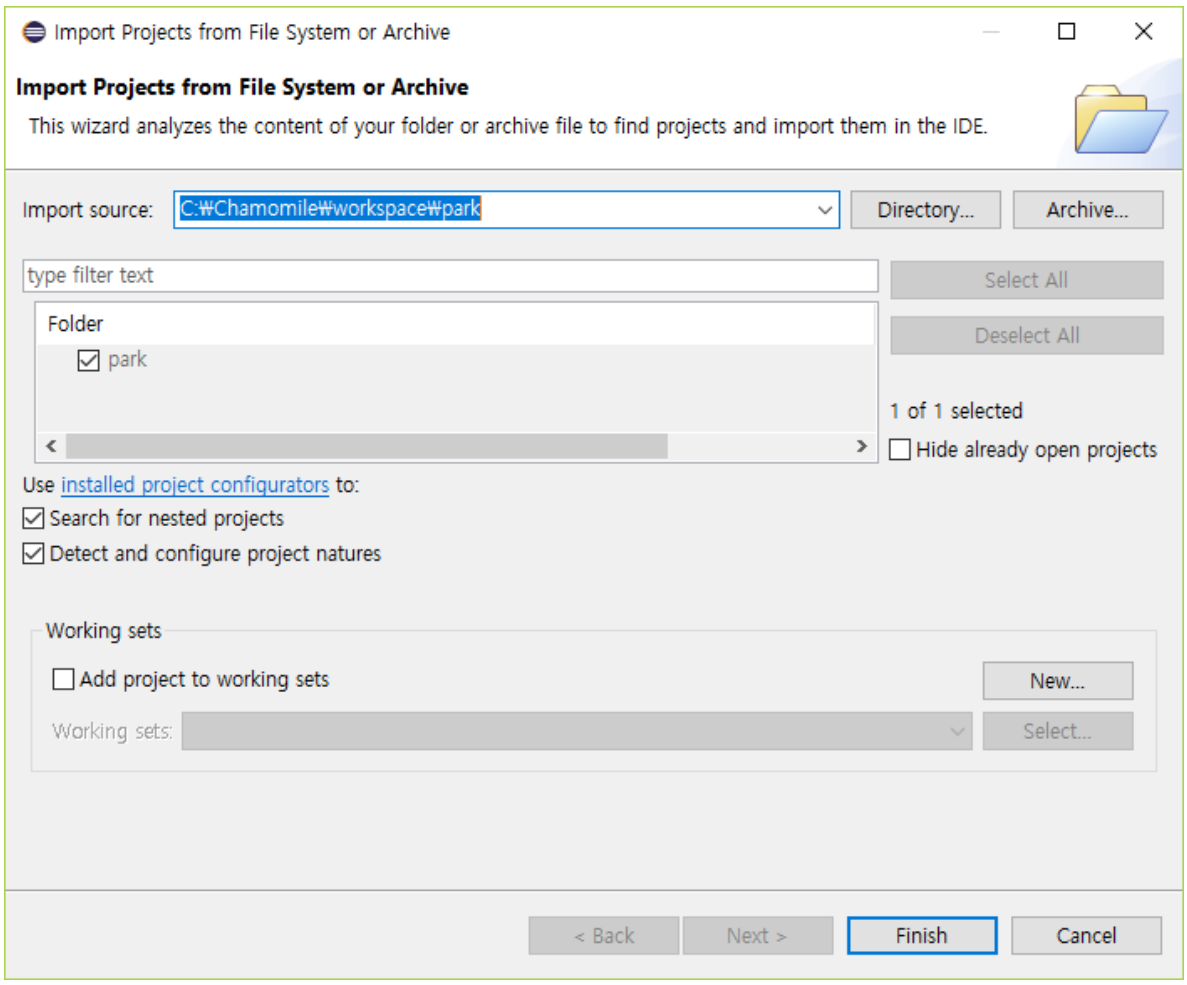
[그림] 프로젝트 경로 지정

5) 프로젝트 import

clone된 프로젝트의 working tree에서 마우스 오른쪽 버튼을 클릭하여 Import Project를 선택하여 import를 진행한다.



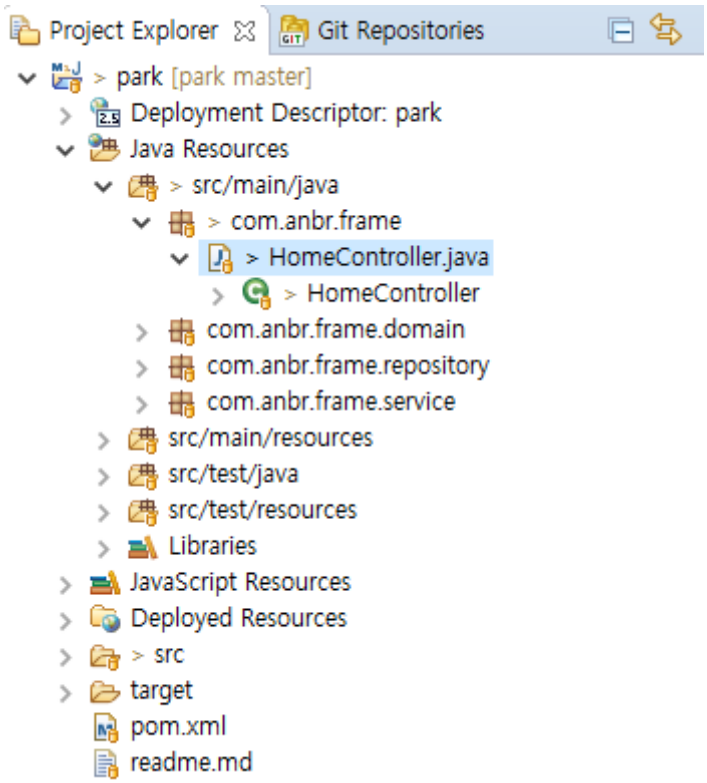
[그림] 프로젝트 import 1



[그림] 프로젝트 import 2

commit/push

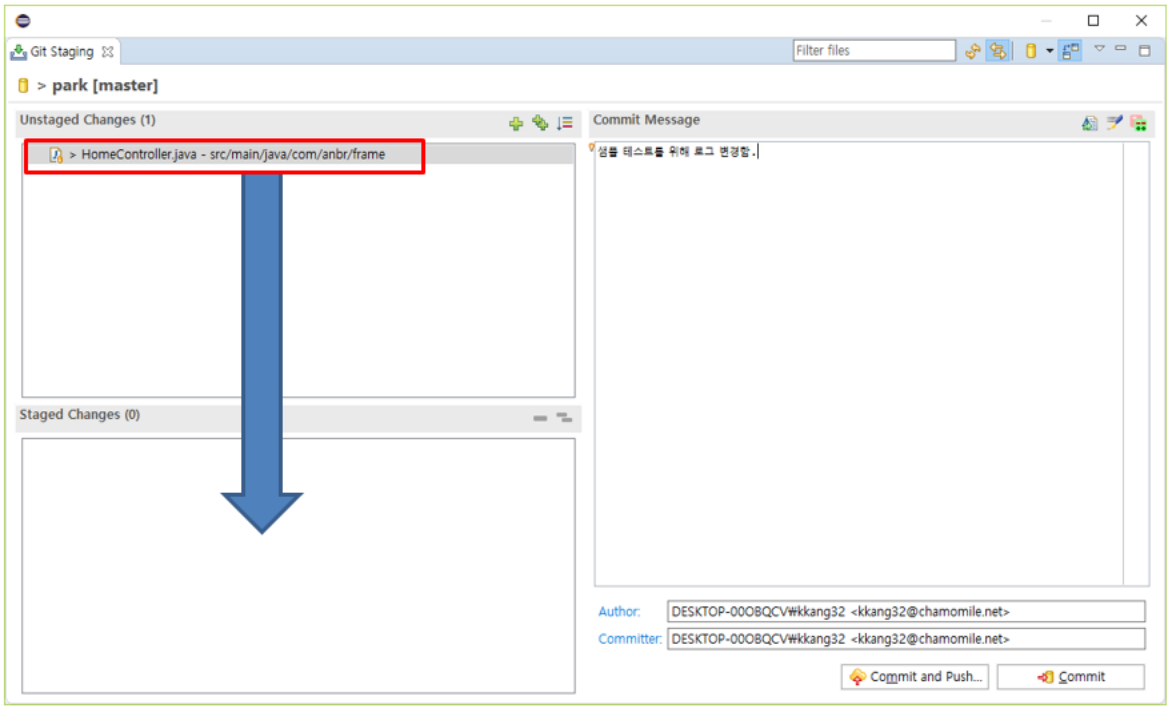
수정사항이 생길경우 아래 그림처럼 수정된 파일에 > 표시가 추가된다.



[그림] 파일 변경

이 상태에서 Git staging View를 확인해보면 Unstaged Changes 영역에 수정된 파일이 있는것을 확인할 수 있다.

저장소에 적용하고자 하는 파일들을 Staged Changes영역으로 드래그 하여 옮긴 후 Commit Message를 상세하게 작성한다.



[그림] 변경내역 commit/push

이후 Commit을 하거나 Commit and Push버튼으로 저장소에 저장한다. Commit만 수행했을 경우 추후에 [생성된 Project] 마우스오른쪽 버튼 -> Team -> push to upstream으로 commit 내역들을 저장소에 적용해야 한다.

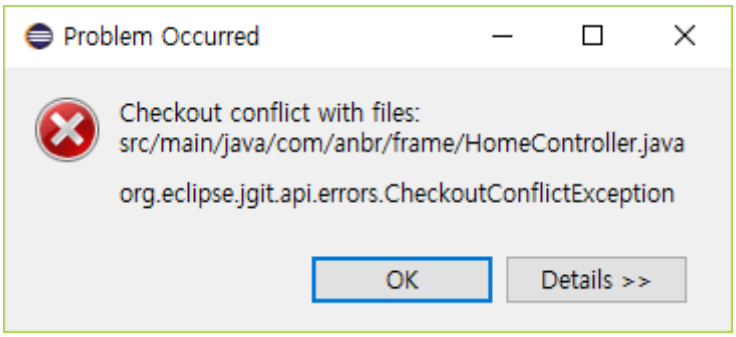
pull

push전에 항상 자신의 로컬저장소와 원격저장소간의 정보를 동일하게 해야 한다. 로컬저장소의 버전이 원격저장소 버전과 다를경우 push는 reject된다.(non fast-forward)

pull을 실행하여 원격 저장소에서 최신정보들과 head정보를 받아온다. 그 다음 push를 수행한다.

[생성된 Project] 마우스오른쪽 버튼 -> Team -> pull

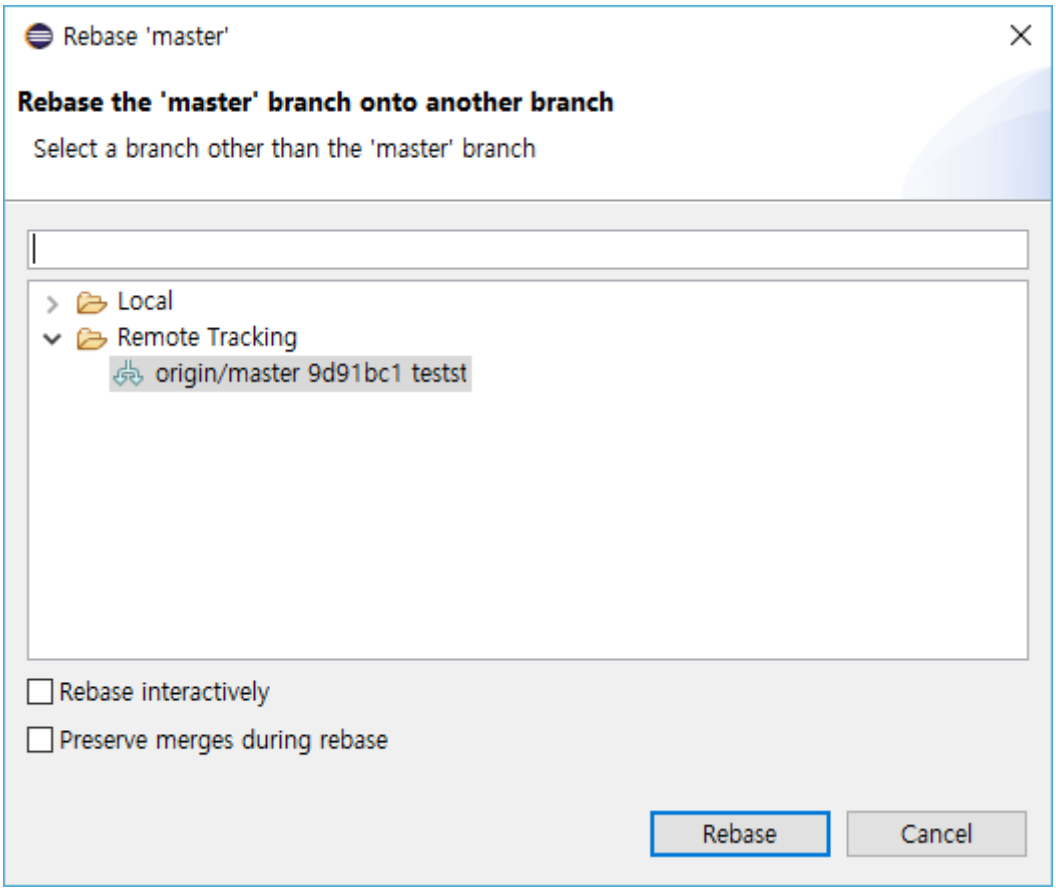
pull을 수행할 때 내가 수정한 파일과 원격지에서 내려받게되는 파일이 동일할 경우
충돌이 발생한다.(conflict)



[그림] conflict발생

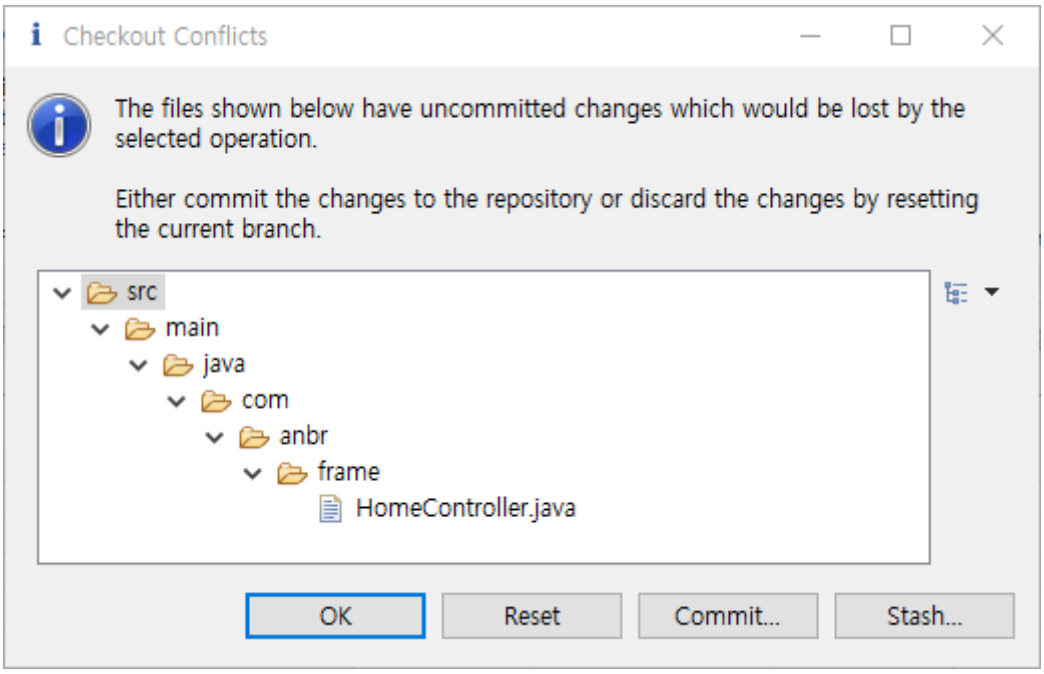
이경우 아래와 같이 조치한다.

[생성된 Project] 마우스오른쪽 버튼 -> Team -> Rebase..



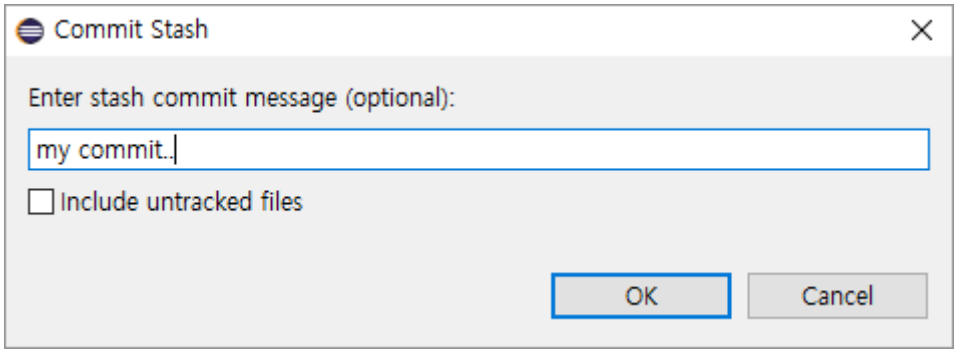
[그림] Rebase 1

위의 화면에서Rebase를 클릭



[그림] 충돌이 발생한 파일 확인

위 대화상자에서 충돌이 발생한 파일을 확인하고 Stash.. 버튼을 클릭하여 나의 변경 내용을 저장해둔다.

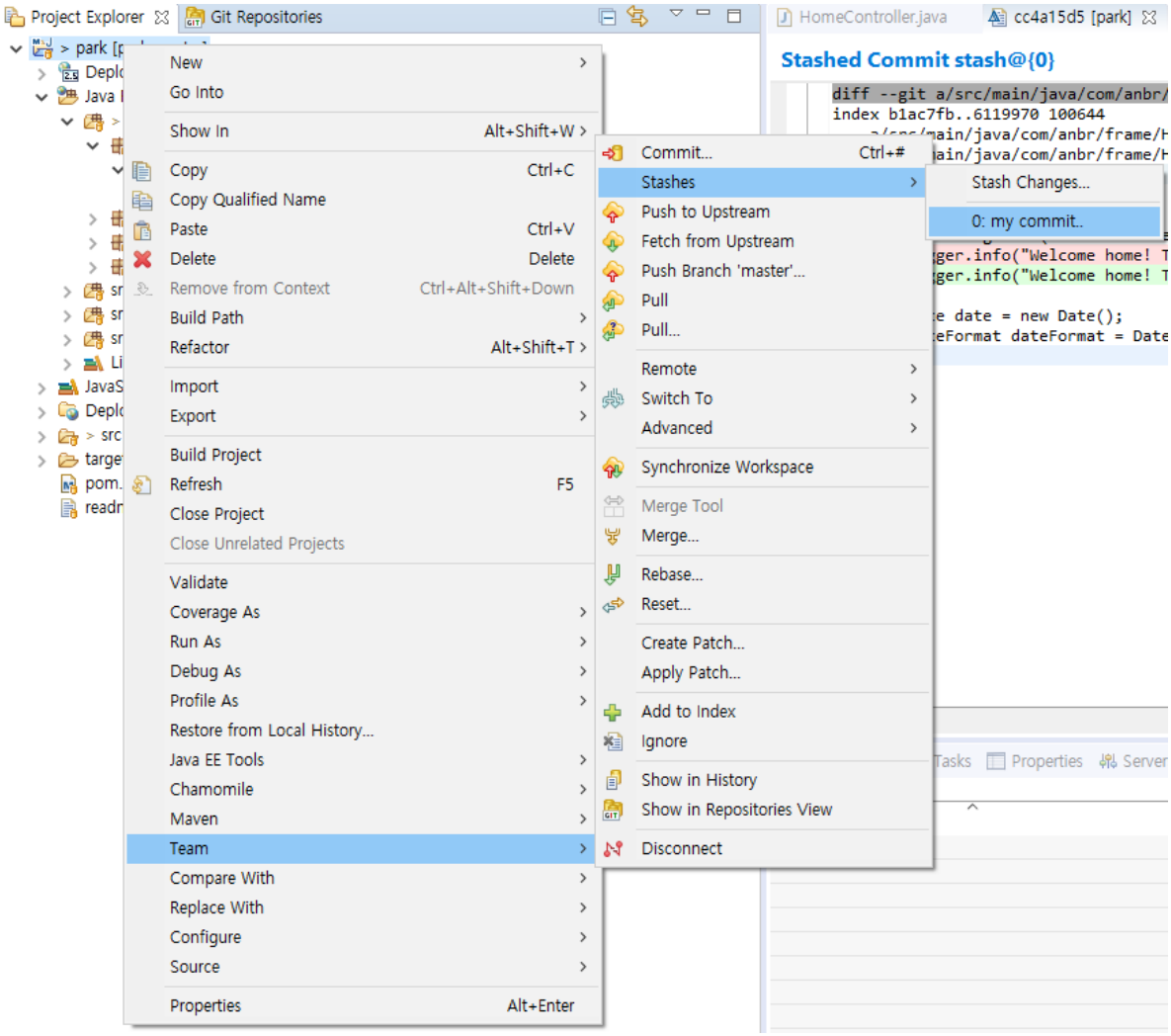


[그림] Stash저장

그다음 다시 pull을 한번더 수행하여 최신 상태로 적용 될 수 있도록 한다.

그리고 본인이 변경한 내용을 다시 적용시킨다.

[생성된 Project] 마우스오른쪽 버튼 -> Team -> Stashs를 클릭하여 방금 적용한 내용을 살펴본다.



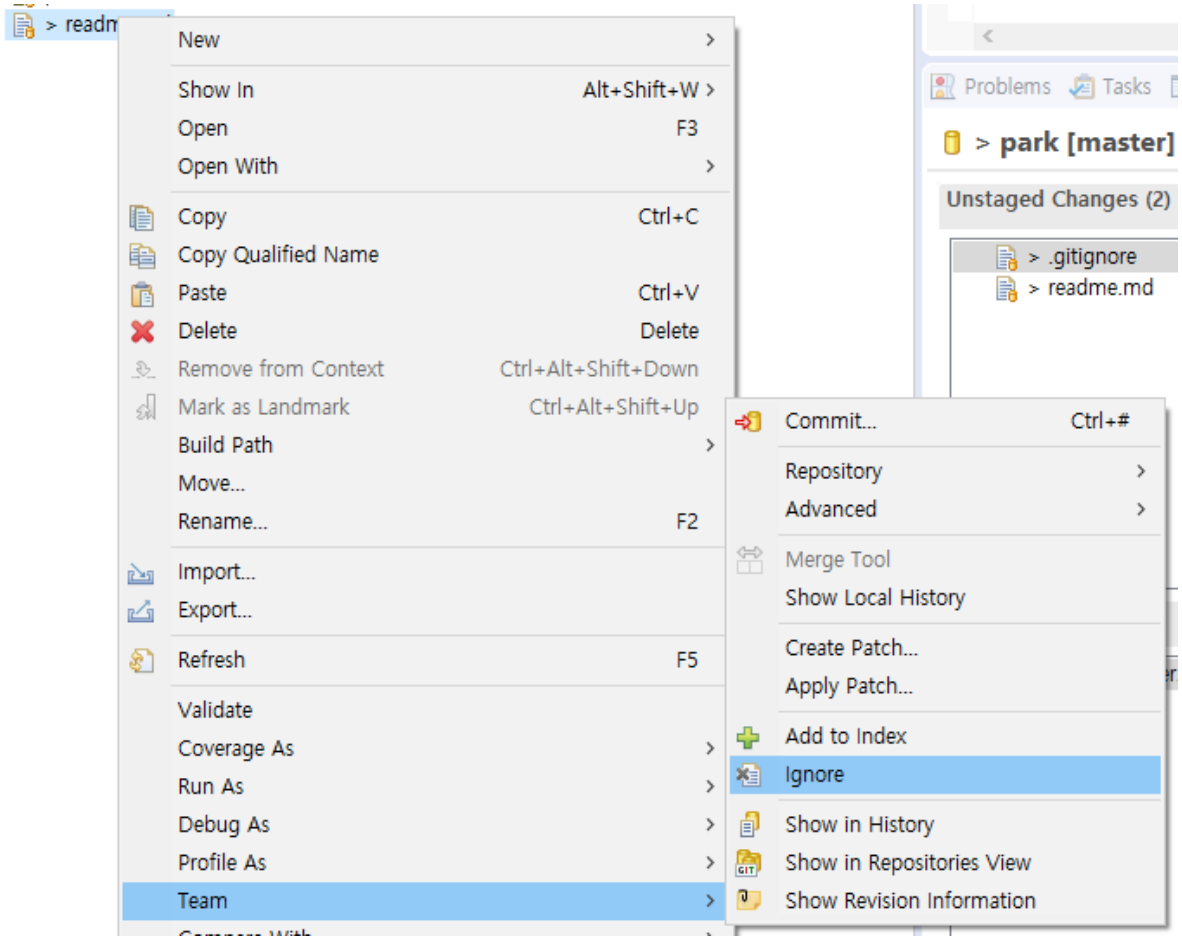
[그림] Stash 확인

파일이 열리면 오른쪽 상단의 (Apply Stashed Changes) 버튼으로 바로 적용하거나 변경된 내용을 하나씩 살펴보며 변경해주고 다시 Commit/push를 진행한다.

Ignore설정

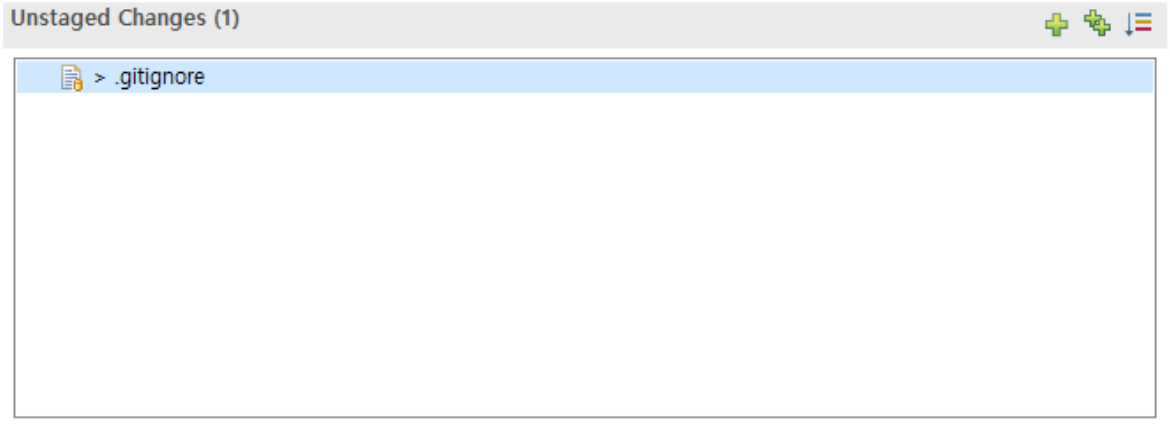
공동작업을 하다보면 이클립스 설정파일까지 push되어 다른 사용자가 해당 파일을 내려받았을 때 프로젝트가 정상적으로 동작하지 않는 경우가 있다. 원격저장소에는 항상 소스만 위치할 수 있도록 하며 설정파일들은 ignore처리하여 다른사용자에게 영향을 주지 않도록 한다.

Ignore처리할 파일/폴더에서 마우스오른쪽 버튼 -> Team -> Ignore



[그림] Ignore설정

위와 같이 할경우 .gitignore파일이 생성/변경되며 해당 파일을 서버에 push시켜준다.



[그림] .gitignore파일

.gitignore파일에 선택된 경로가 저장되어 다음에 해당 파일/폴더가 변경 되더라도 unstage영역에 노출되지 않는다.

.gitignore파일을 열어보면 아래와 같이 형상에 포함하지 않아야할 파일이나 폴더 목록이 나열되어있은 것을 확인 할 수 있다.

```
/target/
*.class
.settings/
.classpath
.project
/bin/
/logs/

*.jar
*.war
*.ear
/readme.md
```

PMD

정적 분석

정적 분석은 소프트웨어를 분석하는 방법의 하나로, 프로그램을 실행하지 않은 상태에서 코드 레벨에서 분석하는 방법이다. 올바른 형식의 소스 파일만 처리할 수 있기 때문에 컴파일 오류를 보고하지는 않으며, 개발자들이 표준 코드에 부합할 수 있는 코드를 작성하여 소스 코드 품질을 높이기 위해 사용한다.

PMD

PMD는 대표적인 오픈소스 기반 정적 소스 코드 분석기다. 사용되지 않는 변수, 비어 있는 try-catch 블록, 불필요한 객체 생성 등 전반적으로 프로젝트 내의 소스를 검토하여 다소 비효율적이거나 치명적일 수 있는 프로그래밍 결함을 찾는다.

PMD에는 내장 규칙 세트(룰셋)가 포함되어 있으며, 사용자 정의 규칙을 작성하는 기능을 제공한다.

주로 Java 및 Apex을 포함하여 6개의 다른 언어를 지원한다. 지원하는 언어는 Java, JavaScript, Salesforce.com Apex and Visualforce, Modelica, PLSQL, Apache Velocity, XML, XSL, Scala 등이다.

PMD 주요 점검 내용

PMD는 소스 코드를 점검하여 표준 코드 기준, 코드 안티패턴, CPD(Cut and Paste Detector)를 기준으로 위반 여부를 확인한다.

PMD 룰셋 (Ruleset)

룰셋은 PMD에서 실행될 규칙들을 설명하는 XML 파일이다. 개발자가 작성한 소스 코드를 검사하며, 위험 요인 및 오류를 발견하여 알려 주기 위한 기준이 된다.

번호	점검 항목	점검 내용	예시
1	Basic (rulesets/basic.xml)	대부분의 개발자들이 동의하는 규칙	catch 블록들은 비어있어서는 안 된다, equals()를 overriding 할 때 마다 hashCode()를 override한다.
2	Naming (rulesets/naming.xml)	표준 자바 naming 규약을 위한 테스트	변수 이름들은 너무 짧아서는 안 된다, 메소드 이름은 너무 길어서는 안 된다, 클래스 이름은 대문자로 시작해야 한다, 메소드와 필드 이름들은 소문자로 시작해야 한다
3	Unused code (rulesets/unusedcode.xml)	불필요한 코드 제거	결코 읽히지 않은 private field와 local variable, 접근할 수 없는 문장, 결코 호출되지 않는private method 등
4	Design (rulesets/design.xml)	디자인 원리 체크	switch 문장은 default 블록을 갖고 있어야 한다, 심하게 중첩된 if 블록은 피해야 한다
5	Import statements (rulesets/imports.xml)	import 문장에 대한 문제들점검	같은 클래스를 두 번 반입하는 것, java.lang에서 클래스를 import하는 것
6	JUnit tests (rulesets/junit.xml)	Test Case와 Test Method 관련 특정 문제 검색	Method 이름의 정확한 스펠링 등
8	Strings (rulesets/string.xml)	스트링 관련 작업을 할 때 발생하는 일반적인 문제들 규명	
9	Braces (rulesets/braces.xml)	for, if, while, else 문장이 괄호를 사용하는지 여부 검사	
9	Code size (rulesets/codesize.xml)	과도하게 긴 메소드, 너무 많은 메소드를 가진 클래스 검사	
10	Javabeans (rulesets/javabeans.xml)	직렬화 될 수 없는 bean 클래스같이 JavaBeans 코딩 규약을 위배하는 JavaBeans 컴포넌트 검사	
11	Finalizers	finalize() 메소드 관련한 다양한 문제들을 검사	비어있는 finalizer, 다른 메소드를 호출하는 finalize() 메소드,finalize()로의 호출 등
12	Clone (rulesets/clone.xml)	clone() 메소드에 대한 규칙	clone()을 오버라이드하는 클래스는 Cloneable을 구현해야 한다
13	Coupling (rulesets/coupling.xml)	클래스들간 과도한 커플링 표시 검색	지나치게 많은 import, 너무 적은 필드, 변수, 클래스 내의 리턴 유형 등
14	Strict exceptions (rulesets/ strictexception.xml)	예외 테스트	메소드는 java.lang.Exception을 발생 시키도록 선언되어서는 안 됨, 예외는 플로우 제어에 사용되어서는 안 됨
15	Controversial (rulesets/controversial.xml)	좀 더 의심스러운 검사	변수에 null 할당하기
16	Logging (rulesets/logging-java.xml)	java.util.logging.Logger를 위험하게 사용하는 경우 검색	끝나지 않고 정적이지 않은 logger와 한 클래스에 한 개 이상의 logger 등

PMD에서 기본적으로 제공하는 룰셋이 있다. 사용자가 개별 프로젝트를 위해 룰셋을 커스터마이징하여 사용할 수 있다. 룰셋을 커스터마이징하는 방법은 사용자 지정 룰셋 파일 생성 방법을 참고한다.

자바 룰셋 카테고리

1) Best Practices

사용자가 일반적으로 통용되는 모범 사례를 따를 수 있도록 하는 규칙이다. 파라미터 재사용 방지, 인터페이스 내의 상수 지정 방지, default가 없는 switch 문 방지, 사용하지 않는 private 메서드 제거 등이 있다.

2) Code Style

사용자가 특정한 코드 스타일을 따를 수 있도록 하는 규칙이다. 필드, 파라미터, 지역 변수, 메서드, 클래스의 네이밍 컨벤션 규칙과 사용하지 않는 괄호 삭제 등이 있다. 3) Design

사용자가 디자인 이슈를 발견할 수 있도록 하는 규칙이다. 메서드가 존재하지 않는 추상 클래스 방지, 널 포인터 예외 처리 방지 등이 있다.

4) Documentation

사용자가 코드 문서를 적절하게 작성할 수 있도록 하는 규칙이다. 보디가 비어 있는 메서드에 대해 주석을 권장하는 규칙, 너무 긴 주석 길이를 권장하지 않는 규칙 등이 있다.

5) Error Prone

발생하기 쉬운 오류를 방지할 수 있도록 하는 규칙이다. 생성자 안에 오버라이드 할 수 있는 메서드를 호출하는 것을 권장하지 않는 규칙, 비어있는 try-catch 문 제거 등이 있다.

6) Multithreading

다중 스레드 실행 시 발생하기 쉬운 이슈를 방지할 수 있도록 하는 규칙이다. 스레드 그룹 사용을 권장하지 않는 규칙, Thread.run() 메서드를 권장하지 않는 규칙 등이 있다.

7) Performance

사용자에게 차선책을 요구하는 규칙이다. 새로운 Integer 객체 인스턴스를 권장하지 않는 규칙, short 타입 사용을 권장하지 않는 규칙 등이 있다.

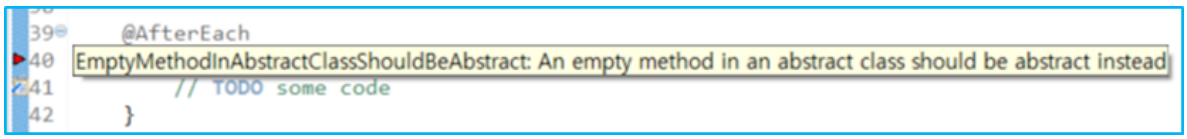
8) Security

잠재적인 보안 결함을 방지할 수 있도록 하는 규칙이다. 암호화 작업에 하드 코딩된 값을 권장하지 않는 규칙 등이 있다.

우선순위 (Priority)

PMD는 규칙의 우선순위를 1단계에서 5단계로 분류하며 값이 낮을수록 높은 위험성을 가지고 있다. PMD 실행 후 소스 코드 옆의 마커로 우선순위를 파악할 수 있다.

Symbol	Value	Name	PMD Name
	1	Blocker	High
	2	Critical	Medium High
	3	Urgent	Medium
	4	Important	Medium Low
	5	Warning	Low



속성 (Property)

PMD는 규칙을 XML 파일에서 직접 커스터마이징 할 수 있도록 속성값을 지정할 수 있다. 규칙을 바로 참조하여 사용하면 기본값으로 설정이 유지된다.

다음은 Error Prone에 해당하는 EmptyCatchBlock 규칙의 설명이다. 해당 규칙은 allowCommentedBlocks, allowExceptionNameRegex라는 두 가지 속성값을 가지고 있으며 아래 코드와 같이 <property>태그 안에서 속성값을 지정할 수도 있다.

This rule has the following properties:

Name	Default Value	Description	Multivalued
allowCommentedBlocks	false	Empty blocks containing comments will be skipped	no
allowExceptionNameRegex	^(ignored|expected)$	Empty blocks catching exceptions with names matching this regular expression will be skipped	no

Use this rule with the default properties by just referencing it:

```
<rule ref="category/java/errorprone.xml/EmptyCatchBlock" />
```

Use this rule and customize it:

```
<rule ref="category/java/errorprone.xml/EmptyCatchBlock">
    <properties>
        <property name="allowCommentedBlocks" value="false" />
        <property name="allowExceptionNameRegex" value="^(ignored|expected)$" />
    </properties>
</rule>
```

캐모마일 룰셋 (67건)

캐모마일에서 선정한 룰셋은 다음과 같다.

Blocker, Critical 단계(심각도 Medium High 이상)에 해당하는 Java Global Ruleset 선정 후, 통용되는 Medium, Medium Low, Low 단계의 규칙을 추가로 선정하였다. 룰셋 목록은 다음과 같다.

#	Rule	Priority	Type
1	AbstractClassWithoutAnyMethod	High	Design
2	AssignmentInOperand	Medium	Error Prone
3	AssignmentToNonFinalStatic	Medium	Error Prone
4	AvoidArrayLoops	Medium	Performance
5	AvoidAssertAsIdentifier	Medium High	Error Prone
6	AvoidBranchingStatementAsLastInLoop	Medium High	Error Prone
7	AvoidEnumAsIdentifier	Medium High	Error Prone
8	AvoidFileStream	High	Performance
9	AvoidLosingExceptionInformation	Medium High	Error Prone
10	AvoidMultipleUnaryOperators	Medium High	Error Prone
11	AvoidReassigningParameters	Medium High	Best Practices
12	AvoidSynchronizedAtMethodLevel	Medium	Multithreading
13	AvoidThrowingNullPointerException	High	Design
14	AvoidThrowingRawExceptionTypes	High	Design
15	AvoidUsingNativeCode	Medium High	Code Style
16	AvoidUsingShortType	High	Performance
17	AvoidUsingVolatile	Medium High	Multithreading
18	BooleanInstantiation	Medium High	Performance
19	BrokenNullCheck	Medium High	Error Prone
20	ByteInstantiation	Medium High	Performance
21	ClassNamingConventions	High	Code Style
22	ClassWithOnlyPrivateConstructorsShouldBeFinal	High	Design
23	ConstantsInInterface	Medium	Best Practices
24	ConstructorCallsOverridableMethod	High	Error Prone
25	DoNotCallGarbageCollectionExplicitly	Medium High	Error Prone
26	DoubleCheckedLocking	High	Multithreading
27	DuplicateImports	Medium Low	Code Style
28	EmptyCatchBlock	Medium	Error Prone
29	EmptyFinallyBlock	Medium	Error Prone
30	EmptyIfStmt	Medium	Error Prone
31	EmptyMethodInAbstractClassShouldBeAbstract	High	Code Style
32	EmptyStatementNotInLoop	Medium	Error Prone

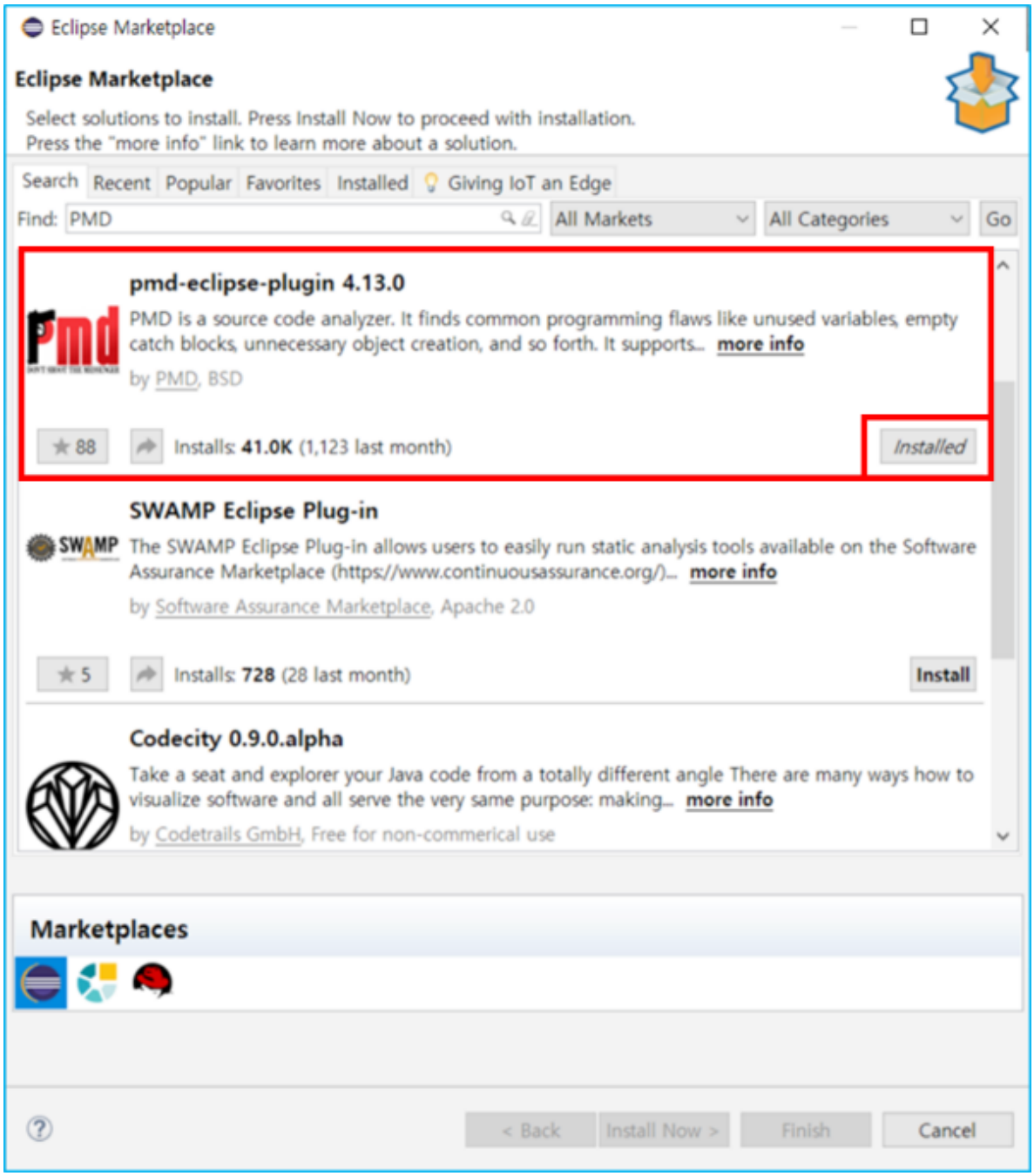
#	Rule	Priority	Type
33	EmptyTryBlock	Medium	Error Prone
34	EmptyWhileStmt	Medium	Error Prone
35	EqualsNull	High	Error Prone
36	FieldNamingConventions	High	Code Style
37	FinalFieldCouldBeStatic	Medium	Design
38	FormalParameterNamingConventions	High	Code Style
39	ImmutableField	Medium	Design
40	ImportFromSamePackage	Medium	Error Prone
41	InefficientEmptyStringCheck	Medium	Performance
42	InefficientStringBuffering	Medium	Performance
43	IntegerInstantiation	Medium High	Performance
44	LocalVariableNamingConventions	High	Code Style
45	LongInstantiation	Medium High	Performance
46	MethodNamingConventions	High	Code Style
47	MoreThanOneLogger	Medium High	Error Prone
48	ProperCloneImplementation	Medium High	Error Prone
49	ReturnEmptyArrayRatherThanNull	High	Error Prone
50	ShortInstantiation	Medium High	Performance
51	SimpleDateFormatNeedsLocale	Medium	Error Prone
52	SimplifyBooleanExpressions	Medium	Design
53	SingleMethodSingleton	Medium High	Error Prone
54	SingletonClassReturningNewInstance	Medium High	Error Prone
55	StringInstantiation	Medium High	Performance
56	StringToString	Medium	Performance
57	SuspiciousEqualsMethodName	Medium High	Error Prone
58	SwitchStmtsShouldHaveDefault	Medium	Best Practices
59	SystemPrintln	Medium High	Best Practices
60	UncommentedEmptyMethodBody	Medium	Documentation
61	UnnecessaryConversionTemporary	Medium	Error Prone
62	UnnecessaryWrapperObjectCreation	Medium	Performance
63	UnusedFormalParameter	Medium	Best Practices
64	UnusedPrivateField	Medium	Best Practices

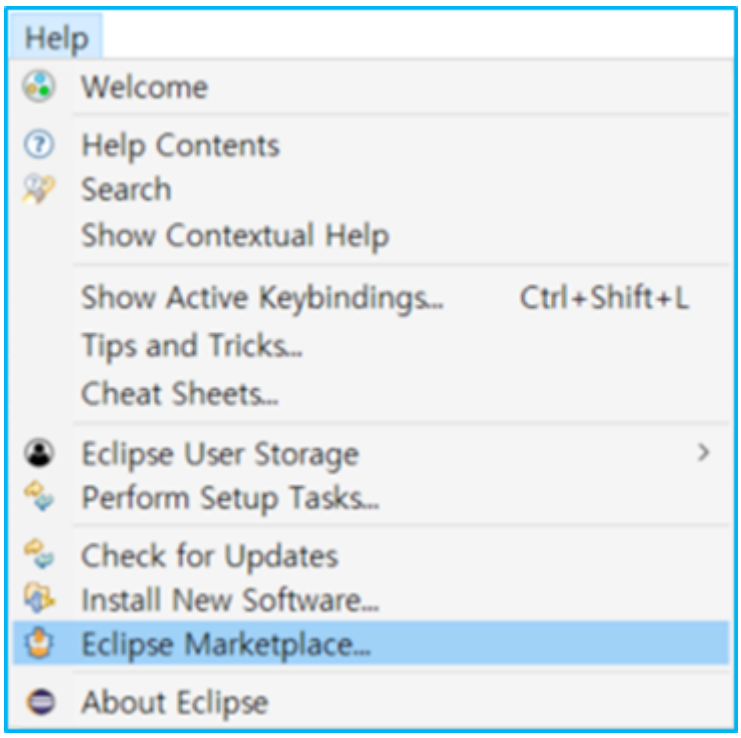
#	Rule	Priority	Type
65	UnusedPrivateMethod	Medium	Best Practices
66	UselessParentheses	Medium Low	Code Style
67	UselessStringValueOf	Medium	Performance

PMD 설치

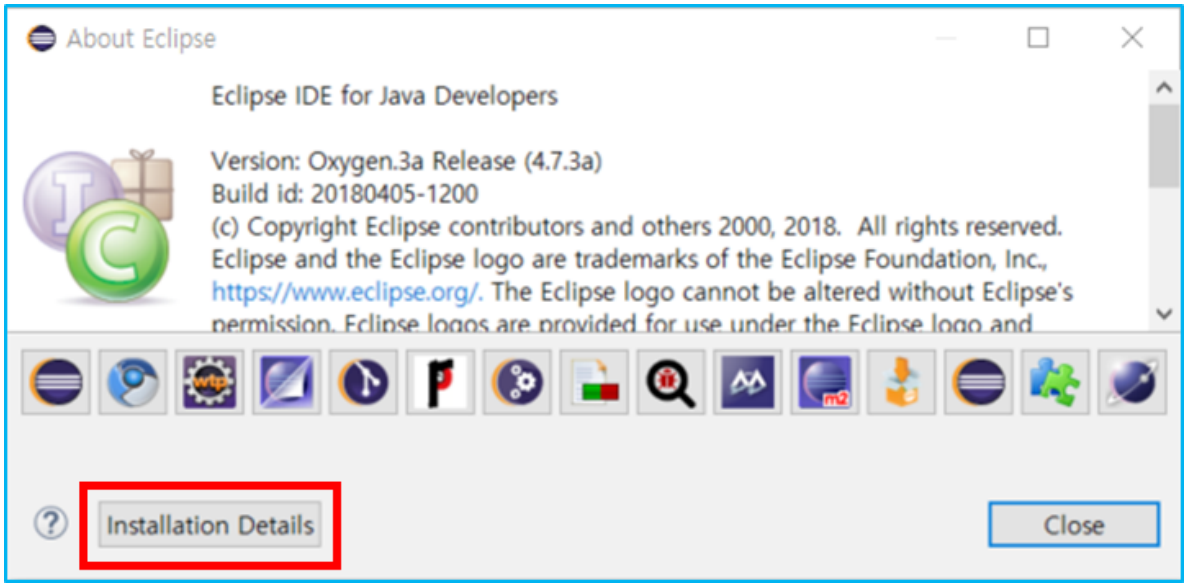
본 가이드는 이클립스 PMD 플러그인 설치 방법을 안내한다. 커맨드라인 사용 시 다음 링크를 참조한다.
https://sourceforge.net/projects/pmd/files/

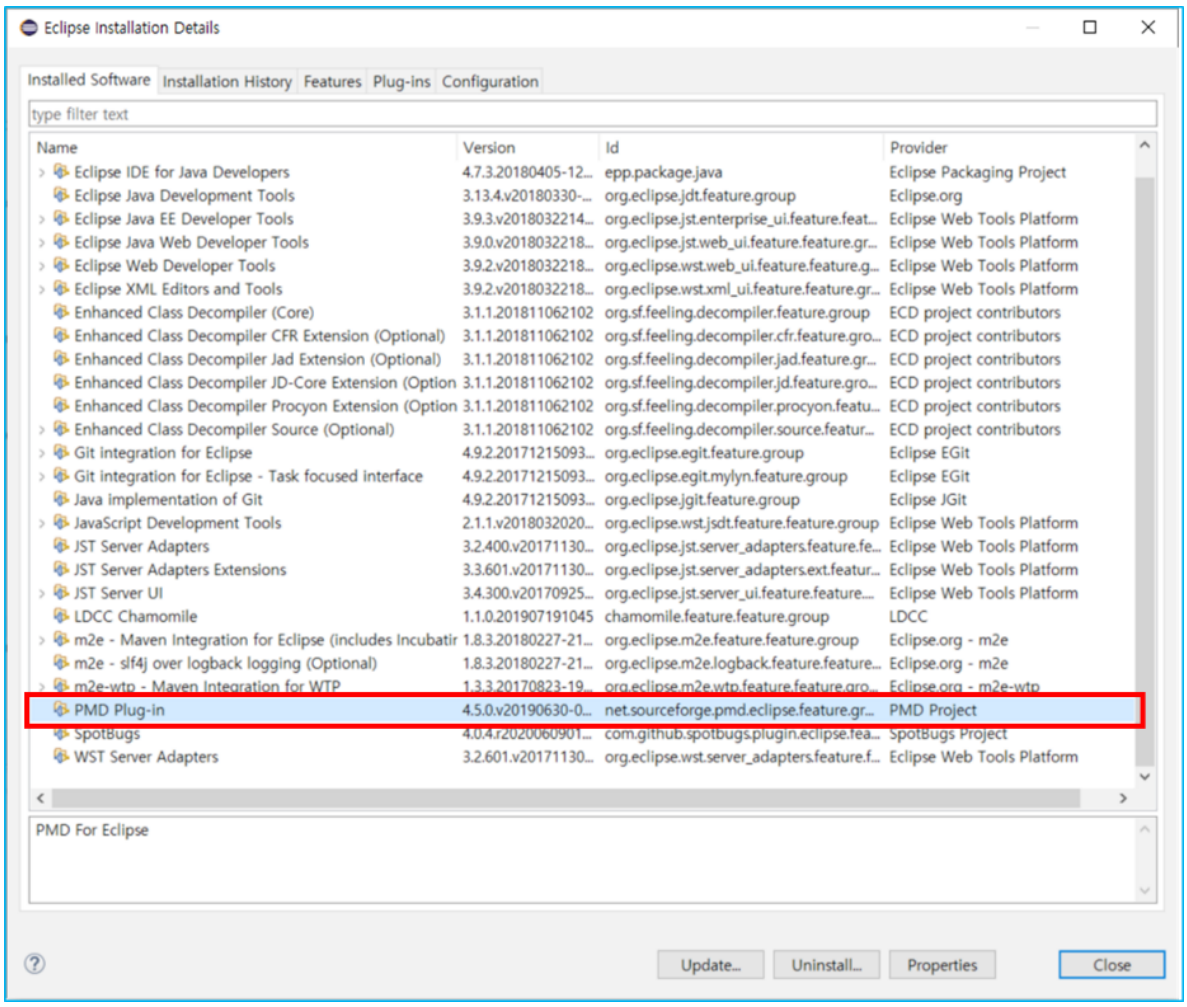
1) Help → Eclipse Marketplace에서 pmd-eclipse-plugin을 검색 후 설치한다.





2) Help → About Eclipse 클릭 후 Installation Details를 클릭한다.





3) Eclipse Installation Details에서 PMD Plug-in의 버전을 확인할 수 있다.

사용자 지정 룰셋 파일 생성 방법

기존 룰셋 참조

템플릿을 사용하여 비어있는 룰셋 파일을 작성한다.

```
<?xml version="1.0"?>
<ruleset name="Custom Rules"
    xmlns="http://pmd.sourceforge.net/ruleset/2.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://pmd.sourceforge.net/ruleset/2.0.0
https://pmd.sourceforge.io/ruleset_2_0_0.xsd">
    <description>
        My custom rules
    </description>
    <!-- Your rules will come here -->
</ruleset>
```

PMD가 제공하는 룰셋을 사용하려면 이에 대한 참조를 추가해야 한다. 기본적으로 제공하는 룰셋과 각 규칙에 대한 설명은 다음을 참고한다.
https://pmd.github.io/latest/pmd_rules_java.html

아래와 같이 검색한 규칙의 설명, 우선순위, 규칙이 정의된 클래스, 간단한 예시와 참조를 확인할 수 있다.

EmptyCatchBlock

Since: PMD 0.1

Priority: Medium (3)

Empty Catch Block finds instances where an exception is caught, but nothing is done. In most circumstances, this swallows an exception which should either be acted on or reported.

This rule is defined by the following XPath expression:

```
//CatchStatement
 [not(Block/BlockStatement)]
 [$allowCommentedBlocks != true() or Block/@containsComment = false()]
 [FormalParameter/Type/ReferenceType
   /ClassOrInterfaceType[@Image != 'InterruptedException' and @Image != 'CloneNotSupportedException']
 ]
 [FormalParameter/VariableDeclaratorId[not(matches(@Name, $allowExceptionNameRegex))]]
```

Example(s):

```
public void doSomething() {
    try {
        FileInputStream fis = new FileInputStream("/tmp/bugger");
    } catch (IOException ioe) {
        // not good
    }
}
```

This rule has the following properties:

Name	Default Value	Description	Multivalued
allowCommentedBlocks	false	Empty blocks containing comments will be skipped	no
allowExceptionNameRegex	^(ignored|expected)$	Empty blocks catching exceptions with names matching this regular expression will be skipped	no

Use this rule with the default properties by just referencing it:

```
<rule ref="category/java/errorprone.xml/EmptyCatchBlock" />
```

Use this rule and customize it:

```
<rule ref="category/java/errorprone.xml/EmptyCatchBlock">
    <properties>
        <property name="allowCommentedBlocks" value="false" />
        <property name="allowExceptionNameRegex" value="^(ignored|expected)$" />
    </properties>
</rule>
```

속성이 명시된 규칙의 경우 각 속성의 설명과 기본값, 그리고 커스터마이징할 수 있는 예시를 확인할 수 있다.

해당 라인을 ruleset에 추가하면 규칙 EmptyCatchBlock이 룰셋에 추가된다.

```
<rule ref="category/java/errorprone.xml/EmptyCatchBlock" />
```

대량으로 추가하고 싶다면 다음과 같이 특정 규칙을 제외한 나머지 전체 룰셋을 참조할 수도 있다.

참조할 때는 액세스 가능한 파일 시스템 경로 혹은 상대 경로를 사용할 수 있다.

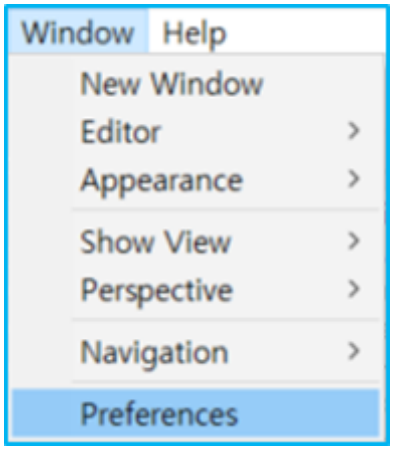
```
<rule ref="category/java/codestyle.xml">
    <exclude name="WhileLoopsMustUseBraces"/>
    <exclude name="IfElseStmtsMustUseBraces"/>
</rule>
```

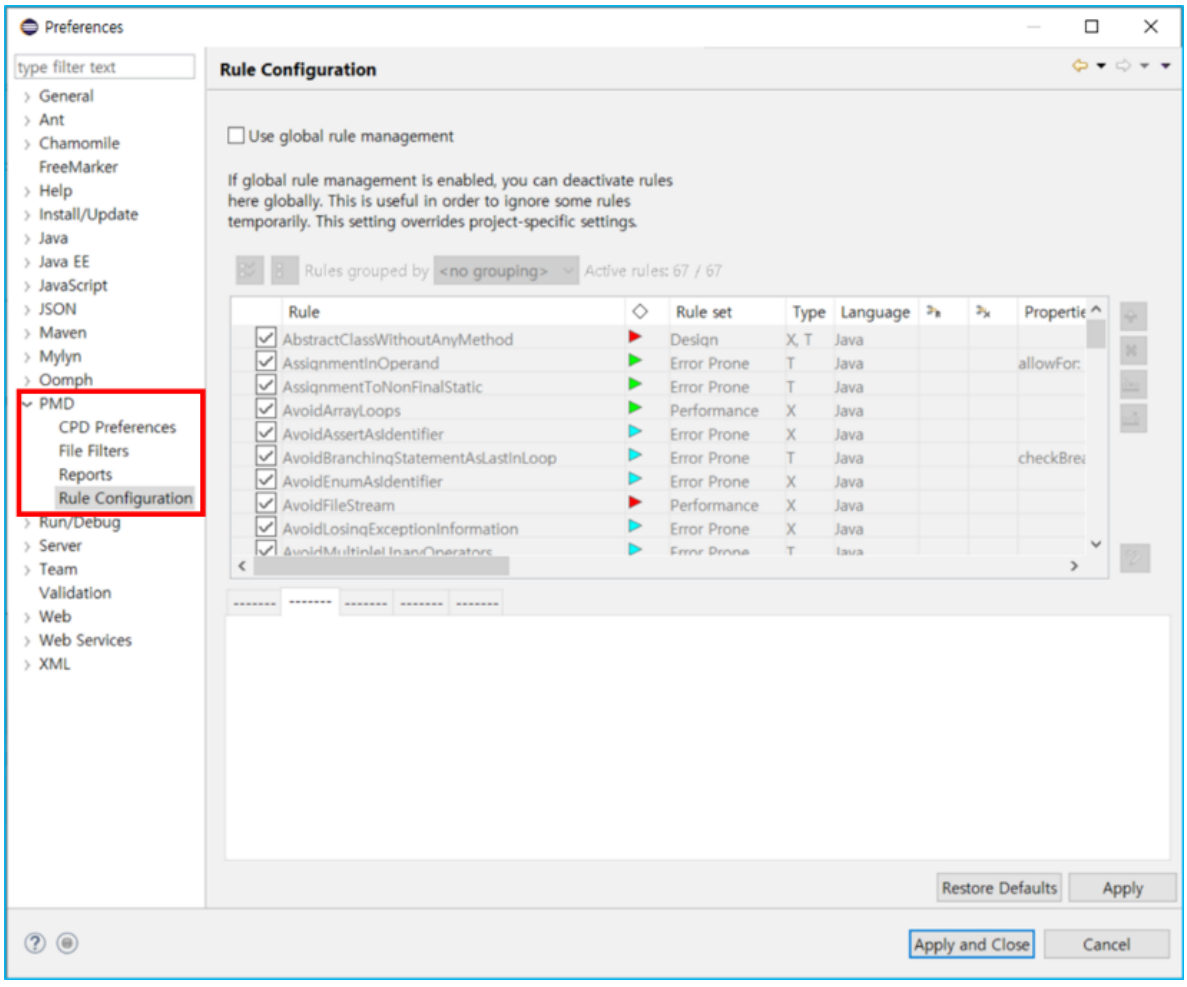
우선순위를 지정하거나 속성을 변경하는 등 더 자세한 룰셋 설정은 아래 링크를 참고한다.

https://pmd.github.io/pmd-6.9.0/pmd_userdocs_configuring_rules.html

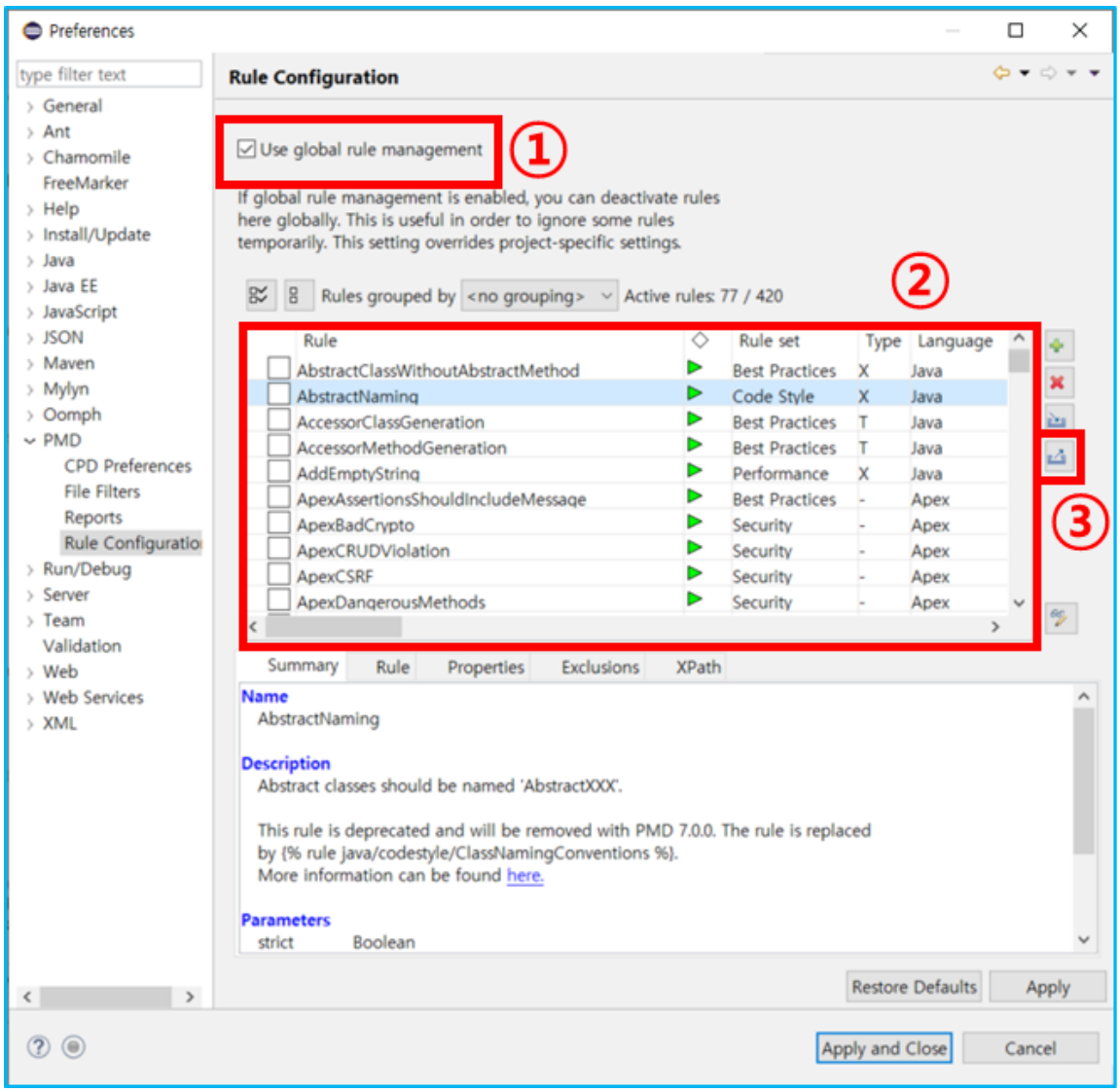
특정 룰만 추출

1) Eclipse 메뉴에서 Windows → Preferences 선택 후 PMD → Rule Configuration을 클릭한다.





2) 상단의 Use global rule management를 체크하여 활성화된 룰셋을 확인한다. 규칙 선택 후 export 버튼을 클릭한다. 주의할 점은 추출할 규칙을 선택할 때 체크 표시가 아닌 규칙 항목 자체를 클릭해야 한다. 다중 선택할 경우 Ctrl 버튼으로 한 항목씩 여러 번 선택, 혹은 Shift 버튼으로 다중 항목 선택 후 export 버튼을 클릭한다.



새로운 룰 생성

PMD는 기존에 제공하던 룰셋 이외에도 Java, XPath를 이용한 두 가지 방법으로 룰을 새로 정의할 수 있다. 새로 생성할 룰은 참조되기 전에 룰셋에 정의되어 있어야 한다.

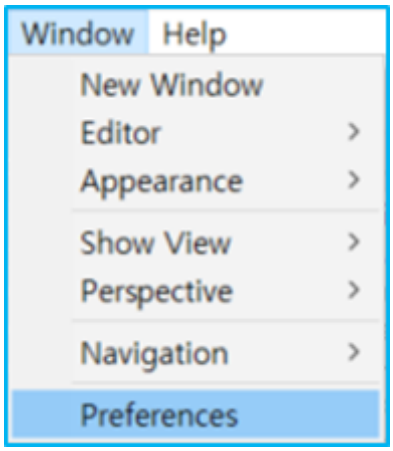
자세한 사항은 아래 링크를 참조한다.
https://pmd.github.io/pmd-6.9.0/pmd_userdocs_extending_writing_pmd_rules.html

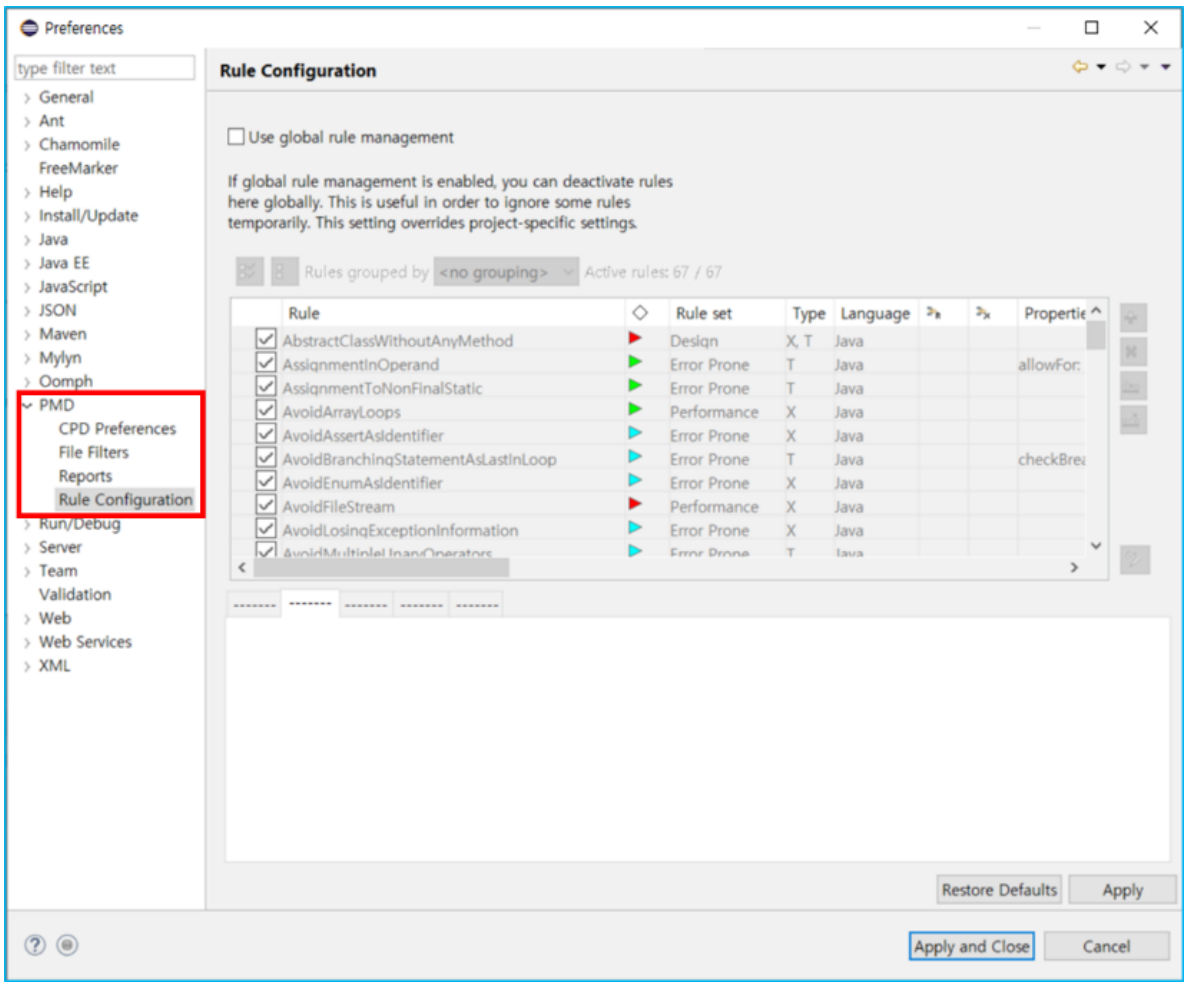
---

PMD 사용

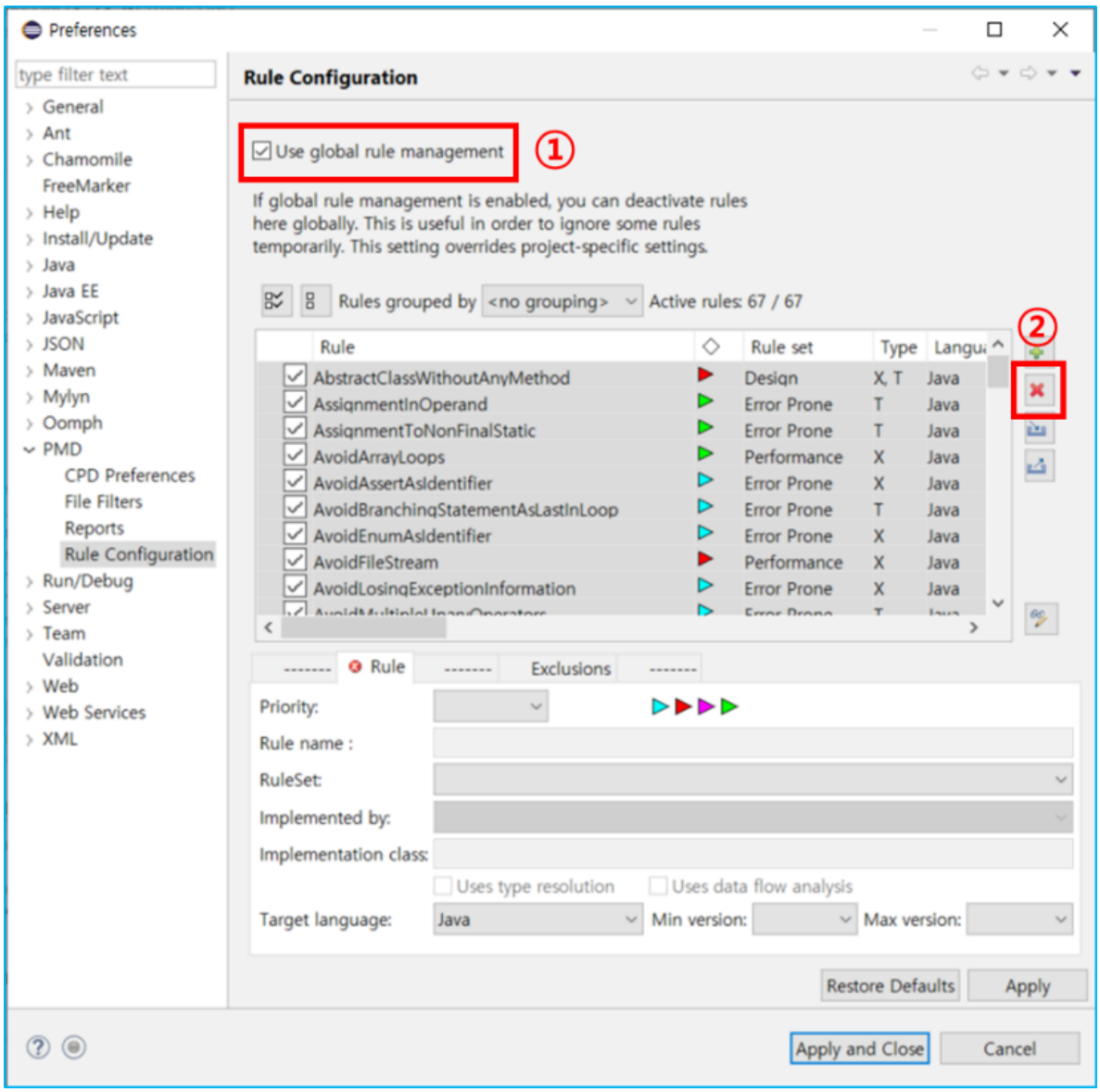
이클립스 플러그인으로 분석

1) Eclipse 메뉴에서 Windows → Preferences 선택 후 PMD → Rule Configuration을 클릭한다.

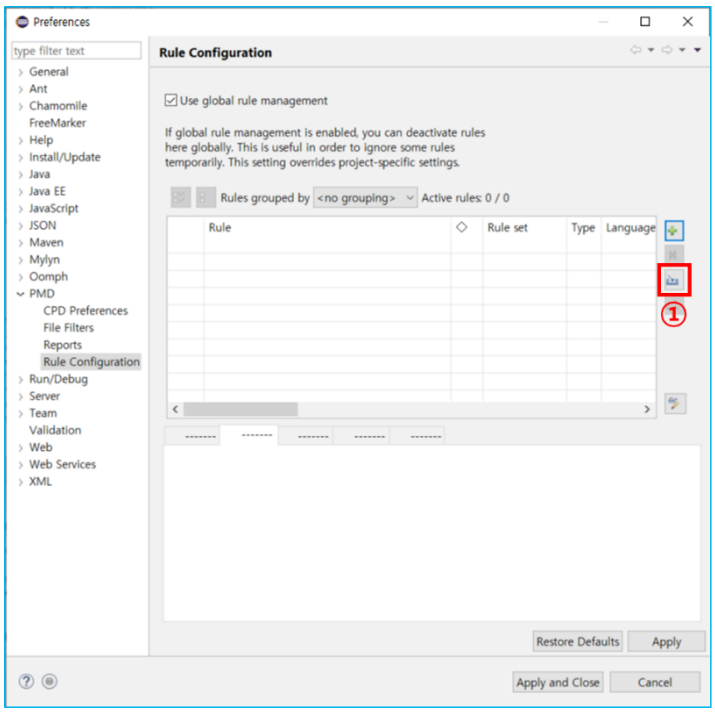


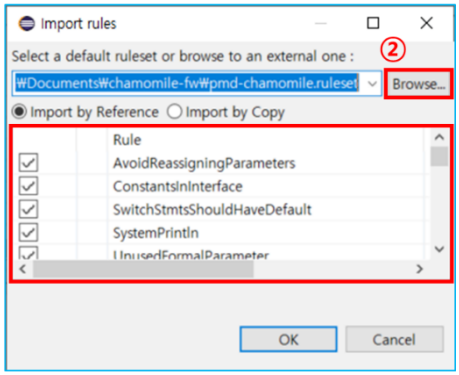


2) 상단의 Use global rule management를 클릭하여 활성화한 후 X 버튼을 눌러 모든 룰셋을 삭제한다.

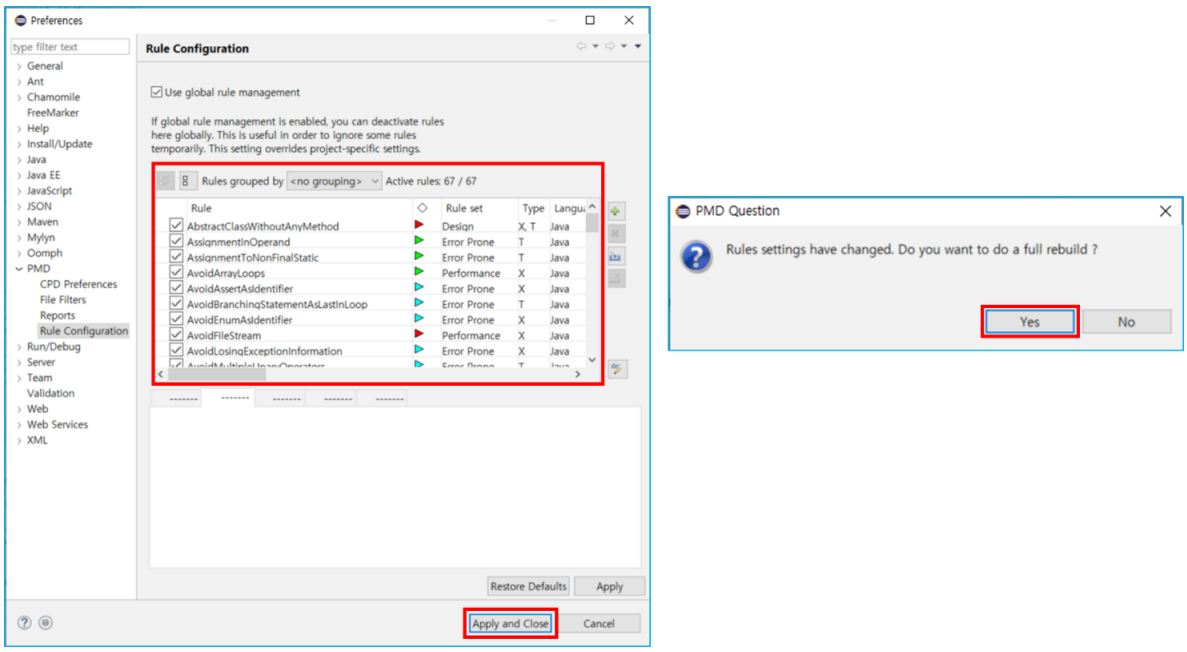


3) Import 버튼을 클릭한다. Browse 버튼을 클릭하여 룰셋 파일을 선택한다.

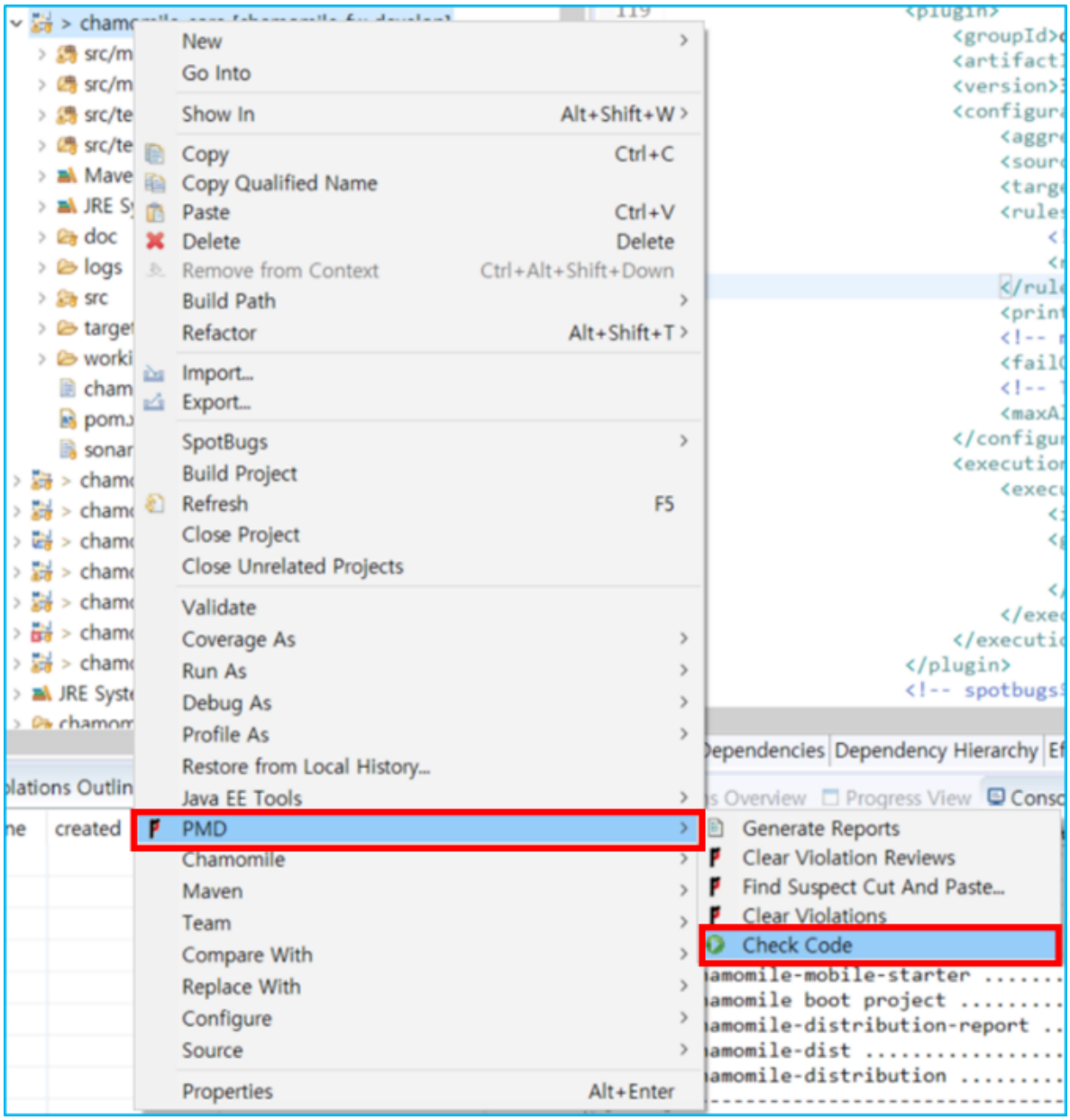




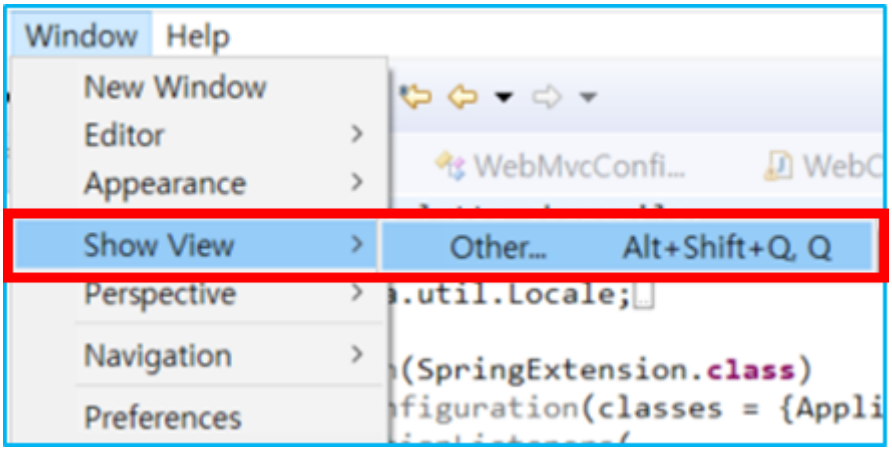
4) 모든 룰셋을 체크하고 하단의 Apply and Close 버튼을 클릭하면 자동으로 프로젝트를 재빌드한다.

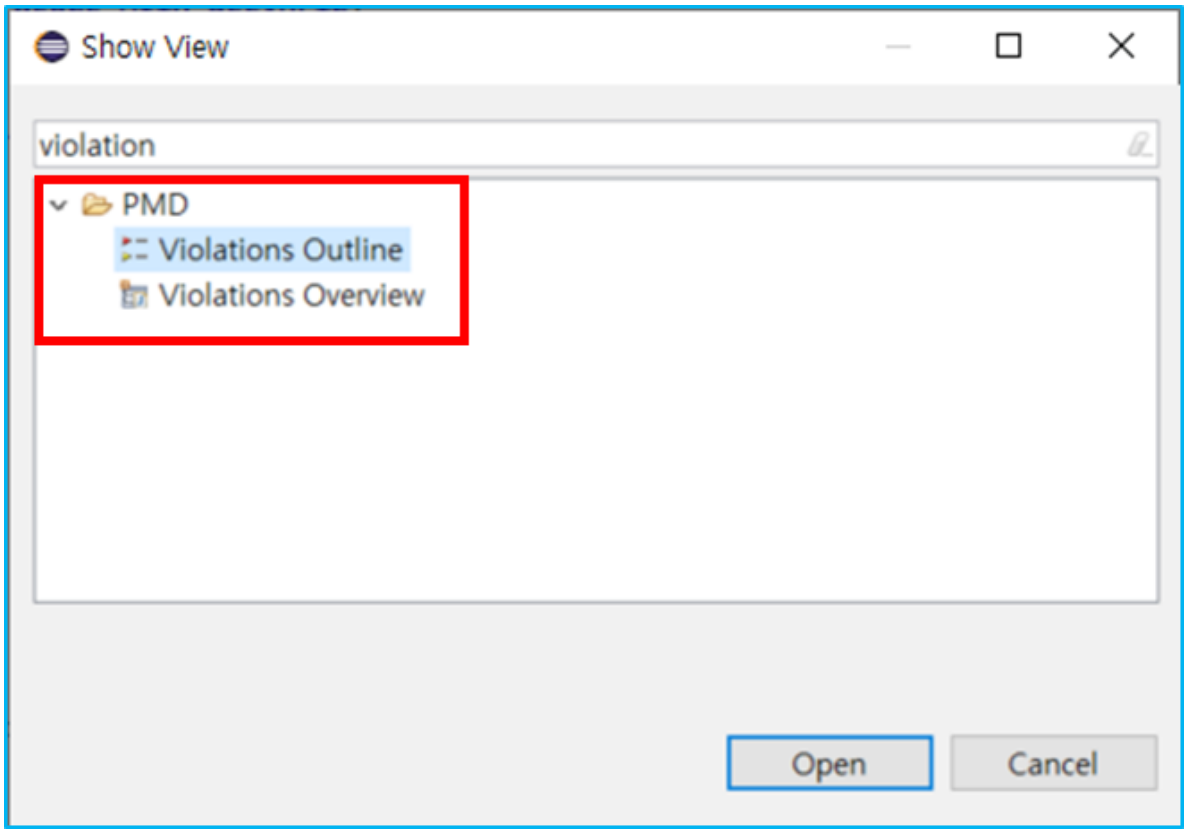


5) 적용할 프로젝트를 우클릭 후 PMD → Check Code를 클릭한다.

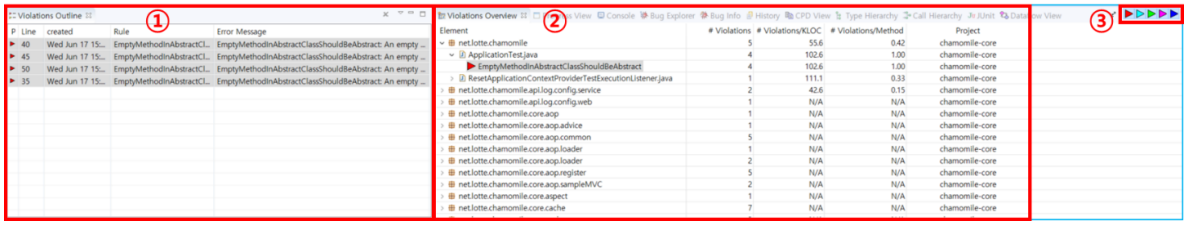


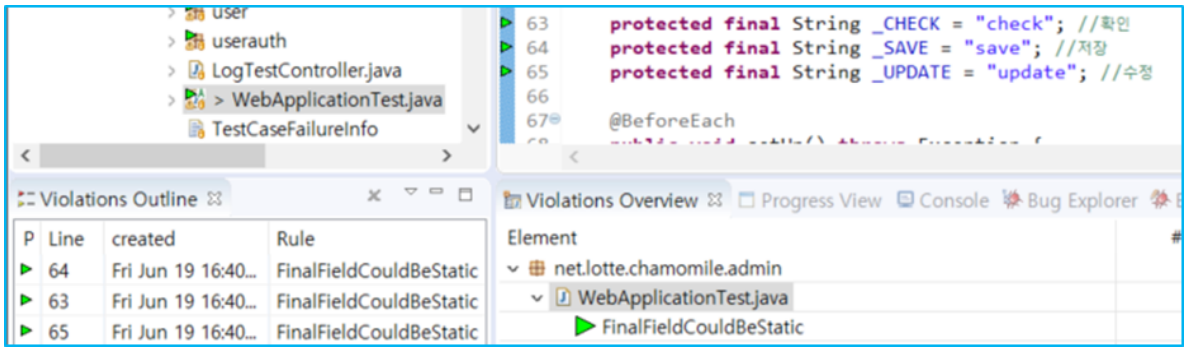
6) 코드 분석이 완료되면 PMD 뷰로 전환되며 규칙에 위반된 코드를 표시한 결과가 나타난다. 뷰가 전환되지 않는다면 Window → Show View → Other를 클릭하여 Violations Outline, Violations Overview를 선택하면 분석 결과를 확인할 수 있다.



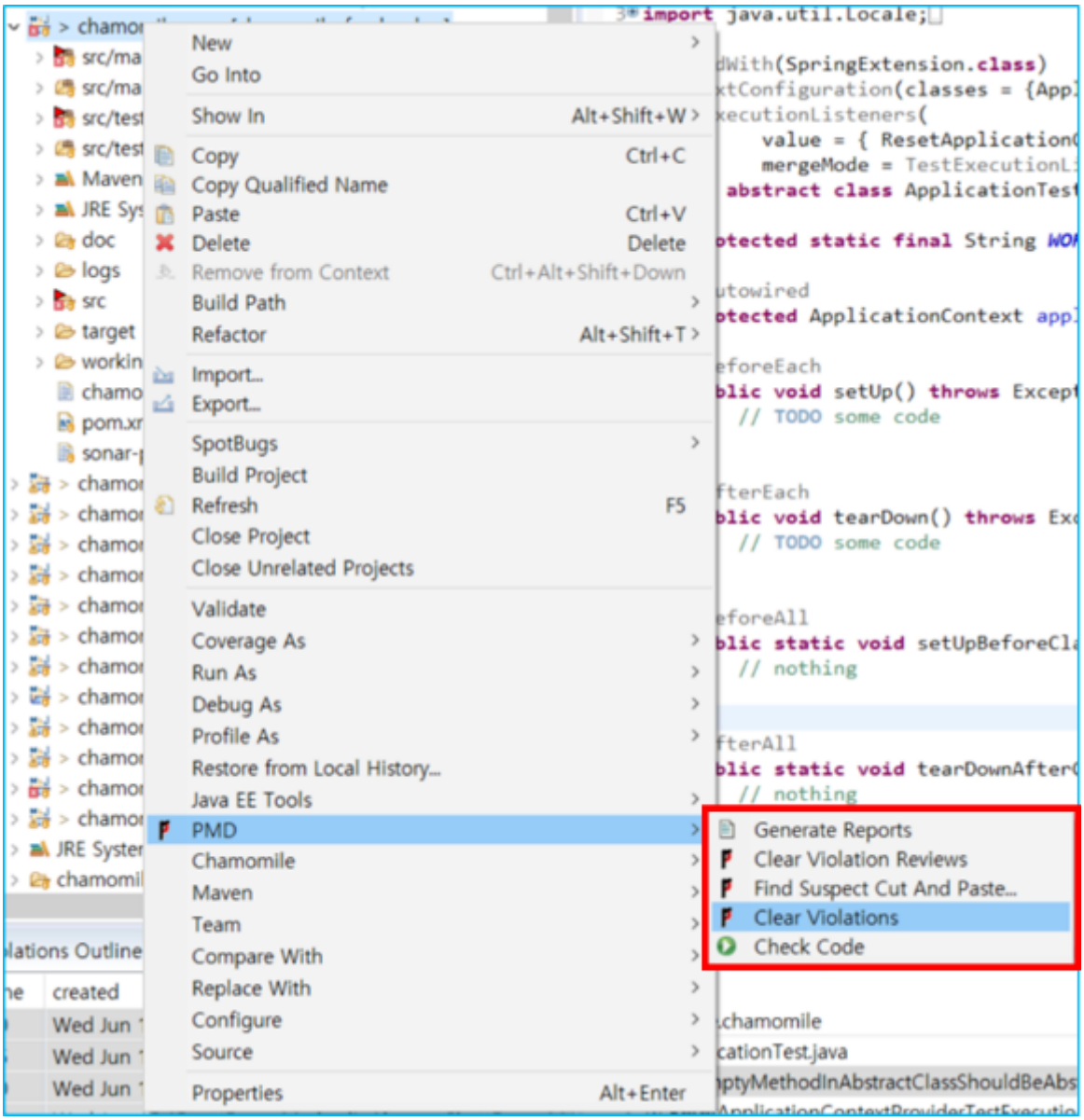


7) Violations Outline과 Violations Overview에서 파일 이름과 에러 메시지 등을 확인할 수 있으며 코드에도 표시된다. Violations Outline에서는 간략한 위반 규칙 내용을 확인할 수 있다. Violations Overview에서는 위반된 소스 코드의 패키지명, 각 파일의 위반 사항 수, 라인당 위반 수, 그리고 메소드당 위반 수를 볼 수 있다. 오른쪽 상단의 마커를 클릭하여 우선순위에 따른 Violation을 확인할 수 있다.





8) PMD 비활성화 시 프로젝트 우클릭 후 PMD → Clear Violations, Clear Violation Reviews를 클릭한다.



Maven으로 분석

1) Maven 사용 시 프로젝트 pom.xml 파일에 maven-pmd-plugin을 추가한다. 기본적인 포맷은 다음과 같다. Maven 환경 설정에 대한 세부 사항은 다음 링크를 참고한다.
https://maven.apache.org/plugins/maven-pmd-plugin/usage.html

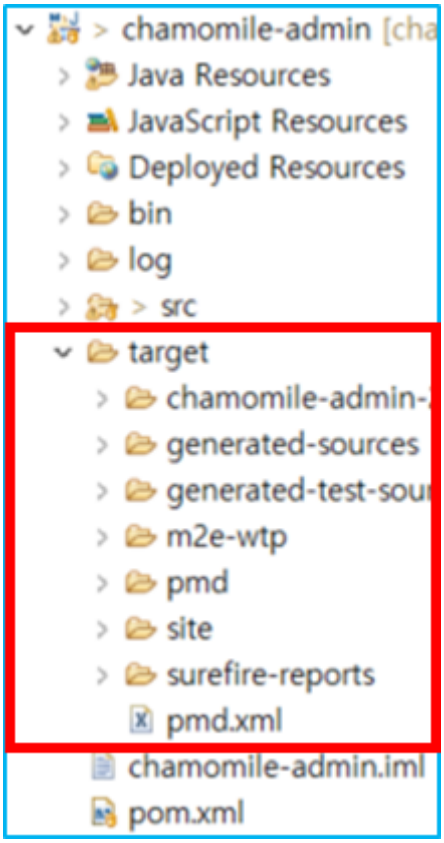
```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.13.0</version>
        <configuration>
          <linkXref>true</linkXref>
          <sourceEncoding>utf-8</sourceEncoding>
          <minimumTokens>100</minimumTokens>
          <targetJdk>1.5</targetJdk>
          <excludes>
            <exclude>**/*Bean.java</exclude>
```

```
            <exclude>**/generated/*.java</exclude>
          </excludes>
          <excludeRoots>
            <excludeRoot>target/generated-sources/stubs</excludeRoot>
          </excludeRoots>
        </configuration>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

2) 특정 룰셋 지정 시 마찬가지로 <configuration></configuration> 안에 명시한다. 프로젝트 구조에 따라 경로가 변하기 때문에 파일 위치 작성에 주의한다. 하단의 예시를 참고한다.

```
<rulesets>
  <!-- A rule set, that comes bundled with PMD -->
  <ruleset>/category/java/bestpractices.xml</ruleset>
  <!-- Custom local file system rule set -->
  <ruleset>d:\rulesets\strings.xml</ruleset>
  <!-- Custom remote rule set accessed via a URL -->
  <ruleset>http://localhost/design.xml</ruleset>
</rulesets>
```

3) 메이븐으로 pmd 실행 시 mvn pmd:pmd 명령어로 Run한다. 해당 프로젝트의 target 폴더에 pmd.xml 실행 결과 파일이 추가된다.



오탐 (False Positive)

PMD 의 경우 해당 라인에 //NOPMD 를 추가하면 해당 라인은 검사를 하지 않고 진행한다 소스 코드의 해당 라인을 예외 처리하는 것이기 때문에 해당 라인에서 발견되는 모든 보안 약점이 예외 처리된다.

구분	판정기준
정탐(True Positive)	취약점을 정확하게 탐지함
오탐(False Positive)	취약점이 해결된 내용을 탐지함
미탐(False Negative)	취약점을 탐지 못함

Chamomile Boot

Spring Boot기반의 경량 application이 제작 가능하도록 Chamomile Framework를 적용한 프레임워크.

Spring boot란?

Tomcat, Jetty 또는 Undertow등이 직접 임베드 되어 WAR파일을 사용하지 않고도 독립형 application으로 생성해주는 프레임워크 이다. 빌드 구성을 단순화 하기위해 starter종속성을 제공하며 이를 통해 Spring및 3rd party 라이브러리를 자동으로 구성해준다. 코드 생성 및 XML로 설정을 하지 않아도 되며 메트릭, 상태 확인 및 외부화 된 구성과 같은 프로덕션 지원 기능을 제공한다.

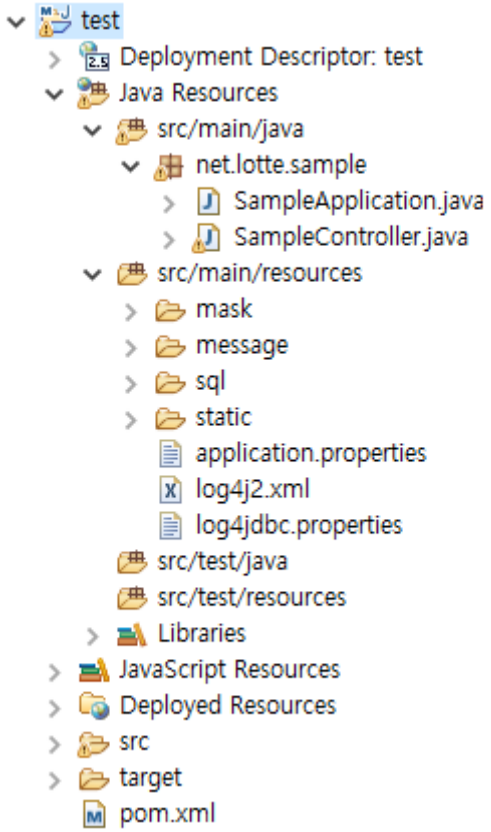
Quick Start

배포되는 IDE(eclipse)에서 새로운 프로젝트를 생성한다.

프로젝트 생성

File -> New -> Project -> Chamomile Framework -> Create Chamomile Project -> chamomile boot 관련 프로젝트 선택 후 완료

프로젝트 파일 구조



프로젝트 생성시 위 그림과 같이 기본적으로 구동시킬수 있는 형태의 프로젝트가 생성된다.

- SampleApplication.java
  - application을 구동시키는 클래스이다. 해당 파일을 열고 단축키 [ctrl + F11] 으로 application을 구동시킬 수 있다.
- SampleController.java
  - [/test] 주소로 호출 해 볼 수 있는 Controller이다.

- application.properties
  - application 구동에 필요한 property들이 설정 되어 있다.
- pom.xml
  - Chamomile Boot 를 이용하기 위한 starter가 정의 되어있다.

주요 파일 내용

SampleApplication.java

```
package net.lotte.sample;
import net.lotte.chamomile.autoconfigure.core.ChamomileApplication;
import net.lotte.chamomile.autoconfigure.core.ChamomileBootApplication;

//chamomile boot application을 구동하기위한 Annotation이다.
@ChamomileBootApplication
public class SampleApplication {
  //application을 구동하는 main method이다.
  public static void main(String[] args) {
    ChamomileApplication.run(SampleApplication.class, args);
  }
}
```

- component scan을 위한 base package명은 별도로 적어 주지 않아도 되며 해당 파일 하위로 위치시켜주기면 하면 자동으로 scan이 된다.

pom.xml

```
<properties>
    <chamomile.version>2.3.0-RELEASE</chamomile.version>
</properties>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>net.lotte.chamomile</groupId>
            <artifactId>chamomile-dependencies</artifactId>
            <version>${chamomile.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<repositories>
    <repository>
        <id>chamomile</id>
        <name>chamomile</name>
        <url>http://210.93.182.16:80/repository/maven-public/</url>
    </repository>
</repositories>

<dependency>
  <groupId>net.lotte.chamomile</groupId>
  <artifactId>chamomile-core-starter</artifactId>
</dependency>
<dependency>
```

```
  <groupId>net.lotte.chamomile</groupId>
  <artifactId>chamomile-common-starter</artifactId>
</dependency>
<dependency>
  <groupId>net.lotte.chamomile</groupId>
  <artifactId>chamomile-security-starter</artifactId>
</dependency>
```

- application에 필요한 설정들이 자동으로 처리 되도록 chamomile의 starter가 등록이 되어 있다.
- chamomile-core-starter : 개발자가 직접 개입하지 않아도 Framework에서 자동으로 처리 되는 설정들이 포함 되어있다.
- chamomile-common-starter : messageSource, fileutil 등 개발자가 직접 사용하는 컴포넌트들이 설정 되어 있는 starter이다.
- chamomile-security-starter : chamomile 인증/인가 처리를 할 수 있도록 자동으로 설정 되어 있는 starter이다.

Web 어드민 설정

어드민은 아래와 같은 설정을 변경할 수 있다.

- 데이터 베이스 설정
- Redis 설정
- 멀티 인스턴스 관리를 위한 웹 프로젝트 인증 설정
- 모니터링 (optional)
- 비밀번호 인코더 설정

데이터베이스 설정

어드민에서 관리하는 데이터에 접근하기 위한 데이터베이스 설정을 수행한다.

데이터베이스 설정 파일은 `{어드민 설치 디렉토리}/WEB-INF/classes/application.properties` 파일을 편집한다.

기본 설정은 아래와 같다.

- `chmm.jdbc.driverClassName`
- JDBC로 연결하기 위한 드라이버 클래스 이름을 명시한다.
- net.sf.log4jdbc.DriverSpy (SQL을 로깅하기 위해 고정된 값을 사용한다.)
- `chmm.jdbc.jdbc-url`
- 데이터베이스에 접근하기 위한 URL을 명시한다.
- JDBC 접근 URL 앞에 `jdbc:log4` prefix를 명시한다.
- `chmm.jdbc.username`
- 데이터베이스에 접근할 사용자 계정을 명시한다.
- `chmm.jdbc.password`
- 사용자 계정의 비밀번호를 입력한다.
- `chmm.jdbc.maxTotal`
- 최대 생성 스레드 개수
- `chmm.jdbc.connectionTimeout`
- 연결 시도시 최대 기다릴수 있는 시간
- `chmm.jdbc.maxLifetime`
- 풀에서 Connection 꺼낼때 최대 기다릴수 있는 시간

아래는 설정예시이다.

```
chmm.jdbc.driverClassName=net.sf.log4jdbc.DriverSpy
chmm.jdbc.url=jdbc:log4jdbc:mysql://127.0.0.1:3306/chamomile?autoReconnect=true
chmm.jdbc.username=user
chmm.jdbc.password=password
chmm.jdbc.maximumPoolSize=8
chmm.jdbc.connectionTimeout=30000
chmm.jdbc.maxLifetime=1800000
```

어드민은 DBCP를 이용하여 JDBC 커넥션을 관리하고 있으며, DBCP 설정을 추가하고자 한다면 `{어드민 설치 디렉토리}/WEB-INF/classes/spring/context-datasource.xml` 파일 내 dataSource 설정항목에 추가하여 사용한다.

이와 관련한 자세한 내용은hikariCP 설명을 참고한다.

> 캐모마일 **부트**에서는 context-datasource의 직접적인 수정없이 접두사 `chmm.jdbc`를 앞에 붙여 기본 설정 외의 설정값을 추가할수 있다.
>
> 예시
>
> ```
> #JMX management Beans에 등록되는 될지 여부를 지정한다
> chmm.jdbc.registerMbeans=true
> ```

Redis 설정

어드민은 성능향상(캐시)과 사용자가 개발한 응용프로그램(웹 프로젝트)을 관리하기 위해 Redis를 이용한다.

어드민 설치 시 기본적으로 내장 된 Redis를 사용하고 있으며, 로컬 개발 시 설치 된 어드민 수정없이 사용가능하다.

하지만 개발서버 혹은 운영 시 별도로 Redis를 설치하여 관리하도록 한다.

Redis 설정은 '{어드민 설치 디렉토리}/WEB-INF/classes/application.properties' 파일을 편집한다.

- `chmm.redis.hostName`
  - Redis가 설치 된 호스트 정보를 입력한다. (e.g. 127.0.0.1)
- `chmm.redis.port`
  - Redis에 접속가능한 포트를 입력한다. (e.g. 6379)
- `chmm.redis.usePool`
- Redis에 접속 된 Connection의 pool 사용여부를 입력한다.(`true` | `false`)
- `chmm.redis.embedded.enabled`
  - 내장 레디스의 활성화 여부를 입력한다. 기본값은 true이다. (`true` | `false`)

멀티인스턴스를 위한 인증 설정

어드민서버와 사용자가 개발한 어플리케이션과 통신을 위한 인증 모듈을 설정한다.

기본적으로 제공되는 모듈은 아래와 같다.

- 어드민서버와 개발한 어플리케이션이 세션 클러스터링이 된 경우
  - `net.lotte.chamomile.integration.service.api.SessionBasedAuthenticationHandler`
- 프레임워크에서 제공하는 인증 모듈을 사용하는 경우 (기본 설정)
  - `net.lotte.chamomile.integration.service.api.OncePerRequestAuthenticationHandler`
- 개발 된 어플리케이션으로 로그인 하기 위한 URL과 계정정보 변경은 '{어드민 설치 디렉토리}/WEB-INF/classes/application.properties' 파일을 수정한다.

인증 모듈을 변경하고자 하는 경우 '{어드민 설치 디렉토리}/WEB-INF/classes/spring/context-integration.xml' 파일의 integrationApiService 빈의 authenticationHandler 속성을 변경하면 된다.

사용자가 개발하는 어플리케이션이 별도의 인증 시스템을 사용하는 경우에는 별도의 인증 모듈을 추가하도록 한다.

인증모듈은
`net.lotte.chamomile.integration.service.api.IntegrationAuthenticationHandler`
인터페이스를 구현하도록 한다. 상세한 내용은 javadoc 문서를 참고한다.

모니터링 (Scouter 설정)

Scouter를 설치하여 모니터링을 수행하는 경우 Scouter의 Collector 서버의 정보를 입력해야 한다.

‘{어드민 설치 디렉토리}/WEB-INF/classes/application.properties’ 파일을 수정한다.

- `chmm.scouter.ip`
  - Scouter Collector 서버의 아이피
- `chmm.scouter.port`
  - Scouter Collector 서버의 Web API port
- `chmm.scouter.timeLength`
- 현재시간 기준으로 데이터를 조회하기 위한 시간 설정 (단위: 초(sec))

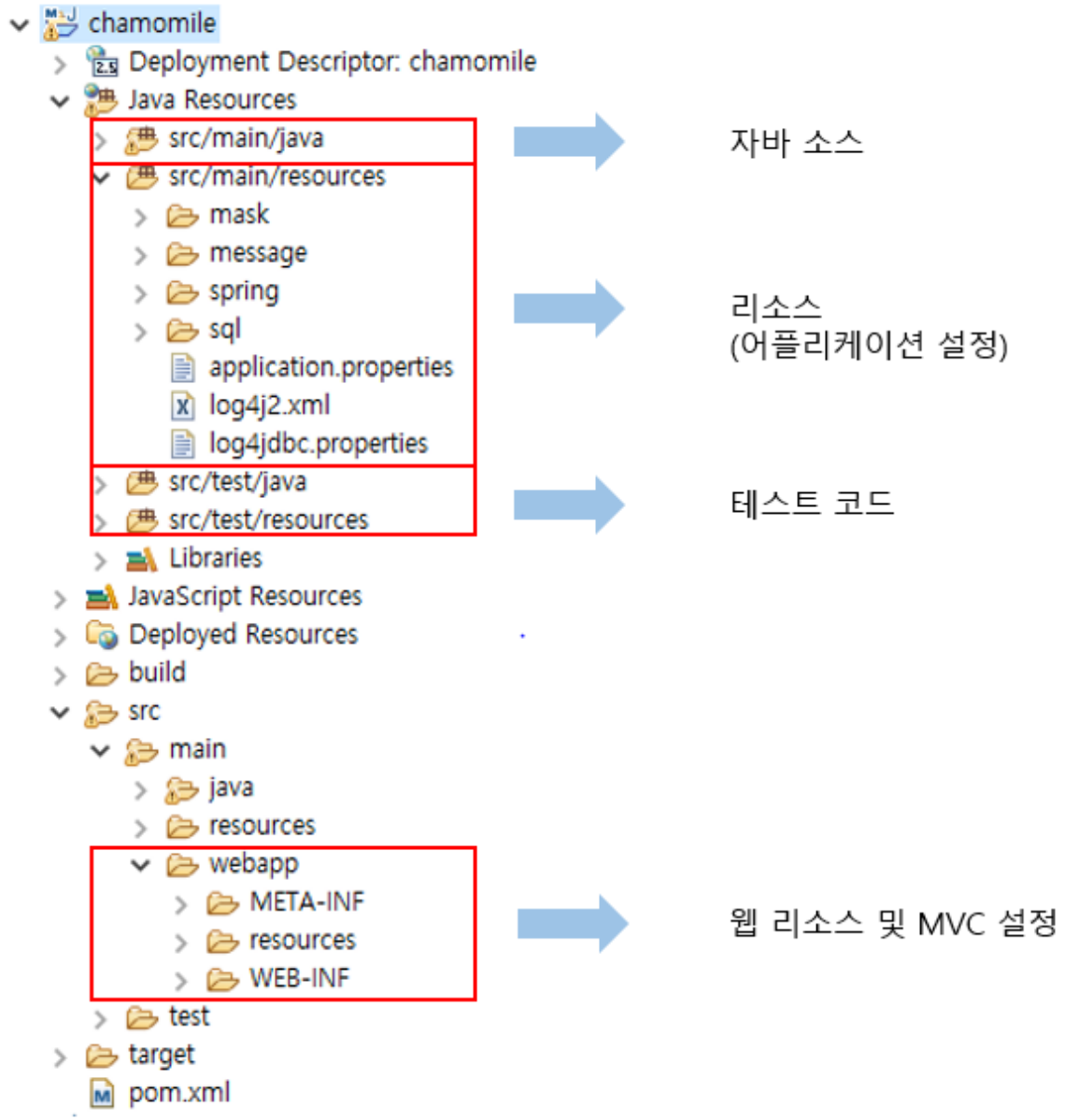
사용자 계정 비밀번호 인코더 설정

사용자 계정 비밀번호를 인코딩할 인코더의 알고리즘을 설정한다. 추가로 캐모마일에서는 스프링에서 제공하는 noop, bcrypt, pbkdf2, scrypt,ldap,sha256,argon2 알고리즘을 제공한다.

- `chmm.security.defaultPasswordEncoder`
- 기본 패스워드 인코더 알고리즘명 (기본은 sha256)
- `chmm.security.passwordEncoderList`
- 사용할 패스워드 인코더 알고리즘 목록 (기본은 sha256 하나이다)

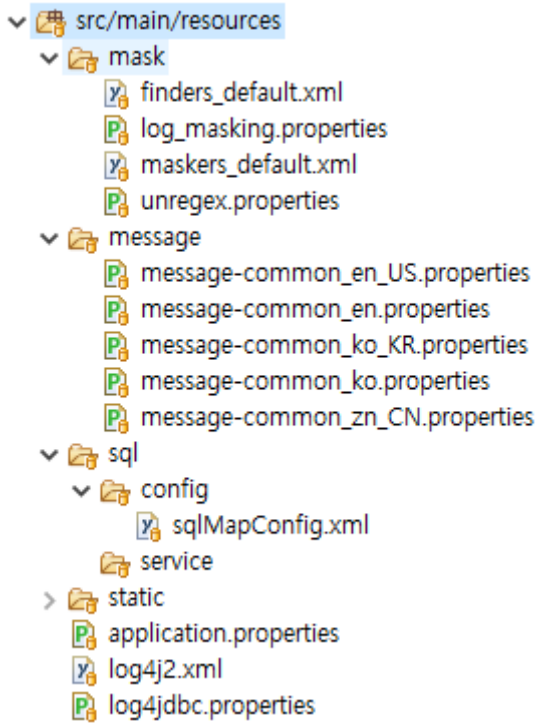
Web 프로젝트 설정

개발도구를 통해 생성된 웹 프로젝트는 아래와 같다.



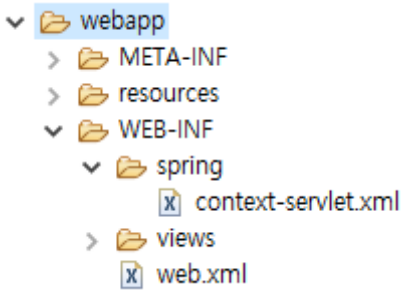
프로젝트 구조

웹 프로젝트의 전반적인 디렉토리 구조 및 파일의 용도는 아래와 같다. (이탤릭체 항목은 캐모마일 부트에는 해당되지 않으며, 밑줄 항목은 Webflux에서 사용되지 않음)


src/main/resources
mask
finders_default.xml
log_masking.properties
maskers_default.xml
unregex.properties
message
message-common_en_US.properties
message-common_en.properties
message-common_ko_KR.properties
message-common_ko.properties
message-common_zn_CN.properties
sql
config
sqlMapConfig.xml
service
static
application.properties
log4j2.xml
log4jdbc.properties

디렉토리 및 파일			설명
mask	finders_default.xml		로그마스킹 처리시 참조하는 정규표현식 패턴에 대해서 정의해 놓은 파일
	log_masking.properties		로그마스킹 정규표현식 패턴 매칭한 문자열에 대해서 치환할 대체문자열 형태를 정의해놓은 프로퍼티 파일
	maskers_default.xml		로그마스킹 관련 실행 클래스 정의 및 로그마스킹 패턴 정보에 대한 타켓 (XML, DB), 프로퍼티 파일에 대한 정의된 XML 파일 (최초 로그마스킹 API 실행시 로드되는 XML 파일)
	unregex.properties		특정 문자열에 대해 마스킹 처리할 경우 문자열을 등록할 수 있는 프로퍼티 파일
application.properties			- 스프링부트 설정(부트 한정) - 메서드 트레이스 AOP on/off 설정 - 파일업로드/다운로드 관련 설정(파일 저장 경로 등) - DB 연동 정보 설정 - property util 설정 (테이블과 컬럼 정보입력) - 알림톡 및 텔레그램 발송을 위한 연동 정보 설정 - Redis접속 정보 설정 - RequestMapping 정보와 DB 동기화 옵션 설정 - 모니터링 (scouter) 설정 - 멀티인스턴스 설정 - SMTP 서버 관련 설정
config	message		다국어 처리시 사용될 properties파일이 있는 디렉토리
spring(스프링 한정)	*context-aoplog.xml*		메서드 트레이스(이력로그) 관련 bean 설정파일
	context-cache.xml		스프링 캐시 (Ehcache, redis) 관련 bean 설정 파일
	context-common.xml		공통 bean (다국어, 로케일 등) 설정 파일
	context-datasource.xml		DB 관련 bean 설정 파일 (Datasource, Transaction 등)
	context-file.xml		파일업로드/다운로드 관련 bean 설정 파일
	context-integration.xml		멀티인스턴스 bean 설정 파일

디렉토리 및 파일			설명
	context-mapper.xml		Mybatis 관련 bean 설정 파일
	context-redis.xml		Redis 관련 bean 설정 파일
	context-security.xml		인증/인가 관련 bean 설정 파일
sql	config	sqlMapConfig.xml	sqlSessionFactory에서 사용하는 configuration
	service		Mybatis 쿼리 디렉토리



디렉토리 및 파일			설명
resources			정적 리소스 디렉토리(js, html, 이미지 등)
WEB-INF	*spring*	*context-servlet.xml*	Spring MVC 설정 파일
	views		jsp 디렉토리
	web.xml		웹어플리케이션 설정 파일

환경 설정 (어플리케이션 설정)

어플리케이션의 기본적인 환경설정 파일은 src/main/resources/application.properties 파일에 위치하며, 설정은 아래와 같다.

데이터베이스 설정

데이터에 접근하기 위한 데이터베이스 설정을 수행한다.

- `chmm.jdbc.driverClassName`

    - JDBC로 연결하기 위한 드라이버 클래스 이름을 명시한다.
    - `net.sf.log4jdbc.DriverSpy` (SQL을 로깅하기 위해 고정된 값을 사용한다.)
- `chmm.jdbc.jdbc-url`
    - 데이터베이스에 접근하기 위한 URL을 명시한다.
    - JDBC 접근 URL 앞에 `jdbc:log4` prefix를 명시한다.
- `chmm.jdbc.username`
    - 데이터베이스에 접근할 사용자 계정을 명시한다.
- `chmm.Jdbc.password`
    - 사용자 계정의 비밀번호를 입력한다.
- `chmm.Jdbc.maximumPoolSize`
    - 최대 생성 스레드 개수
- `chmm.Jdbc.connectionTimeout`
    - 풀에서 Connection 꺼낼때 최대 기다릴수 있는 시간

AOP 설정

개인이력 및 통계 데이터 적재를 위해 로그를 수집하기 위한 설정이 존재한다.

- `chmm.aoplog.onoff`
    - 0 (로그수집 활성화), 1 (로그수집 비활성화)

파일 관련 설정

파일 업로드/다운로드 관련 설정

- `chmm.file.dir`
    - 파일 다운로드시 파일이 저장되는 경로
- `chmm.file.upload.tableName`
    - 파일 업로드 테이블명
    - `CHMM_SYSTEM_DEFAULT_INFO` (고정)
- `chmm.file.upload.allowExtensionKey`
    - 파일 업로드 가능한 확장자 키
    - `FILEUPLOAD_ALLOWED_EXTENSION` (고정)
- `chmm.file.upload.maxSizeKey`
    - 업로드 최대 파일 크기 키
    - `FILEUPLOAD_MAX_SIZE` (고정)
- `chmm.file.upload.column`
    - 파일 업로드 컬럼
    - `ENV_VALUE` (고정)

기본 환경설정 데이터 설정

환경설정 테이블 정보 관련 프로퍼티 파일

- `chmm.property.table`
    - 테이블 명칭
    - `chmm_property_info` (고정)
- `chmm.property.key.column`
    - 키 컬럼 명칭
    - `property_key` (고정)

- `chmm.property.value.column`
  - 값 컬럼 명칭
  - `property_value` (고정)

알림 설정

카카오 알림톡 및 telegram 서버 연결정보를 설정한다.

아래 상세 정보는 카카오 알림톡 API 정보를 확인한다.

```
# TELEGRAM
chmm.push.telegram.token={telegram token}
# ALIMTALK
chmm.push.alimtalk.host={host}
chmm.push.alimtalk.port={post}
chmm.push.alimtalk.tokenPath=/v1/auth/tokens
chmm.push.alimtalk.clientId={clientid}
chmm.push.alimtalk.clientPwd={clientpassword}
chmm.push.alimtalk.alimTalkPath=/v1/send/kakao-notice
chmm.push.alimtalk.msgId=1
chmm.push.alimtalk.sendTime=2018-07-06 17:08:01
chmm.push.alimtalk.sendPhone=01011112222
chmm.push.alimtalk.templateCode={template code}
chmm.push.alimtalk.senderKey={sender key}
```

Redis 설정

Redis 관련 설정 정보이다.

설치형 레디스를 사용하는 경우 대상 Redis의 정보를 입력하도록 한다.

- `chmm.redis.hostName`
  - Redis가 설치 된 호스트 정보를 입력한다. (e.g. 127.0.0.1)
- `chmm.redis.port`
  - Redis에 접속가능한 포트를 입력한다. (e.g. 6379)
- `chmm.redis.usePool`
  - Redis에 접속 된 Connection의 pool 사용여부를 입력한다.(true|false)
- `chmm.redis.embedded.enabled`
  - 내장 레디스의 활성화 여부를 입력한다. 기본값은 true이다. (`true` | `false`)

MVC 동기화 관련 설정

MVC 서비스 정보를 어드민에서 설정한 데이터베이스 값과 동기화 수행 시 사용하는 값들이다.

- `chmm.requestMappingSync.batch.use` (true|false)
  - 데이터베이스 쿼리 수행 시 statement.addbatch를 사용할지 여부이다.
- `chmm.requestMappingSync.delete.use` (true|false)
  - 실제 소스코드 상에 서비스가 삭제된 경우 데이터베이스 설정 값도 삭제할지 여부이다.
- `chmm.requestMappingSync.serviceId.provider`
  - propertyServiceIdProvider (고정)
- 서비스 어플리케이션들을 구분해주는 식별자를 제공해주는 프로바이더
- `chmm.requestMappingSync.enabled`

    - true
    - MVC서비스 정보 스캔 여부 설정. false로 설정 시 MVC제어, 정보, 시나리오 테스트를 사용할 수 없다.

모니터링 설정

Scouter를 설치하여 모니터링을 수행하는 경우 Scouter의 Collector 서버의 정보를 입력해야 한다.

- `chmm.scouter.ip`
    - Scouter Collector 서버의 아이피
- `chmm.scouter.port`
    - Scouter Collector 서버의 Web API port
- `chmm.scouter.timeLength`
    - 현재시간 기준으로 데이터를 조회하기 위한 시간 설정 (단위: 초(sec))

멀티인스턴스 설정

어드민서버에서 사용자가 개발한 어플리케이션을 관리하기 위한 설정이다.

- `chmm.serviceintegration.serviceId`
    - 어플리케이션의 식별자 값이다. MSA 환경에서 어플리케이션을 구별하기 위해 사용된다. 이중화 된 어플리케이션의 경우 동일한 값으로 설정된다.
- `chmm.serviceintegration.serviceAddress`
    - 어플리케이션이 구동하는 서버의 접속정보를 설정한다. 컨텍스트를 포함한다.
        - E.g. http://127.0.0.1:8080/humanresources
    - 자사 PaaS(clepass)를 이용하는 경우 cluster IP를 기록하도록 한다.
- `chmm.serviceintegration.service.registration.interval`
    - 어플리케이션의 helthcheck를 위한 간격을 설정한다. (단위: ms)

SMTP 설정

SMTP 서버 관련 정보를 설정한다.

- `chmm.smtp. address`
    - SMTP 서버의 아이피
- `chmm.smtp.port`
    - SMTP 서버의 포트
- `chmm.smtp.id`
    - SMTP 서버의 아이디
- `chmm.smtp.pw`
    - SMTP 서버의 패스워드

트랜잭션 설정

트랜잭션 포인트컷 설정한다.

- `chmm.txPointcut.expression`
    - 트랜잭션 포인트컷이 적용될 패키지를 지정한다

메서드 트레이스 설정

메서드 트레이스 포인트컷 설정한다.

- `chmm.methodTracePointcut.expression`

- 메서드 트레이스 포인트컷이 적용될 패키지를 지정한다

시큐리티 설정

시큐리티 정보를 설정한다.

- `chmm.security.loginUrl`
  - 로그인 페이지 URL정보를 명시한다. 인증되지 않은 사용자가 접근했거나, 세션이 파기되거나 혹은 로그아웃을 수행하는 경우 해당 URL로 리다이렉션 된다.
  - 명시되지 않은 경우 기본 로그인 페이지가 로드 된다.
- `chmm.security.loginProcessingUrl`
  - 로그인 요청을 처리하는 URL정보를 명시한다.
- `chmm.security.logoutProcessingUrl`
  - 로그아웃 요청을 처리하는 URL 정보를 명시한다.
- `chmm.security.accessDeniedErrorPage`
  - 리소스 요청에 대한 접근이 거부되는 경우 처리 되는 페이지 정보를 명시한다.
- `chmm.security.usernameParameter`
  - 로그인 요청 시 사용되는 사용자 아이디 파라매터 이름을 명시한다.
- `chmm.security.passwordParameter`
  - 로그인 요청 시 사용되는 사용자 비밀번호 파라매터 이름을 명시한다.
- `chmm.security.ignorePatterns`
  - 시큐리티에서 none처리할 URL 패턴의 명시한다. (콤마로 복수개의 URL 패턴을 명시한다.)
- `chmm.security.ignorePatterns`
  - 시큐리티에서 none처리할 URL 패턴의 명시한다. (콤마로 복수개의 URL 패턴을 명시한다.)

UIAdatper 설정

UIAdapter 정보를 설정한다.

- `chmm.uiadapter.excludePatterns`
  - UIAdapter 사용시 시큐리티에서 exclude 처리할 URL 패턴의 명시한다. (콤마로 복수개의 URL 패턴을 명시한다.)

예외 처리 설정

예외 처리와 관련된 정보를 설정한다.

- `chmm.exception.errorPage`
  - 예외가 발생한 후 처리 될 에러페이지 정보를 명시한다. (기본적으로 설정된 페이지는 error 이다.) : ajax 요청이 아닌 페이지 요청 시 처리되는 정보이다.

log4jdbc.properties

- `log4jdbc.drivers`
  - log4jdbc로 드라이버를 설정할 경우 mariadb와 신규 mysql드라이버 명을 인식 시켜 주기위해 설정한다. 기본적으로 `com.mysql.cj.jdbc.Driver,org.mariadb.jdbc.Driver` 로 설정되어있다.
- `log4jdbc.dump.sql.maxlinelength`
  - 로깅시 sql문을 최대 몇 라인까지 출력할 것인지 설정한다. 0으로 설정할 경우 제한 없이 입력된 포맷으로 출력이 되며 해당 property를 설정하지 않을 경우 한줄로 출력이 된다.

로그마스킹 설정

로그마스킹 설정 파일은 src/main/resources/mask 폴더내 위치하며 설정파일은 아래와 같다.

설정파일	역할
maskers_default.xml	마스킹 처리 관련 실행 클래스 및 마스킹 처리할 패턴 정보에 대한 타켓 정보(XML, DB)를 설정해 놓은 XML 파일
finders_default.xml	마스킹 처리할 정규표현식 패턴을 정리해 놓은 XML 파일
unregex.properties	특정 문자열에 대한 마스킹 처리가 필요할 경우 특정 문자열에 대해 정의해 놓은 프로퍼티 파일 ※ 정규표현식 패턴 형태가 아닌 특정 문자열에 대한 로그 마스킹 처리가 필요한 경우 프로퍼티 설정을 통해서 특정 문자열이 시작되는 로그 메시지에 대해서 마스킹 처리가 가능합니다.
log_masking.properites	정규표현식 패턴이랑 매칭되는 문자열을 대체할 문자열에 대해서 정의해 놓은 프로퍼티 파일 ※ 정규표현식 패턴 정보 의 패턴명이랑 1:1 매칭이 됩니다.

각 파일별 설명은 아래와 같다.

로그마스킹 기본 설정 (maskers_default.xml)

속성	설명	기본값	제공여부
masker	실행클래스 정의		O
name	마스커 이름 정의		O
class	실행클래스		O
configuration	대체문자열을 정의한 프로퍼티 파일명	log_masking.properties	O
logmaskopt	패턴정보 타켓 (XML, DB)	XML	O
default	기본 지정 마스커 정의		O

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<maskers>
  <masker>
    <name>LogMaskingExecutor</name>
    <class>net.lotte.chamomile.core.log.mask.LogMaskingExecutor</class>
    <configuration>classpath:config/mask/log_masking.properties</configuration>
    <logmaskopt>XML</logmaskopt>
  </masker>
  <default>LogMaskingExecutor</default>
</maskers>
```

정규표현식 기반 설정 (finders_default.xml)

속성	설명	기본값	제공 여부
finder	**정규표현식 한개당 각각 1개의 파인더로 구성**		**O**
name	정규표현식 이름		O
pattern	정규표현식 패턴 정보		O
class	실행 클래스	지정이 없을 경우 RegexFinder가 디폴트 실행	O
enabled	실행여부	true	O

> 실행 클래스 속성은 현재 사용하지 않고 있습니다. (CreditCardFinder, CompositedCreditCardFinder)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<finders>
  <finder>
    <name>Email</name>
    <pattern>\b[A-Z0-9._%+-]+@([A-Z0-9.-]+)\.([A-Z]{2,4})\b</pattern>
    <enabled>true</enabled>
  </finder>
  <finder>
    <name>MasterCard</name>
    <pattern>5[1-5][0-9]{2}(\\ |\\-|)[0-9]{4}(\\ |\\-|)[0-9]{4}(\\D|$)</pattern>
    <enabled>true</enabled>
  </finder>
<finders>
```

정규표현식 기반 대체문자열 정의(log_masking.properties)

```
Email=xxxx@xxxx.xxx
CreditCard=XXXX-XXXX-XXXX-XXXX
Visa=XXXX-XXXX-XXXX-XXXX
MasterCard=XXXX-XXXX-XXXX-XXXX
KoreaCard=XXXX-XXXX-XXXX-XXXX
AMEX=XXXX-XXXX-XXXX-XXX
```

비정규 패턴 형식 (unregex.properties)

특정문자열로 시작하는 로그를 마스킹하기 위한 프로퍼티 파일이다.

```
id=xxxx
pass=xxxx
pwd=xxxx
password=xxxx
아이디=xxxx
비밀번호=xxx
```

Web 화면 라이브러리

개요

chamomile프레임워크는 상용UI 도구를 사용하지 않는 프로젝트에서 UI컴포넌트들을 사용 할 수 있도록 가이드를 제공한다. UI는 샘플 프로젝트이 미리 준비가 되어있으며 bootstrap과 jquery를 사용한다. 앞으로 설명될 각 컴포넌트의 css들도 미리 설정 되어 있으므로 사용하기만 하면 된다.

그리드

개요

서버의 데이터를 화면에 표현하기위한 방법으로 table 태그를 이용한 방식이 많이 사용되어왔다. 이 가이드에서는 사용자가 직접 테이블을 그리고 컨트롤하는 것이 아닌 javascript라이브러리를 통해 데이터를 출력하고 관리 하는 방법을 제공한다.

slickgrid

관련 파일 링크

```
<!-- slickgrid -->
<link rel="stylesheet" href="<c:url
value='/resources/css/slickgrid/slick.grid.css'/>"  type="text/css" />
<link rel="stylesheet" href="<c:url
value='/resources/css/slickgrid/slick.pager.css'/>" type="text/css" />
<link rel="stylesheet" href="<c:url
value='/resources/css/slickgrid/slick.custom.css'/>" type="text/css" />
<script src="<c:url value='/resources/js/slickgrid/firebugx.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/jquery.event.drag-
2.3.0.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.core.js'/>"></script>
<script src="<c:url
value='/resources/js/slickgrid/slick.rowselectionmodel.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.grid.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.helper.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.dataview.js'/>">
</script>
<script src="<c:url value='/resources/js/slickgrid/slick.pager.js'/>"></script>
<script src="<c:url
value='/resources/js/slickgrid/slick.checkboxselectcolumn.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.autotooltips.js'/>">
</script>
<script src="<c:url
value='/resources/js/slickgrid/slick.cellrangedecorator.js'/>"></script>
<script src="<c:url
value='/resources/js/slickgrid/slick.cellrangeselector.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.cellcopymanager.js'/>">
</script>
<script src="<c:url
value='/resources/js/slickgrid/slick.cellselectionmodel.js'/>"></script>
<script src="<c:url value='/resources/js/slickgrid/slick.columnpicker.js'/>">
</script>
<script src="<c:url value='/resources/js/slickgrid/slick.formatters.js'/>">
</script>
```

```
<script src="<c:url value='/resources/js/slickgrid/slick.editors.js'/>">
</script>
<!-- paging -->
<script src="<c:url
value='/resources/js/pagination/jquery.twbsPagination.js'/>"></script>
```

데이터 출력 및 조회

- grid와 pager가 위치할 영역 정의

```
<div id="SlickGrid1" style="width: 100%;height: 500px"></div>
<div id="pager1" style="width: 100%;text-align:center"></div>
```

[설명]

그리드가 위치할 영역에 div 태그로 ID를 지정해준다. 가로와 세로 길이를 지정해주게 되면 해당 영역안에 생성된다. grid에서 기본적으로 제공하는 페이징 처리기능이 있지만 좀더 친숙한 형태의 UX를 제공해주기위해 별도의 pager 라이브러리(twbsPagination)를 이용한다.

- 그리드 생성

```
//dataView생성
dataView = new Slick.Data.DataView();
//dataView에 data 적용
dataView.beginUpdate();
dataView.setItems(data);
dataView.endUpdate();
//grid생성
grid = slickhelper.createSlickGrid(container, dataView, column, options);
grid.setSelectionModel(new Slick.RowSelectionModel({selectActiveRow: false}));
//행 클릭이벤트 적용
slickhelper.setRowClickEvent2(rowClickFunc, grid);
//dataView에 grid sync설정
dataView.syncGridSelection(grid, true);
```

[설명]

slickgrid는 그리드를 만들 container, 데이터를 담고 있는 DataView, 컬럼 정의, 기타 옵션을 parameter로 하여 생성시킨다.

DataView에 셋팅되는 data형태는 아래와 같다.

[model]

```
data──┬row┬id
  |   |   ├column 1
    |   └column 2
    ├row┬id
    |   ├column 1
    |   └column 2
    ├row┬id
    |   ├column 1
    |   └column 2
    ...
```

[json data]

```
data : [
    {
        id : 순번,
        column1 : column1 value,
        column2 : column2 value,
        ...
    }
    ,
    {
        id : 순번,
        column1 : column1 value,
        column2 : column2 value,
        ...
    }
        ...
]
```

- 컬럼에 대한 정의는 아래와 같다.

```
var columns = [
      {id: "code", name: "코드", field: "code", width:350, sortable: true},
      {id: "userId", name: "아이디", field: "userId", width:200},
      {id: "userName", name: "이름", field: "userName", width:200},
      {id: "address", name: "주소", field: "message", width:350}
  ];
```

[설명]

id : 컬럼 구분자

name : 그리드 헤더에 표시할 내용

field : dataview에 셋팅된 item과 매핑 할 값

width : 가로길이

sortable : 정렬 여부

- 데이터 매핑

```
function mappingData(rstData){
    var data;
    for (var i = 0; i < rstData.resultList.length; i++) {
        data[i] = {
            id : i,
            code : rstData.resultList[i].code,
            languageCode : rstData.resultList[i].languageCode,
            countryCode : rstData.resultList[i].countryCode,
            message : rstData.resultList[i].message,
        };
    }
    return data;
}
```

[설명]

dataView에 setItem으로 넘겨지는 data는 서버에서 값을 받아오거나 로컬에서 데이터를 만들어 구성한다.

- 옵션

```
var options = {
    enableCellNavigation: true,
    enableColumnReorder: false,
    headerRowHeight: 100,
    rowHeight:40,
    editable: true
};
```

[설명]

행 높이나 헤더의 행 높이 드래깅여부등을 설정한다.

체크박스

- 체크박스 컬럼을 정의 한다.

```
var columns = [
    checkboxSelector.getColumnDefinition(),
    ...
]
```

[설명]

체크박스 사용을 위한 정의를 추가한다.

- 체크박스 사용을 위해 플러그인을 설정한다.

```
var checkboxSelector = new Slick.CheckboxSelectColumn({
    cssClass: "slick-cell-checkboxsel"
});
grid.registerPlugin(checkboxSelector);
```

[설명]

slickgrid는 제공하는 기능들을 plugin형태로 제공한다. checkbox selector도 plugin으로 제공하며 registerPlugin을 통해 등록한다.

체크된 값 얻어오기

일괄 삭제등을 수행하기위해 체크된 항목의 키값 등을 얻어오기 위해 사용한다.

- 얻어올 키값 정의

```
var reqKey : ['code','languageCode', 'countryCode'];
```

[설명]

컬럼에 정의된 field 값을 기준으로 키값들을 나열하여 배열로 정의 한다.

- 선택된 값 얻어오기

```
var keys = slickhelper.getSelectedRowsKey2(grid, reqKey, data);
```

[설명]

키값은 아래와 같은 형태로 반환된다.

```
keys : [
    {
        요청키 1 : 값,
        요청키 2 : 값,
        요청키 3 : 값
    },
    {
        요청키 1 : 값,
        요청키 2 : 값,
        요청키 3 : 값
    }
    ...
]
```

- 반환된값을 참조하기위해서 아래와 같은 코드로 변환한다.

```
var jsonObj = [];
for (var idx = 0 ;idx < keys.length ; idx++){
    var obj = {
        code : keys[idx]['code']
        ,languageCode : keys[idx]['languageCode']
        ,countryCode : keys[idx]['countryCode']
    };
    jsonObj.push(obj);
}
```

[설명]

반환된 키값의 내용을 참조하기위해 배열 인자에 직접 요청된 키값을 적어주어 값을 참조한다.

내부 편집

그리드 내부에서 직접 수정하고자 할때 아래와 같이 한다.

- 수정대상 컬럼 정의

```
{id: "userId", name: "아이디", field: "userId", width:180,formatter:
Slick.Formatters.TextEditor},
{id: "userName", name: "이름", field: "userName", width:180,formatter:
Slick.Formatters.SelectEditor},
{id: "control", name: "기능", field: "control", width:150,formatter:
Slick.Formatters.EditControlPanel, excludeEvent:true}
```

[설명]

slickgrid는 라이브러리를 사용하는 사용자에게 customizing해서 사용하기 편리하도록 컴포넌트화 되어있다. 데이터를 표현하기 위해 formatter를 제공 하는데 chamomile에서는 cell 수정용 formatter를 제공한다. text입력창을 생성해주는 TextEditor, 콤보박스를 생성해주는 SelectEditor, 저장/수정/취소를 수행하는 EditControlPanel을 제공한다. 각 formatter는 값을 표현하는 value-area와 수정하는 editor-area로 나뉘어져 있고 class로 지정되어있다.

- DataView에 아이템으로 들어가는 data 구조는 아래와 같다.

[TextEditor의 경우]

```
data : [
    {
        id : 순번,
        column1 : {
            text : 데이터
            className : 'input-sm'
        }
    }
]
```

[설명]

text : 화면에 표시할 값, text box에도 셋팅된다.

className : text box에 적용할 css클래스

[SelectEditor의 경우]

```
data : [
    {
        id : 순번,
        column1 : {
            text : 데이터,
            selectedValue : '1',
            className : 'input-sm form-control',
            element :[
                {value:'1', text:'인자1'},
                {value:'0', text:'인자2'}
            ]
        }
    }
]
```

[설명]

text : 화면에 표시할 값

selectedValue : 선택값(selected상태로 만들 option value)

className : select에 적용할 css클래스

element : select에 option 목록 정의, 배열로 정의 되며 value 는 option value에

나타낼 값, text 는 option태그의 text값

[EditControlPanel의 경우]

```
data : [
    {
        id : 순번,
        column1 : {
            editText : '수정',
            editClassName : 'btn btn-info btn-sm',
            editclick : 'editClickFunction',
            saveText : '저장',
            saveClassName : 'btn btn-danger btn-sm',
            saveclick : 'saveClickFunction',
            cancelText : '취소',
            cancelClassName : 'btn btn-success btn-sm',
            cancelclick : 'cancelClickFunction'
        }
    }
]
```

[설명]

editText : 수정상태로 만들기위한 버튼에 표시할 버튼 text

editClassName : edit 버튼에 적용할 css클래스

editclick : edit버튼 클릭시 실행할 함수(text형태)

saveText : 저장에 표시할 버튼 text

saveClassName : save 버튼에 적용할 css클래스

saveclick : save버튼 클릭시 실행할 함수(text형태)

cancelText : 취소에 표시할 버튼 text

cancelClassName : cancel 버튼에 적용할 css클래스

cancelclick : cancel버튼 클릭시 실행할 함수(text형태)

- 수정모드/보기모드 전환

```
slickhelper.enableRowEditMode(obj);
slickhelper.disableRowEditMode(obj);
```

[설명]

editClick이벤트로 지정한 함수에서 obj를 파라미터로 받아 slickhelper.enableRowEditMode(obj); 를 수행하면 수정모드로 변경된다. 마찬가지로 cancelclick이벤트로 지정한 함수에서 obj를 파라미터로 받아 slickhelper.disableRowEditMode(obj); 를 수행하면 보기모드로 변경된다.

삭제

view상에서 데이터를 삭제할경우 화면에서만 삭제 되기 때문에 데이터베이스상의 데이터는 별도로 삭제되지 않는다.

따라서 삭제 처리시 아래의 순서대로 삭제를 진행해야 한다.

1. check된값 들을 얻어와 서버에 삭제 요청
2. 완료시 데이터 조회
3. dataview에 얻어진 데이터 셋팅
4. grid에 적용

- 단순히 view영역에서만 데이터를 삭제하고자 하는경우 아래와 같이 수행한다.

```
dataView.deleteItem(id);
grid.invalidate();
grid.updateRowCount();
```

[설명]

dataview에서만 변경되므로 grid에 이를 반영해주는 작업이 필요한데, 이때 invalidate를 실행해주면 grid에 반영이 된다.

추가

미리 정의된 row를 grid에 추가해준다.

dataView에 setting하는 data배열의 element를 한개 미리 준비해두고 insert시켜주면 된다.

- row데이터 정의

```
var newData = {
    id : 순번,
    column1 : column1 value,
    column2 : column2 value,
    ...
}
```

[설명]

서버에서 mapping 시키는 예제에서 보여지는것처럼 row 데이터 한개를 만들어준다.

- 행 추가

```
dataView.insertItem(0, newData);
grid.invalidate();
grid.updateRowCount();
```

[설명]

insertItem에 추가할 위치와 추가할 row데이터를 넘겨주어 insert를 수행한다.

dataview에서만 변경되므로 grid에 이를 반영해주는 작업이 필요한데, 이때 invalidate를 실행해주면 grid에 반영이 된다.

context menu

- 그리드에서 마우스 오른쪽 버튼을 클릭해서 context menu를 노출하고자 할경우 아래와 같이 작성한다.

```
slickhelper.createContextMenu(container, grid객체, 실행할함수,
autoClose(true|false));
```

[설명]

container : context메뉴를 구성할 UI를 미리 만들어두고 이를 생성하기 위한 id나 class명을 넘거준다.

grid객체 : context를 띄울 grid객체를 넘겨준다.

실행할함수 : context메뉴가 화면에 보여지기 전에 해야할 작업들이 있다면 이 함수에서 작업한다.(e.g. 데이터 셋팅 등)

autoClose : context menu 외 영역을 클릭하였을 경우 context menu가 자동으로 사라지게 하고자 할 경우 true를 셋팅한다.

드래그

- drag & drop을 통해 행을 이동하고자 하는경우 아래와 같이 작성한다.

```
slickhelper.enableRowReordering(grid객체, dataView, 현재 데이터를 얻어올함수, drop이
후 처리될 함수);
```

[설명]

grid객체 : 드래깅을 수행할 grid객체

dataview : 현재 상태의 dataView

현재 데이터를 얻어올함수 : dataView에 셋팅된 data

drop이후 처리될 함수 : drop이후 서버에 새롭게 데이터를 저장하거나 별도의 동작이

필요할 경우 수행됨. (params : 이벤트 객체, grid데이터)

- 드래깅을 할 때 핸들이 필요한데, column정의에 아래와 같이 제일 첫번째 인자로 적용한다.

```
var columns = [
    {id: "#", name: "",
width:40,behavior:"selectAndMove",selectable:false,resizable:false,cssClass:"cel
l-reorder dnd"},
    {id: "code", name: "코드", field: "code", width:350, sortable: true},
    {id: "userId", name: "아이디", field: "userId", width:200},
    {id: "userName", name: "이름", field: "userName", width:200},
    {id: "address", name: "주소", field: "message", width:350}
];
```

[설명]

id를 #으로 지정해 준다. 이후 데이터 mapping시에 핸들로 지정된 행은 mapping데이터를 설정하지 않는다.

정렬

테이블 정렬이 필요할 경우 아래와 같이 정의한다.

- 컬럼정의

```
var columns = [
    {id: "code", name: "코드", field: "code", width:350, sortable: true},
    {id: "userId", name: "아이디", field: "userId", width:200},
    {id: "userName", name: "이름", field: "userName", width:200},
    {id: "address", name: "주소", field: "message", width:350}
];
```

[설명]

정렬을 적용할 컬럼에 sortable : true를 셋팅한다.

정렬할 컬럼이 클릭되었을경우 아래와 같이 작성한다.

```
grid.onSort.subscribe(function(e, args){
    var currentSortCol = args.sortCol;
    var isAsc = args.sortAsc;
})
```

[설명]

정렬수행시 화면에서만 정렬되는것이 아닌 실제 서버에서 정렬된 데이터를 얻어와야 하므로 onsort시 서버에서 데이터를 다시 받아오는 로직을 채워넣도록 한다.

args.sortCol : 정렬을 수행할 컬럼명

args.sortAsc : ascending여부

파일업로더

fine uploader

파일업로드를 지원하기위해 fine uploader를 UI로 선정하여 제공한다.

파일업로드창을 구성할 템플릿 파일과 업로드 스크립트를 제공한다.(fileupload.jsp)

파일 업로드 기본

- 업로더 생성

```
var fileUploader = new qq.FineUploader({...});
```

- 옵션

```
element : 업로더 템플릿을 적용할 container
template : 템플릿
request.endpoint : 업로드를 수행할 서버 주소
request.method : POST
autoUpload : 파일 선택(drag&drop)시 바로 업로드 수행
callbacks.onComplete : 업로드 성공시 처리
callbacks.onCancel : 취소버튼 클릭시 처리
callbacks.onError : 업로드 에러시 처리
```

확장자 제한

- 옵션

```
validation.allowedExtensions : ['png', 'jpg'];
```

허용할 확장자를 배열형태로 지정. '*' 입력시 모두 허용.

업로드 개수제한

- 옵션

```
validation.itemLimit : [n];
```

정수를 입력한다.

클라이언트 예제

파일업로드의 경우 Multipart 형식으로 전송하게 되며, 파일업로드에 대한 클라이언트 예제 파일은 배포 된 개발환경 내 'C:/Chamomile/교육/edu/fileupload.jsp' 파일로 예제를 제공한다. (fileupload.jsp는 엑셀을 업로드 하는 예제이다.)

```
아래와 같이 예제 파일(fileupload.jsp)을 프로젝트 내 resources 폴더 내 적절한 위치에 포함시킨 뒤 include 한다.
…
<script src="<c:url value='/resources/js/slickgrid/slick.formatters.js'/>">
</script>
<script src="<c:url value='/resources/js/slickgrid/slick.editors.js'/>">
</script>
<script src="<c:url
value='/resources/js/pagination/jquery.twbsPagination.js'/>"></script>

<%@include  file="/resources/excel/fileupload.jsp" %>

    <title>페이지 타이틀(목록보기)</title>
    </head>
<body>
    …

    엑셀을 업로드 하는 버튼을 생성하여 fileupload.jsp 파일의 excelUploadInit()을 수행하여 엑셀을 업로드 한다.
<a href="#" id="buttonExcelInsert" onclick="javascript:excelUploadInit();"
class="btn btn-primary">
    <i class="fa fa-file-excel-o"></i>
        엑셀업로드
</a>
```

서버단 코드

```
@RequestMapping(value = "/excelUpload", method = RequestMethod.POST) protected
ModelAndView excelUpload(
    ModelMap model,
    HttpServletRequest req,
    @RequestParam("qqfile") MultipartFile file,
    @RequestParam("qquuid") String uuid,
    @RequestParam("qqfilename") String fileName,
    @RequestParam(value = "qqpartindex", required = false, defaultValue = "-1")
int partIndex,
    @RequestParam(value = "qqtotalparts", required = false, defaultValue = "-1")
int totalParts,
    @RequestParam(value = "qqtotalfilesize", required = false, defaultValue =
"-1") long totalFileSize) throws Exception {
    ...
        //fileupload컴포넌트를 이용하여 업로드 수행
}
```

[설명]

qqfile : multipart객체

qquuid : 클라이언트에서 생성된 uuid

qqfilename : 원본 파일명

qqpartindex : 순번(여러파일의 경우)

qqtotalparts : 전체 개수

qqtotalfilesize : 전체 파일 크기(byte)

만약 여러건이 업로드 될 경우 이 메소드가 파일 개수만큼 수행된다.

공통

개요

유용하게 사용될 javascript, jquery기능들을 제공한다. cmml내부에 위치하므로 cmml.replaceAll등으로 호출한다.

validation

[html]

```
<!--각각의 폼 구성요소들은 id와 name이 동일해야 한다.-->
<form class="cmxform" id="commentForm" method="get" action="">
    <p>
        <label for="cname">username</label>
        <input id="username" name="username" minlength="2" type="text" required>
    </p>
    <p>
        <label for="cemail">pwd</label>
        <input id="password" type="password" required>
    </p>
    <p>
        <label for="curl">confir_pwd</label>
        <input id="confirm_password" type="password" name="confirm_password">
    </p>
```

```
    <p>
        <label for="ccomment">email</label>
        <input id="email" type="text" name="email">
    </p>
    <p>
        <label for="ccomment">뉴스레터</label>
        <input id="newsletter" type="checkbox" name="newsletter">
    </p>
    <p>
        <label for="ccomment">토픽</label>
        <input id="topic" type="text" name="topic">
    </p>
</form>
<input type='button' onclick="save()" value="Submit">
```

[설명]

validation기능의 경우 form태그 내에 위치한 값들에 대해서만 수행이 가능하다.

[js]

```
<script>
    $.validator.setDefaults({
    success: "valid"
});
$().ready(function() {
    $("#commentForm").validate({
        rules: {
            username: {
                required: true,
                minlength: 2
            },
            password: {
                required: true,
                minlength: 5
            },
            confirm_password: {
                required: true,
                minlength: 5,
                equalTo: "#password"
            },
            email: {
                required: true,
                email: true
            },
            topic: {
                required: "#newsletter:checked",
                minlength: 2
            }
        },
        messages: {
            username: {
                required: "Please enter a username",
                minlength: "Your username must consist of at least 2 characters"
            },
            password: {
                required: "Please provide a password",
                minlength: "Your password must be at least 5 characters long"
```

```
            },
            confirm_password: {
                required: "Please provide a password",
                minlength: "Your password must be at least 5 characters long",
                equalTo: "Please enter the same password as above"
            },
            email: "Please enter a valid email address",
            agree: "Please accept our policy",
            topic: "Please select at least 2 topics"
        },
        errorPlacement: function(error, element) {
            console.log(error);
        },
        invalidHandler: function(form, validator) {
            var errors = validator.numberOfInvalids();
            if (errors) {
                alert(validator.errorList[0].message);
                validator.errorList[0].element.focus();
            }
        }
    });
});
function save(){
    $("#commentForm").valid();
}
</script>
```

옵션 설명

```
rules
    - required
        .설명 : 필수 입력검증
        .값 : true/false
        .대상 : text, password, select, radio, checkbox

    - remote
        .설명 : 서버에서 검증내용을 받아와 체크, 서버에서는 true, false를 리턴해 주면 된다.
        e.g.
        remote : {
            url : "/checkUserId.do",
                type : "post",
                    data : {
                        username : function() {
                            return $("#username").val();
                        }
                    }
        }

    - equalTo
        .설명 : 다른 항목과 같은지 체크
        .값 : 다른오브젝트 ID, Name등 식별자
        e.g.
        equalTo: "#password"

    - minlength
        .설명:최소길이체크
```

```
    e.g.
    minlength: 3

- maxlength
    .설명 : 최대길이체크
    e.g.
    maxlength: 3

- rangelength
    .설명 : 길이 범위 체크.
    e.g.
    rangelength:[2, 6] (2글자 이상 6글자 이하)

- min
    .설명 : 숫자의 최솟값 체크.
    e.g.
    min: 13 (13보다 작을 경우 false)

- max
    .설명 : 숫자의 최댓값 체크.
    e.g.
    max: 5  (5보다 클 경우 false)

- range
    .설명 : 숫자의 범위 체크.
    e.g.
    range: [13, 24] (13보다 작거나 24보다 클 경우 false)

- email
    .설명 : 이메일 형식의 값인지 체크.
    e.g.
    email: true

- url
    .설명 : 유효한 url 형식인지 체크.
    e.g.
    url: true

- date
    .설명 : 유효한 날짜 형식의 값인지 체크

- dateISO (https://ko.wikipedia.org/wiki/ISO_8601)
    .설명 : 유효한 국제표준 날짜 형식인지 체크.
    e.g.
    dateISO: true

- number
    .설명 : 유효한 숫자인지 체크.
    e.g.
    number: true

- digits
    .설명 : 유효한 digit 값인지 체크. number와 다른점은 양의 정수만 허용한다.
    즉, 소수와 음수일 경우 false
    e.g.
    digits: true

- step
```

```
        .설명 : Makes the element require a given step.
        e.g.
        step: 10

    - rrn
        .설명 : 주민등록번호 체크
        e.g.
        rrn : true

    - phone
        .설명 : 핸드폰번호체크
        e.g.
        phone : true

    - accountpassword
        . 설명 : 비밀번호 규칙체크
        e.g.
        accountpassword : {
            minlength : 8, /*최소길이*/
            uppper: true; /*대문자 포함*/
            letter: true; /*문자포함*/
            number: true; /*숫자포함*/
            specialchar: true; /*특수문자포함*/
            repeatcount : 4 /*같은문자 4번이상 반복금지*/
            excludeValue : ['#userId','#userName'] /*지정된 ID에 있는 value포함금지
*/
        }

    - accountid
        . 설명 : 아이디 규칙체크
        e.g.
        accountid : {
            minlength : 8, /*최소길이*/
            excludeString: ['root','admin','select']/*포함시키지 않을 문자열*/
            excludeValue : ['#birth','#userName'] /*지정된 ID에 있는 value 포함금지
*/
        }

    - normalizer
        . 설명 : validation을 수행하기전에 작업해야 할 일이 있으면 이곳에서 작업
        e.g.
            validation수행전에 마스킹되어있는 부분 제거
            moneycheck : {
            required: true,
            number : true,
            range : [100,10000],
            normalizer: function( value ) {
                return cmml._masking.unmask("#moneycheck");
            }
        }
errorPlacement
    - validtion 오류처리(정의하지 않으면 validator에서 label을 추가로 만듬.)
invalidHandler
    - validation 실패시 핸들러
debug
    - true일경우 validation후 submit수행하지 않음.
onfocusout
    - blur시에 항목을 validation할것인지 여부
```

```
message
    - rules에서 설정한 항목들에 대해 실패시 화면에 표시할 문자열정의, rules에 있는 항목과 일치시켜 적용한다.
submitHandler
    - submit전에 처리할 핸들러 정의, 사용자 confirm, 데이터 재가공시 사용한다. false 리턴시 submit 중단.
```

inputmask

Inputmask 클래스 생성

```
var selector = document.getElementById("selector");
var im = new Inputmask("99-9999999");
im.mask(selector);

//or

Inputmask({"mask": "(999) 999-9999", .... other options .....}).mask(selector);
Inputmask("9-a{1,3}9{1,3}").mask(selector);
Inputmask("9", { repeat: 10 }).mask(selector);

Inputmask({ regex: "\\d*" }).mask(selector);
Inputmask({ regex: String.raw`\d*` }).mask(selector);
```

jquery 플러그인으로 사용

```
$(document).ready(function(){
    $(selector).inputmask("99-9999999");  //static mask
    $(selector).inputmask({"mask": "(999) 999-9999"}); //specifying options
    $(selector).inputmask("9-a{1,3}9{1,3}"); //mask with dynamic syntax
});
```

html태그 내의 attribute로 적용

```
<input data-inputmask="'alias': 'datetime'" />
<input data-inputmask="'mask': '9', 'repeat': 10, 'greedy' : false" />
<input data-inputmask="'mask': '99-9999999'" />
$(document).ready(function(){
    $(":input").inputmask();
    or
    Inputmask().mask(document.querySelectorAll("input"));
});
```

chamomile마스킹 사용

```
- onlyEng : 영문 대/소문자만 사용
    e.g. cmml._masking.onlyEng("#objectid");
- numeric : 1000단위마다 comma생성
    e.g. cmml._masking.numeric("#objectid", [최대길이]);
- numeric_arab : 1000단위마다 dot 생성(for arab)
    e.g. cmml._masking.numeric_arab("#objectid", [최대길이]);
- numeric_digits : 소수점길이 제한
    e.g. cmml._masking.numeric_digits("#objectid", [최대길이], [소수점자리수]);
- cellphone : 핸드폰번호(xxx-xxxx-xxxx)
```

```
    e.g. cmml._masking.cellphone("#objectid");
- date : 날짜(yyyy-mm-dd)
    e.g. cmml._masking.date("#objectid");
- rrn : 주민번호(xxxxxx-xxxxxxx)
    e.g. cmml._masking.rrn("#objectid");
- currency : 화폐
    e.g. cmml._masking.currency("#objectid", [접두문자], [접미문자]);
- currency_arab : 화폐(for arab)
    e.g. cmml._masking.currency_arab("#objectid");
- email : 이메일
    e.g. cmml._masking.email("#objectid");
- unmask : 마스킹해제
    e.g. cmml._masking.unmask("#objectid");
```

ajax

cmml._ajaxSync(url, jsonObj, successCallback, errorCallback) : sync방식. ajax통신(다음 로직을 처리하지않고 완료 될때까지 기다림)

parameters

- url : 주소
- jsonObj : 값이 할당된 객체
- successCallback : 성공시 수행할 함수 (서버결과데이터, 상태값 전달)
- errorCallback : 실패시 수행할 함수 (오류값, 서버결과데이터)

cmml._ajaxAsync(url, jsonObj, successCallback, errorCallback) : async방식. ajax통신(서버에 요청후 바로 다음로직 수행)

parameters

- url : 주소
- jsonObj : 값이 할당된 객체
- successCallback : 성공시 수행할 함수 (서버결과데이터, 상태값 전달)
- errorCallback : 실패시 수행할 함수 (오류값, 서버결과데이터)

cmml._ajaxFormAsync(url, jsonObj, successCallback, extra, errorCallback) : Form객체를 submit

parameters

- url : 주소
- jsonObj : 값이 할당된 객체
- successCallback : 성공시 수행할 함수 (서버결과데이터, 상태값 전달)
- extra : successCallback에 추가로 전달할 데이터
- errorCallback : 실패시 수행할 함수 (오류값, 서버결과데이터)

cmml._ajaxAsyncMethod(url, jsonObj, successCallback, errorCallback, method) : 메서드 형태를 다르게 하여ajax호출(restful)

parameters

- url : 주소
- jsonObj : 값이 할당된 객체
- successCallback : 성공시 수행할 함수 (서버결과데이터, 상태값 전달)
- errorCallback : 실패시 수행할 함수 (오류값, 서버결과데이터)
- method : method값

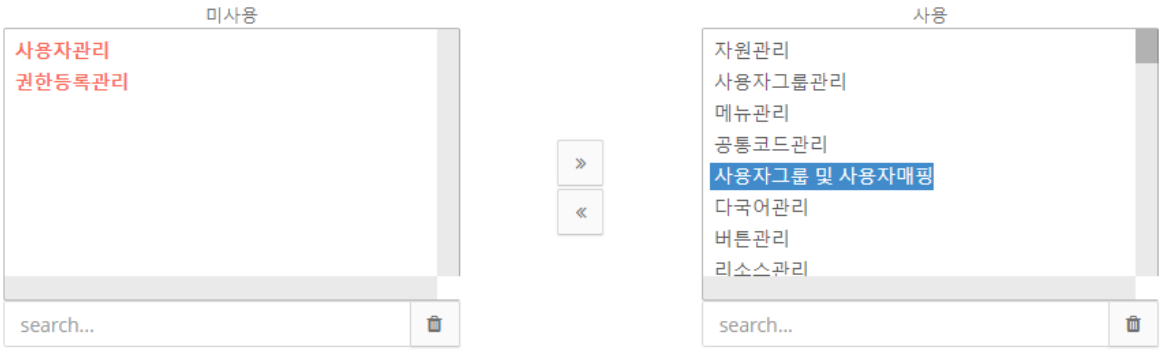
기타

- date유틸
  - 날짜 비교(일수 반환)
    - e.g. cmml.date.diff([시작 년월일], [종료 년월일])
  - 날자 더하기(Date객체로 반환)
    - e.g. cmml.date.add([시작 년월일], [추가할 일 수])
- 매칭되는 문자개수세기
  - e.g. cmml.matchCount([문자열], [검색할문자(열)])
- Cookie
  - cmml.cookie.setCookie([쿠키명], [값], [만료일]) : 쿠키저장
  - cmml.cookie.getCookie([쿠키명]) : 쿠키 읽기
  - cmml.cookie.deleteCookie([쿠키명]) : 쿠키삭제
  - cmml.cookie.listCookies : 쿠키목록 반환
- local storage
  - cmml.localStorage.setItem([키], [값]) : 항목저장
  - cmml.localStorage.getItem([키]) : 항목읽기
  - cmml.localStorage.removeItem([키]) : 항목삭제
- replaceAll
  - cmml.replaceAll([원본], [변경대상문자], [변경할문자]) : 문자열 전체 치환

---

add&remove

아래와 같이 사용/미사용 등을 설정하기위한 컴포넌트를 제공한다.



[그림] add & remove

[html]

```
<div id="addremove" style='height:150px;'></div>
```

[js]

```
new Chamomile.AddRemove(컨테이너, 왼쪽 selectbox 라벨 내용, 오른쪽 selectbox 라벨 내용);
```

e.g.

```
var _addRemove = new Chamomile.AddRemove("#addremove", '미사용', '사용');
```

기능목록

- addLeft
  - 설명 : 왼쪽 selectbox에 값을 추가한다.
  - param : 삽입할 데이터
  - param format :

```
  [
{val : "value값1", text:"표시할 값1", fixed:true}
    ,{val : "value값2", text:"표시할 값2"}
,{val : "value값3", text:"표시할 값3"}
  ]
```

*fixed : 움직이지 못하게 할 항목에 대해서 true로 설정함.

- addRight
  - 설명 : 오른쪽 selectbox에 값을 추가한다.
  - param : 삽입할 데이터
  - param format :

```
[
  {val : "value값1", text:"표시할 값1", fixed:true}
  ,{val : "value값2", text:"표시할 값2"}
  ,{val : "value값3", text:"표시할 값3"}
]
```

*fixed : 움직이지 못하게 할 항목에 대해서 true로 설정함.

- moveRight
  - 설명 : 왼쪽 -> 오른쪽으로 이동. 단, fixed='true' 속성은 옮기지 않음
  - param : N/A
- moveLeft
  - 설명 : 오른쪽 -> 왼쪽으로 이동. 단, fixed='true' 속성은 옮기지 않음
  - param : N/A
- getRightValues
  - 설명 : 오른쪽에 데이터 목록을 가져온다.
  - param : N/A
- getLeftValues
  - 설명 : 왼쪽 데이터 목록을 가져온다.
  - param : N/A
- clearLeft
  - 설명 : 왼쪽 목록을 비운다.
  - param : N/A
- clearRight
  - 설명 : 오른쪽 목록을 비운다.
  - param : N/A
- clear
  - 설명 : 전부 비운다.
  - param : N/A

- setLeft
  - 설명 : 왼쪽 selectbox에 값을 넣어준다.(기존데이터 삭제)
  - param : 삽입할 데이터
  - param format :

```
  [
{val : "value값1", text:"표시할 값1", fixed:true}
   ,{val : "value값2", text:"표시할 값2"}
,{val : "value값3", text:"표시할 값3"}
  ]
```

*fixed : 움직이지 못하게 할 항목에 대해서 true로 설정함.

- setRight
  - 설명 : 오른쪽 selectbox에 값을 넣어준다.(기존데이터 삭제)
  - param : 삽입할 데이터
  - param format :

```
  [
{val : "value값1", text:"표시할 값1", fixed:true}
   ,{val : "value값2", text:"표시할 값2"}
,{val : "value값3", text:"표시할 값3"}
  ]
```

*fixed : 움직이지 못하게 할 항목에 대해서 true로 설정함.

---

addon

QuickSearch

항목이 많을 경우 검색을 통해 이동시킬 수 있도록 quicksearch입력창을 추가한다.

```
var quickSearchOptions = {};
quickSearchOptions.addRight = true;
quickSearchOptions.addLeft = true;
_addRemove.addon("quickSearch",quickSearchOptions);
```

[설명]

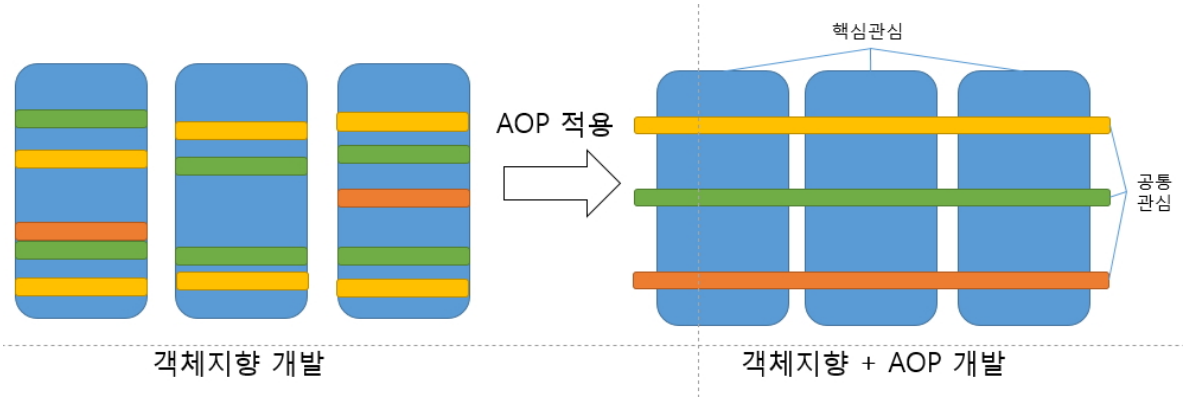
addRight : 오른쪽에 추가할경우 true

addLeft : 왼쪽에 추가할경우 true

실행 기능

AOP

AOP 소개



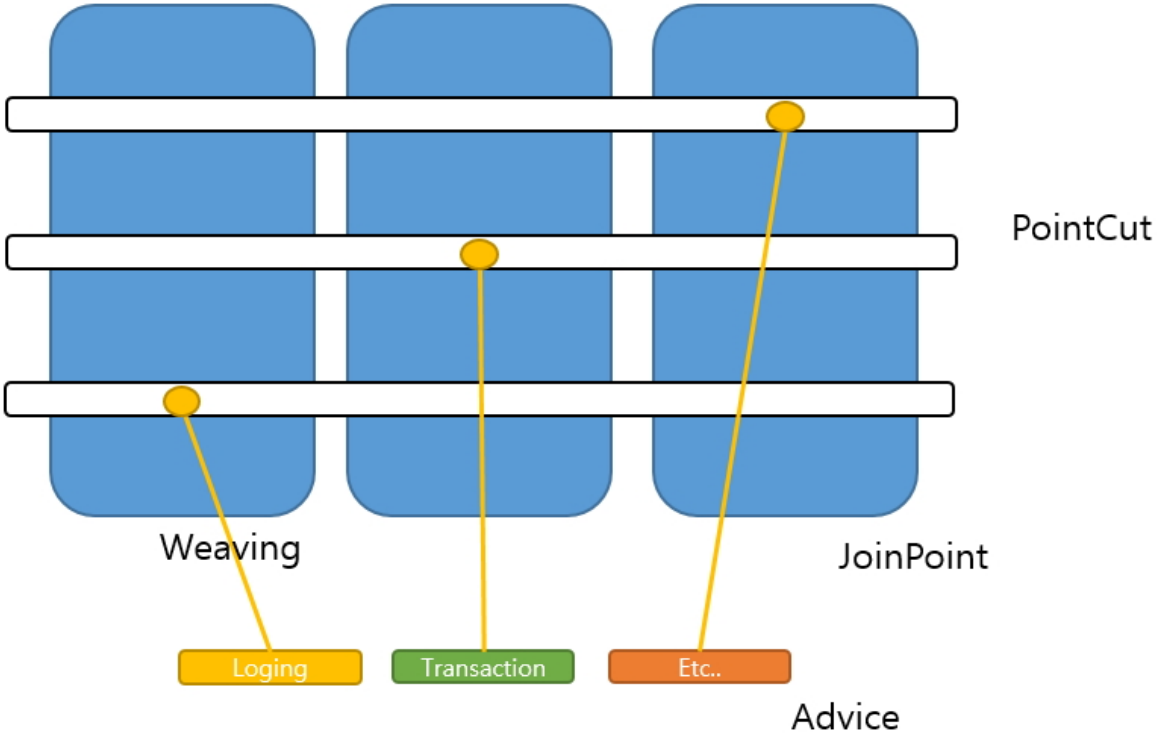
[그림] AOP 기본 설명

AOP는 Aspect Oriented Programming의 약자로써 관점지향 프로그래밍이라 불린다.

객체지향프로그래밍을 보완하는 개념으로 어플리케이션을 객체지향적으로 모듈화 하여 작성했을 때, 나타날 수 있는 중복코드들을 횡단관심으로 분리하여 핵심관심과 엮어서 처리할 수 있는 방법을 제공한다.

로깅, 보안, 트랜잭션등의 공통적인 기능의 활용을 기존의 비지니스 로직에 영향을 주지 않고 모듈화처리를 가능할 수 있게 한다.

AOP 용어

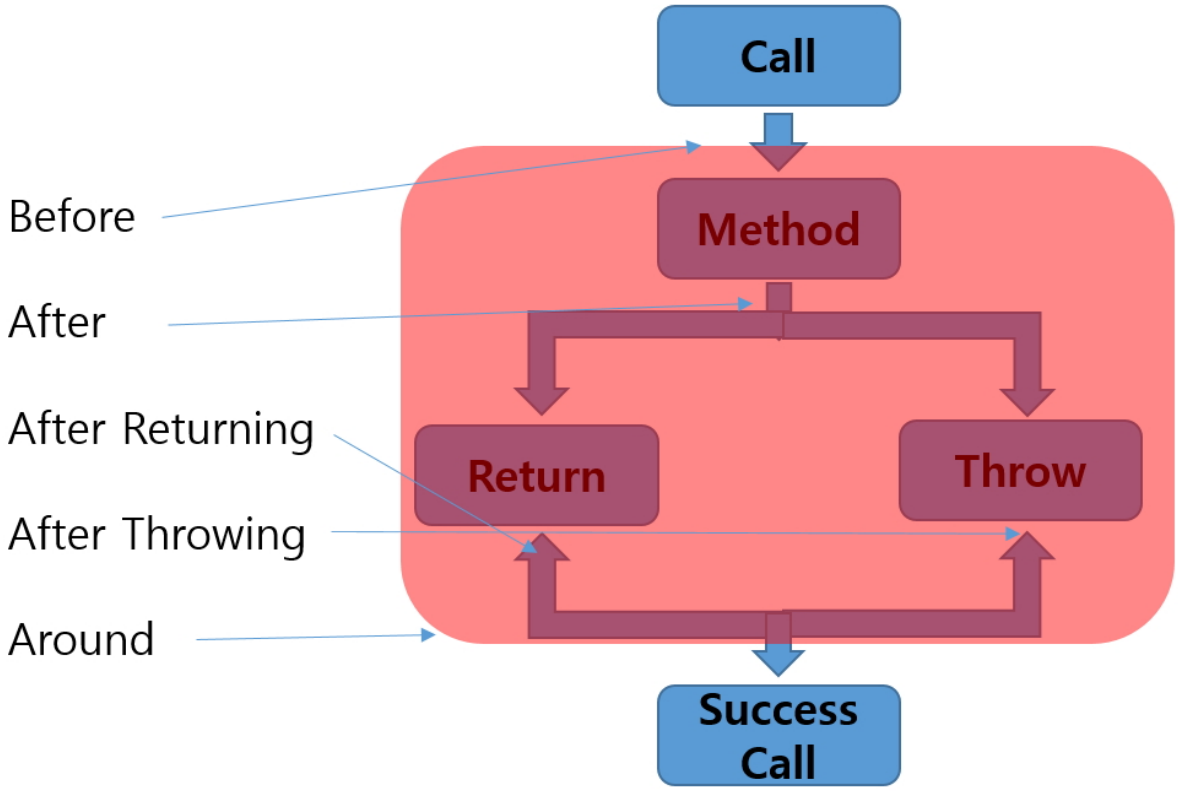


[그림] AOP의 세분화

- Join Point

    - 횡단관심(Crosscutting Concerns)의 모듈(Advice)이 삽입되어 동작할 수 있는 실행 가능한 특정 위치를 말한다.
- Pointcut
    - 포인트컷이란 필터링된 조인포인트를 말한다. 실제로 Advice가 삽입될 포인트이다.
- Advice
    - 횡단 관심에 해당하는 공통기능을 하는 코드이다.
- Weaving
    - Pointcut 으로 지정한 핵심관심 메소드가 호출 될때, 어드바이스에 해당하는 횡당관심 메소드(Advice)가 삽입되는 과정이다.
    - 크게 컴파일 타임 위빙, 로딩타임 위빙, 런타임 위빙이 있지만 스프링에서는 런타임 방식 위빙만 제공한다.
- Aspect
    - 포인트 컷과 어드바이스의 결합 그 자체이다. (Pointcut(어디에서) + Advice(무엇을 할 것인지))

Advice의 Type



[그림] Advice의 Type

- Before advice: joinpoint 전에 수행되는 advice 이다.
- After returning advice: joinpoint가 성공적으로 리턴된 후에 동작하는 advice이다.
- After throwing advice: 예외가 발생하여 joinpoint가 빠져나갈 때 수행되는 advice 이다.
- After (finally) advice: join point를 빠져나가는(정상적이거나 예외적인 반환) 방법에 상관없이 수행 되는 advice 이다.
- Around advice: joinpoint 전, 후 에 수행되는 advice 이다.

사용자별 접속 로그

사용자별 접속 로그 개요

접속로그관리는 사용자가 시스템 로그인/아웃한 로그를 검색, 조회하는 기능을 제공한다.

접속로그관리는 로그인,아웃 로그의 등록, 조회, 목록의 기능을 수반한다.

1. 로그인, 로그아웃 로그의 등록 - AOP 기능 및 Session Destroy Listener 를 이용한다. (로그아웃시에 세션객체가 파괴되는 데 이를 Listener로 감지)
2. 접속 관련 로그의 상세내용을 조회한다.
3. 접속 관련 로그의 목록을 검색, 조회한다.

> 로그 확인에 대한 내용은 Web운영가이드를 참조한다.

접속 로그 설정

기본적으로 모든 내용이 설정되어있으므로 별도로 설정할 필요는 없다. 참고용으로 활용하도록 한다.

접속로그 기능을 사용하기 위해선 프로젝트에 Logger Register Bean, Aop Advice 및 Session Listener 를 등록해 주어야 한다.

접속 로그 dependency

```
<dependency>
  <groupId>com.lmax</groupId>
  <artifactId>disruptor</artifactId>
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j</groupId>
  <artifactId>log4j-core</artifactId>
</dependency>
```

접속 로그 기록 적용

로거설정

가장먼저 log4j2.xml에 다음과 같은 이름의 비동기 로거를 추가해 주어야 한다.

```
<Loggers>
        ...
        <AsyncLogger name="UserLoginLogger" level="TRACE" additivity="false"/>
```

JDBCLoggerRegister Bean 을 등록한다.(context-aoplog.xml)

이는 비동기적으로 DB IO를 하기 위한 Logger를 등록하는 Bean이다. Bean이 생성될 때 실행되는 init-method옵션을 이용하여 Async Logge(UserLoginLogger)를 Log4j2 설정에 등록한다. DB연결을 위하여 dataSource정보도 설정한다.

```
<bean id="JDBCLoggerRegister"
      class="net.lotte.chamomile.core.aop.register.JDBCLoggerRegister"
      init-method="loggerRegist">
  <property name="userLoginLoggerName"value="UserLoginLogger"/>
  <property name="dataSource" ref="dataSource"/>
  <!-- DATA SOURCE의 경우 공통 jdbc 설정 빈을 참고 -->
</bean>
```

접속 로그 적용

logInOutAdvice Bean 을 context-aoplog.xml파일 내에 등록한다. 로그인시 로그인정보를 로깅할때 사용되는 AOP Advice 이다. 기본 포인트컷은 Spring Security 로그인 성공시 호출되는 콜백 함수인 onAuthenticationSuccess 메소드로 정의되어 있다.

```
<!-- 로그인시 기록을 저장할때 사용되는 AOP Advice 이다. JDBCLoggerRegister 빈 에 종속성이 있으므로 Depends On 옵션이 서술됨-->
<bean id="logInOutAdvice"
      class="net.lotte.chamomile.commons.aop.advice.LogInOutAdvice"
      depends-on="JDBCLoggerRegister"/>
```

sessionDestoryListener를 등록한다. 이 컴포넌트는 로그아웃 시점을 캐치하여 로그를 기록하는 리스너가 정의된 빈이다.

```
<!-- 로그아웃시점을 캐치하여 로그를 기록하는 리스너가 정의된 빈이다. -->
<bean id="sessionDestoryListener"

 class="net.lotte.chamomile.commons.aop.interceptor.SessionDestoryListener"/>
```

리스너를 사용하기위해선 추가로 HttpSessionEventPublisher 클래스를 web.xml에 등록해주어야 한다.

```
<listener>
    <listener-class>
        org.springframework.security.web.session.HttpSessionEventPublisher
    </listener-class>
</listener>
```

로그아웃 시점을 Aop로 캐치하지 않고, Session Listener로 캐치하는 이유는 사용자가 직접 로그아웃하는 것 외에 Timeout으로 로그아웃 되는 시점을 AOP로 캐치 할 수는 없다. Timeout 시점까지 캐치하기 위하여 세션의 만료를 감지할 수 있도록 하기 위해 HttpSessionEventPublisher 클래스를 web.xml에 등록 한다.

Method 추적로그

Method 추적로그 기록 적용

로거설정

가장먼저 Log4j.xml에 다음과 같은 이름의 비동기 로거를 추가해 주어야 한다.

```
<Loggers>
        ...
    <AsyncLogger name="MethodTraceLogger" level="TRACE" additivity="false"/>
```

JDBCLoggerRegister Bean 을 등록한다.(context-aoplog.xml)

이는 비동기적으로 DB IO를 하기 위한 Logger를 등록하는 Bean이다. Bean이 생성될 때 실행되는 init-method옵션을 이용하여 Async Logger(MethodTraceLogger)를 Log4j2 설정에 등록한다. DB연결을 위하여 dataSource정보도 설정한다.

```
<bean id="JDBCLoggerRegister"
      class="net.lotte.chamomile.core.aop.register.JDBCLoggerRegister"
      init-method="loggerRegist">
  <property name="userLoginLoggerName"value="UserLoginLogger"/>
  <property name="methodTraceLoggerName"value="MethodTraceLogger"/>
  <property name="dataSource" ref="dataSource"/>
  <!-- DATA SOURCE의 경우 공통 jdbc 설정 빈을 참고 -->
</bean>
```

Method 추적로그 적용

context-aoplog.xml 파일 내에 aopLogPropertyLoader 라는 이름의 Bean 생성 구문을 추가해주어야 한다. 이 빈은 Aop/On Off 기능을 수행할때 사용되며, On Off 에대한 설정 값을 DB에서 불러올지 아니면 .properties 파일에서 불러올지 설정해주는 loader 이다.

데이터 베이스에서 Log ON/OFF 설정 정보를 불러오고 싶다면, 다음과 같이 빈을 생성한다.

```
<bean id="aopLogPropertyLoader"
      class="net.lotte.chamomile.core.aop.loader.AopLogPropertyDbLoader">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

단 CHMM_PROPERTY_INFO 테이블내에 key값이 AOPLOGONOFF인 설정값이 존재해야한다. (0=false,1=true)

.property 파일에서 Log ON/OFF 설정을 읽어오려면 다음과 같이 aopLogPropertyLoader 빈을 생성 하고, 빈 생성시에 property 값에 . properties 설정파일에 위치를 명시해주어야 한다.

```
<bean id="aopLogPropertyLoader"
      class="net.lotte.chamomile.core.aop.loader.AopLogPropertyFileLoader">
    <property name="location"value="classpath:application.properties"/>
</bean>
```

application.properties파일 내에 아래의 항목으로 값을 설정한다. (0=false,1=true)

```
##AOP LOG
chmm.aoplog.onoff = 0
```

aopLogPropertyLoader 생성이 끝났다면 methodAdvice 빈을 생성한다. 그리고 위에서 만든 aopLogPropertyLoader를 property를 지정해 준다. methodAdvice는 메소드별 기록을 남길때 사용되는 AOP의 Advice이다.

```
<bean id="methodAdvice"
      class="net.lotte.chamomile.core.aop.advice.MethodTraceAdvice"
      init-method="initLogOption"
      depends-on="JDBCLoggerRegister">
  <property name="aopLogPropertyLoader“ ref="aopLogPropertyLoader“/>
</bean>
```

Advice 까지 만들었으므로, 이제 AOP Advice를 포인트컷에 위빙 하도록 한다. 포인트컷은 기본적으로 AOP는 Controller, Service, DAO 클래스에만 걸어주어야 하며, 만약 Mybatis mapper를 통해 구현체 없이 DAO를 개발한 경우는 mybatis interceptor를 추가해주어야한다. 아래는 메서드를 추적하기위해 설정된 샘플 point cut이다.

(* Dispatcher Servlet Context 쪽 xml 파일에는 Controller 를 걸어주고 Root Context에는 Service, DAO를 입력해야 한다.).

context-servlet.xml 예시)

```
<!-- method 트레이스 로그를 남기기위해 적용시킨 AOP -->
<aop:config proxy-target-class="true">
    <!-- method trace aop -->
    <aop:pointcut id="logPointcut"
                  expression="(execution(* net.lotte.sample..*Controller.*
(..)))"/>
    <aop:aspect ref="methodAdvice">
        <aop:around pointcut-ref="logPointcut" method="logMethod" />
    </aop:aspect>
</aop:config>
```

context-common.xml 예시)

```
<!-- method 트레이스 로그를 남기기위해 적용시킨 AOP -->
<aop:config proxy-target-class="true">
    <!-- method trace aop -->
    <aop:pointcut id="logPointcut"
                  expression="(execution(* net.lotte.sample..*ServiceImpl.*(..))
or execution(* net.lotte.sample..*DAO.*(..)))"/>
    <aop:aspect ref="methodAdvice">
        <aop:around pointcut-ref="logPointcut" method="logMethod" />
    </aop:aspect>
</aop:config>
```

스프링 부트와 같은 경우는 설정 xml파일이 없기 때문에 properties파일에 포인트컷을 명시해줘야 한다. 이때는 Controller, Service, DAO 클래스모두 걸어준다.

```
##MethodTrace Pointcut
chmm.methodTracePointcut.expression=(execution(* net.lotte.sample..*Controller.*
(..)) or execution(* net.lotte.sample..*ServiceImpl.*(..)) or execution(*
net.lotte.sample..*DAO.*(..)))
```

다음으로 인터셉트를 정의해주어야 한다. 인터셉터는 리퀘스트마다 Request ID를 발급하여 Request 별로 메소드를 구분할 수 있게 엮어주는 역할을 한다. 아래와 같이 dispathcer servlet xml 파일에 Interceptor설정값을 넣어주면 된다.

```
<!-- method trace aop -->
<interceptors>
 <interceptor>
  <mapping path="/**"></mapping>
  <exclude-mapping path="/resources/**" />
  <exclude-mapping path="/admin/login" />
  <beans:bean
     class="net.lotte.chamomile.commons.aop.interceptor.ServiceLogInterCeptor"/>
  </interceptor>
</interceptors>
```

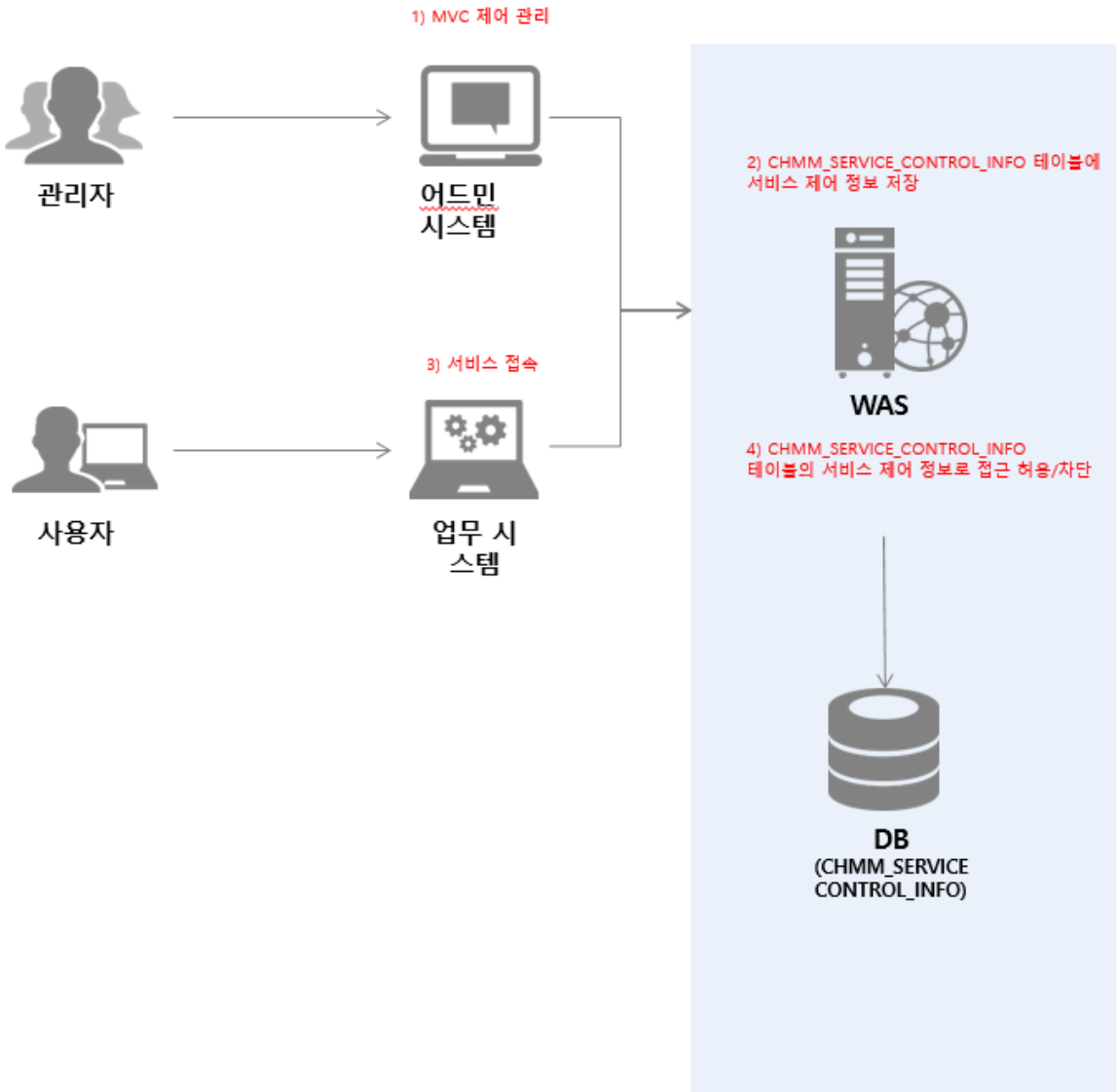
MVC

MVC 제어

MVC제어 개요

Chamomile Framework 에서는 MVC 서비스를 제어할 수 있는 기능을 제공한다.

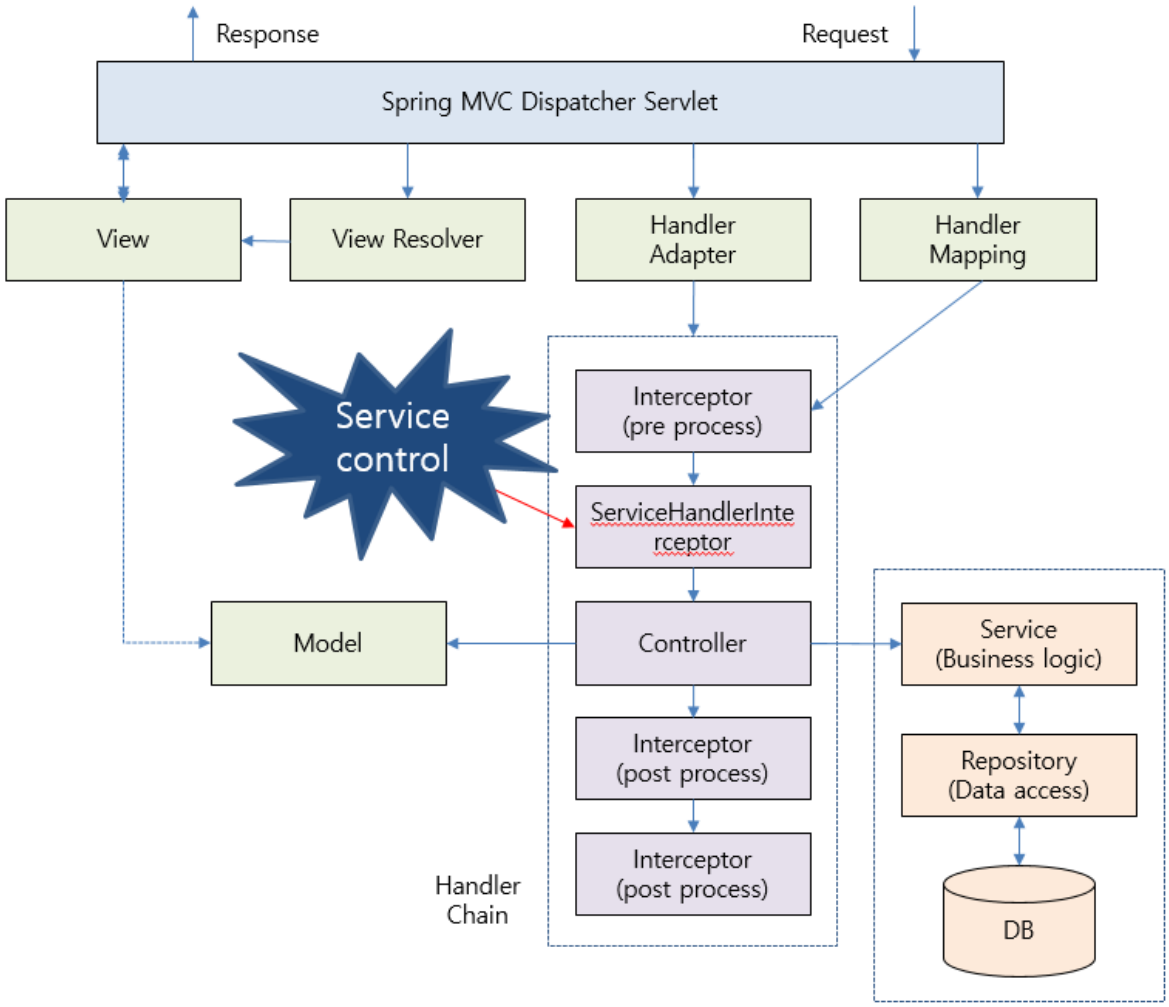
DB에 등록된 MVC 서비스 제어 정보를 통해 서비스 접근을 허용할지 판단한다.



[그림] MVC제어 아키텍처

Dispatcher Servlet 이 적절한 handler 를 찾으면 그 handler 와 연결된handler chain을 순서대로 실행한다. 실행 중 Chomomile framework 에서 제공하는 ServiceHandlerInterceptor 에서 서비스를 제어한다.

Chamomile ServiceHandlerInterceptor 에서 handler 제어 정보를 읽고 제어 여부를 판단한다. 판단 기준은 관리자의 승인 여부, 기간, 활성화 여부3가지이며 이를 위해서는 @ServiceContent 어노테이션이 필요하다.



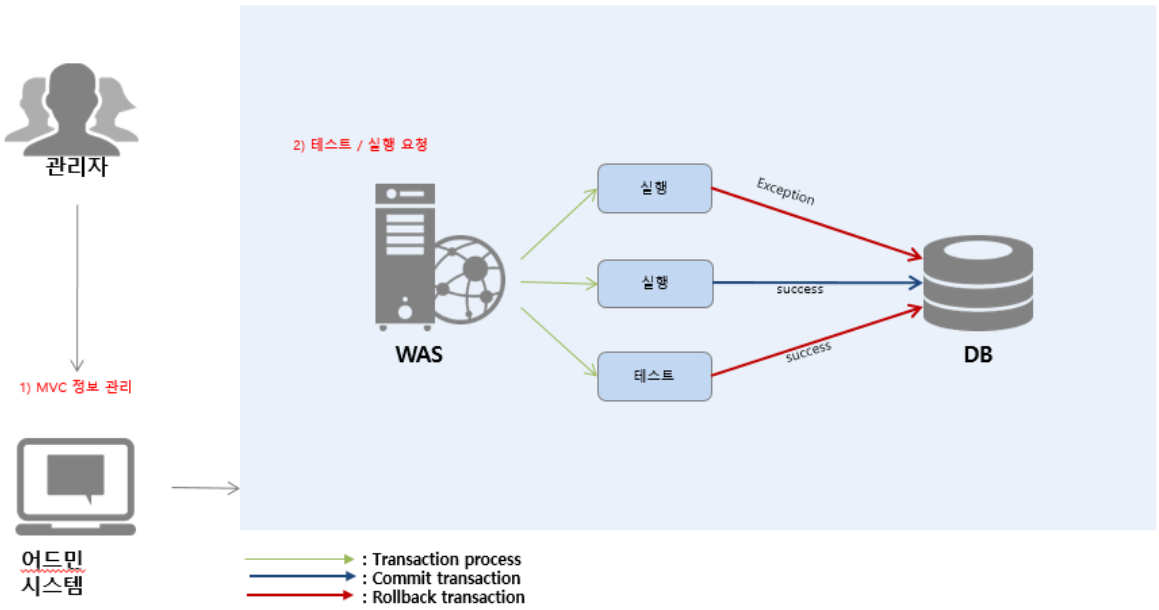
[그림] MVC제어 흐름도

MVC 테스트

MVC테스트 개요

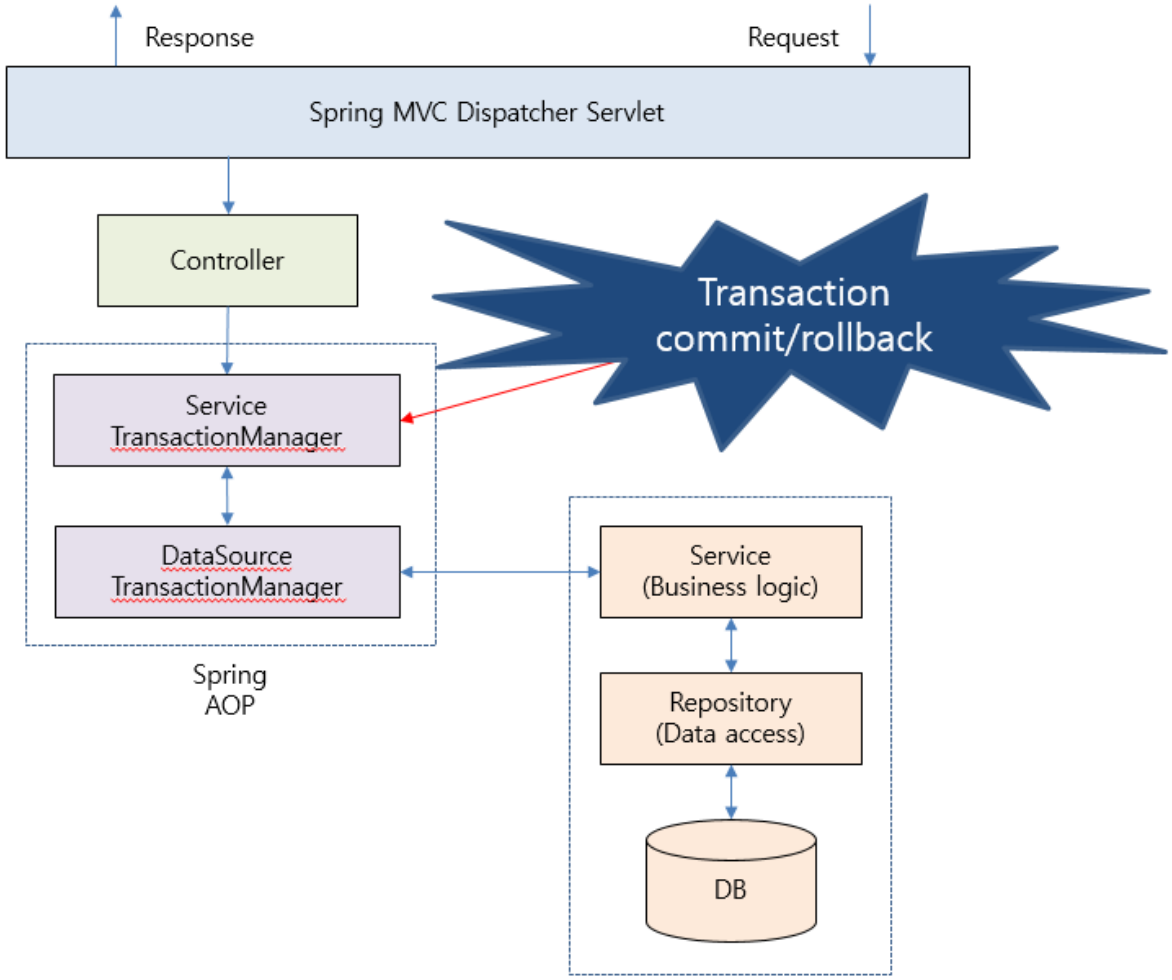
Chamomile Framework에서는 MVC 서비스를 테스트/실행할 수 있는 기능을 제공한다.

테스트를 요청하면 DB transaction이 rollback 되고 실행을 요청하면 commit 된다.



[그림] MVC테스트 아키텍처

Chamomile Framework 의 transactionmanager 인 ServiceTransactionManager 에서 request 정보를 읽고 rollback 할 것인지 commit 할 것인지 판단한다. '테스트' 요청이면 해당 transaction을 rollback 수행하고 '실행' 요청이면 commit 수행한다. 그리고 '테스트', '실행' 요청이 없으면 해당transaction을 commit 수행한다.



[그림] MVC테스트 흐름도

설정

기본적으로 모든 내용이 설정되어있으므로 별도로 설정할 필요는 없다. 참고용으로 활용하도록 한다.

- 동기화 옵션 설정

서비스가 기동되는 시점에 request mapping으로 설정되있는 주소를 모두 스캔하여 데이터베이스에 저장하게된다. 그러나 다중 인스턴스환경하에서(각 개발자들의 로컬PC에서 구동되는 서버 형태 포함) 상이한 주소가 있을경우 동기화시 문제가 발생하는 경우가 있다. 아래는 다양한 환경에 대비하기위한 동기화 옵션이다.

[application.properties]

```
##REQUEST MAPPING SYNC
chmm.requestMappingSync.batch.use=true
chmm.requestMappingSync.delete.use=true
chmm.requestMappingSync.enabled=true
```

설정	내용
chmm.requestMappingSync.batch.use	동기화 방식을 결정한다. true : 배치 방식 사용, 구동시 일괄적으로 저장한다.(운영시 사용권장) false : 낱개 동기화 방식 사용, 데이터 유무에 따라 insert또는 update처리를 한다.(개발시 사용권장)
chmm.requestMappingSync.delete.use	메모리상에 존재하지 않는 건을 DB 에서도 삭제할 건지 결정한다. (※ 개발시에는 false로 설정할 것을 권고한다.) true : 삭제 수행 false : 삭제 미수행
chmm.requestMappingSync.enabled	기능 사용 여부를 설정한다. true : 기능을 사용하며 MVC서비스 목록을 스캔하고 DB에 값을 반영한다. false : 기능을 사용하지 않는다. 서비스 목록을 스캔하지 않고 DB에 값을 반영하지 않는다.(이전에 설정된 값 유지)

- bean정의

Chamomile framework 에서 제공하는 구현체(ServiceRequestMappingHandlerMapping)를 context-servlet.xml 에 선언한다.

```
<context:component-scan />

<!-- Enables the Spring MVC @Controller programming model -->
<beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.annotation.ServiceRequestMappingHandlerMapping" />
<annotation-driven />
```

Chamomile framework 에서 제공하는 구현체(DocumentationConfiguration, RequestMappingInfoSynchronizationConfiguration)와 spring에서 제공하는 ParameterNameDiscoverer 의 구현체(LocalVariableTableParameterNameDiscoverer)를 context-servlet.xml 에 선언한다

```
<beans:bean
class="org.springframework.core.LocalVariableTableParameterNameDiscoverer" />
<beans:bean
class="net.lotte.chamomile.core.documentation.configuration.DocumentationConfigu
ration" />
<beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.configuration.RequestMappingI
nfoSynchronizationConfiguration" />
```

Spring 에서 제공하는 객체(JdbcTemplate)를 context-datasource.xml 에 선언한다.

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <constructor-arg ref="dataSource" />
</bean>
```

Xml 설정 파일에 bean 선언되어 있는 PlatformTransactionManager 의 구현체 id를 useTransactionManager 로 바꾸고 chamomile framework에서 제공하는 구현체 (ServiceTransactionManager)를 context-datasource.xml 에 선언한다.

```
<bean id="useTransactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<bean id="transactionManager"

 class="net.lotte.chamomile.core.jdbc.datasource.ServiceTransactionManager">
    <property name="transactionManager" ref="useTransactionManager"/>
</bean>
```

- transaction 설정

MVC테스트에서 롤백을 수행하기위해 트랜잭션을 설정해준다.(context-datasource.xml)

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
    <tx:attributes>
        <tx:method name="*" rollback-for="Exception" />
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="txPointcut"
    expression= "${chmm.txPointcut.expression}" />

    <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />
</aop:config>
```

- interceptor설정

Chamomile framework 에서 제공하는 interceptor 구현체(ServiceTestInterceptor, ServiceHandlerInterceptor)를 context-servlet.xml 에 선언한다.

```
<interceptors>
    <interceptor>
        <mapping path="/**"/>
```

```
        <exclude-mapping path="/resources/**"/>
        <exclude-mapping path="/admin/login" />
        <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.test.ServiceTestInterceptor"/>
    </interceptor>
    <interceptor>
        <mapping path="/admin/**"/>
        <exclude-mapping path="/resources/**"/>
        <exclude-mapping path="/admin/login" />
        <exclude-mapping path="/admin/mvc/**" />
        <beans:bean
class="net.lotte.chamomile.core.web.servlet.handler.ServiceHandlerInterceptor"/>
    </interceptor>
</interceptors>
```

테스트를 위한 ServiceTestInterceptor 는 /resources/** 를 제외한 모든URL 이 interceptor 를 거치도록 설정한다.

제어를 위한 ServiceHandlerInterceptor 는 MVC 제어 관리 페이지 관련URL(예시: `/admin/mvc/**`), `/resources/**` 를 제외한 모든 URL 이 interceptor 를 거치도록 설정한다. 물론 `<exclude-mapping path=""/>` 에 추가하여 다른 URL 도 제어하지 못하게 할 수 있다. (※ 제어 interceptor는 프로젝트 개발 완료 전에 설정할 것을 권고한다.)

enabled/disable

enabled

- boot에서 사용 시
  - `application.properties` 내 설정 변경

```
    chmm.requestMappingSync.enabled=true
```

- legacy에서 사용시
  - `application.properties` 내 설정 변경

```
chmm.requestMappingSync.enabled=true
```

  - `context-servlet.xml` 내 설정 변경

```
    <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거 -->
    <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.annotation.ServiceRequest
MappingHandlerMapping" />

...
    <interceptors>
        ...

        <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거 -->
        <!-- 서비스 실행/테스트 인터셉터 -->
        <interceptor>
            <mapping path="/**"/>
            <exclude-mapping path="/resources/**"/>
            <!-- <exclude-mapping path="/admin/login" /> -->
```

```
            <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.test.ServiceTestInterceptor"/>
        </interceptor>
        <!-- 서비스 제어 인터셉터 -->
        <interceptor>
            <mapping path="/admin/**"/>
            <exclude-mapping path="/resources/**"/>
            <!-- <exclude-mapping path="/admin/login" />
            <exclude-mapping path="/admin/mvc/**" /> -->
            <beans:bean
class="net.lotte.chamomile.core.web.servlet.handler.ServiceHandlerIntercepto
r"/>
        </interceptor>

    </interceptors>
...
    <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거 -->
    <!-- MVC 관리를 위한 Bean -->
    <beans:bean
class="org.springframework.core.LocalVariableTableParameterNameDiscoverer">
</beans:bean>
    <beans:bean
class="net.lotte.chamomile.core.documentation.configuration.DocumentationCon
figuration"></beans:bean>
    <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.configuration.RequestMapp
ingInfoSynchronizationConfiguration"></beans:bean>
```

disabled

- boot에서 사용 시
  - `application.properties` 내 설정 변경

```
chmm.requestMappingSync.enabled=false
```

- legacy에서 사용시
  - `application.properties` 내 설정 변경

```
chmm.requestMappingSync.enabled=false
```

  - `context-servlet.xml` 내 설정 변경

```
    <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거
    <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.annotation.ServiceRequest
MappingHandlerMapping" />
    -->
...
    <interceptors>
        ...

        <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거 -->
        <!-- 서비스 실행/테스트 인터셉터
        <interceptor>
```

```
            <mapping path="/**"/>
            <exclude-mapping path="/resources/**"/>
            < ! - - <exclude-mapping path="/admin/login" /> - - >
            <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.test.ServiceTestInterceptor"/>
        </interceptor>
        < ! - - 서비스 제어 인터셉터 - - >
        <interceptor>
            <mapping path="/admin/**"/>
            <exclude-mapping path="/resources/**"/>
            < ! - - <exclude-mapping path="/admin/login" />
            <exclude-mapping path="/admin/mvc/**" /> - - >
            <beans:bean
class="net.lotte.chamomile.core.web.servlet.handler.ServiceHandlerIntercepto
r"/>
        </interceptor>
        -->
    </interceptors>
...
    <!-- MVC제어기능, 시나리오 테스트 enable시 아래 주석 제거 -->
    <!-- MVC 관리를 위한 Bean
    <beans:bean
class="org.springframework.core.LocalVariableTableParameterNameDiscoverer">
</beans:bean>
    <beans:bean
class="net.lotte.chamomile.core.documentation.configuration.DocumentationCon
figuration"></beans:bean>
    <beans:bean
class="net.lotte.chamomile.core.servlet.mvc.method.configuration.RequestMapp
ingInfoSynchronizationConfiguration"></beans:bean>
-->
```

업무시스템에 적용

MVC제어 및 테스트 기능은 MVC패턴으로 개발시 효율성을 제공하고자 하는데 목적이 있다.

우선 서버에서 기능구현을 완료한 후에 MVC테스트를 통해서 테스트를 완료한 후 화면을 개발하게 되면 개발자들간 원할한 커뮤니케이션을 도모할 수 있다.

개발절차

- 기능 정의
  - 설계서를 통해 필요한 기능을 도출한다.
  - 도출한 기능이 관리자의 승인이 필요한 것인지 판단한다. 필요하다면 annotation을 통해 설정한다.
- 설정
  - 동기화 옵션을 설정한다.(운영환경과 개발환경 구분)
  - 기능정의를 통해 설계한 내용을 바탕으로 XML을 작성한다.
- 트랜잭션의 경우 업무시스템에도 적용해야 하므로 여러 개를 설정하기위해서는 아래와 같이 pointcut의 표현식에 “or” 로 연reloadableFilterInvocationSecurityMetadataSource결하여 설정한다.

application.properteis

```
##Transaction Pointcut
chmm.txPointcut.expression=(execution(* net.lotte.sample..*ServiceImpl.*(..)) or
execution(*
net.lotte.chamomile.core.servlet.mvc.method.RequestMappingInfoSynchronization.*
(..)))
```

- Java class작성
  - 기능을 구현하는 class를 작성한다.
- Controller의 설명추가 및 제어를 하고자 할경우 @ServiceContents 어노테이션을 추가한다.

```
@RequestMapping(value = "/test03", method = RequestMethod.GET)
@ServiceContents(description = "승인 요청 있음", approvalRequired = true)
public String test03() {
    return "success";
}
```

@ServiceContents의 속성

속성	타입	설명
description	String	어떤 기능을 하는지를 부여
approvalRequired (기본값: false)	boolean	true : 관리자 승인이 필요한 기능이면 true를 명시 false : 관리자 승인이 필요 없는 기능이면 false를 명시

관리자 승인은 운영 가이드를 참조한다.

- MVC제어 대상에서 제외하고자 할경우 @ApiIgnore 어노테이션을 추가한다. 대상은 클래스에 적용 할 수 있으며 클래스에 포함된 request mapping이 적용된 메서드는 전부 제외 시킬 수 있다.

```
@Controller
@ApiIgnore
@RequestMapping(value = "/admin/mvc")
public class ServiceController {
```

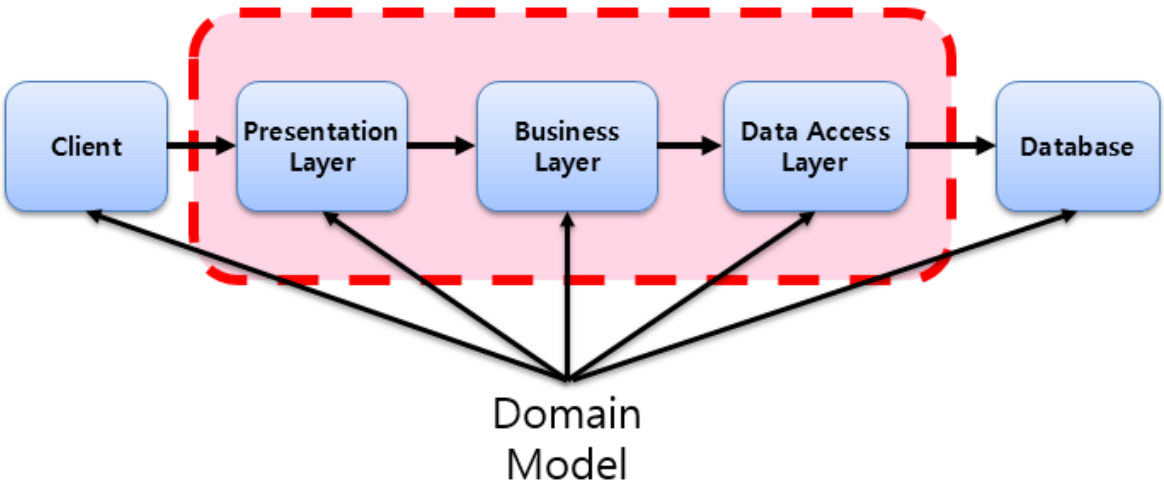
- 테스트
  - 구현한 기능을 테스트한다.

Validation

개요

- Bean Validation

Bean Validation이란 데이터를 검증하기 위한 JAVA 표준 기술이다. 검증 규칙은 Annotation으로 사용하며 데이터의 검증은 Runtime 시에 이루어진다.



[그림] validation

- Valid Annotation

Valid Annotation을 이용하여 Validation 체크한다. Spring에서는 사용자가 입력한 값에 대한 유효성을 체크하기 위해 Spring Validator 또는 JSR-303 Validator를 사용할 수 있도록 지원하고 있다.

Spring 3.0 에서 `mvc:annotation-driven` 을 통해 간단하게 Bean Validation 을 사용할 수 있다.

Validation제공범위

JSR303 spec 기본제공 Constraint Annotation

Annotation	제약조건
@AssertFalse	거짓인가?
@AssertTrue	참인가?
@DecimalMax	지정값 이하 실수인가?
@DecimalMin	지정값 이상 실수인가?
@Digits(integer=, fraction=)	대상수가 지정된 정수, 소수자리수이내인가?
@Future	미래날짜인가?
@Past	과거날짜인가?
@Max	지정값 이하인가?
@Min	지정값 이상인가?
@NotNull	NotNull이 아닌가?
@Null	Null인가?
@Pattern(regex=, flag=)	정규식을 만족하는가?
@@Size(min=, max=)	문자열,배열등의 크기가 지정크기를 만족하는가?

- Hibernate 제공 Constraint Annotation

Annotation	제약조건
@CreditCardNumber (ignoreNonDigitCharacters=)	주석 된 문자 시퀀스가 Luhn 체크섬 테스트를 통과하는지 확인한다. 참고로,이 유효성 검사는 신용 카드 유효 기간이 아닌 사용자 실수를 확인하는 것을 목표로한다. 신용 카드 번호 분석을 참조한다. ignoreNonDigitCharacters 비 숫자 문자는 무시할 수 있다. 기본값은 false 이다.
@EAN	주석 된 문자 시퀀스가 유효한 EAN 바코드인지 확인한다. type은 바코드의 유형을 결정한다. 기본값은 EAN-13 이다.
@Email	지정된 문자 시퀀스가 유효한 전자 메일 주소인지 여부를 확인한다.
@Length(min=, max=)	순서가 그 사이 min에 max포함되어 있는지를 검증한다.
@LuhnCheck (startIndex= , endIndex=, checkDigitIndex=, ignoreNonDigitCharacters=)	startIndex그리고 endIndex는 지정된 하위 문자열에 알고리즘을 실행할 수 있다. checkDigitIndex 문자 시퀀스 내의 임의의 숫자를 체크 디지트로 사용할 수 있다. 지정하지 않으면 검사 숫자가 지정된 범위의 일부인 것으로 간주된다. 마지막으로 중요한 것은 ignoreNonDigitCharacters 비 숫자 문자를 무시할 수 있다.
@Mod10Check (multiplier=, weight=, startIndex=, endIndex=, checkDigitIndex=, ignoreNonDigitCharacters=)	multiplier홀수 (기본값은 3)에 대한 승수, 짝수에 weight대한 가중치 (기본값은 1)를 결정한다. startIndex그리고 endIndex는 지정된 하위 문자열에 알고리즘을 실행할 수 있다. checkDigitIndex 문자 시퀀스 내의 임의의 숫자를 체크 디지트로 사용할 수 있습니다. 지정하지 않으면 검사 숫자가 지정된 범위의 일부인 것으로 간주된다. 마지막으로 중요한 것은 ignoreNonDigitCharacters 비 숫자 문자를 무시할 수 있다.
@Mod11Check(threshold=, startIndex=, endIndex=, checkDigitIndex=, ignoreNonDigitCharacters=, treatCheck10As=, treatCheck11As=)	thresholdmod11 배율 증가에 대한 임계 값을 지정합니다. 값을 지정하지 않으면 승수가 무기한 증가한다. treatCheck10As 및 treatCheck11As유행의 체크섬 (11)가 10 또는 11와 동일 할 때, 각각 사용되는 체크 디지트를 지정. 기본값은 각각 X와 0이다. startIndex에서 endIndex checkDigitIndex와 ignoreNonDigitCharacters동일한 의미를 지닌다 @Mod10Check.
@NotBlank	주석이 붙은 문자 순서가 null는 아니고, 잘라 버린 길이가 0보다 큰지를 판정한다.
@NotEmpty	null도 공백도 아닌지를 판정한다.
@Range(min=, max=)	값이 지정된 최소값과 최대 값 사이에 있는지 여부를 확인한다.

Constraint Annotation (chamomile)

Annotation	제약조건
@AccountNumber	신한은행(구), 신한은행, 국민은행, 우리은행, IBK, 하나은행, 대구은행, 대구NoPay은행, 부산은행
@CreditCardNumber	MasterCard, Visa, KoreaCard, AMEX, DinnerClub1, DinnerClub2
@JuminNumber	주민번호 체크
@PassportNumber	여권번호 체크
@PhoneNumber	UKPhone, KRPhone

사용법

- dependency

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
</dependency>
```

- Validation적용하기

유효성을 검사할 VO객체의 필드에 annotation을 설정한다.

```
public class UserVO {

    @NotNull @NotEmpty @Size(min=5, max=50)
    private String userId = ""; /* 사용자아이디 */

    @NotNull @NotEmpty @Email @Size(max=255)
    private String userEmail = ""; /* 사용자이메일 */
...
```

유효성을 검사할 controller 메서드의 parameter에 `@valid` annotation을 추가한다. 입력값이 유효하지 않을경우 403오류가 발생한다.

```
@RequestMapping("/create")
public ModelAndView regist(@Valid UserVO userVO
                           , HttpServletRequest req
                           , HttpServletResponse res) throws Exception {
  logger.debug("데이터 추가");
  ModelAndView mv = new ModelAndView("service/demoBoard/demoBoardForm");

  return mv;
}
```

아래와 같이 `BindingResult`를 통하여 결과를 직접 제어할 수 있다.

```
@RequestMapping(value = "/userSave", method = RequestMethod.POST)
protected ModelAndView userSave(@Valid UserVO userVO
        , BindingResult bindingResult
        , ModelMap model) throws Exception {

    /* Vo Validation */
    if (bindingResult.hasErrors()) {
        FieldError fieldError = bindingResult.getFieldError();
        logger.info("Validation Error Field : {}", fieldError.getField());
        logger.info("Validation Error Code : {}", fieldError.getCode());
        throw new ValidationException(fieldError.getDefaultMessage());
    }

    ...
}
```

Custom validation

업무시스템에 맞게 Validation을 추가할 수있다. 전화번호의 유효성을 체크하는 Validation을 만들어 보고자한다.

- Annotation정의

```
@Documented
//검증을 수행할 클래스를 지정한다.
@Constraint(validatedBy = PhoneNumberValidator.class)
//annotation을 적용할 타겟을 설정한다.
@Target( { ElementType.METHOD, ElementType.FIELD, ElementType.PARAMETER })
@Retention(RetentionPolicy.RUNTIME)
public @interface PhoneNumber {
    String message() default "Invalid Phone Number"; //오류메시지 정의
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

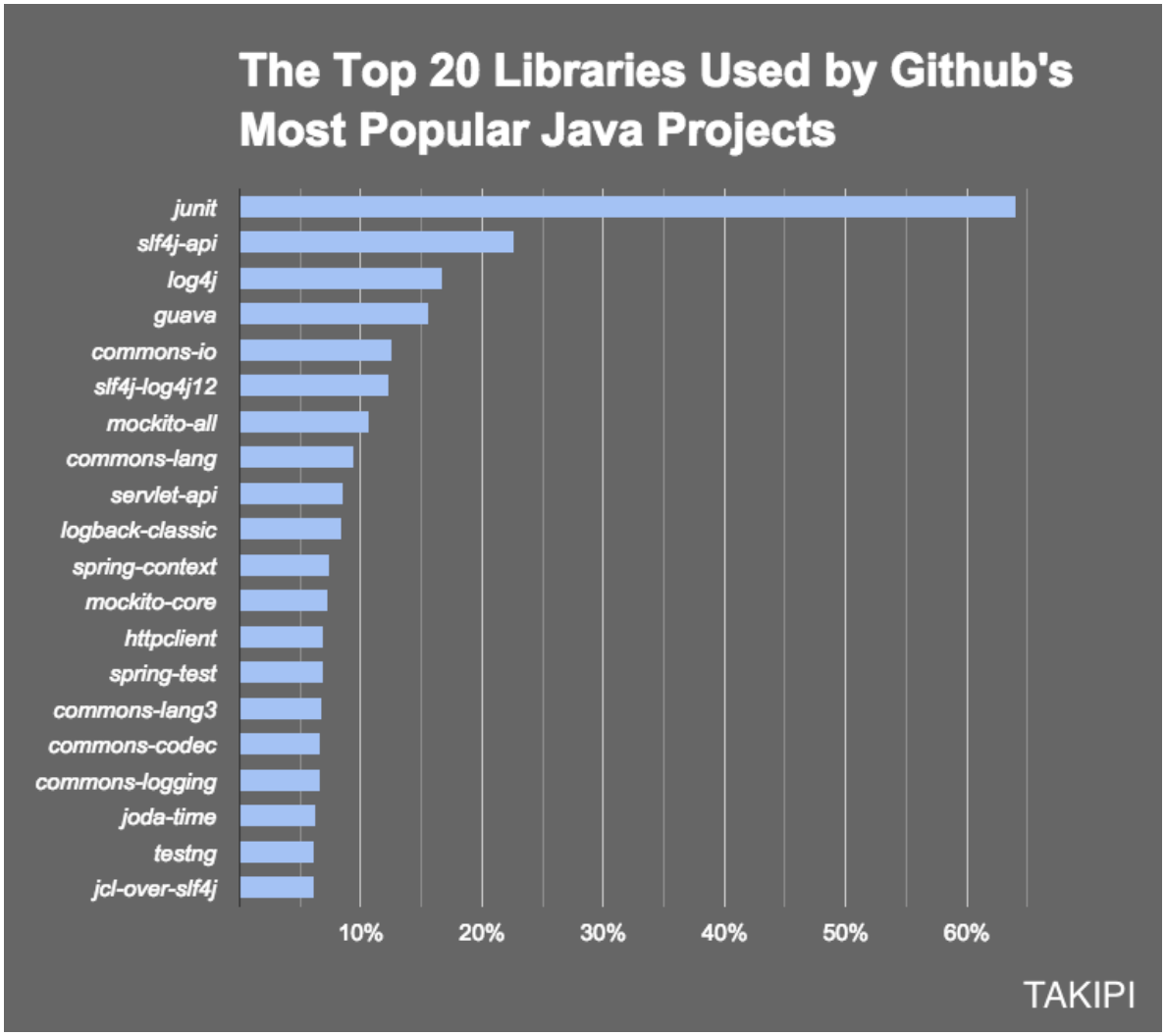
- Validator 생성

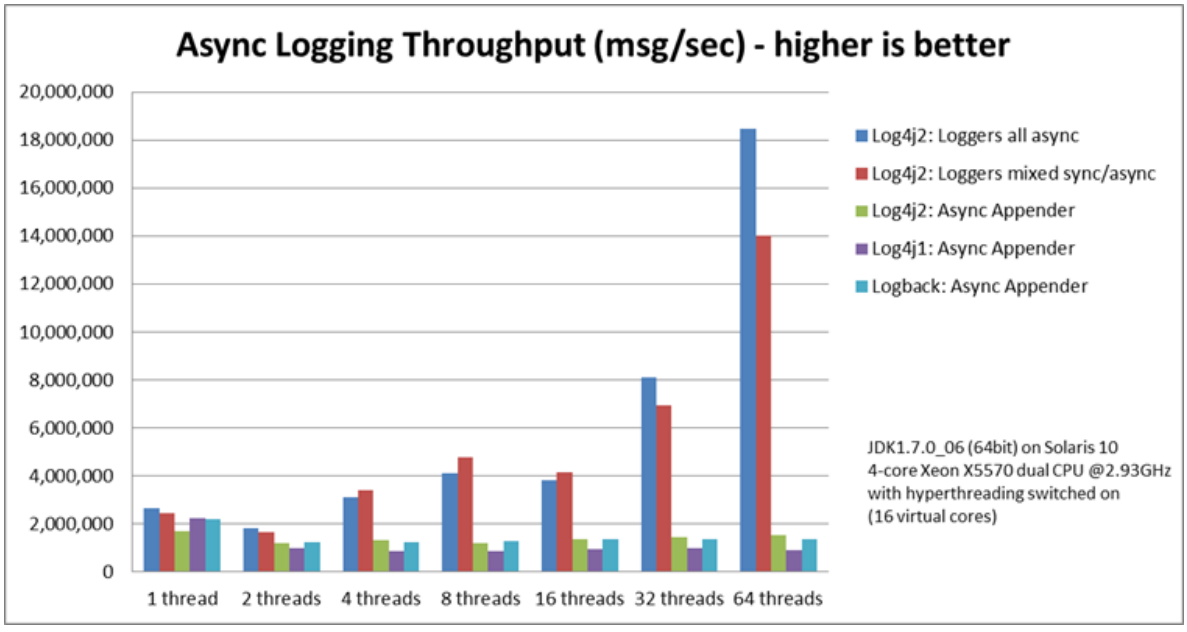
```
public class PhoneNumberValidator implements ConstraintValidator<PhoneNumber,
String> {
    @Override
    public void initialize(PhoneNumber phoneNumber) {
    }
    //Validation수행
    @Override
    public boolean isValid(String contactField,
      ConstraintValidatorContext cxt) {
        //데이터 검증
        return contactField != null
                && contactField.matches("[0-9]+")
                && (contactField.length() < 8)
                && (contactField.length() < 14);
    }
}
```

Logging

개요

프레임워크에서 사용되는 로깅 프레임워크는 slf4j와 log4j2를 이용하여 로깅을 수행한다.로깅에 대한 성능과 안정성을 확보하기 위해 선정 하였다. (비동기 로거)





자세한 내용은 아래 분석자료를 참조한다.

- OverOps (소프트웨어분석회사), loggly (로그관리 및 분석 서비스업체) 분석 자료

https://www.sitepoint.com/which-java-logging-framework-has-the-best-performance/

https://www.loggly.com/blog/benchmarking-java-logging-frameworks/

https://blog.takipi.com/the-top-100-java-libraries-in-2017-based-on-259885-source-files/

https://logging.apache.org/log4j/2.0/performance.html

로깅

- 디버깅과 로깅의 비교

CATEGORY /OPERATION	DEBUGGING	LOGGING
관리자 개발자 중재	관리자, 개발자의 중재와 개입이 필요	관리자 등의 개입이 필요 없음
영속성 매체	영속성 저장소와 통합될 수 없음	파일, 데이타베이스, NoSQL등의 매체와 통합 운용 가능
감사	감사 추적 방식으로 적용되기 어려움	감사, 추적에 효율적으로 사용 가능
복잡한 구조와 절차에 적합여부	특정 절차 등에서 수행되기 어려움. 부적합	적합
생산성	생산성 낮음	생산성 높음

- 로그유형

로그레벨	분류	사용목적	로그내용
TRACE	성능로그	요청처리시간 측정 (운영환경 제외)	프로세스 시작 및 종료시간, 측정된 처리 소요시간 실행 처리 프로세스를 정의할 수 있는 정보 (컨트롤러, 메소드, 요청 URL 등)
DEBUG	디버그로그	개발시 디버깅 용도 (운영환경에서 제외)	선택적(실행된 질의정보, 입력 파라미터, 리턴 값 등)
INFO	접근로그	업무프로세스 접근로그	접근 시간, 날짜, 사용자 (IP, 인증정보), 실행 프로세스 요청 URL, 추적정보
INFO	외부통신로그	외부시스템과의 통신 간에 발생하는 오류 분석	송수신 시간, 송수신 데이터
WARN	업무프로세스 오류 로그	업무프로세스 오류 기록	업무 프로세스 오류 시간, 메시지 ID, 메시지 대응 오류 정보 (입력정보, 예외 메시지)
ERROR	시스템 오류 로그	시스템기능 실행 오류 이벤트 기록	시스템 오류 시간, 메시지, 메시지 ID, 입력정보, 예외 메세지
ERROR	모니터링로그	예외발생 모니터링	예외 발생 시간, 메시지 ID

로그마스킹

개요

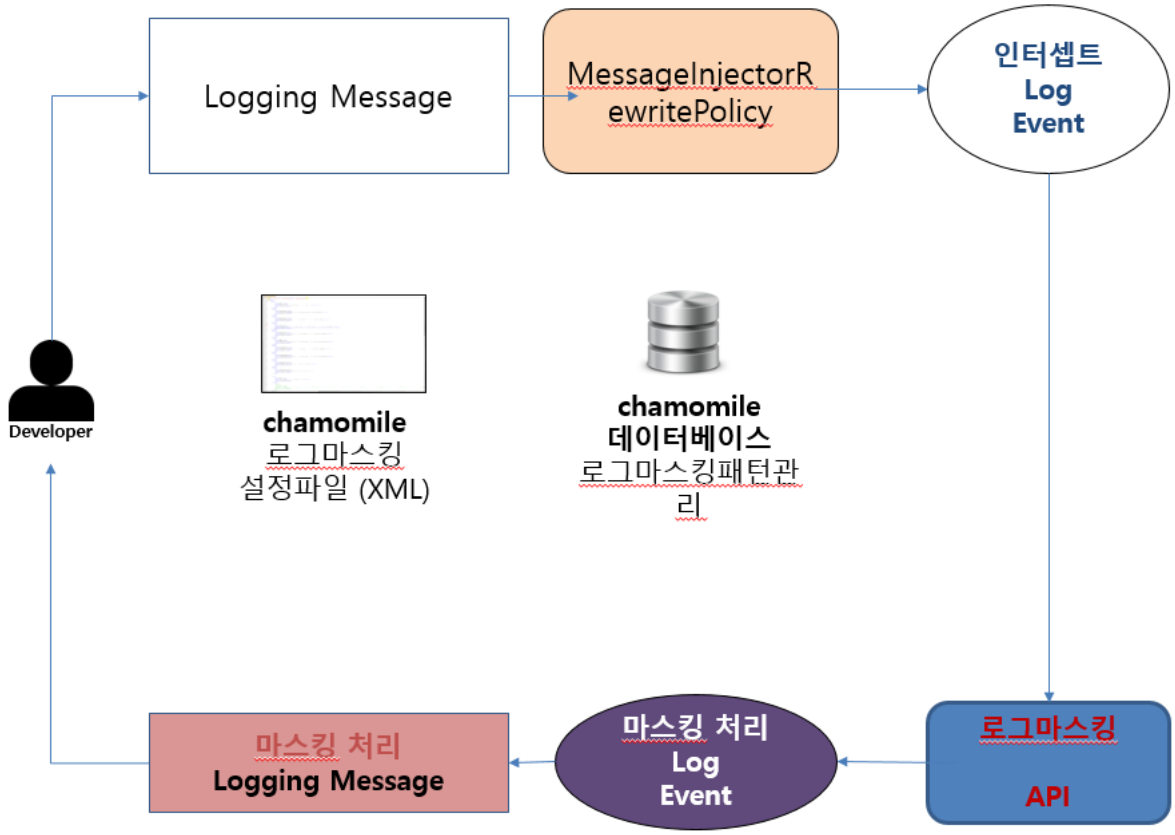
로그마스킹 이란 로깅 시 중요정보(주민번호, 비밀번호, 카드번호 등)에 대해 알아볼 수 없는 형태의 데이터로 변환하여 로깅하는 것을 말한다.

무분별하고 적절하지 못한 로그마스킹 처리를 하면 예상 외의 결과가 발생할 수 있기 때문에 로그마스킹할 기밀정보는 신중하게 선별해서 개발해야 한다.

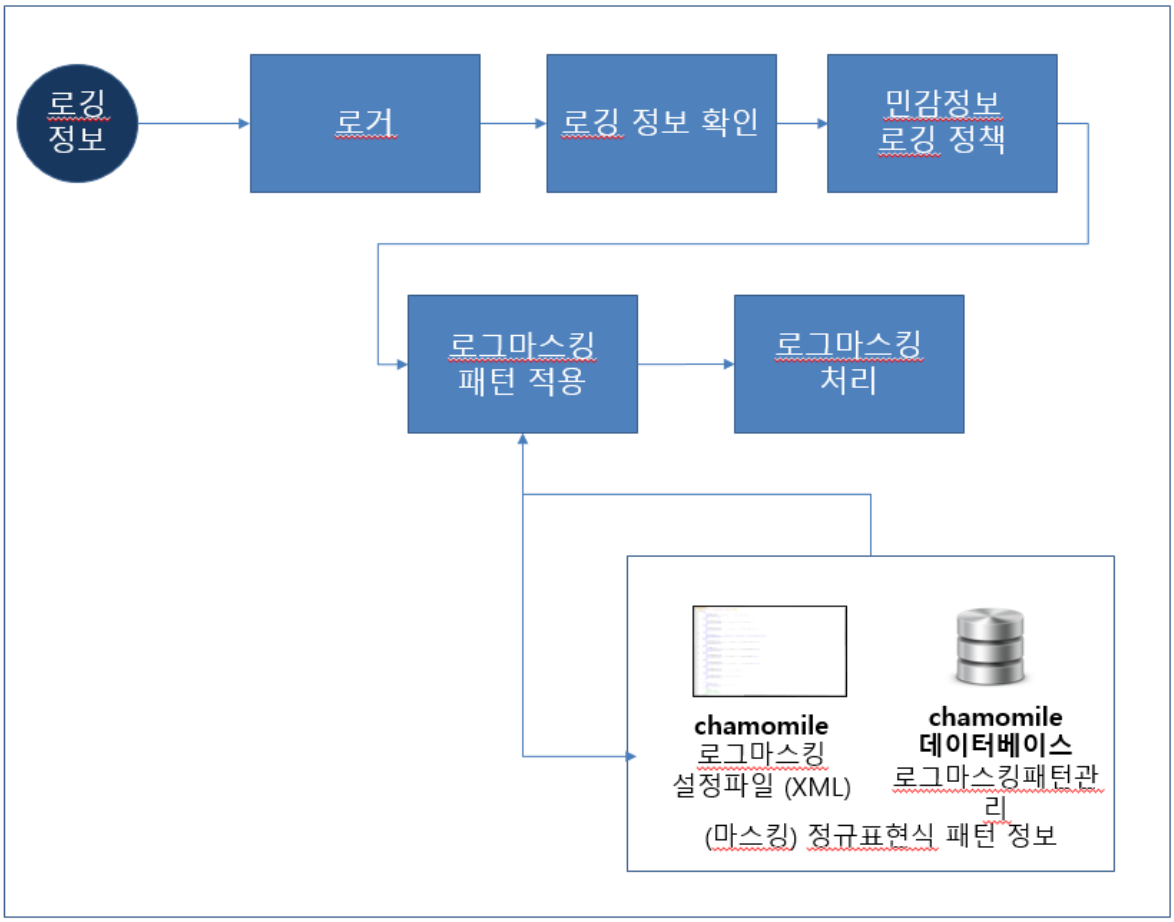
Chamomile Framework 에서는 로그마스킹 처리할 대상에 대한 정규표현식을 추가하거나, 제외할 수 있는 기능을 제공한다. (정규표현식 참고 https://www.regular-expressions.info/examples.html)

자바에서 제공하는 Matcher 를 통해 패턴 매칭하고, 설정파일(XML 파일), 어드민 시스템설정 > 로그마스킹 패턴관리 기능을 통해 로그마스킹 처리할 대상을 추가 설정하거나, 삭제할 수 있다. 추가된 패턴 정보는 enabled 상태에 따라 ON/OFF 설정 기능을 제공한다.

Chamomile 로그마스킹 기능은 설정파일이나, Annotation 을 통한 중요필드 설정에 대한 개발 가이드를 제공한다.



[그림]로그마스킹 프로세스



[그림] 로그마스킹 순서도

- 로그마스킹처리대상

정보대상	제공방식	기본 설정 유무	Chamomile프레임워크 제공 여부
아이디	비정규 표현 형태 (특정 문자열)	O	O
비밀번호	비정규 표현 형태 (특정 문자열)	O	O
주민 번호	정규표현식	O	O
카드 번호	정규표현식	O	O
휴대폰 번호	정규표현식	O	O
여권 번호	정규표현식	O	O
은행권 계좌 번호	정규표현식	O	O

- 로그마스킹기능목록

항목	제공방식	Chamomile프레임워크 제공 여부
Email	정규표현식	O
신용카드 (MasterCard, Visa, KoreaCard, AMEX, DinnerClub,	정규표현식	O
아이피 주소 (IPv4, IPv6)	정규표현식	O
핸드폰 번호 (US, UK, Korea)	정규표현식	O
URL 주소	정규표현식	O
여권 번호 (US, Korea)	정규표현식	O
우편번호 (Korea)	정규표현식	O

사용법

설정

기본적으로 모든 내용이 설정되어있으므로 별도로 설정할 필요는 없다. 참고용으로 활용하도록 한다.

- log4j설정

Chamomile 프레임워크에서는 기본적으로 log4j2 를 사용해서 로깅하며 그와 관련해서 로그마스킹 처리를 위한 가이드를 별도로 제공한다.

Chamomile 프레임워크 에서는 SLF4j (Logging Façade) 인터페이스와 log4j2 구현체를 기본으로 사용한다.

로그마스킹 처리를 위해서 Appender 를 통해서 내려오는 LogEvent 에 대한 처리가 필요한데,
org.apache.logging.log4j.core.filter.AbstractFilter 를 사용하거나,
org.apache.logging.log4j.core.appender.rewrite.RewritePolicy를 사용하는 방법이 있다.
AbstractFilter를 사용하는 경우 로그마스킹 처리 전에 LogEvent 가 내려지는 문제가 발생해,
RewritePolicy 를 사용해서 Log Event에 대한 로그마스킹 처리한다.

로그마스킹 처리를 위해서는 log4j2.xml 파일 내에 플러그인 설정이 필요하다.

무분별하게 사용하는 경우 성능에 영향을 미칠 수 있다.

[log4j2.xml]

```
<Configuration status="error"
packages="net.lotte.chamomile.admin.log.monitor.filter,
net.lotte.chamomile.core.log.mask.filter" monitorInterval="1">
    <Appenders>
        ...
        <!-- Rewite를 많이 처리 하는 경우 성능 저하 발생. -->
        <!-- 마스킹 플러그인 설정(LogMaskingPolicy)-->
        <Rewrite name="maskDailyRollingFile">
```

```
            <LogMaskingPolicy />
        </Rewrite>
        <Rewrite name="maskErrorRollingFile">
            <LogMaskingPolicy />
        </Rewrite>
        <Rewrite name="maskSqlDailyRollingFile">
            <LogMaskingPolicy />
        </Rewrite>
        <Rewrite name="maskSqlErrorDailyRollingFile">
            <LogMaskingPolicy />
        </Rewrite>

        ...
    <Appenders>

    <Loggers>
        <!-- 로거 appender에 마스킹 적용 -->
        <!-- sql logging -->
        <Logger name="java.sql" level="FATAL" additivity="false">
            <AppenderRef ref="maskSqlDailyRollingFile" />
        </Logger>
        ...
    </Loggers>

    <!-- Root Logger -->
    <Root level="INFO">
        <AppenderRef ref="maskDailyRollingFile" />
    </Root>
</Loggers>
```

- 로그마스킹 관련 설정 파일 목록

설정파일	역할
maskers_default.xml	마스킹 처리 관련 실행 클래스 및 마스킹 처리할 패턴 정보에 대한 타켓 정보(XML, DB)를 설정해 놓은 XML 파일
finders_default.xml	마스킹 처리할 정규표현식 패턴을 정리해 놓은 XML 파일
log_masking.properites	정규표현식 패턴이랑 매칭되는 문자열을 대체할 문자열에 대해서 정의해 놓은 프로퍼티 파일 ※ 정규표현식 패턴 정보 의 패턴 명이랑 1:1 매칭이다.
unregex.properties	특정 문자열에 대한 마스킹 처리가 필요할 경우 특정 문자열에 대해 정의해 놓은 프로퍼티 파일 ※ 정규표현식 패턴 형태가 아닌 특정 문자열에 대한 로그 마스킹 처리가 필요한 경우 프로퍼티 설정을 통해서 특정 문자열이 시작되는 로그 메시지에 대해서 마스킹 처리가 가능하다.

가) maskers_default.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<maskers>
    <masker>
        <name>LogMaskingExecutor</name>
        <class>net.lotte.chamomile.core.log.mask.LogMaskingExecutor</class>

<configuration>classpath:config/mask/log_masking.properties</configuration>
        <logmaskopt>DB</logmaskopt>
    </masker>
    <default>LogMaskingExecutor</default>
</maskers>
```

property설명

속성	설명	기본값	제공여부
masker	실행클래스 정의		O
name	마스커 이름 정의		O
class	실행클래스		O
configuration	대체문자열을 정의한 프로퍼티 파일명	log_masking.properties	O

속성	설명	기본값	제공여부
logmaskopt	패턴정보 타켓(XML, DB)	XML	O
dataSourceName	프레임워크에서 사용되는 datasource명이 변경되었을 경우 사용한다.	dataSource	O
maskingEndCharacter	비정규식으로 마스킹 할 경우 마스킹을 종료할 문자를 설정한다. 정규식으로 적용한다. 문자를 한 글자씩 비교하여 처리한다. 지정하지 않을 경우 마스킹 대상 문자열을 만났을 때 부터 제일 마지막 글자까지 마스킹 처리한다.	[정규식 예제] : 컴마, 온점, 공백을 종료문자로 하고자 할 경우 [,.]	O
default	기본 지정 마스커 정의		O

나) finders_default.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<finders>
   <finder>
         <name>Email</name>
         <pattern>\b[A-ZO-9._%+-]+@([A-ZO-9.-]+)\.([A-Z]{2,4})\b</pattern>
         <enabled>false</enabled>
    </finder>
    <finder>
         <name>MasterCard</name>
         <pattern>
5[1-5][0-9]{2}(\\ |\\-|)[0-9]{4}(\\ |\\-|)[0-9]{4}(\\D|$)
</pattern>
         <enabled>false</enabled>
    </finder>
    <finder>
         <name>Visa</name>
         <pattern>
4[0-9]{3}(\\ |\\-|)[0-9]{4}(\\ |\\-|)[0-9]{4}(\\ |\\-|)[0-9]{4}(\\D|$)
</pattern>
         <enabled>false</enabled>
    </finder>
...
```

property설명

속성	설명	기본값	제공 여부
finder	정규표현식 한개당 각각 1개의 파인더로 구성		O
name	정규표현식 이름		O
pattern	정규표현식 패턴 정보		O
class	실행 클래스	지정이 없을 경우 RegexFinder가 디폴트 실행	O
enabled	실행여부	true	O

다) log_masking.properties

```
Email=xxxx@xxxx.xxx
CreditCard=XXXX-XXXX-XXXX-XXXX
Visa=XXXX-XXXX-XXXX-XXXX
MasterCard=XXXX-XXXX-XXXX-XXXX
KoreaCard=XXXX-XXXX-XXXX-XXXX
AMEX=XXXX-XXXX-XXXX-XXXX
DinnerClub1=XXXX-XXXX-XXXX-XXXX
DinnerClub2=XXXX-XXXX-XXXX-XXXX
USPhone=XXX-XXX-XXXX
UKPhone=XXX-XXX-XXXX
KRPhone=XXX-XXXX-XXXX
URL=http://xxx.xxxxx.xxxxx
Hostname=hostname.redated
IPV4=XXX.XXX.XXX.XXX
IPV6=XXXX.XXXX.XXXX.XXXX.XXXX.XXXX.XXXX.XXXX
PostCode=XXX-XXX
Passport=XXXXXXXXXX
JuminNumber=XXXXXX-XXXXXXX
KoreaPassport=XXXXXXXXX
BankShinHanOld=XXX-XX-XXXXXX
BankShinHan=XXX-XXX-XXXXXX
BankKookmin=XXXXXX-XX-XXXXXX
BankWoori=XXXX-XXX-XXXXXX
BankIBK=XXX-XXXXXX-XX-XXX
BankHana=XXX-XXXXXX-XXXXX
BankDaegu=XXX-XX-XXXXXX-X
BankDaeguNoPay=XXX-XX-XXXXX
BankPusan= XXX-XXXX-XXXX-XX
```

라) unregex.properties

```
id=XXXX
pass=XXXX
pwd=XXXX
password=XXXX
아이디=XXXX
비밀번호=XXXX
```

> finders_default.xml, log_masking.properties 파일은 어드민 화면에서도 조작이 가능하며 자세한 내용은 운영자 가이드를 통해 확인한다.

업무시스템 적용

- 마스킹 api사용법

```
//마스킹 정보를 얻어오기위해 데이터 소스 사용
@Autowired
DataSource dataSource;
…

FinderEngine engine = new FinderEngine((List<Finder>) null, true, true);
MaskFactory maskFactory = new MaskFactory(dataSource, engine);
Masker masker = maskFactory.getMasker();

String result = masker.mask("마스킹할 문자열");
```

- 어노테이션으로 적용하는 방법

  @MaskingAnnotation으로 적용한다. regex에 사용할 정규 표현식을 사용할 수도 있고 별도로 지정하지 않을경우 설정된 마스킹으로 처리한다.

```
public class MemberVO {

    private String userId = "";
    @MaskingAnnotation
    private String password = "";

    @MaskingAnnotation(
        regex="^(?:[0-9]{2}(?:0[1-9]|1[0-2])(?:0[1-9]|[1,2][0-9]|3[0,1]))-[1-4]
[0-9]{6}$"
    // 사용할 정규 표현식
    )
    private String juminNo = "";
    ...
```

- Finder를 커스터마이징 하는 방법

Chamomile 로그마스킹은 설정파일을 통한 마스킹 처리의 경우 기본적으로 정규 표현식을 이용해서 패턴 매칭을 통한 문자열을 검색해 마스킹 처리한다. 이때 패턴 매칭에 사용하는 클래스가 RegexFinder를 사용하는데, Chamomile 프레임워크를 사용하는 개발자는 Finder 인터페이스를 이용해서 Custom Finder를 개발할 수 있으며,
설정 파일 (finder_default.xml) 사용할 클래스 지정을 통해 해당 정보 항목에 대해 Custom Finder를 사용할 수 있다.

```
public class CustomFinder implements Finder {// Finder 인터페이스 구현
    /**
     * Finder 명 리턴
     * @return name finder 이름
     */
    @Override
    public String getName() {
        return null;
    }
```

```
    /**
     * 입력 문자열에 대해서 매칭 결과 값을 찾는 메소드
     * @param input 입력 로그 파일
     * @return FinderResult Finder 매칭 결과값
     */
    @Override
    public FinderResult find(String input) {

        return null;
    }
}
```

- 패턴매칭을 이용한 Finder예시

find메소드의 리턴값은 FinderResult에 찾은 문자열 리스트와 변환할 문자열을 인자로 하여 객체를 생성하여 리턴시킨다.

```
public class CustomFinder implements Finder {// Finder 인터페이스 구현
    public static final int DEFAULT_FLAGS = Pattern.CANON_EQ |
Pattern.CASE_INSENSITIVE | Pattern.MULTILINE;

    protected String name = "";             // 명칭
    protected Pattern pattern;              // 패턴명

    /**
     * 생성자
     */
    public CustomFinder() {
    }

    /**
     * 생성자
     * @param name          명칭
     * @param pattern       패턴명
     */
    public CustomFinder(String name, String pattern) {
        this(name, pattern, DEFAULT_FLAGS);
    }

    /**
     * 생성자
     * @param name          명칭
     * @param pattern       패턴명
     * @param flags         플래그값
     */
    public CustomFinder(String name, String pattern, int flags) {
        this.name = name;
        try {
            this.pattern = Pattern.compile(pattern, flags);
        } catch (PatternSyntaxException e) {
            LOGGER.error(e.getMessage(), e);
        } catch (IllegalArgumentException e) {
            LOGGER.error(e.getMessage(), e);
        }
```

```
    }

    /**
     * 패턴 명칭 조회
     * @return name 패턴 명칭
     */
    @Override
    public String getName() {
        return name;
    }

    /**
     * 입력 문자열에서 정규표현식에 따른 매칭 문자열을 찾아서 리턴시켜 주는 메소드
     * @param input 입력 문자열
     * @return FinderResult 찾은 결과
     */
    @Override
    public FinderResult find(String input) {
        List<String> matches = new ArrayList<String>();

        if(pattern != null) {
            Matcher matcher = pattern.matcher(input);
            // boolean isFinder = matcher.find();

            while(matcher.find()) {
                // LOGGER.debug("finder >>> " + this.getName());
                matches.add(input.substring(matcher.start(), matcher.end()));
                // LOGGER.debug("matcher >>> " +
input.substring(matcher.start(), matcher.end()));

            }
        }
        return new FinderResult(matches, replace(input, ""));
    }
}
```

설정파일에 생성한 클래스 적용하기.(finders_default.xml)

```
<finders>
    ...
    <finder>
         <name>CustomFinder</name>
         <class>com.lotte.finder.demo.CustomFinder</class>
         <enabled>true</enabled>
    </finder>
    ...
</finders>
```

Security

개요

어플리케이션 보안이란 서버 보안의 한 영역으로써 응용 소프트웨어의 결함, 시스템 개발의 취약점, 코드의 생명주기 전체를 포함하고 있다.

대부분의 기업들은 네트워크 보안과 시스템 보안 측면에 대해서는 이해를 하고 보안을 구축하려고 노력한다.

어플리케이션 보안은 네트워크/시스템 계층보다 고도화 되어 있으며, 대부분의 보안 책임자들이 적용하는데 어려움을 겪고 있다.

대부분의 어플리케이션이 웹을 통해 접근하고 있으며, 웹 보안에 있어 가장 중요도가 높은 부분이 어플리케이션 보안이다.

- 인증 (Authentication)
  인증은 자신이 누구라고 주장하는 것을 확인하거나 확증하는 행위이다.
  인증의 분류는 아래와 같다.
  1. 자격(Credential) 기반 인증
     - 웹에서 사용하는 대부분의 인증 방식이며, 로그인 요청을 통해 통신상의 보내는 사람의 정보를 확인하는 과정을 말한다.
  2. 이중 (Two-factor) 인증
     - 한번에 2가지 방식으로 인증받는 것을 말하며, 예를들어 OTP(일회용암호)와 같은 방식을 말한다.
  3. 물리적 인증
     - 가장 효과적인 보안수단 중 하나이며, 생체인식(지문, 홍채, 얼굴 등)을 통한 인증을 말한다.
- 권한 부여 (Authorization) 혹은 접근제어
  권한 부여는 누군가에게 무엇을 할 수 있게 하거나, 원하는 정보를 얻을 수 있도록 허용하는 과정이다.
  권한 부여는 아래 내용을 포함한다.
  1. 부여된 권한 (Granted Authority)
     - 적절한 절차로 사용자가 인증되었다면 권한을 부여해야 한다.
     - 해당 권한을 토대로 시스템에 접근하게 된다.
  2. 리소스의 권한
     - 리소스로의 접근은 권한이 존재하는 경우에만 허용해야 한다. (물론 예외적인 리소스도 존재한다.)
     - 리소스에 접근할 때 설정 된 권한의 유무를 확인하는 절차가 필요하다.
- Spring security
  Chamomile 프레임워크의 Security기능은 spring security 기반하에 작업되었다. Spring security에 대한 자세한 내용은 부록을 참고한다.

사용법

depencency

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-core</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
```

```
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
  </dependency>
```

설정(web.xml)

```
<!-- spring security -->
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>
org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

컨텍스트에서 filter bean을 찾을 때 springSecurityFilterChain으로 찾기 때문에 해당이름으로 등록한다.

위 설정의 filter bean은 HttpSecurityBeanDefinitionParser에 의해 자동으로 등록된다.

authentication

개요

- 인증은 사용자의 신원을 확인하는 과정을 의미한다.
- Chamomile에서의 인증은 Spring Security에서 제공하는 사용자 인증 정보를 RDB로 관리하기 위해서 JdbcDaoImpl(UserDetailsService의 구현체)를 확장하여 제공한다.
- 사용자의 정보, 사용자가 가진 권한, 사용자가 속한 그룹에 대한 권한 정보를 바탕으로 인증이 이루어진다.
- 기본적인 인증 매커니즘은 Spring Security의 인증 매커니즘을 그대로 사용하고 있으며, 아래와 같은 흐름으로 이루어진다.

가) 인증 요청 인 경우 UsernamePasswordAuthenticationFilter에서 사용자가 입력한 아이디와 비밀번호를 토대로 UsernamePasswordAuthenticationToken을 생성한다.

나) AuthenticationManager에 AuthenticationToken을 전달하여 인증을 위임한다.

다) AuthenticationManager에 등록 된 AuthenticationProvider를 통해 Authentication 정보를 획득하게 된다. (기본적으로 DaoAuthenticationProvider가 등록된다.)

라) RDB를 통해 사용자 아이디를 통한 Authentication 정보를 획득하고 사용자가 가진 권한과, 사용자가 속한 그룹의 권한 정보를 획득한다.

마) 사용자의 기본적인 상태(계정이 잠겨있는지, 계정이 비활성화 되어있는지, 계정이 만료되었는지)를 확인하고, 정상적인 경우 패스워드를 검사하여 인증을 수행한다. 그 후 비밀번호의 상태(비밀번호가 만료되었는지)를 확인한다.

바) 성공 되는 경우 인증 매커니즘에 의해 원래 요청 한 페이지로 redirect 된다.

사용법

http설정

http 설정은 기본적인 인증 매커니즘을 적용하고, URL을 보호하며, 로그인 및 오류에 대한 처리 적용되는 Filter들을 설정할 수 있다.

```
<http>
    <intercept-url pattern="/**" access="hasRole('USER')" />
    <form-login />
    <logout />
</http>
```

항목에 대한 설명은 아래와 같다.

가) intercept-url: 애플리케이션의 모든 요청은 인증을 요구하도록 하며, ROLE_USER 권한이 존재하는 경우에만 접근할 수 있다. (권한이 존재하지 않는 경우는 Access Denied 오류가 발생한다.) – chamomile에서는 RDB를 통한 관리

나) pattern : 보호하고자 하는 URL 패턴을 등록한다. Ant 형식의 패턴 으로 등록가능하다.

다) access : 등록 된 URL 패턴에 대해 접근을 제어하기 위한 정보를 설정한다. 권한 혹은 표현식을 통한 제어가 가능하다.

기타 항목은 로그인 항목에서 자세하게 확인이 가능하다.

authentication 설정

인증을 위한 설정은 authentication-manager를 이용하여 설정하며, 아래는 RDB를 이용하는 설정의 예이다.

```
<!-- authentication -->
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service data-source-ref="dataSourceFactoryBean"
                       id="jdbcUserDetailsService"/>
  </authentication-provider>
</authentication-manager>


<beans:bean id="jdbcUserDetailsService"

class="net.lotte.chamomile.security.userdetails.JdbcUserDetailsService" >
    <beans:property name="dataSource" ref="dataSourceFactoryBean" />
</beans:bean>
```

항목에 대한 설명은 아래와 같다.

가) jdbc-user-service : 사용자 정보를 RDB를 이용하여 처리 한다.

나) jdbcUserDetailsService : chamomile에서 제공하는 테이블 형태의 구조로 사용자 정보를 제공한다.

- dataSource : RDB에 연결하기 위한 데이터소스를 설정한다.
- usersByUsernameQuery (optional) : 입력받은 사용자 아이디로 사용자를 조회하는 쿼리를 설정한다.
    - 쿼리의 결과는 기본적으로 6개의 항목을 조회하고 있다.
    - userId(아이디), userPassword(비밀번호), enabled(활성화여부), accountNonExpired(계정이만료되지 않았는지), accountNonLocked(계정이 잠겨있지 않았는지), credentialsNonExpired(비밀번호가 만료되지 않았는지): 순서는 변경되어서는 안된다.

    - 예) select userId, userPassword, enabled, accountNonExpired, accountNonLocked, credentialsNonExpired from {your table} where {start_date} < ? and {end_date} > ? and userId = ?
- authoritiesByUsernameQuery (optional) : 입력받은 사용자 아이디에 부여 된 권한 목록을 조회하는 쿼리를 설정한다.
    - 쿼리의 결과는 기본적으로 2개의 항목을 조회하고 있다.
    - userId(아이디), roleId(권한): 순서가 변경되어서는 안된다.
- groupAuthoritiesByUsernameQuery (optional) : 사용자가 속한 그룹의 권한 목록을 조회하는 쿼리를 설정한다.
    - 쿼리의 결과는 기본적으로 3개의 항목을 조회하고 있다.
    - userId(아이디), groupId(그룹), roleId(권한): 순서가 변경되어서는 안된다.

기본 설정 (authentication)

사용자 정보를 직접 정의한다면 아래와 같이 정의 할 수 있다.

```
 <authentication-manager>
     <authentication-provider>
         <user-service>
           <user name="jimi" password="jimispassword" authorities="ROLE_USER,
ROLE_ADMIN" />
           <user name="bob" password="bobspassword" authorities="ROLE_USER" />
         </user-service>
     </authentication-provider>
 </authentication-manager>
```

위와 같이 정의 한다면 InMemoryUserDetailsManager가 설정 되어 처리 된다.

그 외 추가적으로 Spring에서는 아래와 같은 인증 형식을 제공한다.

- Basic Authentication
- Digest Authentication
- X.509 Authentication
- JAASLDAP
- ...

상세한 내용은 아래에서 확인할 수 있다.

https://docs.spring.io/spring-security/site/docs/4.2.7.RELEASE/reference/htmlsingle/#what-is-acegi-security

Authorization

개요

- authorization은 누군가에게 무엇을 할 수 있게 하거나, 원하는 정보를 얻을 수 있도록 허용하는 과정이다. (즉, 웹 사이트 개발 시 리소스에 접근하기 위한 권한을 부여/허용하는 과정을 말한다.)
- Chamomile에서의 authorization은 권한에 대한 정보를 RDB로 관리 할 수 있도록 기능을 확장하였다.

가) 보호되는 자원(리소스, 버튼)에 대한 DB 권한 처리

나) 권한 계층 정보의 DB 화

- 기본적인 권한 부여 과정은 아래와 같다.

가) 보호되는 자원에 접근하게 되는 경우 FilterSecurityInterceptor에서 요청을 받아 처리하게 된다.

나) 실제 대상 보호되는 자원에 접근하기 전 beforeInvocation()을 수행하고 권한이 존재하는지 확인하게 된다.

- 먼저 현재 요청에 대한 권한 정보를 획득한다. (리소스에 부여된 권한 정보 – RDB로 관리된다.)
- 인증이 필요한 경우 인증을 수행한다.
- AccessDecisionManager.decide()를 통해 리소스에 대한 접근의 가부를 결정한다. (Spring Security에서는 투표방식으로 접근 여부를 결정한다.)
    - 총 3개의 AccessDecisionManager가 제공된다.
    - AffirmativeBased(하나라도 접근허용 한다면) - 기본설정, ConsensusBased(허용하는것이 접근거부보다 많다면, UnanimousBased(만장일치)
    - 등록 된 AccessDecisionVoter의 투표에 의한 결정.

다) 접근이 허용되는 경우 요청을 전달하게 되고, AfterInvocationManager를 통한 반환되는 결과에 대한 후처리 작업을 진행하게 된다.

설정

- Authorization은 기본적으로 Filter 기반으로 동작하게 된다.
- authorization을 수행하는 Filter는 FilterSecurityInterceptor이다.
- Spring Security에서 제공되는 Filter는 기본적으로 XML 기반으로 처리되게 되어 있으며, chamomile에서는 RDB 기반으로 처리되게 된다.
- chamomile에서 제공되는 filter를 설정하기 위해 아래와 같이 설정한다.

```
 <http access-decision-manager-ref="accessDecisionManager">
   <expression-handler ref="webSecurityExpressionHandler"/>

       <!-- for authorization -->
     <custom-filter before="FILTER_SECURITY_INTERCEPTOR"
                    ref="filterSecurityInterceptor"/>
     <custom-filter after="FILTER_SECURITY_INTERCEPTOR"
                    ref="internalResourceUrlFilter"/>
     </http>
   ...
   <beans:bean id="accessDecisionManager"
               class="org.springframework.security.access.vote.AffirmativeBased"
>
   <beans:constructor-arg>
     <beans:list>
       <beans:bean
class="org.springframework.security.web.access.expression.WebExpressionVoter">
         <beans:property name="expressionHandler"
 ref="webSecurityExpressionHandler"/>
       </beans:bean>
     </beans:list>
   </beans:constructor-arg>
 </beans:bean>
```

가) accessDecisionManager

- 리소스에 대한 접근 여부를 결정하기 위한 판단주체이다.
- 설정 된 AccessDecisionManager는 등록 된 voter 중 하나라도 리소스에 대한 접근을 허용하게 되는 경우 접근하도록 결정한다.

- 2개의 AccessDecisionVoter가 설정되어 있으며, expression 기반의 처리와 권한계층을 포함하는 Role Voter가 설정되어져 있다.

위 항목에 설정 된 Filter는 아래와 같다.

```
 <beans:bean id="filterSecurityInterceptor"
class="org.springframework.security.web.access.intercept.FilterSecurityIntercept
or">
     <beans:property name="authenticationManager"
ref="org.springframework.security.authenticationManager" />
     <beans:property name="accessDecisionManager" ref="accessDecisionManager" />
     <beans:property name="securityMetadataSource"
ref="reloadableFilterInvocationSecurityMetadataSource" />
     <beans:property name="rejectPublicInvocations" value="true" />
 </beans:bean>
 ...
 <beans:bean id="internalResourceUrlFilter"
class="net.lotte.chamomile.security.filter.InternalResourceUrlFilter" />
```

가) filterSecurityInterceptor

- 보호되는 자원을 RDB로 처리하기 위한 FilterSecurityInterceptor이다.
    - 모든 요청에 대해 권한이 적절한지 검사를 수행한다.
- authenticationManager : 추가인증이 필요한경우 사용하게 될 인증 처리 주체
- accessDecisionManager : 리소스에 대한 접근 여부를 판단하게 될 AccessDecisionManager를 설정한다.
- securityMetadataSource : RDB로 관리되는 보호되는 자원(리소스)를 제공하는 메타데이터소스
- rejectPublicInvocations : 권한을 부여하지 않은 리소스에 대한 접근 시 오류 발생 여부 설정

나) internalResourceUrlFilter

- Spring에서 개발 시 기본적으로 jsp를 WEB-INF 폴더 하단에 위치시키게 된다.
- JSP에 대한 리소스 권한 부여시 WEB-INF 폴더를 명시적으로 지정하지 않아도 처리 될 수 있도록 하는 Filter이다.
- JSP 화면에서 security 관련 태그 라이브러리를 손쉽게 사용할 수 있다.

RDB로 관리되는 보호되는 자원(리소스)의 권한 정보

```
 <beans:bean id="reloadableFilterInvocationSecurityMetadataSource"
class="net.lotte.chamomile.security.access.intercept.ReloadableFilterInvocationS
ecurityMetadataSource" >
   <beans:property name="securedUrlResourceService"
 ref="securedUrlResourceService"/>
 </beans:bean>

 <beans:bean id="securedUrlResourceService"
class="net.lotte.chamomile.security.access.secured.url.SecuredUrlResourceService
">
   <beans:property name="dataSource" ref="dataSourceFactoryBean" />
 </beans:bean>
```

가) reloadableFilterInvocationSecurityMetadataSource

- FilterSecurityInterceptor에서 리소스의 권한 정보를 제공하는 메타데이터소스
- 초기 로딩 시 리소스에 대한 권한정보를 캐시하고 있으며, 권한 정보가 변경되는 경우 갱신을 수행한다.
    - 단, 자동적으로 갱신을 수행하지는 않으며, API를 호출해야 한다. 그렇지만 admin 시스템을 사용하는 경우 자동으로 갱신된다.

나) securedUrlResourceService

- URL 리소스에 대한 권한정보를 제공하는 서비스
- dataSource : RDB의 데이터소스를 설정한다.
- rolesQuery : URL 리소스에 대한 권한 정보를 조회하는 쿼리를 설정한다.
    - 기본적으로 url(리소스 url 정보), authority(권한)을 조회하게 되어 있다.
    - 순서와 명칭이 동일해야 한다.

RDB로 관리되는 보호되는 자원(버튼)의 권한 정보

```
 <!-- secured button resources -->
 <beans:bean id="reloadableUrlButtonSecurityMetadataSource"
class="net.lotte.chamomile.security.access.secured.button.ReloadableUrlButtonSec
urityMetadataSource">
   <beans:property name="securedUrlButtonResourceService"
ref="securedUrlButtonResourceService" />
 </beans:bean>

 <beans:bean id="securedUrlButtonResourceService"
class="net.lotte.chamomile.security.access.secured.button.SecuredUrlButtonResour
ceService">
   <beans:property name="dataSource" ref="dataSourceFactoryBean" />
 </beans:bean>
```

가) reloadableUrlButtonSecurityMetadataSource

- 버튼 태그라이브러리에서 사용되며 리소스에 부여된 버튼의 권한 정보를 제공하는 메타데이터소스
- 초기 로딩 시 리소스에 부여된 권한정보를 캐시하고 있으며, 권한 정보가 변경되는 경우 갱신을 수행한다.
    - 단, 자동적으로 갱신을 수행하지는 않으며, API를 호출해야 한다. 그렇지만 admin 시스템을 사용하는 경우 자동으로 갱신된다.

나) securedUrlButtonResourceService

- 리소스에 부여된 버튼의 권한정보를 제공하는 서비스
- dataSource : RDB의 데이터소스를 설정한다.
- rolesQuery : 리소스에 부여된 버튼의 권한 정보를 조회하는 쿼리를 설정한다.
    - 기본적으로 url(리소스 url 정보), buttonCode(버튼코드), buttonName(버튼이름), authority(권한)을 조회하게 되어 있다.
    - 순서와 명칭이 동일해야 한다.

권한 계층 정보 (RDB로 관리)

```
 <beans:bean id="reloadableRoleHierarchy"
class="net.lotte.chamomile.security.access.hierarchicalroles.ReloadableRoleHiera
rchy">
   <beans:property name="roleHierarchyService" ref="jdbcRoleHierarchyService"/>
 </beans:bean>
 <beans:bean id="jdbcRoleHierarchyService"
class="net.lotte.chamomile.security.access.hierarchicalroles.JdbcRoleHierarchySe
rvice">
   <beans:property name="dataSource" ref="dataSourceFactoryBean" />
 </beans:bean>
```

가) reloadableRoleHierarchy

- 권한의 계층정보를 제공한다.
- 초기 로딩 시 리소스에 부여된 권한정보를 캐시하고 있으며, 권한 정보가 변경되는 경우 갱신을 수행한다.
    - 단, 자동적으로 갱신을 수행하지는 않으며, API를 호출해야 한다. 그렇지만 admin 시스템을 사용하는 경우 자동으로 갱신된다.

나) jdbcRoleHierarchyService

- 권한의 계층 정보를 제공하는 서비스
- dataSource : RDB의 데이터소스를 설정한다.
- hierarchicalRolesQuery : 권한의 계층 정보를 조회하는 쿼리를 설정한다.
    - 기본적으로 parent(상위 권한), child(하위 권한)을 조회하게 되어 있다.
    - 순서와 명칭이 동일해야 한다.
    - 기타 설정

```
<!-- dataSource FactorBeans -->
<beans:bean id="dataSourceFactoryBean"

 class="net.lotte.chamomile.security.config.support.DataSourceFactoryBean" >
  <beans:property name="dataSource" ref="dataSource" />
</beans:bean>
<!-- expresssion handler -->
<beans:bean id="webSecurityExpressionHandler"
class="org.springframework.security.web.access.expression.DefaultWebSecurityExpr
essionHandler" >
  <beans:property name="roleHierarchy" ref="reloadableRoleHierarchy"/>
</beans:bean>
<!-- api Service -->
<beans:bean id="securityService"
class="net.lotte.chamomile.security.SecurityService">
  <beans:property name="reloadableFilterInvocationSecurityMetadataSource"

                  ref="reloadableFilterInvocationSecurityMetadataSource" />
  <beans:property name="reloadableUrlButtonSecurityMetadataSource"

                  ref="reloadableUrlButtonSecurityMetadataSource" />
  <beans:property name="reloadableRoleHierarchy"
                  ref="reloadableRoleHierarchy" />
</beans:bean>
```

가) dataSourceFactoryBean

- 데이터소스를 생성하기 위한 DataSource Factory Bean
- 데이터소스를 설정하는 property에서 dataSource Bean을 설정한다.

나) webSecurityExpressionHandler

- 요청에 대한 expression 검사를 수행하기 위한 핸들러

다) securityService

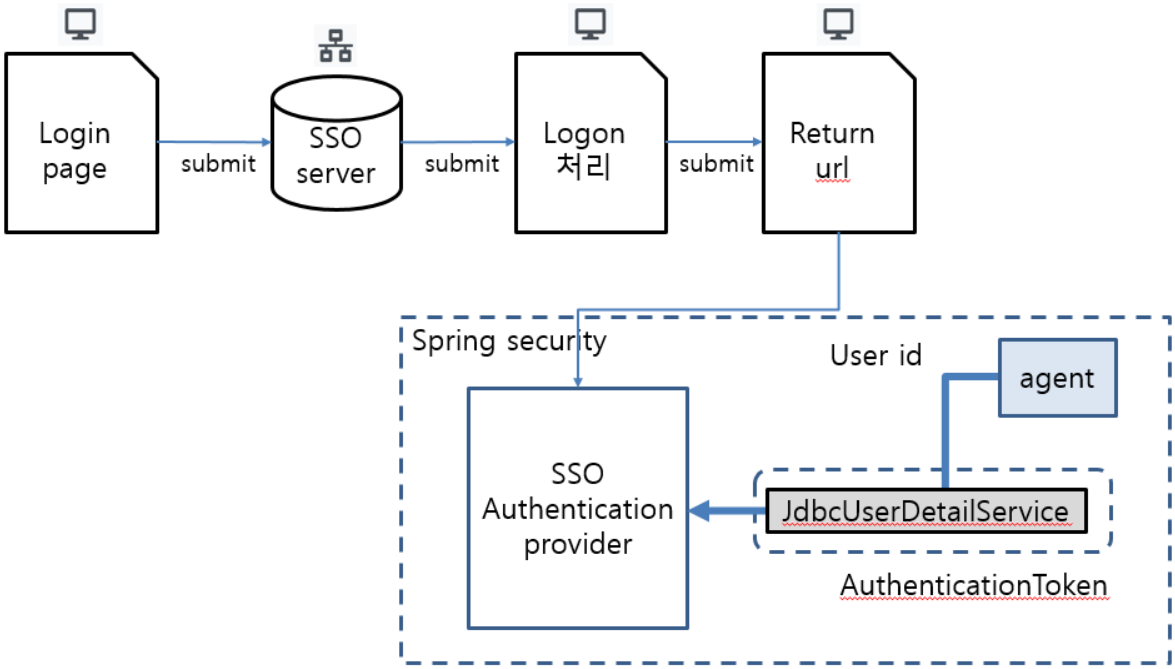
- Security 관련 된 API를 제공하는 서비스
- Java doc을 참고한다.

SSO적용하기(롯데 DWP SSO)

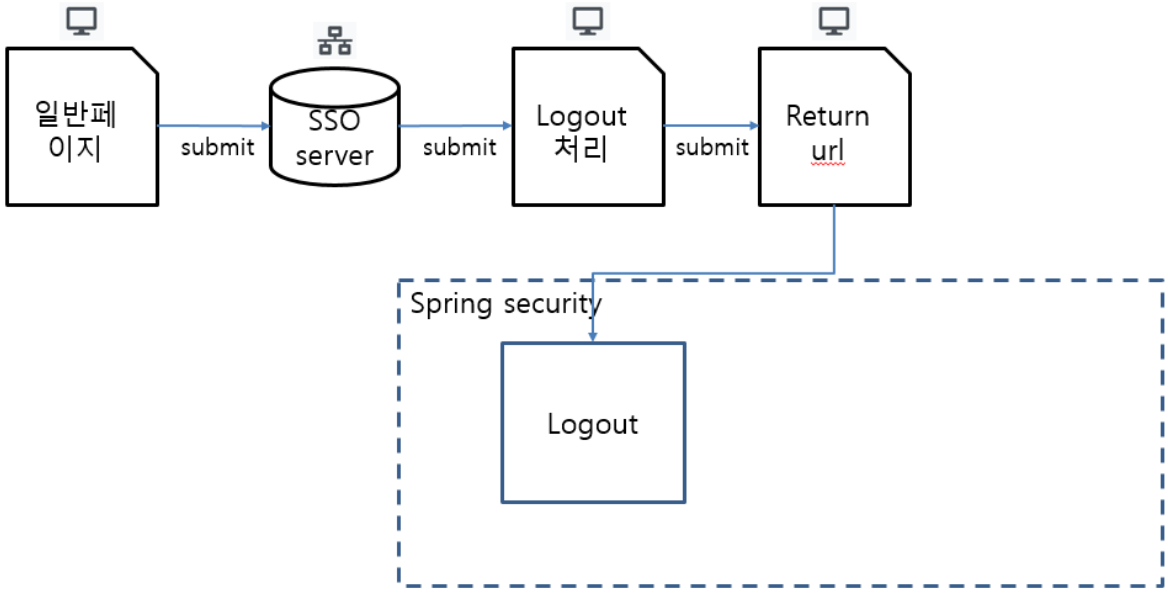
개요

- ssosite에 묶여 있는 사이트들을 대상으로 한곳에서 로그인을 하면 관련된 다른사이트들도 동시에 다른곳도 로그인되어 서비스를 이용할 수 있게 해준다.
- SSO적용 가이드는 아래 주소를 참고하여 적용하도록 한다. https://dwp-sso.gitbook.io/developers/
- 이 가이드 문서에서는 SSO를 적용하여 spring security 와 연계하는 방법을 설명한다.
- SSO인증을 모두 완료하고 나면 구현하고 있는 application 의 주소를 호출하게되는데 해당 시점부터 spring security의 인증을 수행한다.
- SSO를 적용한 로그인과 로그아웃의 흐름도는 아래와 같다.

<로그인처리>



<로그아웃처리>



설정

각 프로퍼티 설명 중 required의 기준은 소스에서는 강제되지 않지만 해당 기능 사용시 프레임워크에서 반드시 지정해서 사용해야 하는 경우를 의미한다.

가) security 예외 설정

- SSO연동 가이드에서는 인증에 필요한 몇가지 페이지 들이 있다. 해당 페이지 들은 spring security의 영향을 받지 않도록 제외시켜준다.

```
<!-- sso인증용 페이지, 아래 페이지들은 security영역으로 관리하지 않는다. -->
<http pattern="/loginPage" security="none"/>
<http pattern="/logonService" security="none"/>
<http pattern="/logoffService" security="none"/>
<http pattern="/resources/**" security="none"/>
```

나) 로그인 처리 설정

- SSO인증 완료 후, 혹은 다른 사이트에서 인증을 완료한 후 현재 사이트로 바로 접근했을 경우 application의 인증을 수행해야 한다.
- 해당 인증을 바로 수행할 수 있도록 구성요소 들을 제공한다.
- 이 설정은 authentication filter - Authentication Manager - Authentication Provider 흐름으로 진행되며 설정 방법은 아래와 같다.

authentication filter 설정

```
<http …>
…
<custom-filter before="FORM_LOGIN_FILTER" ref="ssoAuthenticationFilter"/>
…
</http>
…
<beans:bean id="ssoAuthenticationFilter"
class="net.lotte.chamomile.security.sso.oneid.SSOAuthenticationFilter">
<beans:property name="filterProcessesUrl"
value="${chmm.security.loginProcessingUrl}" />
```

```
<beans:property name="authenticationManager"
ref="org.springframework.security.authenticationManager" />
<beans:property name="usernameParameter"
value="${chmm.security.usernameParameter}" />
<beans:property name="passwordParameter"
value="${chmm.security.passwordParameter}" />
<beans:property name="authenticationSuccessHandler"
ref="sampleLoginSuccessHandler" />
<beans:property name="authenticationFailureHandler"
ref="sampleLoginFailureHandler" />
<beans:property name="postOnly" value="false"/>
</beans:bean>


<beans:bean id="sampleLoginSuccessHandler"
class="com.ldcc.login.handler.SampleLoginSuccessHandler" />
<beans:bean id="sampleLoginFailureHandler"
class="com.ldcc.login.handler.SampleLoginFailureHandler" />
```

- FORM_LOGIN_FILTER이전에 로그인 역할을 수행하게 되므로 <form-login> 태그에 작성하는 내용들을 bean에 직접 작성해야 한다.
- filterProcessesUrl(required) : 로그인을 처리할 URL지정 pattern으로 지정할 수 있다.
    - authenticationManager(required) : 로그인을 처리해줄 authentication-provider 를 관리하는 매니저를 지정한다.
    - authenticationSuccessHandler(required) : 인증이 성공했을 경우 최종적으로 수행해야 하는 작업이나 client에게 response해줄 내용을 정의하는 handler이다.
- authenticationFailureHandler(required) : 인증이 실패 했을 경우 client에게 인증 실패 처리를 수행해 줄 handler이다.
    - postOnly(option) : SSO인증이 완료되고 나서 호출 되는 페이지에서 로그인이 수행되거나, application 에 인증이 수행되기 전에 페이지에 접근했을경우에도 바로 인증을 수행해야 하기때문에 로그인 요청이 POST가 아닐경우도 처리될 수 있도록 한다.

Authentication Provider 설정

```
<!-- authentication -->
<authentication-manager>
    <authentication-provider ref="ssoAuthenticationProvider"/>
</authentication-manager>

<beans:bean id="ssoAuthenticationProvider"
class="net.lotte.chamomile.security.sso.oneid.SSOAuthenticationProvider" >
    <beans:property name="jdbcUserDetailsService" ref="jdbcUserDetailsService"/>
</beans:bean>
```

- SSO서버에서 사용자 ID를 받아와 처리하기 때문에 provider를 별도로 구성한다.
- jdbcUserDetailsService(required) : 사용자 정보를 얻어오는 클래스를 지정한다.

다) 로그아웃 처리 설정

- SSO는 로그아웃에 대해 global logout과 local logout을 사용할 수 있다. global logout은 SSO서비스 자체에서 로그아웃을 하여 연관되어있는 모든 사이트에서 로그아웃 처리가 되는것을 의미하며 local logout은 자신이 이용하고 있는 사이트에서만 logout을 수행하는 것을 말한다.
- DWP에서는 타 사이트 영향도를 고려하여 local logout을 하도록 권장하고 있다. local logout은 기존에 가이드 되어있는 방법으로 로그아웃 처리를 진행한다.
- global logout처리
    - 위 흐름도에 나와있는 것처럼 SSO서버에서 로그아웃을 처리한 후 현재 사이트에 정의되어있는 logout주소를 매핑시켜주도록 한다.
    - 이 경우 서버로 요청이 발생할 때 마다 sso서버에 로그인 상태를 체크하는 기능이 구현되어야 한다. 자세한 내용은 SSO개발가이드를 참고하도록 한다.
- local logout처리
    - 기존에 가이드된 로그아웃 처리를 사용한다.

Multi factor authentication(다중 요소 인증)

개요

기존의 ID/Password기반의 로그인인증 방식에서 보안성을 강화하고자 다요소 인증 방식이 많이 사용되고 있다. 많은 곳에서 2 factor 이상의 인증을 구현하여 ID/Password외에 OTP나 생체 인증등을 사용하고 있으며, 이에 따라 프레임워크에서도 다양한 방식의 인증을 처리할 수 있도록 지원한다.

설정

각 프로퍼티 설명 중 required의 기준은 소스에서는 강제되지 않지만 해당 기능 사용시 프레임워크에서 반드시 지정해서 사용해야 하는 경우를 의미한다.

가) 로그인 처리 설정

- 로그인을 수행 할 때 프레임워크에서 제공되는 구성 요소는 Authentication filter(확장), Authentication provider(확장), mfaAuthenticationTokenManager(신규), loginSuccessHandler(확장), loginFailureHandler(확장) 가 필요하다. 이 중 mfaAuthenticationTokenManager의 경우 로그인을 수행할 때 필요한 토큰들을 관리하는 클래스로 다양한 인증요소를 관리 하기 위해 사용된다.

Authentication filter 설정

```
 <http …>
 …
 <custom-filter before="FORM_LOGIN_FILTER" ref="mfaAuthenticationFilter"/>
 …
 </http>
 …
 <beans:bean id="mfaAuthenticationFilter"
class="net.lotte.chamomile.security.mfa.filter.MfaAuthenticationFilter">
 <beans:property name="filterProcessesUrl" value="/loginProcess-*" />
 <beans:property name="authenticationManager"
ref="org.springframework.security.authenticationManager" />
 <beans:property name="usernameParameter"
value="${chmm.security.usernameParameter}" />
 <beans:property name="passwordParameter"
value="${chmm.security.passwordParameter}" />
 <beans:property name="authenticationSuccessHandler"
ref="sampleLoginSuccessHandler" />
 <beans:property name="authenticationFailureHandler"
ref="sampleLoginFailureHandler" />
```

```
 <beans:property name="mfaAuthenticationTokenManager"
ref="mfaAuthenticationTokenManager"/>
 <beans:property name="postOnly" value="false"/>
 </beans:bean>
```

- FORM_LOGIN_FILTER이전에 로그인 역할을 수행하게 되므로 태그에 작성하는 내용들을 bean에 직접 작성해야 한다.
- filterProcessesUrl(required) : 로그인을 처리할 URL지정 pattern으로 지정할 수 있다.
- authenticationManager(required) : 로그인을 처리해줄 authentication-provider 를 관리하는 매니저를 지정한다.
- usernameParameter(required) : 프레임워크 에서 제공하는 필터가 UsernamePasswordAuthenticationFilter를 상속받아 구현되어있다. 사용자 정보에서 권한등을 얻어오기 위해서는 최초에는 ID/PW기반의 인증을 수행해야 하기 때문인데, 이 때 사용되는 property중 하나로 ID를 넘겨받을 parameter명을 정의한다.
- passwordParameter(required) : usernameParameter 와 마찬가지의 이유로 사용되며 password를 넘겨받을 parameter명을 정의한다.
- authenticationSuccessHandler(required) : 각 단계마다 인증이 성공했을 경우 최종적으로 수행해야 하는 작업이나 client에게 response해줄 내용을 정의하는 handler이다.
- authenticationFailureHandler(required) : 인증이 실패 했을 경우 client에게 인증 실패 처리를 수행해 줄 handler이다.
- mfaAuthenticationTokenManager(required) : 인증에 필요한 토큰들을 관리해주는 매니저를 지정한다.
- postOnly(option) : 로그인 요청이 POST가 아닐경우도 처리될 수 있도록 한다.(SSO인증시에만 사용된다.)
- sso사용의 경우 MfaWithSsoAuthenticationFilter 를 사용한다.

loginSuccessHandler 설정

```
 <beans:bean id="sampleLoginSuccessHandler"
class="net.lotte.chamomile.security.mfa.handler.MfaLoginSuccessHandler"  >
   <beans:property name="mfaAuthenticationTokenManager"
ref="mfaAuthenticationTokenManager"/>
 </beans:bean>
```

- mfaAuthenticationTokenManager(required) : 각 로그인 단계를 참조할때 사용된다. 최종적으로 로그인을 모두 수행 했는지 여부, 다음 인증으로 이동할 URL정보등을 얻을 때 사용된다.

loginFailureHandler 설정

```
 <beans:bean id="sampleLoginFailureHandler"
class="net.lotte.chamomile.security.mfa.handler.MfaLoginFailureHandler" />
```

Authentication provider 설정

```
 <!-- authentication -->
 <authentication-manager>
     <authentication-provider>
         <password-encoder ref="passwordEncoder"/>
         <jdbc-user-service data-source-ref="dataSourceFactoryBean"
id="jdbcUserDetailsService"/>
     </authentication-provider>
     <authentication-provider ref="otpAuthenticationProvider"/>
 </authentication-manager>
```

- provider는 authentication manager에 복수개를 설정 할 수 있다.
- 인증 절차는 Authentication filter - Authentication manager - Authentication provider 의 순서로 진행되는데, 내부적으로 Authentication Token이 사용된다. Filter에서 생성되는 Token의 종류에 따라 해당 인증을 수행할 Authentication provider가 결정된다. 해당 토큰들은 mfaAuthenticationTokenManager 에서 관리된다.

mfaAuthenticationTokenManager 설정

```
 <beans:bean id="mfaAuthenticationTokenManager"
class="net.lotte.chamomile.security.mfa.MfaAuthenticationTokenManager">
 <beans:property name="mfaAuthenticationStepFilterOperator">
 <beans:bean
class="net.lotte.chamomile.security.mfa.token.filter.MfaAuthenticationStepStepDo
mainFilter" />
 </beans:property>
 <beans:property name="authenticationStep">
 <beans:list>
 <beans:bean class="net.lotte.chamomile.security.mfa.AuthenticationTokenInfo"

p:tokenName="org.springframework.security.authentication.UsernamePasswordAuthent
icationToken"
       p:redirectNextAuthenticationUrl="/otp"
       p:processUrl="${chmm.security.loginProcessingUrl}"/>
      <beans:bean
class="net.lotte.chamomile.security.mfa.AuthenticationTokenInfo"

p:tokenName="net.lotte.chamomile.security.mfa.token.OtpAuthenticationToken"
       p:availableRoles="ROLE_ADMIN_ID"
       p:availableCustomValue="localhost"
 p:redirectNextAuthenticationUrl=""
       p:processUrl="/loginProcess-otp"/>
 </beans:list>
 </beans:property>
 </beans:bean>
```

- authenticationStep : authentication provider에서 처리할 토큰목록을 정의한다.
  - AuthenticationTokenInfo
    - tokenName : 토큰 클래스명(전체 패키지명)
    - availableCustomValue : 인증 대상을 다르게 설정하기 위해 Custom정보를 설정한다. 인증 요청시 Custom에 지정된 값에 따라 인증 토큰이 filtering된다. 필터링 기능은 기본적으로 header와 domain기반의 filter를 제공한다.
    - availableRoles : 인증 대상을 다르게 설정하기 위해 role정보를 설정한다. 첫번째 인증으로 수행하게 되는 id/pw기반의 인증을 수행하고나서 얻어지는 role기반으로 filtering된다. comma를 구분자로 여러개를 지정할 수 있다.
    - redirectNextAuthenticationUrl : 인증 완료 후 이동할 페이지 경로를 설정한다.
- mfaAuthenticationStepFilterOperator(option) : 인증토큰은 2단계로 필터링이 된다. 헤더나 도메인등 서비스 운영 방식에 따라 인증방법을 다르게 설정하고자 하는 경우에 대한 필터링과 사용자 권한에 따른 인증방법을 다르게 설정 하는 방법을 설정 할 수 있다. 두 단계에 걸친 인증 방법 필터링중 첫번재 단계에서는 다양한 환경에서의 인증 방식을 결정하는데 사용자가 설정하기 용이하도록 interface를 제공한다.
  net.lotte.chamomile.security.mfa.token.filter.MfaAuthenticationStepFilterOperator 인터페이

스를 구현하여 설정할 수 있으며 프레임워크에서는 기본적으로 domain과 header를 이용한 인증 방식 filter를 제공한다.

- header를 이용한 인증방식 filter설정
  - 헤더값은 기본적으로 mfaHeader라는 값으로 설정되어있으며 변경 하고자 할 경우 headerName property로 변경한다.

```
 <beans:bean id="mfaAuthenticationTokenManager" …>
   <beans:property name="mfaAuthenticationStepFilterOperator">
     <beans:bean
class="net.lotte.chamomile.security.mfa.token.filter.MfaAuthenticationStepStepHe
aderFilter" />
   </beans:property>
 …
```

- domain을 이용한 인증 방식 filter설정

```
 <beans:bean id="mfaAuthenticationTokenManager" …>
   <beans:property name="mfaAuthenticationStepFilterOperator">
     <beans:bean
class="net.lotte.chamomile.security.mfa.token.filter.MfaAuthenticationStepStepDo
mainFilter" />
   </beans:property>
```

- 두번째 인증단계 필터링인 role기반은 인증은 availableRoles에 값을 설정하여 자동으로 처리된다.

나) 로그인 성공/실패 처리 설정

- 로그인 성공/실패 처리는 loginSuccessHandler/loginFailureHandler에서 담당한다.
- 로그인이 성공한 경우 Authentication filter에 설정된 loginSuccessHandler에서 후 처리를 담당하도록 한다.
- 기본적으로 MfaLoginSuccessHandler는 로그인이 성공한 경우 아래와 같은 JSON데이터가 response된다.

```
 {"result":"success"}
```

MfaLoginSuccessHandler 설정

```
 <beans:bean id="sampleLoginSuccessHandler"
class="net.lotte.chamomile.security.mfa.handler.MfaLoginSuccessHandler"  >
   <beans:property name="mfaAuthenticationTokenManager"
ref="mfaAuthenticationTokenManager"/>
 </beans:bean>
```

- 업무시스템에 맞게 구현할 경우 net.lotte.chamomile.security.mfa.handler.AbstractMfaLoginSuccessHandler 를 상속받아 재구현한다.
  - AbstractMfaLoginFailureHandler는 내부적으로 AuthenticationSuccessHandler 인터페이스의 구현체로 동작하며 onAuthenticationSuccess 를 재정의 하여 로그인 성공에 대한 처리를 수행한다.
  - 메서드
    - isFinalToken

        - return : boolean
        - parameter : HttpServletRequest, Authentication
        - 설명 : 현재 request의 내용을 기준으로 필터링된 인증 토큰 목록이 마지막 토큰인지 판단한다. 즉, 모든인증을 완료 했는지 판단한다.
- 로그인이 실패했을 경우 Authentication filter에 설정된 loginFailureHandler에서 후 처리를 담당하도록 한다.
- 기본적으로 MfaLoginFailureHandler 로그인이 실패한 경우 아래와 같은 JSON데이터가 response 된다.

```
{"result" : "fail", "message" : "에러메시지"}
```

- MfaLoginFailureHandler 설정

```
 <beans:bean id="sampleLoginFailureHandler"
class="net.lotte.chamomile.security.mfa.handler.MfaLoginFailureHandler" />
```

- 업무시스템에 맞게 구현할 경우 net.lotte.chamomile.security.mfa.handler.AbstractMfaLoginFailureHandler 를 상속받아 재구현한다. AbstractMfaLoginFailureHandler는 AuthenticationFailureHandler인터페이스의 구현체로 동작하며 onAuthenticationFailure 메서드를 재정의 하여 로그인 실패에 대한 처리를 수행한다.

다) 미인증 상태에 대한 처리 설정 (voter-entry point)

- 정의 된 인증단계를 모두 거치지 않고 업무 페이지로 접근하고자 하는 경우에 인증을 모두 수행 할 수 있도록 적절한 조치를 취해야 한다.
- 프레임워크에서는 인가(authority) 검증과 비인증 예외 발생 후 처리 클래스를 제공한다.
- 인증 단계를 모두 거치지 않은 사용자가 업무 페이지로 접근하는 경우 access decision manager에 적용된 Voter를 통해 인증여부를 판단하게 된다.

net.lotte.chamomile.security.mfa.MfaVoter 설정

```
 <!-- access secured url -->
 <beans:bean id="accessDecisionManager"
class="org.springframework.security.access.vote.AffirmativeBased" >
     <beans:constructor-arg>
         <beans:list>
             <beans:bean class="net.lotte.chamomile.security.mfa.MfaVoter"/>
             <beans:bean
class="org.springframework.security.web.access.expression.WebExpressionVoter">
                 <beans:property name="expressionHandler"
ref="webSecurityExpressionHandler"/>
             </beans:bean>
             <beans:bean
class="org.springframework.security.access.vote.RoleHierarchyVoter">
                 <beans:constructor-arg  ref="reloadableRoleHierarchy" />
                 <beans:property name="rolePrefix" value=""/>
             </beans:bean>
         </beans:list>
     </beans:constructor-arg>
 </beans:bean>
```

- 현재 인증단계를 판단하기위해 인증단계를 확인할 수 있는 MfaAuthenticationTokenManager를 사용 하게 되는데 이는 자동으로 Autowired에 의해 자동으로 주입된다.

- 잘못된 접근일때 MfaVoter에서는 AuthenticationException이 발생하게 되며 이는 entry-point에서 최종적으로 처리를 담당하게된다.
- 프레임워크에서는 net.lotte.chamomile.security.mfa.entrypoint.MfaAuthenticationEntryPoint를 제공하며 설정 방법은 아래와 같다.

{설정}

```
 <http pattern="/**" auto-config="true" entry-point-ref="mfaEntryPoint" disable-
url-rewriting="true" use-expressions="true" access-decision-manager-
ref="accessDecisionManager">
 …
 </http>
 …
 <beans:bean id="mfaEntryPoint"
class="net.lotte.chamomile.security.mfa.entrypoint.MfaAuthenticationEntryPoint">
</beans:bean>
```

- customResponse : useCustomResponse 옵션이 true일경우 클라이언트에게 response할 문자열을 지정한다. json, xml등을 설정 할 수 있다. 기본값은 {"error" : "authentication exception"}
    - useCustomResponse : customResponse를 클라이언트에게 response할 경우 true로 설정한다. 기본값은 false
    - redirectPage : 인증이 실패했을 경우 redirect시킬 페이지를 지정한다. 기본값은 "/loginPage"
    - request의 contentType이 "text/html"이거나 "application/x-www-form-urlencoded"으로 설정되어있을 경우 기본적으로 401 http status가 response된다.
    - 클라이언트에게 response되는 조건들의 우선순위는 customResponse -> 401 http status -> redirectPage이다.
- 업무시스템에 맞게 구현할 경우 net.lotte.chamomile.security.mfa.entrypoint.AbstractMfaAuthenticationEntryPoint을 상속받아 재 구현 한다. AbstractMfaAuthenticationEntryPoint 는 AuthenticationEntryPoint 인터페이스의 구현체로 commence메서드를 재정의 하여 인증되지 않은 사용자의 URL접근에 대한 처리를 수행한다.

라) 로그아웃 설정

- 기존에 security상에서 logout하는 항목을 참조하여 구현한다.

한번의 요청으로 모든 인증요소 처리

개요

단일 페이지에서 multi factor authentication을 사용하는 경우 ID와 password외에 추가 정보를 받아들여 인증을 수행해야 한다.

MfaAuthenticationTokenManager에는 인증 단계를 설정하지 않고 필터링 대상 별 한개씩 만 설정 한다. 이 경우 authentication manager에 적용된 authentication provider들을 재정의 해야 한다.

설정

프레임워크에서 제공되는 net.lotte.chamomile.security.mfa.provider.MfaAuthenticationProvider를 상속 할 경우 추가 인증정보를 받아 처리할 수 있도록 메서드 및 property가 제공된다.

- authenticate
    - return : Authentication
    - paramter : Authentication

    - 설명 : 인증을 수행하고 Authentication객체를 생성하여 반환한다.
- supports
    - return : boolean
    - 설명 : provider가 다루게 되는 Authentication 객체인지 판단하는 로직을 추가한다.
- getRequest
    - return : HttpServletRequest
    - 설명 : parameter나 header를 참조하기 위해 request객체를 얻어온다.
- getResponse
    - return : HttpServletResponse
    - 설명 : 사용자에게 response 할 내용들을 정의 하기위해 사용한다.
- getParameter
    - return : String
    - parameter : String parameterName
    - 설명 : 추가 parameter를 얻어오기위해 사용된다.
- getParameter
    - return : String
    - parameter : String parameterName, String defaultValue
    - 설명 : 추가 parameter를 얻어오기위해 사용된다. 값이 null일경우 defaultValue로 대체된다.
- createUsernamePasswordAuthenticationToken
    - return : UsernamePasswordAuthenticationToken
    - parameter : String userId, String password
    - 설명 : 인증 완료 후 Authentication을 반환 할 UsernamePasswordAuthenticationToken 을 생성하기위한 메서드. 해당 메서드를 구현하기 위해서는 jdbcUserDetailsService 가 정의되어있어야 한다.

> jdbcUserDetailsService : 사용자정보(UserDetails)를 얻어오기 위한 컴포넌트

업무시스템에 적용

동시세션 제어

```
<http pattern="/resources/**" security="none" />
<http pattern="/js/**" security="none" />
<http auto-config="true">
<form-login login-page="${chmm.security.loginUrl}“
        login-processing-url=“${chmm.security.loginProcessingUrl}"
        username-parameter=“${chmm.security.usernameParameter}"
        password-parameter="${chmm.security.passwordParameter}""
        authentication-success-handler-ref="loginSuccessHandler"
        authentication-failure-handler-ref="loginFailureHandler" />
<logout logout-url="${chmm.security.logoutProcessingUrl" logout-success-
url="${chmm.security.loginUrl}" />
<csrf />
<session-management invalid-session-url=“${chmm.security.loginUrl}">
       <concurrency-control max-sessions="1" error-if-maximum-exceeded="true"
     expired-url="${chmm.security.loginUrl}"/>
</session-management>
</http>
```

- session-management를 이용하여 세션을 제어할 수 있다.

가) concurrency-control

- max-sessions : 동일한 사용자로 생성가능한 세션 수를 설정한다.
- error-if-maximum-exceeded : max-sessions에 도달했을때 에러 처리를 수행할 수 있다.
    - true로 설정 되는 경우 max-sessions에 도달했을때 오류를 발생시킨다. (즉, 나중에 로그인한 계정을 제한한다.)
    - false로 설정 되는 경우 max-sessions에 도달했을 때 오류를 발생시키지 않는다.
      (기존의 세션을 파기한다. 즉, 기존에 로그인 된 계정을 파기한다.)

태그라이브러리

Chamomile에서는 Spring에서 제공하는 태그 라이브러리 외 추가적인 형태의 태그라이브러리를 제공한다.

- 리소스(화면)에 부여된 버튼의 권한 처리
- WEB-INF 내 외부에 존재하는 리소스의 버튼의 권한 처리

선언

```
<%@ taglib prefix="chamomile-sec"
uri="https://www.ldcc.co.kr/chamomile/security/tags" %>
```

사용

```
<chamomile-sec:button code="처리대상 코드명칭">
    "처리대상 코드명칭"에 대한 권한이 존재하는 경우 출력"
</chamomile-sec:button>
```

사용 예

```
<chamomile-sec:button code="buttonSelect">
    <div class="form-group abs-right">
    <button class="btn btn-info btn-sm" type="button" >
    ...
</chamomile-sec:button>
```

태그라이브러리 미사용 시 버튼권한 기능 사용

마크업 작성

```
<!-- 버튼 권한제어 html-->
<button id="buttonDelete"></button>
<button id="buttonInsert"></button>
<button id="buttonModify"></button>
<button id="buttonInsertTest"></button>
<button id="buttonExcel"></button>
```

태그변수 생성

```
//할당된 버튼 권한  buttonInsertTest
var btn1 = new TagDataBuilder()
        .setId("buttonInsertTest")
        .setEventCallback({
            event:"click",
            callback:function(){
                console.log("btn1");
            }
        })
        .build();
var btn2 = new TagDataBuilder()
        .setId("buttonDelete")
        .setEventCallback({
            event:"click",
            callback:function(){
                console.log("btn2");
            }
        })
        .build();
```

요청

```
_cmmCode.initComponentAuth(window.location.pathname,[btn1,btn2]);
```

결과

CSRF

CSRF는 사이트간 요청 위조라고 하며, 웹취약점 공격 중 하나이다.

Spring Security에서는 CSRF를 예방하기 위해서 임의난수(토큰)를 발급하여 해당 토큰이 없는 경우에는 요청을 거부하는 형태로 제공되고 있다.

- 기본설정 (security 설정)

기본적으로 Spring Security의 CSRF는 HTTP 403 응답코드로 전달되게 된다.

CSRF를 활성화 시키는 설정은 아래와 같다. (context-security.xml 파일내)

```
<http>
    <!-- ... -->
    <csrf />
</ http>
```

- 태그 라이브러리 선언

JSP에서 Spring Security에서 제공하는 태그라이브러리를 선언한다.

```
<% @taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>
```

- CSRF 태그 사용 (csrfInput)

가) CSRF가 활성화 된 경우 CSRF 토큰에 대해 서버에서 정의 된 이름과 값을 가진 숨겨진 태그(hidden 태그)를 삽입한다.

나) CSRF가 비활성화 되어있는 경우 아무것도 출력하지 않는다.

다) CSRF 사용시에는 반드시 POST 방식을 사용한다.

라) CSRF 태그는 HTML의 <form> ... </form> 블록 내 위치시켜야 하며, spring의 <form:form> </form:form> 블록 내에는 두지 않는다.

마) 사용법은 아래와 같다.

```
<form method="post" action="/do/something">
    <sec:csrfInput />
    Name:<br />
    <input type="text" name="name" />
    ...
</form>
```

- CSRF 메타태그 (csrfMetaTags)

가) CSRF가 활성화 된 경우 CSRF 토큰에 대해 서버에서 정의 된 이름과 값을 가진 메타 태그를 삽입한다.

나) 비활성화 되어 있는 경우 아무것도 출력하지 않는다.

다) Java Script에서 CSRF를 적용하는데 사용할 수 있다.

라) HTML의 <head> ... </head> 블록 내 csrfMetaTags를 배치한다.

마) 태그를 통해 CSRF 토큰의 헤더 이름, 값에 쉽게 접근할 수 있다.

```
<!DOCTYPE html>
<html>
   <head>
      <title>CSRF Protected JavaScript Page</title>
      <meta name="description" content="This is the description for this page"
/>
      <sec:csrfMetaTags />
      <script type="text/javascript" language="javascript">
         var csrfParameter = $("meta[name='_csrf_parameter']").attr("content");
         var csrfHeader = $("meta[name='_csrf_header']").attr("content");
         var csrfToken = $("meta[name='_csrf']").attr("content");
         // XMLHttpRequest를 직접 사용하여 x-www-form-urlencoded 요청 보내기
         var ajax = new XMLHttpRequest();
         ajax.open("POST", "http://www.example.org/do/something", true);
         ajax.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded data");
         ajax.send(csrfParameter + "=" + csrfToken + "&name=John&...");
         // XMLHttpRequest를 직접 사용하여 non-x-www-form-urlencoded 요청 보내기
         var ajax = new XMLHttpRequest();
         ajax.open("POST", "http://www.example.org/do/something", true);
         ajax.setRequestHeader(csrfHeader, csrfToken);
         ajax.send("...");
<script>
```

```
    </head>
    <body>
        ...
    </body>
</html>
```

jQuery ajax통신시 CSRF사용하기

```
// JQuery를 사용하여 x-www-form-urlencoded 요청 보내기
var data = {};
data[csrfParameter] = csrfToken;
data["name"] = "John";
...
$.ajax({
url: "http://www.example.org/do/something",
type: "POST",
data: data,
...
});
// JQuery를 사용하여 non-x-www-form-urlencoded 요청 보내기
var headers = {};
headers[csrfHeader] = csrfToken;
$.ajax({
url: "http://www.example.org/do/something",
type: "POST",
headers: headers,
...
});
```

HTTP 응답 헤더

Spring Security에서는 기본적으로 HTTP 응답에 아래와 같은 헤더를 추가하여 전송한다.

```
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
```

위 응답에 대한 기본 설정을 변경할 수 있다. 목록은 아래와 같으며, 사이트 환경에 맞게 내용을 적용할 필요가 있다. (context-security.xml내 <http> 태그의 <headers> 태그안에 명시 된다.)

- Cache Control (브라우저 캐시 제어)
- Security-Transport-Security (hsts 설정. 브라우저가 자동으로 https 요청으로 처리)
- X-Frame-Options (ClickJacking 공격 방어. 감춰진 링크를 사용자가 클릭하게 함.)
- X-XSS-Protection (브라우저의 XSS Filter 활성화. 구글에서 사용.)
- X-Content-Type-Options (mime 기반 공격 방지)

- Public-Key-Pinning or Public-Key-Pinning-Report-Only - 중간자공경 방지 (특정 사이트에 대한 인증서를 명시)
- Content-Security-Policy - 브라우저가 신뢰할 수 있는 Content의 출처 정의
- Referrer-Policy

참고 :
https://docs.spring.io/spring-security/site/docs/4.2.7.RELEASE/reference/htmlsingle/#headers

버튼 권한 기능 API

- 예제 및 사용법

```
import java.util.Collection;
import net.lotte.chamomile.security.SecurityService;
import
net.lotte.chamomile.security.access.secured.button.UrlButtonSecurityConfig;

// SecurityService 객체 생성
private SecurityService securityService;

// url 파라미터로 전달하여 api호출받을 객체 선언
Collection<UrlButtonSecurityConfig> list =
securityService.getStringUrlButtonAuthorities("/url");
```

- API

Package

net.lotte.chamomile.security

Class

클래스명	메서드명	파라미터	반환값	설명
SecurityService	getStringUrlButtonAuthorities	String url	Collection<UrlButtonSecurityConfig> urlAuthorites	url에 해당하는 버튼 권한 목록을 반환한다.

url : 입력받은 url 경로를 의미한다.

> urlAuthorites는 접근가능한 버튼 권한 목록을 반환한다.
> UrlButtonSecurityConfig 객체에는 버튼코드, 버튼명, 권한 속성을 갖고 있다.

사용자 계정 비밀번호 다중 인코더

캐모마일 2.0에서는 사용자 계정 비밀번호를 인코딩할 인코더를 여러개 설정할 수 있는 MultiplePasswordEncoder 제공한다. 추가로 캐모마일에서는 스프링에서 제공하는 noop, bcrypt, pbkdf2, scrypt,ldap,sha256,argon2 알고리즘을 제공한다.

Xml 적용 예시

```
<beans:bean id="passwordEncoder"
class="net.lotte.chamomile.security.password.encoder.MultiplePasswordEncoder">
    <beans:constructor-arg name="encoderId"
value="${chmm.security.defaultPasswordEncoder:sha256}"/>
    <beans:constructor-arg name="encoderList"
value="${chmm.security.passwordEncoderList:sha256}"/>
</beans:bean>
```

Java 코드 적용 예시

```
@Bean
@ConditionalOnMissingBean(PasswordEncoder.class)
public PasswordEncoder passwordEncoder() {
    return new MultiplePasswordEncoder(
            securityProperties.getDefaultPasswordEncoder(),
            securityProperties.getPasswordEncoderList());
}
```

매개변수는 다음과 같다.

- defaultPasswordEncoder
  - 기본 패스워드 인코더 알고리즘명 (기본은 sha256)
- passwordEncoderList
  - 사용할 패스워드 인코더 알고리즘 목록 (다음 중 선택 noop, bcrypt, pbkdf2, scrypt,ldap,sha256,argon2)

UI어댑터

개요

타 프레임워크에서 상용 UI 솔루션과 연동을 위한 모듈을 제공하나, 해당 프레임워크에 종속적인 형태로 개발이 된다.

Chamomile에서는 프레임워크에 대한 종속성을 탈피하기 위해 Spring MVC에서 제공하는 기술들을 이용하여 상용 UI 솔루션과 연동할 수 있는 방법을 제시한다.

즉, 다시말해 백엔드에서 개발되어지는 코드 자체가 프레임워크에 대한 종속성, 상용 UI 솔루션에 대한 종속성이 제거된다. 상용 UI 솔루션을 사용하지 않는 경우에도 동일한 코드로 서비스를 제공할 수 있다. (단, 예외적인 경우도 존재한다.)

지원하는 상용 UI 솔루션은 아래와 같다.

1. nexacro (투비소프트)
2. xplatform (투비소프트)
3. websquare (인스웨이브시스템즈)

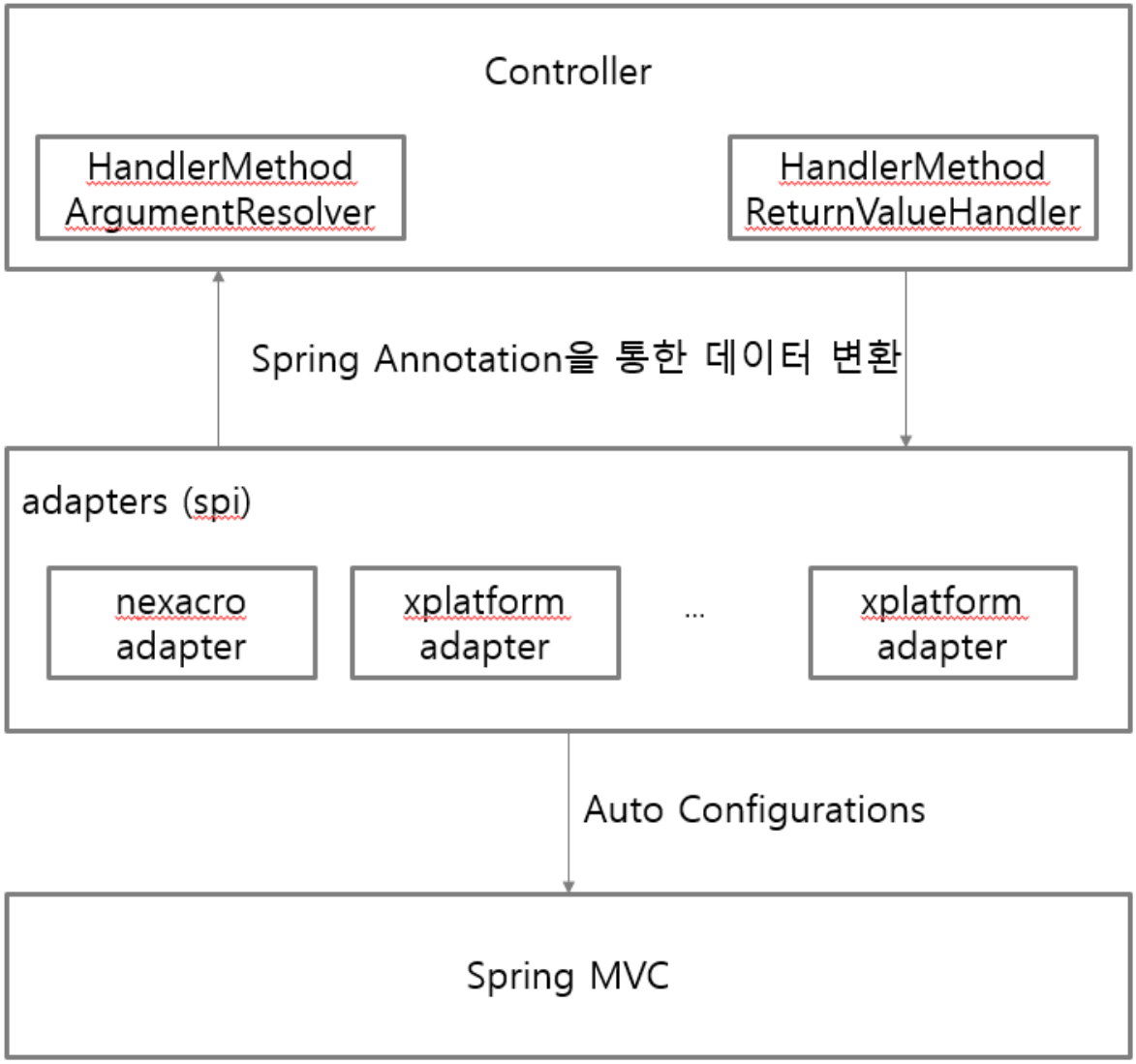
흐름도

- UIAdapter는 상용 UI 솔루션과 연동하기 위한 모듈이다.
- adapter 들은 java의 SPI (Service Provider Interfaces) mechanism을 이용하여 확장 가능하도록 구성되어져 있다.

- UIAdapter의 로드는 Java의 java.util.ServiceLoader에 의해 이루어지고, 해당 jar 파일은 아래와 같은 설정 정보를 포함한다.

1. META-INF/services/net.lotte.chamomile.core.web.adapter.UiAdapter
2. net.lotte.chamomile.core.web.adapter.UiAdapter의 상세한 내용은 java doc을 참고한다.

- UiAdapter의 설치 및 제거는 클래스패스에 존재하는 경우 자동으로 설치되며,
  클래스패스에서 삭제하는 경우 제거 된다.
- Spring MVC에서 제공하는 기술들을 그대로 사용할 수 있도록 구성된다.



[그림] UIAdapter흐름도

기능 Overview

- 다중 UI Adapter를 지원한다.

1. 요청에 따라 동적으로 UI Adapter의 변경을 지원한다.

- 예) 하나의 Controller가 존재할 때 여러 UI 솔루션에서 호출할 수 있다.
- 클래스패스에 존재하는 UIAdapter를 자동으로 로드한다.
- 상용 UI 솔루션이 제공하는 통신타입 처리를 지원한다.
- Spring의 Controller 관련 기술 및 annotation을 지원한다.

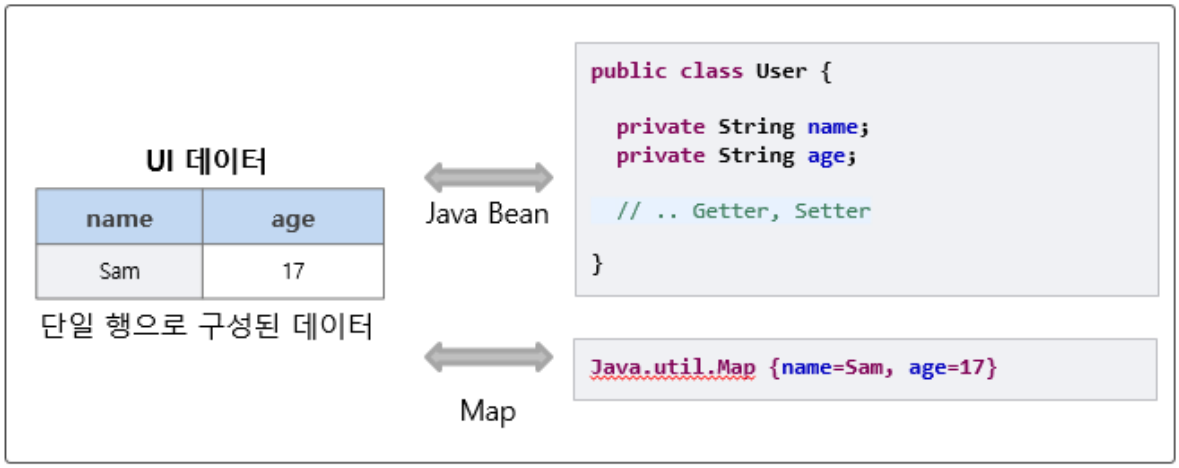
1. @RequestParam, @ModelAttribute, @RequestBody, @ResponseBody, HttpEntity, RequestEntity, ResponseEntity

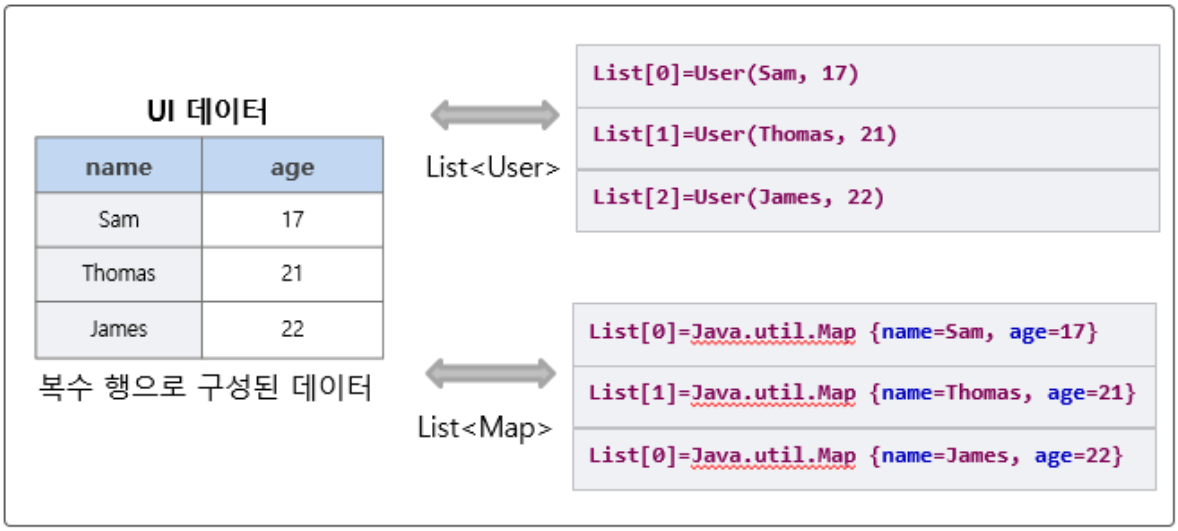
- 입/출력 데이터 변환, 및 에러처리를 지원한다.

기능설명

데이터 변환

- 상용 UI 솔루션에서 제공되는 데이터 형식은 아래와 같다.
  - 단일 데이터 (e.g. Variable)
    - Primitive 형식의 데이터이며, 식별자(name)와 값(value)를 가지고 있다.
  - 2차원 데이터 (e.g. DataSet)
    - DB의 테이블 형식과 유사한 형태이며, 단일 데이터를 복수개를 가지고 있는 데이터이다.
    - 데이터 형식에 따라 복수의 행을 가지고 있을 수 있다.
      (websquare의 경우 map 형식과 list 형식으로 구분할 수 있다.)
- UI 솔루션의 종속성을 탈피하기 위해 Spring의 Controller 레벨에서 데이터 변환을 수행하게 된다.
  (UI 솔루션 ↔ Java Bean)
  - 단일 데이터 변환
    - byte[], int, float, double, Boolean (Support wrapper class)
    - java.lang.String, java.math.BigDecimal, java.util.Date, java.util.Object
    - UI 솔루션에서 제공되는 타입만을 변환한다.
  - 2차원 데이터
    - List<Java Bean | Map>
    - Java Bean (ValueObject) | java.util.Map
    - Java Bean으로 변환하게 되는 경우 멤버 변수의 변환 가능한 타입은 단일 데이터 변환 시 지원하는 타입만을 제공한다. (Bean에 getter/setter가 존재 해야 한다.)
- 2차원 데이터 변환 시 상세한 내용은 아래와 같다.
  - 2차원 데이터의 열의 명칭은 Java Bean의 멤버변수와 대응된다.
    - Map으로 변환하게 되는 경우 식별자로 사용된다.
    - 첫 번째 행만을 대상으로 하며, 나머지 행의 데이터는 무시된다.
  - 복수개의 행으로 구성되는 경우 List 형태로 변환할 수 있다.
    - 모든 행의 데이터가 변환된다.





[그림] 데이터 변환

데이터타입 매핑

- 상용 UI 솔루션과 java 간의 데이터 타입에 대한 매핑 정보는 아래와 같다.
  - nexacro/xplatform의 경우는 지정 된 타입 외에는 데이터 손실의 우려가 있기 때문에 변환하지 않는다.
  - Websquare의 경우 지정 된 타입 외는 String으로 변환을 수행한다.

java types	nexacro	Xplatform	websquare
byte[], java.lang.Byte[]	BLOB	BLOB	
int, java.lang.Integer	INT	INT	int
long, java.lang,Long	LONG	LONG	
float, java.lang.Float	FLOAT	FLOAT	float
double, java.lang.Double	DOUBLE	DOUBLE	double
boolean, java.lang.Boolean	BOOLEAN	BOOLEAN	
java.lang.String	STRING	STRING	String
java.math.BigDecimal	BIG_DECIMAL	BIG_DECIMAL	BigDecimal

java.util.Date	DATE	DATE	Date
java.types	nexacro	xplatform	websquare
	DATE_TIME	DATE_TIME	
	TIME	TIME	
Java.lang.Object	UNDEFINED	UNDEFINED	

데이터 입력

- UI 솔루션에서 요청하는 입력 데이터를 Spring의 Controller 레벨에서 손쉽게 처리 할 수 있는 방법을 제공한다.
    - Spring MVC의 기술을 그대로 이용할 수 있으며, 벤더사에서 제공되는 데이터는 아래와 같은 항목을 통해 변환을 수행한다.

Controller에서 입력 파라매터로 다음과 같은 데이터 형식을 사용할 수 있다.

Argument 타입	설명	옵션	비고
@RequestParam	일반적으로 key/value 형식의 데이터를 획득할 때 사용된다. **(단일 데이터 처리)** 기본적으로는 어노테이션에 명시된 이름을 통해 데이터를 획득하고, 이름이 명시되지 않은 경우 파라매터의 타입을 통해 데이터를 획득한다. 옵션 : required와 defaultValue를 추가적으로 설정할 수 있다.	required defaultValue	어노테이션을 지정하지 않아도 동작한다. 예제 : public void handle(@RequestParam("userId") String userId)
@ModelAttribute	요청 된 파라매터 정보와 선언 된 객체와의 매핑을 수행한다. **(2차원 데이터 처리)** @RequestParam과 동일한 이름 정책을 가진다. binding을 추가적으로 설정할 수 있다. @Valid 어노테이션과 사용 가능하다.	binding	메서드 레벨에 선언되는 경우 선처리 기능으로 동작 가능 어노테이션을 지정하지 않아도 동작한다. 예제 : public void handle(@ModelAttribute("address") Address address)
HttpEntity<T> or RequestEntity<T>	Http 요청의 헤더와 본문으로 구성 된 엔티티이다. **(단일/2차원 데이터가 합쳐진 데이터)** 내부적으로는 HttpMessageConverter를 이용하여 본문를 구성한다.		전체 요청 데이터를 하나의 객체로 변환한다. 예제 : public void handle(HttpEntity<User> httpEntity)
@RequestBody	Http 요청의 본문을 명시 된 파라매터로 구성한다. **(단일/2차원 데이터가 합쳐진 데이터)** 내부적으로 HttpMessageConverter를 이용하여 본문을 구성한다. @Valid 어노테이션과 사용 가능하다.	required	전체 요청 데이터를 하나의 객체로 변환한다. 예제 : public void handle(@RequestBody User user)

Spring에서 제공하는 Controller의 메서드 레벨에서 사용가능한 추가 타입들은 아래에서 확인할 수 있다.

https://docs.spring.io/spring/docs/4.3.18.RELEASE/spring-framework-reference/htmlsingle/#mvc-ann-arguments

데이터 출력

- 처리 된 데이터를 UI 솔루션으로 응답하기 위해선 Model, ModelAndView를 이용할 수 있으며, 데이터 자체를 바로 전달 할 수 있다.
    - ModelAndView를 이용하더라도 View는 요청 된 UI 솔루션을 처리하기 위한 View는 자동으로 지정된다.

Controller에서 출력 파라매터로 다음과 같은 데이터 형식을 사용할 수 있다.

Argument 타입	설명	비고
Model	Model 인터페이스를 구현한 객체 반환을 허용한다. 추가된 데이터는 primitive 타입인 경우 단일데이터로 판단하고, 그 외 데이터는 2차원 데이터로 판단하여 응답한다.	
ModelAndView	Model과 동일한형태이며, UI 솔루션을 사용하게 되는 경우 View는 명시적으로 지정할 필요가 없다. 단, UserController와 같은 UI 솔루션과 일반 JSP에서 동일하게 호출한다면 ModelAndView를 사용하여 view를 지정할 수 있습니다.	
HttpEntity<T> or ResponseEntity<T>	Http 응답의 헤더와 본문으로 구성 된 엔티티이다. View를 처리하지 않으며, Entity(데이터) 자체를 응답으로 구성한다. Bean으로 반환되는 경우 해당 Bean에 단일데이터와 2차원데이터가 합쳐져 있어야 한다.	
@ResponseBody	Http 응답의 본문으로 구성한다. View를 처리하지 않으며, 대상 자체를 응답으로 구성한다. Bean으로 반환되는 경우 해당 Bean에 단일데이터와 2차원데이터가 합쳐져 있어야 한다.	List로 반환되는 경우 2차원 데이터로 인식하여 응답한다.

Spring에서 제공하는 Controller의 메서드 레벨에서 사용가능한 추가 타입들은 아래에서 확인할 수 있다.

https://docs.spring.io/spring/docs/4.3.18.RELEASE/spring-framework-reference/htmlsingle/#mvc-ann-return-types

프로젝트 설정

기본적으로 생성 된 프로젝트에는 UI Adapter의 설정이 포함되어져 있지 않다.

UI Adapter를 사용하기 위해선 아래와 같은 설정을 변경해야 한다.

1. adapter 라이브러리 추가.
    - 배포되는 zip 파일 내 adapter-jars/{벤더사}/ 폴더 내 파일들을 WEB-INF/lib 폴더로 복사한다.
2. XSS 필터 제거 (TOBESOFT 제품군 한정)
    - src/main/webapp/WEB-INF/web.xml 파일 내 XSS filter 제거 (데이터 변환 시 XSS 처리모듈 내장)
3. src/main/resources/spring/context-uiadapter.xml 파일 추가.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- ui adapter -->
    <bean name="uiAdapterSelector"
class="net.lotte.chamomile.core.web.adapter.ParameterUiAdapterSelector">
        <property name="uiAdapterManager" ref="uiAdapterManager" />
    </bean>

    <bean name="uiAdapterManager"
class="net.lotte.chamomile.core.web.adapter.DefaultUiAdapterManager">
        <property name="defaultType" value="nexacro17" />
    </bean>

    <bean name="uiAdapterViewResolver"
class="net.lotte.chamomile.core.web.adapter.UiAdapterViewResolver" p:order="0">
        <property name="uiAdapterManager" ref="uiAdapterManager" />
    </bean>

    <bean id="uiAdapterExceptionResolver"
class="net.lotte.chamomile.core.web.adapter.UiAdapterHandlerExceptionResolver"
p:order="0">
        <property name="uiAdapterManager" ref="uiAdapterManager" />
        <property name="exceptionService" ref="jdbcExceptionService" />
        <property name="exceptionTransfers">
            <list>
                <ref bean="exceptionLogInfo" />
                <ref bean="exceptionInfoSave" />
            </list>
        </property>
    </bean>
    <!-- end ui adapter -->

</beans>
```

4. src/main/resources/spring/context-uiadapter-security.xml 파일 추가.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- uiadapter security -->
    <!-- framework security가 로드 된 후 적재. /admin/** 외의 모든 요청을 제어한다. -
->
```

```
    <http pattern="/**" auto-config="true"  disable-url-rewriting="true" use-
expressions="true"
            entry-point-ref="uiAdapterAuthenticationEntryPoint"
            access-decision-manager-ref="accessDecisionManager">

        <csrf disabled="true" />

        <!-- for authentication -->
        <custom-filter after="SECURITY_CONTEXT_FILTER"
ref="uiAdapterSelectFilter"/>
        <custom-filter before="FORM_LOGIN_FILTER"
ref="uiAdapterAuthenticationFilter"/>

        <access-denied-handler ref="uiAdapterAccessDeniedHandler" />
        <logout logout-url="/logout" success-handler-
ref="uiAdapterLogoutSuccessHandler" />
        <expression-handler ref="webSecurityExpressionHandler"/>

        <!-- for authorization -->
        <custom-filter before="FILTER_SECURITY_INTERCEPTOR"
ref="filterSecurityInterceptor"/>
        <custom-filter after="FILTER_SECURITY_INTERCEPTOR"
ref="internalResourceUrlFilter"/>
    </http>

    <beans:bean id="uiAdapterAuthenticationEntryPoint"
class="net.lotte.chamomile.security.uiadapter.UiAdapterAuthenticationEntryPoint"
/>
    <beans:bean id="uiAdapterLogoutSuccessHandler"
class="net.lotte.chamomile.security.uiadapter.UiAdapterLogoutSuccessHandler" />
    <beans:bean id="uiAdapterAccessDeniedHandler"
class="net.lotte.chamomile.security.uiadapter.UiAdapterAccessDeniedHandler"
depends-on="applicationContextProvider" />

    <!-- ui adapter selector -->
    <beans:bean id="uiAdapterSelectFilter"
class="net.lotte.chamomile.security.uiadapter.UiAdapterSelectFilter" >
        <beans:property name="uiAdapterSelector" ref="uiAdapterSelector" />
    </beans:bean>

    <!-- direct login filter -->
    <beans:bean id="uiAdapterAuthenticationFilter"
class="net.lotte.chamomile.security.uiadapter.UiAdapterAuthenticationFilter" >
        <beans:property name="authenticationManager"
ref="org.springframework.security.authenticationManager" />
        <beans:property name="filterProcessesUrl" value="/login" />
        <beans:property name="usernameParameter" value="username" />
        <beans:property name="passwordParameter" value="password" />
    </beans:bean>

</beans:beans>
```

5. src/main/webapp/WEB-INF/spring/context-servlet.xml 파일 내 import 구문 추가.

- ◦ <beans:import resource="context-servlet-uiadapter.xml"/

6. src/main/webapp/WEB-INF/spring/context-servlet-uiadapter.xml 파일 추가.

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
 http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

    <!-- ui adapter -->
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <mvc:exclude-mapping path="/your application resources/**"/>
            <mvc:exclude-mapping path="/admin/**" />
            <ref bean="uiAdapterSelector" />
        </mvc:interceptor>
    </mvc:interceptors>

    <!-- handle http request -->
    <bean id="requestMapping"
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandl
erMapping" />
    <bean
class="net.lotte.chamomile.core.web.adapter.UiRequestMappingHandlerAdapter"
p:order="1">
        <property name="uiAdapterManager" ref="uiAdapterManager" />
    </bean>

</beans>
```

업무시스템 개발하기

Controller

- UI 솔루션과 연동하기 위해서는 Controller를 다음과 같이 구성하여 사용한다.
- Controller 구성
  - 프로젝트 생성 시 입력한 패키지 하단에 web 패키지를 생성한 후 클래스를 생성한다.
  - E.g. com.company.module.user.web.UserController
  - Controller 클래스 상단에 @Controller 어노테이션을 선언한다.
  - UI 솔루션에서 호출하는 URL은 메서드 상단의 @RequestMapping의 value="/user"이다.
- Controller의 입출력
  - 입력
    - 입력은 @ModelAttribute, @RequestParam, @RequestBody, RequestEntity와 같은 Spring에서 제공하는 기술을 그대로 사용할 수 있다.
  - 출력
    - 반환되는 값은 ModelAndView 혹은 Model, @ResponseBody 등 Spring에서 제공하는 기술을 그대로 사용할 수 있다.
- Controller 예제

```
// 예제 1: Model 관련 데이터 사용
@Controller
public class UserController {
    @RequestMapping(value="/user")
    public ModelAndView user(@ModelAttribute(“dsUser”) User user
            , @RequestParam(“userId”) String userId){
        // some logic…
        List<User> userList= getUserList(userId);
        ModelAndView mv = new ModelAndView();
        mv.addObject(“userList", userList);
        return mv;
    }

// 예제 2: Json 처리방식의 데이터 처리
    @Controller
    public class UserController {
    @RequestMapping(value="/handleRequestBody")
    public @ResponseBody User handleRequestBody(@RequestBody List<User> users) {
        // some logic…
        int count = insertUsers(users);
        List<User> userList= getUserList();
        return userList;
    }
}
```

Service

- Controller에서 호출 되는 Service는 다음과 같이 구성한다.
- Service 구성
  - 프로젝트 생성 시 입력한 패키지 하단에 service 패키지를 생성한 후 interface를 생성한 후 구현한다.
  - e.g. com.company.module.user.service.UserService (interface)
  - e.g. com.company.module.user.service.UserServiceImpl (구현체)
- 전자정부 표준프레임워크를 사용하는 경우 Service의 구현체 클래스는 EgovAbstractServiceImpl 를 상속받는다.

  - Service 인터페이스
    - Service는 하나의 단위 모듈이며, 자바의 기본 개념인 캡슐화, 다형성을 위해 interface 를 생성한다. 소프트웨어 품질의 척도인 loose coupling(느슨한 결합)을 유지 할 수 있다.
    - Service interface는 다음과 같이 생성할 수 있다.

```
public interface UserService {
    User getUser(String userId);
}
```

- Service 구현체
  - Service interface를 구현하여 클래스를 작성한다.

```
public class UserServiceImpl implements UserService {
    ...
}
```

DAO

- DAO(Data Access Object)는 데이터에 접근하기 위한 객체이다. 예를 들어 데이터베이스에 접근해 질의하고 응답을 처리하는 클래스를 말한다.
- 데이터베이스에 접근하게 되는 경우 다양한 형태의 방법을 제공한다.
    - Jdbc, mybatis, JPA 등..
- 프레임워크에서 기본적으로 가이드하는 것을 myBatis를 이용한 연동방법이다.
- myBatis를 이용하여 처리하는 방식은 아래와 같은 3가지 형태로 개발할 수 있다.

    - DAO class 방식
        - SqlSessionDaoSupport를 이용하여 직접 SqlSession을 획득하여 질의하는 방법
    - Interface 사용
        - 인터페이스를 작성하면 myBatis에 의해 런타임 시 Mapper 인스턴스화 되어져 동작한다.
        - 기본적으로 mapper 파일의 namespace에 명시 된 이름의 interface와 매칭된다. (자바의 프록시 기술을 이용한다.)

```
public interface UserMapper { User getUserById(Integer id); }
```

- - Mapper Annotation 방식(raw 방식)
    - 별도의 mapper(xml 파일)을 작성할 필요가 없으며, 메서드 상단에 어노테이션(쿼리)을 통해 실행된다.
        - 단, xml 처리 방식에 비해 제한적 기능을 제공한다. (동적 SQL X)

```
@Select("select * from users where id = #{id}") User getUserById(Integer id);
```

- Interface를 사용하는 것을 권장하며, 기본은 @Repository 어노테이션을 사용하고, 전자정부 표준 프레임워크는 @Mapper를 사용한다.

VO

- VO(Value Object)는 값을 저장하는 객체이며, Entity와 구분되어서 사용되며, 기본적으로 불변객체이다. SI 개발에서는 데이터베이스에서 읽은 데이터를 저장하기 위한 용도로 사용된다.
- VO 클래스 구성
    - VO 클래스는 xxx.vo 패키지 하단에 생성한다.
    - 각 벤더사의 Data Type 매핑 정보를 확인하여 Data 변환이 가능한 멤버 변수에만 데이터가 할당된다.
    - 반드시 해당 멤버변수의 이름으로 해당하는 Getter, Setter가 존재해야 한다.
        - Java의 표준 네이밍 규칙을 이용한다.

```
private String name;
public String getName() {}
public void setName(String name) {}
```

인증 처리 (로그인/로그아웃)

- 로그인을 하기 위해서는 아래와 같이 요청한다.
  - http://host/{context}/login
    - 입력 : username(사용자 아이디), password (비밀번호)
    - 응답 : ok
    - 에러코드 : -401
- 로그아웃을 하기 위해서는 아래와 같이 요청한다.
  - http://host/{context}/logout
    - 입력 : -
    - 응답 : ok
- 서비스 호출 시 권한이 존재하지 않는 경우
  - 에러코드 : -403

호출 정보 및 파라매터를 변경하기 위해서는 src/main/resources/spring/context-uiadapter-security.xml 파일을 변경해야 한다.

인증 후 처리 (성공/실패)

인증 성공/실패 시 추가적인 처리를 수행하기 위해서는 아래와 같은 순으로 개발을 진행한다.

- 성공/실패를 위한 클래스 개발
- context-uiadapter-security.xml 파일에 성공/실패 관련 클래스 적용

UiAdapter에서는 인증에 성공/실패 시 사용자계정 lock count 처리와 같은 추가적인 처리를 수행하지 않는다.

인증에 성공하는 경우 추가 처리 예제 (nexacro)

```
package com.sample;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.AuthenticationSuccessHandler;

import net.lotte.chamomile.commons.login.LoginService;
import net.lotte.chamomile.core.exception.FrameworkException;
import net.lotte.chamomile.core.web.adapter.ConvertException;
import net.lotte.chamomile.core.web.adapter.UiContextHolder;
import net.lotte.chamomile.core.web.adapter.nexacro17.NexacroContext;

public class SampleAuthenticationSuccessHandler implements
AuthenticationSuccessHandler {
```

```
    private static final Logger logger =
LoggerFactory.getLogger(SampleAuthenticationSuccessHandler.class);

    private LoginService loginService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response,
            Authentication authentication) throws IOException, ServletException
{

        String user = authentication.getName();
        logger.info("Account that succeeded authentication : {}", user);

        try {
            //락카운트 초기화
            loginService.resetLockCnt(user);
        } catch (FrameworkException e) {
            logger.error("ResetLockCount Exception  : {} " , e.getMessage());
        }

        NexacroContext context = (NexacroContext)
UiContextHolder.getUiContext();
        Map<String, Object> result = new HashMap<String, Object>();
        result.put("ErrorCode", "0");
        result.put("ErrorMsg", "success");

        try {
            context.sendParameter(result);
        } catch (ConvertException e) {
            throw new ServletException("An error occurred while transferring
platformData.", e);
        }

    }

    public LoginService getLoginService() {
        return loginService;
    }

    public void setLoginService(LoginService loginService) {
        this.loginService = loginService;
    }

}
```

인증에 실패하는 경우 추가 처리 예제 (nexacro)

```
Package com.sample;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import
org.springframework.security.web.authentication.AuthenticationFailureHandler;

import com.nexacro17.xapi.data.Variable;

import net.lotte.chamomile.commons.login.LoginService;
import net.lotte.chamomile.core.exception.FrameworkException;
import net.lotte.chamomile.core.web.adapter.ConvertException;
import net.lotte.chamomile.core.web.adapter.UiContextHolder;
import net.lotte.chamomile.core.web.adapter.nexacro17.NexacroContext;

public class SampleAuthenticationFailureHandler implements
AuthenticationFailureHandler {

    private static final Logger logger =
LoggerFactory.getLogger(SampleAuthenticationFailureHandler.class);

    private LoginService loginService;

    @Override
    public void onAuthenticationFailure(HttpServletRequest request,
HttpServletResponse response,
            AuthenticationException exception) throws IOException,
ServletException {

        NexacroContext context = (NexacroContext)
UiContextHolder.getUiContext();
        Variable usernameVariable = context.getVariable("user");

        String user = usernameVariable.getString();
        logger.debug("authentication failed.", exception);
        logger.info("Account that failed authentication : {}", user);

        try {
            //인증에 실패한 계정의 락카운트를 증가시키며  카운트 임계치를 넘으면 계정을 잠
그는 일괄작업
            loginService.lockProcessAccount(user, exception);
        } catch (FrameworkException e) {
            logger.error("LoginService Exception info : {} " , e.getMessage());
        }

        Map<String, Object> result = new HashMap<String, Object>();
        result.put("ErrorCode", "-401");
        result.put("ErrorMsg", "authentication failed");

        try {
            context.sendParameter(result);
        } catch (ConvertException e) {
            throw new ServletException("An error occurred while transferring
platformData.", e);
        }
    }

    public LoginService getLoginService() {
```

```
        return loginService;
    }

    public void setLoginService(LoginService loginService) {
        this.loginService = loginService;
    }

}
```

Security 설정 파일 수정 (개발한 AuthenticationSuccessHandler와 AuthenticationFailureHandler를 등록한다.)

```
src/main/resources/spring/context-uiadapter-security.xml
(아래 내용 중 jdbcLoginService의 경우 context-common.xml 파일에 설정되어져 있다.)

<beans:bean id="uiAdapterAuthenticationFilter"
class="net.lotte.chamomile.security.uiadapter.UiAdapterAuthenticationFilter" >
    ...

    <beans:property name="authenticationSuccessHandler"
ref="customAuthenticationSuccessHandler" />
    <beans:property name="authenticationFailureHandler"
ref="customAuthenticationFailureHandler" />
</beans:bean>

<beans:bean id="customAuthenticationSuccessHandler"
class="com.sample.SampleAuthenticationSuccessHandler" >
    <beans:property name="loginService" ref="jdbcLoginService" />
</beans:bean>
<beans:bean id="customAuthenticationFailureHandler"
class="com.sample.SampleAuthenticationFailureHandler" >
    <beans:property name="loginService" ref="jdbcLoginService" />
</beans:bean>
```

벤더사별 특이사항

- TOBESOFT (nexacro, xplatform)
  - TOBESOFT 라이브러리 Repository
    - http://cacao.tobesoft.co.kr/nexus/service/local/repo_groups/public
  - 데이터 처리
    - 행의 타입 (RowType)
      - DataSet에 존재하는 행들의 상태를 말하며, 추가, 수정, 삭제 된 상태를 확인할 수 있다.
      - 데이터셋의 RowType은 nexacro platform에서 transaction 시 입력 데이터셋의 전송옵션이 :U 혹은 :A일 때 행의 타입을 확인할 수 있다. :N 옵션인 경우 Normal 상태이다.
      - VO 클래스를 생성할 때 DataSetRowTypeAccessor를 구현해야 행의 타입을 확인할 수 있다.

        - nexacro : com.nexacro.uiadapter17.spring.core.data.DataSetRowTypeAccessor
        - Xplatform : com.tobesoft.xplatform.data.DataSetRowTypeAccessor
    - 행 타입 처리 예제

```
public class UserVO implements DataSetRowTypeAccessor {
    private String name;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    private int rowType;
    @Override
    public int getRowType() { return this.rowType; }
    @Override
    public void setRowType(int rowType) { this.rowType = rowType; }
}
```

- 원본데이터 (savedData)
    - DataSet에 데이터가 변경되기 전의 데이터를 저장하고 있다.
        - 데이터셋의 RowType은 nexacro platform에서 transaction 시 입력 데이터셋의 전송옵션이 :A로 전송한 경우에만 원본데이터를 획득할 수 있다.
        - VO 클래스를 생성할 때 DataSetSavedDataAccessor를 구현해야 원본데이터를 획득할 있다.
            - nexacro : com.nexacro.uiadapter17.spring.core.data.DataSetSavedDataAccessor
            - Xplatform : com.tobesoft.xplatform.data.DataSetSavedDataAccessor
        - 원본데이터 처리 예제

```
public class UserVO implements DataSetSavedDataAccessor<UserVO> {
    private String name;
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    private UserVO savedData;
    @Override
    public UserVO getData() { return this.savedData; }
    @Override
    public void setData(UserVO userVO) { this.savedData = userVO; }
}
```

- 대용량 데이터 분할 전송 (firstrow)
    - 대용량 데이터를 전송하기 위해 데이터를 분할하여 전송하는 방식을 지원한다.
        - 즉, 데이터베이스에서 다량의 데이터 조회 시 Out Of Memory 오류를 예방할 수 있다.
    - Controller에서 입력 파라매터로 데이터 분할 전송을 하기 위한 핸들러를 선언한다.
        - nexacro : com.nexacro.uiadapter17.spring.core.data.NexacroFirstRowHandler
        - xplatform : com.tobesoft.xplatform.data.XplatformFirstRowHandler

    - 데이터 분할전송을 처리하는 핸들러의 주요내용은 다음과 같다.
        - 전송 규칙은 Variable이 전송 되고 난 뒤 DataSet을 전송한다.
            - DataSet이 전송되고 난 후 Variable이 전송 될 경우 예외를 발생시킨다.
        - 대용량 처리시 UI에서 에러처리를 하기 위해선 기존의 ErrorCode, ErrorMsg 외에 추가적으로 데이터셋을 통해 에러를 처리해야 한다.
            - DataSet Schema
            - DataSet 이름 : FirstRowStatus
            - Column : ErrorCode, ErrorMsg
        - 대표적인 메서드는 다음과 같다.
            - void sendPlatformData(PlatformData platformData)
    - void sendDataSet(DataSet ds)
        - void sendVariable(Variable var)
    - void setContentType(String contentType)
        - PlatformType.CONTENT_TYPE_XML
    - PlatformType.CONTENT_TYPE_SSV
        - PlatformType.CONTENT_TYPE_BINARY
- Excetion 처리시 다국어 처리
    - 아래와 같이 정의된 messageSource bean을 할당한다.

```
<bean id="uiAdapterExceptionResolver"
class="net.lotte.chamomile.core.web.adapter.UiAdapterHandlerExceptionResolve
r" p:order="0">
        <property name="uiAdapterManager" ref="uiAdapterManager" />
        <property name="messageSource" ref="messageSource" />
        <property name="exceptionService" ref="jdbcExceptionService" />
        <property name="exceptionTransfers">
            <list>
                <ref bean="exceptionLogInfo" />
                <ref bean="exceptionInfoSave" />
            </list>
        </property>
    </bean>
```

    - messageSource 코드에 할당된 메시지로 throw된다.

```
    @RequestMapping(value="/handleException")
    public @ResponseBody User handleException() throws NexacroException {
        throw new NexacroException("alert.msg.err");
    }
```

Exception

개요

예외 처리란 일반적인 실행의 흐름을 바꾸는 몇 가지 조건을 처리하도록 설계한 프로그래밍 언어의 개념이나 컴퓨터 하드웨어 구조를 말한다.

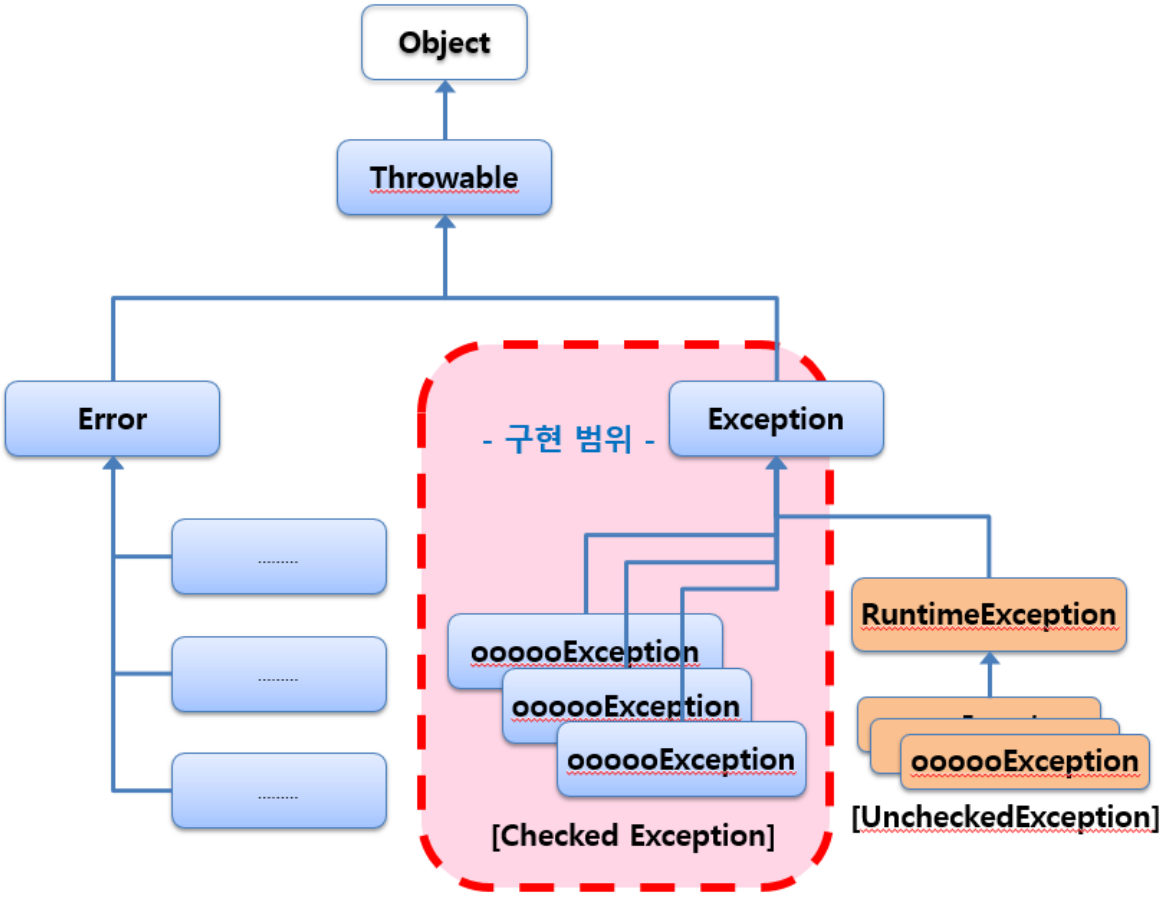
일반적으로 프로그램이 처리되는 동안 특정한 문제가 일어났을 때 처리를 중단하고 다른 처리를 하는 것을 예외 처리라고 한다.

어떤 예외가 발생하더라도 시스템 운영자가 이를 개선시키기 위해 항상 로그를 남겨야 한다.

프로그램이 실행 중 어떤 원인에 의해서 오작동을 하거나 비정상적으로 종료되는 경우 이러한 결과를 초래하는 원인을 에러 또는 오류라고한다

- 에러(Error) & 예외(Exception) : 자바에서는 실행(Runtime) 시 발생할 수 있는 오류를 에러(Error)와 예외(Exception)라고 지칭한다.
    - 에러 (Error) : 프로그램 코드에 의해서 수습될 수 없는 심각한 오류 프로그램이 정상적으로 실행되지 못하는 상황 (H/W)
      <Syntax, 메모리와 관련된>
    - 예외 (Exception) : 프로그램 코드에 의해서 수습될 수 있는 다소 미약한 오류

아키텍처

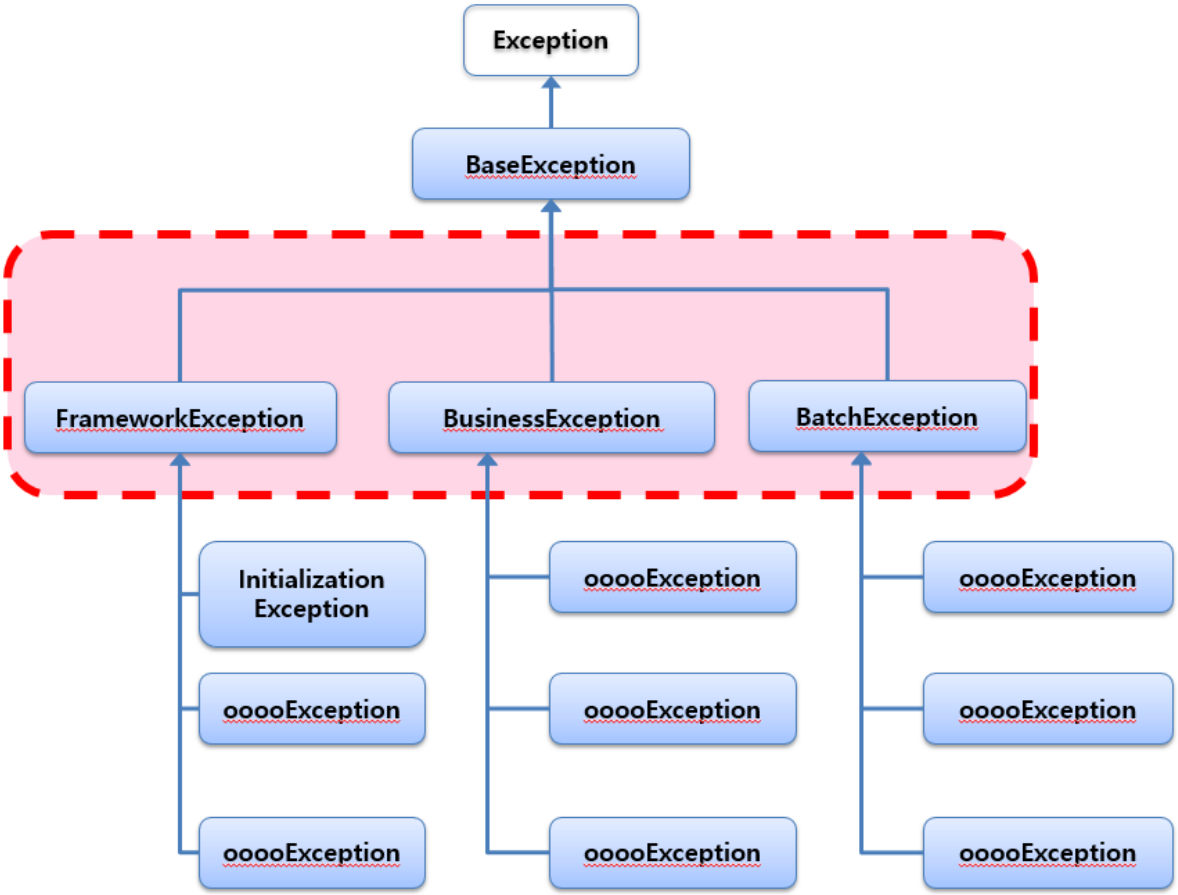


[그림] 예외처리 아키텍처아키텍처

	Checked Exception	Unchecked Exception
처리여부	반드시 예외를 처리해야 함	명시적 처리를 강제하지 않음
확인시점	컴파일 단계	실행 단계
예외발생시 트랜젝션 처리	Roll-back 하지 않음	Roll-back 함
대표예외	Exception의 상속받는 하위 클래스 중 Runtime Exception을 제외한 모든 예외 IOException SQLException	RuntimeException 하위 예외 NullPointException IllegalArgumentException IndexOutOfBoundException

	Checked Exception	Unchecked Exception

Chamomile Exception 제공 범위



[그림] 예외처리 제공범위

	FrameworkException	BusinessException	BatchException
시용 시점	프레임워크 개발 시 사용	업무 로직 개발 시 사용	Batch 업무 개발 시 사용
Package 경로	net.lotte.chamomile.core.exception		

업무시스템에 적용

HandlerExceptionResolver (기본 예외 모듈)

DefaultExceptionResolver는 Controller에서 발생하는 예외를 처리하고 적절한View를 처리 한다. 크게 아래와 같이 2가지 기능을 담당한다.

- 예외 정보 처리 (ExceptionTransfers): 설정 된 exceptionTransfers를 이용하여 예외 정보 처리 (기본 설정 : 로그 파일, DB 예외 정보 적재)
- 예외 발생 시 오류 응답(View) 처리 : 요청 헤더 인 Accept를 기반으로 페이지 요청인 경우 설정 된 오류 페이지로 응답, 그 외 ajax 요청인 경우 설정 된 view를 반환하게 된다.

구성정보는 아래와 같다. (src/main/webapp/WEB-INF/spring/context-servlet.xml에 존재)

```
<!-- exception resolver -->
<beans:bean id="defaultExceptionResolver"
class="net.lotte.chamomile.core.exception.web.DefaultExceptionResolver">
    <beans:property name="messageSource" ref="messageSource" />
    <beans:property name="exceptionService" ref="jdbcExceptionService" />
    <beans:property name="errorPage" value="${chmm.exception.errorPage}" />
    <beans:property name="exceptionTransfers">
        <beans:list>
            <beans:ref bean="exceptionLogInfo" />
            <beans:ref bean="exceptionInfoSave" />
        </beans:list>
    </beans:property>
</beans:bean>
…
```

Controller Based Exception Handling (직접 예외 처리)

@ExceptionHandler 를 특정 Controller에 추가하여 @RequeseMapping에 의해 발생하는 예외처리들을 다룰 수 있다. 즉, Controller내에서 발생하는 예외에 대해서 직접 처리가 가능하다.

- @ResponseStatus 없이 예외처리 (라이브러리나 다른 컴포넌트에 의해 미리 정의된 예외)
- 사용자를 특정 페이지로 redirect
- 커스텀 에러 만들기

```
@Controller
public class ExceptionHandlingController {

    // @RequestHandler methods
     ...

    // Exception handling methods

    // Convert a predefined exception to an HTTP Status code
    @ResponseStatus(value = HttpStatus.CONFLICT, reason = "Data integrity
violation") // 409
    @ExceptionHandler(DataIntegrityViolationException.class)
    public void conflict() {
        // Nothing to do
    }

    // Specify name of a specific view that will be used to display the error:
    @ExceptionHandler({ SQLException.class, DataAccessException.class })
    public String databaseError() {
    // Nothing to do. Returns the logical view name of an error page, passed
    // to the view-resolver(s) in usual way.
    // Note that the exception is NOT available to this view (it is not added
    // to the model) but see "Extending ExceptionHandlerExceptionResolver"
    // below.
        return "databaseError";
    }

    // Total control - setup a model and return the view name yourself. Or
    // consider subclassing ExceptionHandlerExceptionResolver (see below).
    @ExceptionHandler(Exception.class)
```

```
    public ModelAndView handleError(HttpServletRequest req, Exception ex) {
        logger.error("Request: " + req.getRequestURL() + " raised " + ex);

        ModelAndView mav = new ModelAndView();
        mav.addObject("exception", ex);
        mav.addObject("url", req.getRequestURL());
        mav.setViewName("error");
        return mav;
    }
}
```

Global Exception Handling (직접 예외 처리)

@ControllerAdvice 를 이용해 개별 Controller 가 아닌 전체 시스템에 적용할 수 있다.

Annotation Driven Interceptor로 이해하면 된다.

@ControllerAdvice 를 가지는 클래스는 3가지 타입의 Attribute 를 사용할 수 있다.

- @ExceptionHandler 로 처리된 메소드
- @ModelAttribute 로 처리된 메소드
- @InitBinder 로 처리된 메소드

```
@ControllerAdvice
public class SampleCommonExceptionAdvice {

    private static final Logger logger =
LoggerFactory.getLogger(SampleCommonExceptionAdvice.class);

    /* common메소드는 Exception 타입으로 처리하는 모든 예외를 처리하도록 설정 */
    @ExceptionHandler(Exception.class)
    public ModelAndView common(Exception e) {

        logger.info(e.toString());

        ModelAndView mav = new ModelAndView();
        mav.setViewName("/samples/errors/error_common");
        mav.addObject("exception", e); // 예외를 뷰에 전달한다.

        return mav;
    }
}
```

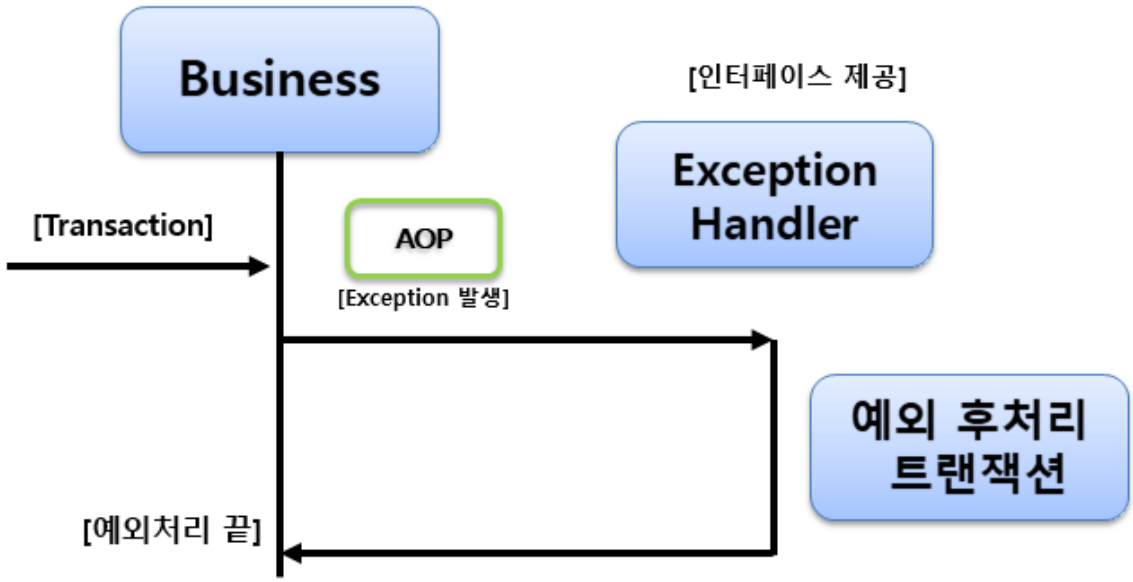
AOP 기반 Exception Handling

시스템 개발 시 Exception 처리, 정확히는 Exception 별 특정 로직(후처리 로직이라고 부르기도 함)을 흐를 수 있도록 하여 Exception 에 따른 적절한 대응이 가능도록 하고자 하는데 목적이 있다.

AOP(After throwing advice)기반으로 예외 후처리 작업을 할 수 있다.

Exception 후처리 방식.

- AOP(pointCut => after-throw) => Exception Transfer.transfer() => ExceptionHandlerService => Handler 순으로 실행된다



[그림] AOP기반 예외 후 처리

Aop Config, ExceptionTransfer 설정 및 설명

Bean 설정

- Exception 후처리와 leaveaTrace 설정을 위해서 샘플에서는context-aspect.xml 파일을 이용한다.
- AOP 사용 설정

```
<aop:config>
        <aop:pointcut id="handlerMethod" expression="execution
                        (* net.lotte.chamomile.core.exception.*(..))" />
        <aop:aspect ref="exceptionTransfer">
                <aop:after-throwing throwing="exception"
               pointcut-ref="handlerMethod" method="occurExceptionHandler" />
        </aop:aspect>
</aop:config>
```

- Java의 Reflection 기능을 사용해서 exceptionTransfer 호출

- ExceptionHandlerAspect 실행되어 exceptionHandleManager 호출

```
<bean id="exceptionTransfer"
        class="net.lotte.chamomile.core.aspect.ExceptionHandlerAspect">
       <property name="exceptionHandlerService">
              <list>
                    <ref bean="exceptionHandleManager" />
              </list>
       </property>
</bean>
```

AOP 실행 시 수행 메소드에 따라 패턴을 비교하여 custom 후처리 핸들러를 호출한다.

```
<bean id="exceptionHandleManager" class="net.lotte.chamomile
           .core.exception.manager.ExceptionHandlerManager">
     <property name="patterns">
          <list>
               <value>**Service.*</value>
          </list>
     </property>
     <property name="handlers">
          <list>
               <ref bean="exceptionLoggingHandler"/>
          </list>
     </property>
</bean>

<bean id="exceptionLoggingHandler"
class="net.lotte.chamomile.core.exception.handler.ExceptionLoggingHandler" />
```

필요한 후처리 기능은 exceptionLoggingHandler와 같이 생성하여 추가

BusinessException사용법

프레임워크상에서 발생하는 Exception은 FrameworkException으로 정의 되어있다.

업무시스템 개발시에는 프레임워크에서 제공하는 BusinessException을 사용한다.

예외처리 적용 형식 예제

접근지정자 반환자료형 메소드이름(매개변수 리스트) throws 예외처리종류 { // 실행문 : Business 로직 내 예외 발생 가능성이 있는 문장 try { 예1) throw new 예외종류(); 예2) throw new 예외종류("메시지[코드]"); 예3) throw new 예외종류("메시지[코드]", 메시지 변수); } catch(예외종류 e) { 메시지 = e.getMeassage(); 예1) 지정된 메시지 반환 예2) 메시지 코드 有 : 해당 메시지, 메시지 코드 無 : 입력된 메시지 예3) 메시지 코드 有 : 해당 메시지, 메시지 코드 無 : 입력된 메시지, 메시지 변수 } }

BusinessException을 예외처리로 사용한 예제

```
접근지정자 반환자료형 메소드이름(매개변수 리스트) throws 예외처리종류 {

    // 실행문 : Business 로직 내 예외 발생 가능성이 있는 문장
    try {
        예1) throw new 예외종류();
        예2) throw new 예외종류("메시지[코드]");
        예3) throw new 예외종류("메시지[코드]", 메시지 변수);
    } catch(예외종류 e) {
        메시지 = e.getMeassage();
        예1) 지정된 메시지 반환
            예2) 메시지 코드 有 : 해당 메시지, 메시지 코드 無 : 입력된 메시지
                예3) 메시지 코드 有 : 해당 메시지, 메시지 코드 無 : 입력된 메시지,
        메시지 변수
    }
}
```

예외처리를 BusinessException 으로 catch하여 BusinessException으로 메시지 코드를 파라미터로 넘긴다.

예외 발생 통계 또는 메일발송, 푸시서비스 등은 운영자 가이드를 참고한다.

공통 기능

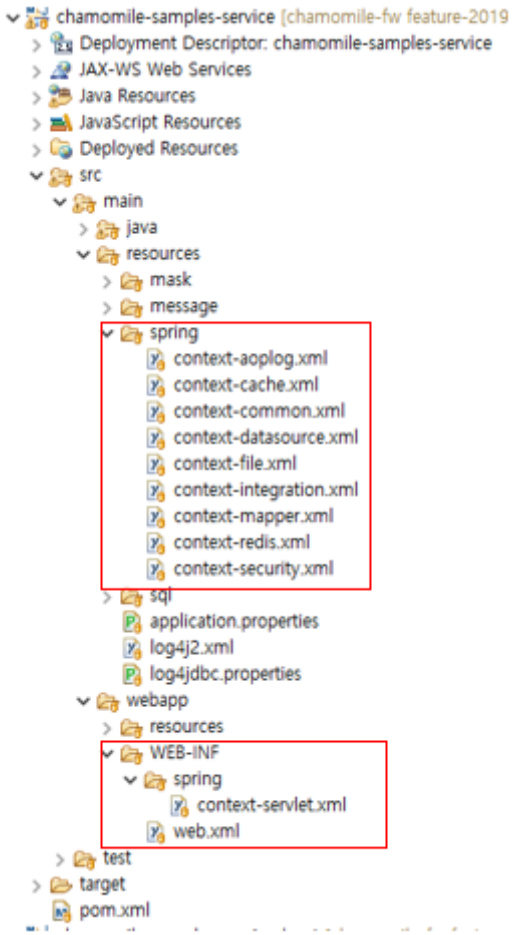
개요

어플리케이션의 공통(Common)기능을 개발하는 가이드이다.

스프링 버전과 스프링부트 버전은 초기 설정만 다소 차이가 있다.

스프링 버전은 통상적인 xml을 통해 어플리케이션 프로젝트에서 직접 설정하며 스프링 부트 버전은 아래와 같은 어노테이션을 통해 자동으로 설정된다.

스프링 버전은 어플리케이션의 다양한 설정들이 프로젝트의 xml파일로 로 직접 설정한다.



스프링부트 버전은 메인클래스에서 @ChamomileBootApplication 어노테이션을 다양한 설정들이 자동으로 주입된다. 이는 pom.xml의 chamomile-core-starter와 chamomile-security-starter에 의해 주입된다.

```
@ChamomileBootApplication
public class DemoApplication {

    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }

}
```

가이드는 **스프링 버전 기준**으로 가이드하며 스프링 부트 또한 각 공통 기능별로 사용방법은 동일하다.

로그인

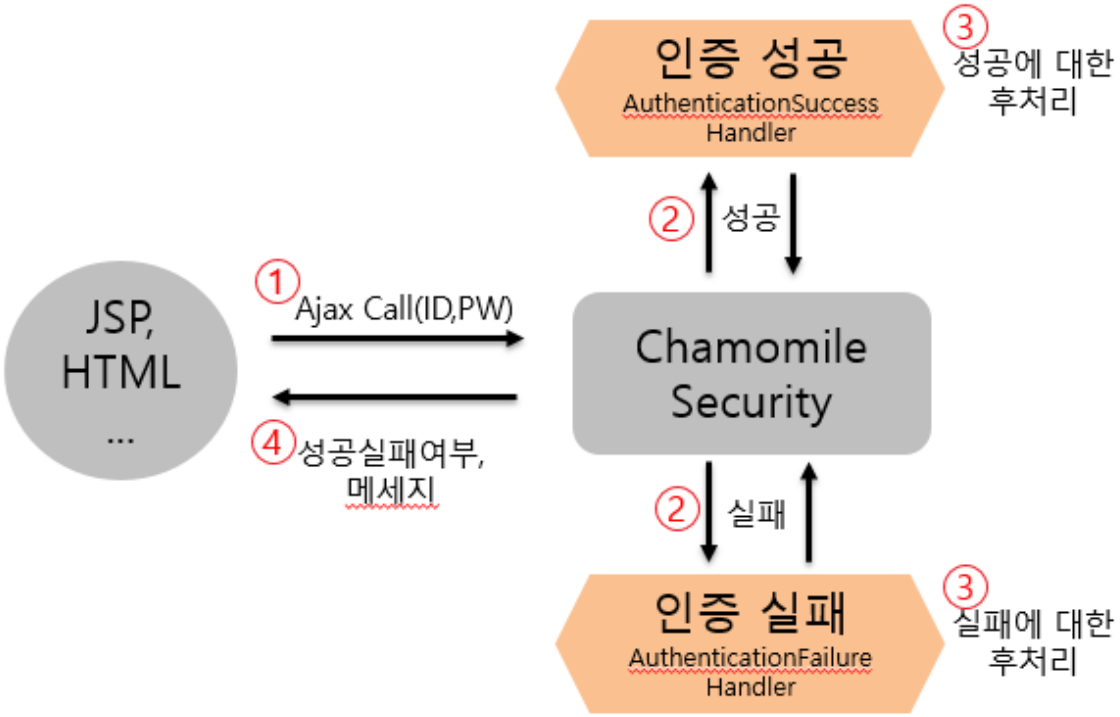
개요

케모마일 로그인 서비스에서는 기본적으로 Spring Security를 통해 인증/인가를 처리한다.

로그인 서비스를 개발하기 이전에 " 시큐리티_개발가이드"를 통해 Spring Security의 인증/인가에 대한 개념을 이해했다는 전제하에 로그인 기능 개발에 대해 중점적으로 설명한다.

케모마일에서 가이드하는 로그인 서비스 개발은 Ajax를 통한 로그인 호출이며 성공/실패에 대한 에러 처리 및 다국어 처리를 위한 다양한 API를 제공하고 있다.

로그인서비스 프로세스



설정 확인

샘플 프로젝트의 루트 컨텍스트에 아래의 빈이 선언되어 있는지 확인한다.
(context-common.xml 확인)

```
<!-- Login API -->
<bean id="jdbcLoginService"
class="net.lotte.chamomile.commons.login.JdbcLoginService">
    <property name="dataSource" ref="dataSource"/>
    <property name="messageSource" ref="messageSource"/>
</bean>
```

- 샘플 프로젝트의 루트 컨텍스트에 시큐리티 설정 파일을 확인한다.
  (context-security.xml 확인)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
```

```
xmlns:aop="http://www.springframework.org/schema/aop"
xsi:schemaLocation="http://www.springframework.org/schema/security
http://www.springfra
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/context
http://www.springframework.org/schema/con
http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spr

<!-- 서비스 http 설정 -->
<http pattern="/**" auto-config="true"  disable-url-rewriting="true" use-
expressions="true" access-decision-manager-ref="accessDecisionManager">
        <access-denied-handler error-
page="${chmm.security.accessDeniedErrorPage}" />
        <csrf disabled="true" />
        <expression-handler ref="webSecurityExpressionHandler"/>
        <custom-filter before="FILTER_SECURITY_INTERCEPTOR"
ref="filterSecurityInterceptor"/>
        <custom-filter after="FILTER_SECURITY_INTERCEPTOR"
ref="internalResourceUrlFilter"/>
    </http>
```

샘플 로그인 서비스 만들기

시큐리티 설정파일 작성

- 앞서 확인해 보았던 context-security.xml에서 `<http>` 태그를 통해 로그인 인증 매커니즘을 적용한다.
- `<http>`태그는 HTTP 보안 구성을위한 컨테이너 요소이며. 가이드에서 언급하지 않은 보다 상세한 항목은 아래에서 확인할 수 있다.
  https://docs.spring.io/spring-security/site/docs/4.2.7.RELEASE/reference/htmlsingle/#nsa-web
- 본 가이드에서는 예제 소스를 통해 Step-by-Step으로 간단한 구성의 로그인 페이지를 만들어본다.
- 우선 `<!-- 서비스 http 설정 -->`주석 아래에 다음과 같이 http 엘리먼트를 작성하고 bean을 등록한다

```
<!-- 서비스 http 설정 -->
<http pattern="${chmm.security.loginUrl}" security="none" />
<http pattern="/sample/**" auto-config="true"
      access-decision-manager-ref="accessDecisionManager">
    <csrf disabled="true" />
    <form-login login-page="${chmm.security.loginUrl}"
                username-parameter="${chmm.security.usernameParameter}"
                password-parameter="${chmm.security.passwordParameter}"
                login-processing-url="${chmm.security.loginProcessingUrl}"
                authentication-success-handler-ref="sampleLoginSuccessHandler"
                authentication-failure-handler-ref="sampleLoginFailureHandler"
/>
    <logout invalidate-session="true" logout-
url="${chmm.security.logoutProcessingUrl}"
            logout-success-url="${chmm.security.loginUrl}" />
    <expression-handler ref="webSecurityExpressionHandler"/>
```

```
    <!-- for authorization -->
    <custom-filter before="FILTER_SECURITY_INTERCEPTOR"
ref="filterSecurityInterceptor"/>
    <custom-filter after="FILTER_SECURITY_INTERCEPTOR"
ref="internalResourceUrlFilter"/>
</http>
<!-- Sample login success handler -->
<beans:bean id="sampleLoginSuccessHandler"
class="com.sample.SampleLoginSuccessHandler" />
<!-- Sample login fail handler -->
-   <beans:bean id="sampleLoginFailureHandler"
class="com.sample.SampleLoginFailureHandler" />
```

- 어플리케이션의 URL 패턴은 “{ContextRoot}/sample/**”로 가정하며 각 설정에 대한 주요 내용은 아래와 같다.

```
<!- 서비스 http 설정 □
<!-- 로그인 페이지 호출 URL은 시큐리티 제외 (컨트롤러) -->
<http pattern="${chmm.security.loginUrl}" security="none" />
<!-- /sample/** URL은 인증 매커니즘 적용 -->
<http pattern="/sample/**" auto-config="true"
access-decision-manager-ref="accessDecisionManager">
<csrf disabled=" true " />
<!-- 로그인 페이지 호출 URL (컨틀롤러) -->
<form-login login-page="${chmm.security.loginUrl}"
<!-- 아이디의 파라미터 명 (로그인 페이지 form에 있는 ID를 저장한 변수 이름) -->
Username-parameter="${chmm.security.usernameParameter}"
<!-- 패스워드의 파라미터 명 (로그인 페이지 form에 있는 패스워드를 저장한 변수 이름) -->
Password-parameter="${chmm.security.passwordParameter}"
<!-- 로그인 페이지 form action에 입력할 주소 -->
Login-processing-url="${chmm.security.loginProcessingUrl}"
<!-- 인증 성공시 호출할 핸들러 Bean-->
Authentication-success-handler-ref="sampleLoginSuccessHandler"
<!-- 인증 실패시 호출할 핸들러 Bean -->
Authentication-failure-handler-ref="sampleLoginFailureHandler" />
<!-- 로그아웃 수행 URL, 로그아웃 성공시 이동할 URL -->
<logout invalidate-session="true" logout-
url="${chmm.security.logoutProcessingUrl}"
logout-success-url="${chmm.security.loginUrl}" />
<expression-handler ref="webSecurityExpressionHandler"/>
<!-- for authorization -->
<custom-filter before="FILTER_SECURITY_INTERCEPTOR"
ref="filterSecurityInterceptor"/>
<custom-filter after="FILTER_SECURITY_INTERCEPTOR"
ref="internalResourceUrlFilter"/>
</http>
<!-- Sample login success handler(인증 성공시 호출할 핸들러 Bean 선언) -->
<beans:bean id="sampleLoginSuccessHandler"
class="com.sample.SampleLoginSuccessHandler" />
<!-- Sample login fail handler(인증 실패시 호출할 핸들러 Bean 선언) -->
-   <beans:bean id="sampleLoginFailureHandler"
class="com.sample.SampleLoginFailureHandler" />
```

로그인페이지개발

사용자에게 브라우저를 통해 노출한 로그인 페이지를 개발한다.

아래의 샘플은 <script src="/resources/js/jquery.form.js"></script>가 필요하다

```
<!-- HTML -->
 <form id="loginForm" action="/login.ajax"" method="post">
   <div>아이디
     <input type=text name='username' id="id">
   </div>
   <div>패스워드
     <input type="password" name='password' id="password">
   </div>
   <button type="submit" onclick="javascript:loginSubmit();">로그인</button>
 </form>

<script>
function loginSubmit() {
    $('#loginForm').ajaxForm({
        dataType : 'json',
        success : function(json, statusText, xhr, $form) {
          //인증 성공시 화면 페이지 이동
          if (json.success == true) {
          location.href = '/sample/main';
          //인증 실패시 alert
          } else {
          alert(json.message);
          }
        },
        error : function(xhr) {
          alert("Ajax error" + xhr.statusText);
      }
  });
}
</script>
```

AuthenticationSuccessHandler 구현

앞서 등록했던 sampleLoginSuccessHandler Bean을 만든다.

AuthenticationSuccessHandler 상속받아 구현한다.

이 핸들러는 인증 성공시 onAuthenticatioinSuccess 메서드를 호출한다.

인증에 성공시에 필요한 후처리 로직을 작성한다.

Ajax Call에 의한 json 리턴 메세지를 스트림으로 작성한다.

```
@Override
public void onAuthenticationSuccess(
        HttpServletRequest request, HttpServletResponse response,
        Authentication authentication) throws IOException, ServletException {
    //인증 성공 후처리 로직 개발
```

```
    //1. 세션에 사용자 정보 저장
    //2. 로그인에 성공하였으므로 락카운트 초기화
    //3. 기타 등등...
    //인증 결과를 성공으로 json return
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("success", true);
    ObjectMapper om = new ObjectMapper();
    String jsonString = om.writeValueAsString(map);
    OutputStream out = response.getOutputStream();
    out.write(jsonString.getBytes());
}
```

AuthenticationFailureHandler 구현

앞서 등록했던 sampleLoginFailureHandler Bean을 만든다.

AuthenticationFailureHandler 상속받아 구현한다.

이 핸들러는 **인증 실패시 onAuthenticatioinFailure 메서드를 호출**한다.

인증에 실패시에 필요한 후처리 로직을 작성한다.

Ajax Call에 의한 json 리턴 메세지를 스트림으로 작성한다.

```
@Override
public void onAuthenticationFailure(
        HttpServletRequest request, HttpServletResponse response,
        AuthenticationException exception) throws IOException, ServletException
{
    //인증 실패 후처리 로직 개발
    //1. 인증 실패 사용자에 대한 로깅
    //2. 로그인 실패하였으므로 락카운트 증가
    //3. 기타 등등...
    //인증 결과를 실패로 json return
    //사용자 브라우저에게 보여줄 실패 메세지 json return
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("success", false);
    map.put("message", "로그인에 실패하였습니다.");
    ObjectMapper om = new ObjectMapper();
    String jsonString = om.writeValueAsString(map);
    OutputStream out = response.getOutputStream();
    out.write(jsonString.getBytes());
}
```

로그인 서비스 API

로그인 서비스에서는 인증 성공과 실패에 따른 후처리에 필요한 다양한 기능을 API로 제공한다.

로그인 서비스 API를 사용하기 위해서는 LoginService 빈이 등록되어 있어야 하며 DI를 통한 객체 주입이 선행 되어야 한다.

로그인 서비스 API의 명세는 아래와 같다.

```
// AuthenticationSuccessHandler에서 인증에 성공한 유저 아이디 조회
String userId = authentication.getName();
// AuthenticationFailureHandler에서 인증에 실패한 유저 아이디 조회
(아규먼트는 context-security.xml에 설정한 username-parameter)
String userId = request.getParameter("serviceid");
```

```
// 시스템에 설정된 락카운트 임계치 조회
int lockCountLimit = loginService.getLockCountLimit();
// 유저의 현재 락카운트 조회
int userlockCnt = loginService.checkLockCnt(userId);
// 유저 락카운트 증가
loginService.increaseLockCnt(userId);
// 유저 락카운트 초기화
loginService.resetLockCnt(userId);
// 유저 계정 잠금
loginService.lockAccount(userId);
// 유저 계정 잠금 해제
loginService.unLockAccount(userId);
// 계정의 유무 확인 (계정이 있으면 '1' return, 없으면 '0' return)
int count = loginService.checkUser(userId);
// 유저의 락카운트를 증가시키며 락카운트 임계치가 넘으면 계정을 잠금는 일괄 작업 (인증 실패
시 AuthenticationFailureHandler에서 사용)
loginService.lockProcessAccount(userId, exception);
// 인증 실패 정보에 따른 메세지 (다국어) 조회 (인증 실패시
AuthenticationFailureHandler에서 사용)
String returnMsg = loginService.getExceptionInfoMsg(exception);
/*
Exception 정보로 판단한 인증 실패의 종류에 따른
리턴되는 메세지(한국어)는 아래와 같다.
- 계정명(ID) 또는 비밀번호를 잘못 입력하셨습니다. (BadCredentialsException)
- 계정이 잠겼습니다. (LockedException)
- 계정이 비활성화 되었습니다. (DisabledException)
- 계정이 만료되었습니다.  (AccountExpiredException)
- 계정의 패스워드 기간이 만료 되었습니다. (CredentialsExpiredException)
- 이미 로그인한 계정이 있습니다. (SessionAuthenticationException)
- 계정에 권한이 부여되지 않았습니다. (InternalAuthenticationServiceException)
- 계정 인증에 실패하였습니다.(위 Exception 이외)
*/
```

(심화) 로그인 서비스 API를 적용한 리펙토링

로그인 서비스 API를 사용하여 앞서 만들었던AuthenticationSuccessHandler를 리펙토링 한다.

```
@Override
public void onAuthenticationSuccess(
    HttpServletRequest request,
    HttpServletResponse response,
    Authentication authentication) throws IOException, ServletException {
    //인증에 성공한 계정 아이디 조회
    String userId = authentication.getName();
    logger.info("Account that succeeded authentication : {}", userId);
    try {
        //인증에 성공하였으므로 락카운트 초기화
        loginService.resetLockCnt(userId);
    } catch (Exception e) {
        logger.error("resetLockCnt Exception  : {} " , e.getMessage());
    }
    //인증 결과를 성공으로 json return
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("success", true);
    ObjectMapper om = new ObjectMapper();
```

```
    String jsonString = om.writeValueAsString(map);
    OutputStream out = response.getOutputStream();
    out.write(jsonString.getBytes());
}
```

로그인 서비스 API를 사용하여 앞서 만들었던AuthenticationFailureHandler를 리펙토링 한다.

```
@Override
public void onAuthenticationFailure(
    HttpServletRequest request, HttpServletResponse response,
    AuthenticationException exception) throws IOException, ServletException {

    //인증에 실패한 계정 아이디 조회
    String userId = request.getParameter("serviced");
    Logger.debug("Authentication failure message : {} : “
                , exception.getMessage());
    Logger.info("Account that failed authentication : {} ", userId);
    String returnMsg = null;
    try {
        //인증에 실패한 계정의 락카운트를 증가시키며
        //카운트 임계치를 넘으면 계정을 잠그는 일괄작업
        loginService.lockProcessAccount(userId, exception);
        //인증 실패 정보에 따라 유저에게 노출할 메시지
        returnMsg = loginService.getExceptionInfoMsg(exception);
    } catch (Exception e) {
        returnMsg = messageSource.getMessage("common.message.manager.fail“
                                             , null,
LocaleContextHolder.getLocale());
    }
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("success", false);
    map.put("message", returnMsg);
    ObjectMapper om = new ObjectMapper();
    String jsonString = om.writeValueAsString(map);
    OutputStream out = response.getOutputStream();
    out.write(jsonString.getBytes());
}
```

공통 코드

개요

사이트에서 사용되는 공통코드들을 사용하기위한 인터페이스를 제공한다.

태그라이브러리제공으로 공통적인 html코드를 생성해준다.

사용법

[commonCode bean 정의] context-common.xml

```
<bean id="applicationContextProvider"
class="net.lotte.chamomile.commons.code.ApplicationContextProvider" />
<bean id="commonCode"
    class="net.lotte.chamomile.commons.code.CommonCode">
    <property name="dataSource" ref="dataSource"/>
    <property name="cacheManager" ref="cacheManager"/>
    <property name="cacheName" value="commonCodeCache"/>
</bean>
```

datasource : 사전에 정의된 datasource를 지정해준다.

cachaManager : 공통코드에서 사용될 캐시 매니저 지정(지정하지 않을경우 호출 할 때 마다 DB 조회)

cachaManager : 캐시명

java에서의 활용

선언 및 정의

```
@Autowired
private CommonCode commonCode;
```

사용

```
// 대분류 기준으로 중분류 목록을 조회
List<CodeVO> listCode = commonCode.listCodes("[대분류코드]", [null | Locale정보]);
// 중분류 기준으로 소분류 목록을 조회
List<CodeVO> listCode = commonCode.listCodes("[대분류코드]", "[중분류코드]", [null
| Locale정보]);
```

Locale정보는 null로 설정할 경우 현재 설정된 언어로 값을 반환해 준다.

특정언어로 받고 싶을 경우 Locale.US등으로 조회 한다.

jsp에서의 활용

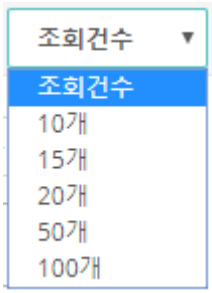
공통코드 tag library 선언 및 정의

```
<%@ taglib uri="https://www.ldcc.co.kr/chamomile/commonCode/tags"
prefix="cmmCode" %>
```

select 태그 만들기

```
<c:set var="rowCount">
    <spring:message code="screen.main.pannel01.listcount.title"/>
</c:set>
<cmmCode:select
                category="category00028"
                code=""
                onchange="changeRowCount1"
                className="form-control input-sm"
                id="test"
                defaultItem="rowCount"
                selectedItem="category00028code00001"
                useRealValue="true"
                extra="data-toggle='tooltip' data-placement='top'"/>
```

```
<select class="form-control input-sm" id="test" onchange="changeRowCount1(this)">
  <option value>조회건수</option>
  <option value="10">10개</option>
  <option value="15">15개</option>
  <option value="20">20개</option>
  <option value="50">50개</option>
  <option value="100">100개</option>
</select>
```



이 밖에 checkbox, radio 오브젝트들도 생성해주고 있으며 각 속성들의 의미는 아래와 같다.

[select]

```
category(필수) : 대분류
code(필수) : 중분류
className : select 태그에 적용할 클래스명
id : select 태그에 적용할 id
name : select 태그에 적용할 name
selectedItem : 생성시 선택할 option value
defaultItem : 최상단에 위치할 빈 value를 가지는 option요소
language : 언어코드. 값이 없을 경우 현재 설정된 언어코드로 설정
country : 국가코드. 값이 없을 경우 현재 설정된 언어코드로 설정
Onchange : onchange이벤트에 적용할 function 명
useRealValue : true일경우 realValue로 설정된값이 option의 value에 셋팅되고, false 일경우 코드값이 value로 적용됨
```

[checkbox]

```
category(필수) : 대분류
code(필수) : 중분류
className : input 태그에 적용할 클래스명
id : input 태그에 적용할 id
name : input 태그에 적용할 name
selectedItem : 생성시 선택할 value
language : 언어코드. 값이 없을 경우 현재 설정된 언어코드로 설정
country : 국가코드. 값이 없을 경우 현재 설정된 언어코드로 설정
useRealValue : true일경우 realValue로 설정된값이 value에 셋팅되고, false 일 경우 코드값이 value로 적용됨
```

[radio]

```
category(필수) : 대분류
code(필수) : 중분류
className : input 태그에 적용할 클래스명
id : input 태그에 적용할 id
name : input 태그에 적용할 name
selectedItem : 생성시 선택할 value
language : 언어코드. 값이 없을 경우 현재 설정된 언어코드로 설정
country : 국가코드. 값이 없을 경우 현재 설정된 언어코드로 설정
useRealValue : true일경우 realValue로 설정된값이 value에 셋팅되고, false 일 경우 코드값이 value로 적용됨
```

태그라이브러리 미 사용 시 공통코드 기능 사용

마크업 작성

```
<!-- 공통코드 html-->
<select id="test-sel"></select>
<input type="checkbox" id="test-check" />
<input type="radio" id="test-radio"/>
```

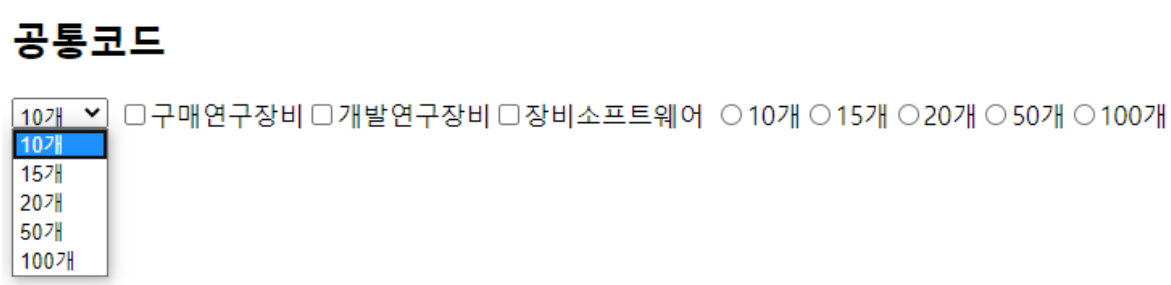
태그변수 생성

```
var tag1= new TagDataBuilder()
        .setId("test-sel")
        .setCategory("category00028")
        .build();

var tag2= new TagDataBuilder()
        .setId("test-check")
        .setCategory("category00052")
        .build();

var tag3 = new TagDataBuilder()
        .setId("test-radio")
        .setCategory("category00028")
        .build();
```

요청

```
_cmmCode.initCommonCode([tag1,tag2,tag3]);
```

결과



엑셀

개요

조회된 데이터를 엑셀파일로 만들어주거나 사용자에게 다운받을 수 있도록 제공해준다.

엑셀파일을 import가능한 형태의 데이터로 만들어주어 개발자에게 편의성을 제공한다.

대용량 엑셀 파일을 생성 해 준다.

사용법

엑셀 데이터 import

파일 업로더를 통해 업로드된 경로를 전달하여 SpreadSheet 객체로 만든다.

```
String path = [파일 경로]
SpreadSheet ss = ImportExcel.importData(path);
```

SpreadSheet객체는 아래와 같은 구조로 이루어져 있다.

```
SpreadSheet
    - sheet목록
        - sheet명
            - 행 목록
                - 셀 목록
```

이 객체에서 vo를 가지는 List형태로 값을 변환하고자 할 경우 아래의 과정으로 변환할 수 있다.

```
List<VO클래스명> listData = (List<VO클래스명>)
ExcelCommonUtil.getListFromExcel([SpreadSheet객체], "[sheet명]", VO클래스
명.class);
```

e.g.

```
List<CodeVO> category = (List<CodeVO>) ExcelCommonUtil.getListFromExcel(ss,
"CHMM_CATEGORY_INFO", CodeVO.class);
```

추출된 데이터는 아래의 예시처럼 Batch 처리로 insert작업을 수행한다.

```
BatchRequest batchRequest = new BatchRequest(1000);
messageDao.messageInsert(messageVO, batchRequest);
```

엑셀데이터 export

export는 서버에 파일로 저장하는 방식과 사용자에게 다운로드 하는 방식을 제공한다.

다운로드방식

[view resolver 설정] context-servlet.xml

```
<beans:bean id="multisheetSpreadSheetView"
class="net.lotte.chamomile.commons.spreadsheet.MultiSheetExcelViewer"/>
```

[view 선언/정의] Controller

```
@Resource(name="multisheetSpreadSheetView")
private ExcelViewer multisheetView;
```

[소스 구성] Controller

```
//데이터 조회
List<VO> list = service.getListData();
//엑셀파일명
String docName = URLEncoder.encode("엑셀파일명","UTF-8"); // UTF-8로 인코딩
//view 초기화
multisheetView.init();
//파일명 셋팅
multisheetView.setDocName(docName);
//데이터 추가
multisheetView.addData(list);
ModelAndView view = new ModelAndView();
//뷰 지정
view.setView(multisheetView);
return view;
```

[엑셀 정보입력] VO

- 엑셀 생성시 VO에 정의된 annotation을 기반으로 엑셀데이터를 만든다.
- sheet정보와 column 정보를 입력한다.

Sheet정보입력

```
@ExcelSheet(name="[sheet명]")
public class VO명 extends {
```

column정보입력

```
@ExcelColumn(name="컬럼명1", index=0)
private String column1;
@ExcelColumn(name="컬럼명2", index=1)
private String column2;
```

파일저장방식

```
//데이터 조회
List<VO> list = service.getListData();

ExcelFile excelFile = new MultiSheetExcelFile();
//초기화
excelFile.init();
//엑셀 데이터 추가
excelFile.addData(list);
//엑셀파일 생성
excelFile.makeExcelFile("[파일경로]");
```

VO구성은 다운로드 방식과 동일하게 구성한다.

대용량 데이터 생성

조회된 ResultSet을 기반으로 대용량 엑셀 데이터를 생성한다.

파일 포맷은 Microsoft Office XML formats 형태로 생성된다.

```
ResultSet rs = ...
LargeDataExport.createLargeExcel(rs, "[파일경로]");
```

커스텀 export

sheet데이터 구성시 자신만의 레이아웃이나 모양으로 데이터를 출력하고자 할 경우ExcelViewer를 상속받음으로서 이를 구현할 수 있다.

[ExcelViwer상속] CustomExcelViwer.java

```
public class CustomExcelViwer extends ExcelViewer {
```

[엑셀 출력 구현]

```
@Override
protected void makeExcel(List<List<?>> data, List<SheetElement> sheetInfo,
Workbook workbook) throws NoSuchFieldException, SecurityException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException,
NoSuchMethodException {

    ...
}
```

엑셀데이터를 만드는 곳은 makeExcel Override하여 구현한다.

data는 데이터 목록 가지고 있는 List로서 sheet별 내용을 채우기 위해 사용된다.

sheetInfo는 Sheet명을 처리하기위해 사용된다.

workbook은 엑셀 파일을 만들기위한 POI 객체 이다.

[view resolver 설정] context-servlet.xml

```
<beans:bean id="customExcelViwer" class="[자신이 만든 viwer클래스 전체 경로"/>
```

[view 선언/정의] Controller

```
@Resource(name="cusomExcelViwer")
private ExcelViewer customExcelView;
```

[구현부] 예시

```
@Override
protected void makeExcel(List<List<?>> data, List<SheetElement> sheetInfo,
Workbook workbook) throws NoSuchFieldException, SecurityException,
IllegalAccessException, IllegalArgumentException, InvocationTargetException,
NoSuchMethodException {
    //엑셀 데이터 핸들러 생성
    ExcelDataHandler edh = new ExcelDataHandler();
    //loop문을 위한 개수 저장
    int listSize = data.size();
    //list 에서 내용을 추출하여 엑셀을생성한다.
    //아래 로직은 sheet별 데이터를 가져오기위한 loop문이다.
    for (int i = 0 ; i < listSize ; i++) {
            //i번째 sheet에 넣을 데이터를 얻어온다.
        List<?> sheetData = data.get(i);
            //i번째 sheet명을 얻어온다.
        String sheetName = sheetInfo.get(i).getSheetName();
            //POI를 이용해 i번째 sheet를 만든다.
        Sheet sheet = workbook.createSheet(sheetName);

        Row row = null;
        Cell cell = null;
            //타이틀 행을 만든다.
        edh.makeViewTitle(sheetData, sheet, row, cell);
```

```
            //데이터가 표시될 행들을 만든다.
            edh.makeViewRows(sheetData, sheet, row, cell, locale);
        }
    }
```

[타이틀 행만들기 예제]

```
public void makeViewTitle(List<?> data, Sheet sheet, Row row, Cell cell) {
    // 제목 셀 생성
    row = sheet.createRow(0);
    // vo에 적용되어있는 annotation에서 정보를 꺼내온다.
    ExcelSheet sheetInfo =
data.get(0).getClass().getAnnotation(ExcelSheet.class);


    List<Field> listColumn =
ExcelCommonUtil.arrangeFields(data.get(0).getClass());
    int colCount = Math.min(listColumn.size(),sheetInfo.maxCols());

    // 스타일을 생성한다.
    CellStyle style = sheet.getWorkbook().createCellStyle();
    style.setFillForegroundColor(sheetInfo.foregroundColor());
    style.setFillPattern(sheetInfo.pattern());


    style.setBorderLeft(sheetInfo.borderLeft());
    style.setBorderTop(sheetInfo.borderTop());
    style.setBorderRight(sheetInfo.borderRight());
    style.setBorderBottom(sheetInfo.borderBottom());


    for (int colIndex = 0 ; colIndex < colCount ; colIndex++) {

        ExcelColumn cellInfo = null;
        try {
            // 컬럼별 정보를 알기위해 annotation을 얻어온다.
            cellInfo =
listColumn.get(colIndex).getAnnotation(ExcelColumn.class);
            if (cellInfo != null) {
                sheet.setColumnWidth(colIndex, cellInfo.width());
                cell = row.createCell(colIndex);
                cell.setCellStyle(style);
                cell.setCellValue("".equals(cellInfo.name()) ?
                                listColumn.get(colIndex).getName() :
cellInfo.name());
            }

        } catch (SecurityException e) {
            logger.error("make title security : " + e.getMessage());
        }
    }
}
```

[데이터 출력 행 만들기 예제]

타이틀만들기 행과 동일하게 작업하면 된다.

데이터를 구하기 위해서는 아래의 코드로 구해온다.

```
Object value = ExcelCommonUtil.invokeMethod(listColumn.get(colIndex).getName(),
data.get(rowIndex));
```

엑셀 api

클래스	메서드명	파라미터	반환값	설명
ExcelCommonUtil	getColumnInfo	Sheet sheet, int firstRowIndex	List<Column>	지정된 sheet의 요청된 행에서 column정보를 얻어온다.
	getExcel	String path	Workbook	파일을 읽어 workbook객체를 반환한다.
	getSheets	Workbook workbook	List<Sheet>	workbook으로 부터 sheet정보들을 읽어와 list로 반환한다.
	invokeMethod	String fieldName, Object obj	Object	필드명으로 get메서드를 실행하여 값을 반환한다.
	getListFromExcel	SpreadSheet ss, String sheetName, Class<?> clazz	List<?>	엑셀에서 추출한 SpreadSheet객체를 List형태로 반환한다.
	arrangeFields	Class<?> clazz	List<Field>	excel export시 annotation에 index로 정의된 순서대로 정렬 한 후 annotation이 정의된 field목록들만 반환한다.
ExcelDataHandler	makeViewTitle	List<?> data, Sheet sheet, Row row, Cell cell		타이틀 행을 만든다. 설정된 최대 열 이상의 데이터는 무시된다.
	makeViewRows	List<?> data, Sheet sheet, Row row, Cell cell, Locale locale	int	데이터 타입별로 알맞게 각 행들을 만든다. 설정된 최대 행/최대 열 이상의 데이터는 무시된다.
	getRows	Sheet sheet, int firstRowIndex	List<RowElement>	엑셀 sheet에서 행/셀 데이터를 추출하여 RowElement로 변환한다.
- ExcelFile - ExcelViewer	setLocale	Locale locale		로케일정보를 저장한다.
	setDocName (ExcelViewer에만 해당됨)	String docName		파일명을 지정한다.
	addData	List<?> data		조회된 데이터를 추가한다. 이 때 sheet명은 annotation에 지정된 이름으로 한다. 중복될 경우 뒤에 '#'이 자동으로 추가된다.

클래스	메서드명	파라미터	반환값	설명
	addData	List<?> data, String sheetName		sheet네임을 지정하여 조회된 데이터를 추가한다. 이때 annotatino에 지정된 이름은 무시된다. 중복될 경우 뒤에 '#'이 자동으로 추가된다.
	init			리스트를 초기화해준다.
	makeExcelFile (ExcelFile에만 해당됨)	String filePath		엑셀 파일을 생성하여 서버에 저장한다.
	makeExcel	List<List<?>> data, List<SheetElement> sheetInfo, Workbook workbook		주어진 데이터로 엑셀파일을 구성하는 메서드. 재정의시 해당 메서드를 이용하여 커스터마이징한다.
ImportExcel	importData	String path	SpreadSheet	엑셀파일의 로컬경로를 매개변수로 넘겨 SpreadSheet객체로 받는다
LargeDataExport	createLargeExcel	ResultSet rs, String path	boolean	대용량 엑셀파일을 생성한다. 이 기능이 완료 되기 전까지 connection, Statement등을 close시키지 말아야 한다. 기존파일은 덮어 쓴다.
SpreadSheet	addSheet	SheetElement sheet		SpreadSheet객체는 엑셀파일 한 개로 인식하면 된다. 이 메소드는 sheet를 추가한다.
	get	String sheetName	SheetElement	sheet명으로 객체내의 sheet를가져온다.
	get	int index	SheetElement	index로 객체내의 sheet를가져온다.
	getSheets		List<SheetElement>	객체내의 모든 sheet를가져온다.
SheetElement	SheetElement	String sheetName		생성자. sheet명을 지정하여 생성한다.
	addRow	RowElement row		sheet내에 행을 추가한다.
	setRows	List<RowElement> rows		sheet내에 행을 지정한다.
	get	int index	RowElement	지정된 행을 얻어온다.
	getSheetName		String	현재 sheet명을 얻어온다.

클래스	메서드명	파라미터	반환값	설명
	setSheetName	String sheetName		sheet명을 지정한다.
	size		int	전체 행 수를 반환한다.
	getRows		List<RowElement>	전체 행을 반한다.
RowElement	get	String columName	String	컬럼 이름으로 데이터를 얻어온다.
	get	int index	String	인덱스 번호로 데이터를 얻어온다.
	addCell	CellElement cell		행에 데이터를 추가한다

Excel Export시 사용되는 annotation정보

anntation명	속성	기본값	설명
ExcelSheet - 클래스에 적용한다.	name	sheet	기본sheet명이며 엑셀파일 생성시 addData에서 sheet명을 넘길경우 무시된다.
	maxCols	500	최대 열 수 cf.) Excel 97~2003 형식(*,xls) : 65,536행/256열 Excel 2007~2013 (*.xlsx/xlsm) : 1,048,576행x16,384열
	maxRows	2000	최대 행 수 cf.) Excel 97~2003 형식(*,xls) : 65,536행/256열 Excel 2007~2013 (*.xlsx/xlsm) : 1,048,576행x16,384열

anntation명	속성	기본값	설명
	foregroundColor	0x16 (HSSFColor .HSSFColorPredefined .GREY_25_PERCENT .getIndex())	배경색 cf. **public enum** HSSFColorPredefined { **BLACK** (0x08, -1, 0x000000), **BROWN** (0x3C, -1, 0x993300), **OLIVE_GREEN** (0x3B, -1, 0x333300), **DARK_GREEN** (0x3A, -1, 0x003300), **DARK_TEAL** (0x38, -1, 0x003366), **DARK_BLUE** (0x12, 0x20, 0x000080), **INDIGO** (0x3E, -1, 0x333399), **GREY_80_PERCENT** (0x3F, -1, 0x333333), **ORANGE** (0x35, -1, 0xFF6600), **DARK_YELLOW** (0x13, -1, 0x808000), **GREEN** (0x11, -1, 0x008000), **TEAL** (0x15, 0x26, 0x008080), **BLUE** (0x0C, 0x27, 0x0000FF), **BLUE_GREY** (0x36, -1, 0x666699), **GREY_50_PERCENT** (0x17, -1, 0x808080), **RED** (0x0A, -1, 0xFF0000), **LIGHT_ORANGE** (0x34, -1, 0xFF9900), **LIME** (0x32, -1, 0x99CC00), **SEA_GREEN** (0x39, -1, 0x339966), **AQUA** (0x31, -1, 0x33CCCC), **LIGHT_BLUE** (0x30, -1, 0x3366FF), **VIOLET** (0x14, 0x24, 0x800080), **GREY_40_PERCENT** (0x37, -1, 0x969696), **PINK** (0x0E, 0x21, 0xFF00FF), **GOLD** (0x33, -1, 0xFFCC00), **YELLOW** (0x0D, 0x22, 0xFFFF00), **BRIGHT_GREEN** (0x0B, -1, 0x00FF00), **TURQUOISE** (0x0F, 0x23, 0x00FFFF), **DARK_RED** (0x10, 0x25, 0x800000), **SKY_BLUE** (0x28, -1, 0x00CCFF), **PLUM** (0x3D, 0x19, 0x993366), **GREY_25_PERCENT** (0x16, -1, 0xC0C0C0), **ROSE** (0x2D, -1, 0xFF99CC), **LIGHT_YELLOW** (0x2B, -1, 0xFFFF99), **LIGHT_GREEN** (0x2A, -1, 0xCCFFCC), **LIGHT_TURQUOISE** (0x29, 0x1B, 0xCCFFFF), **PALE_BLUE** (0x2C, -1, 0x99CCFF), **LAVENDER** (0x2E, -1, 0xCC99FF), **WHITE** (0x09, -1, 0xFFFFFF), **CORNFLOWER_BLUE** (0x18, -1, 0x9999FF), **LEMON_CHIFFON** (0x1A, -1, 0xFFFFCC), **MAROON** (0x19, -1, 0x7F0000), **ORCHID** (0x1C, -1, 0x660066), **CORAL** (0x1D, -1, 0xFF8080), **ROYAL_BLUE** (0x1E, -1, 0x0066CC), **LIGHT_CORNFLOWER_BLUE**(0x1F, -1, 0xCCCCFF), **TAN** (0x2F, -1, 0xFFCC99), **AUTOMATIC** (0x40, -1, 0x000000);

anntation명	속성	기본값	설명
	pattern	FillPatternType .SOLID_FOREGROUND	배경 패턴 cf. **public enum** FillPatternType { /** No background */ **NO_FILL**(0), /** Solidly filled */ **SOLID_FOREGROUND**(1), /** Small fine dots */ **FINE_DOTS**(2), /** Wide dots */ **ALT_BARS**(3), /** Sparse dots */ **SPARSE_DOTS**(4), /** Thick horizontal bands */ **THICK_HORZ_BANDS**(5), /** Thick vertical bands */ **THICK_VERT_BANDS**(6), /** Thick backward facing diagonals */ **THICK_BACKWARD_DIAG**(7), /** Thick forward facing diagonals */ **THICK_FORWARD_DIAG**(8), /** Large spots */ **BIG_SPOTS**(9), /** Brick-like layout */ **BRICKS**(10), /** Thin horizontal bands */ **THIN_HORZ_BANDS**(11), /** Thin vertical bands */ **THIN_VERT_BANDS**(12), /** Thin backward diagonal */ **THIN_BACKWARD_DIAG**(13), /** Thin forward diagonal */ **THIN_FORWARD_DIAG**(14), /** Squares */ **SQUARES**(15), /** Diamonds */ **DIAMONDS**(16), /** Less Dots */ **LESS_DOTS**(17), /** Least Dots */ **LEAST_DOTS**(18);
	borderLeft	BorderStyle.THIN	왼쪽줄 cf. **public enum** BorderStyle { /** * No border (default) */ **NONE**(0x0), /** * Thin border */ **THIN**(0x1), /** * Medium border */ **MEDIUM**(0x2), /** * dash border */ **DASHED**(0x3), /** * dot border */ **DOTTED**(0x4), /** * Thick border */ **THICK**(0x5), /** * double-line border */ **DOUBLE**(0x6), /** * hair-line border */ **HAIR**(0x7), /** * Medium dashed border */ **MEDIUM_DASHED**(0x8), /** * dash-dot border */ **DASH_DOT**(0x9), /** * medium dash-dot border */ **MEDIUM_DASH_DOT**(0xA), /** * dash-dot-dot border */ **DASH_DOT_DOT**(0xB), /** * medium dash-dot-dot border */ **MEDIUM_DASH_DOT_DOT**(0xC), /** * slanted dash-dot border */ **SLANTED_DASH_DOT**(0xD);
	borderRight	BorderStyle.THIN	오른쪽줄
	borderTop	BorderStyle.THIN	윗쪽줄
	borderBottom	BorderStyle.THIN	아래쪽줄
ExcelColumn - 출력 하고자 하는 필드에 적용한다.	name	공백문자	타이틀에 표시할 컬럼명을 입력한다.
	width	4000	컬럼 폭

anntation명	속성	기본값	설명
	l10n	공백	지역화(date, currency)
	bMasking	false	민감정보마스킹여부
	index	0	데이터 표시순서
	foregroundColor	0x16 (HSSFColor .HSSFColorPredefined .GREY_25_PERCENT .getIndex())	배경색
	pattern	FillPatternType .SOLID_FOREGROUND	배경패턴
	borderLeft	BorderStyle.THIN	왼쪽줄
	borderRight	BorderStyle.THIN	오른쪽줄
	borderTop	BorderStyle.THIN	윗쪽줄
	borderBottom	BorderStyle.THIN	아래쪽줄

푸시서비스

개요

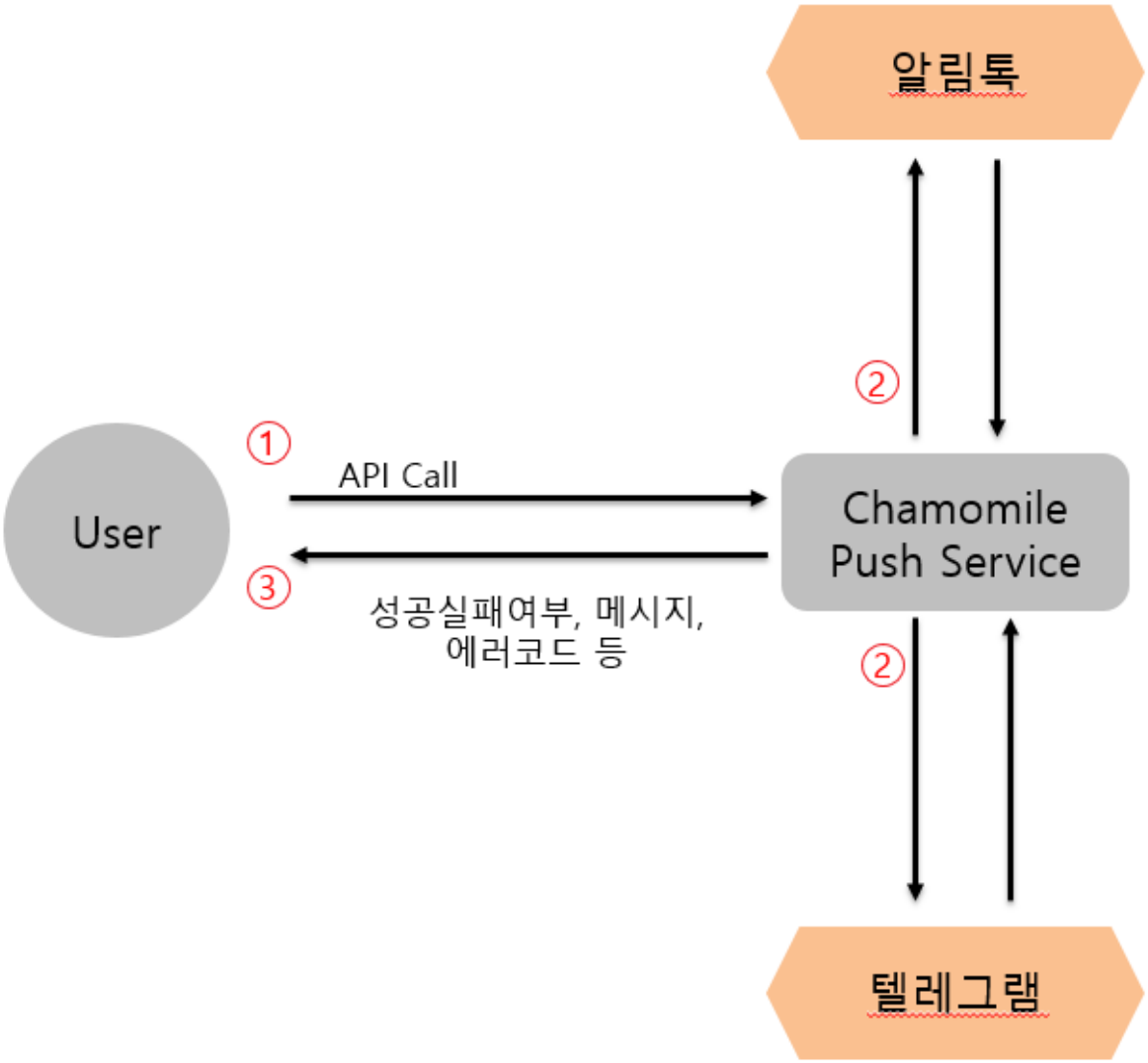
푸시 서비스란?

케모마일 푸시 서비스는 알림톡과 텔레그램 메신저를 통해 메시지를 발송 할 수 있는 서비스를 제공한다.

각 푸시 서비스에 대한 자세한 프로토콜 및 스펙은 본 문서에서 다루지 않는다.

케모마일에서 제공하는 푸시 서비스 bean을 등록하여 메시지를 발송한다.

흐름도



설정

케모마일에서 제공하는 푸시 서비스 빈을 루트 컨텍스트에 등록한다.
(context-common.xml 확인)

```
<!-- Push Message Service -->
<bean id="pushMessageService"
class="net.lotte.chamomile.commons.push.PushMessageService"/>
```

application.properties 파일을 통해 발송을 위해 필요한 값을 설정한다.

알림톡의 각 프로퍼티의 값에 대한 자세한 내용은 알림톡 담당자(롯데정보통신 e-biz 사업팀 신사업담당. 2018년 12월 현재)를 통해 확인한다.

예시)

```
##PUSH
#TELEGRAM
chmm.push.telegram.token = 598847139:AAHI-iPNiLXrVRuyZ_bGuNn9ijsZTrA1TSs
#ALIMTALK
chmm.push.alimtalk.host = 111.222.333.444
chmm.push.alimtalk.port = 1234
chmm.push.alimtalk.tokenPath = /v1/auth/tokens
chmm.push.alimtalk.clientId = lotte
```

```
chmm.push.alimtalk.clientPwd = 1234
chmm.push.alimtalk.alimTalkPath = /v1/send/kakao-notice
chmm.push.alimtalk.msgId = 1
chmm.push.alimtalk.sendTime = 2018-07-06 17:08:01
chmm.push.alimtalk.sendPhone = 01012345678
chmm.push.alimtalk.templateCode = alimtalktest_001
chmm.push.alimtalk.senderKey = 94040debee2faf0734fbd5a200a1a73077869c12
```

사용법

앞서 등록한 빈을 통해 API를 활용하여 푸시 메시지를 발송할 수 있다.

알림톡 발송 API

```
@Test
public void testLAlimtalk() throws Exception {
    pushMessageService.sendAlimTalkMessage
        ("[카카오뮤직] 회원가입 안내 아무게님, 카카오뮤직 회원이 되신 것을
         환영합니다. ▶신규 가입 회원 혜택 #{START_DATE} #{STATUS}",
         "01012345678");
}
```

텔레그램 발송 API

```
@Test
public void testTelegram() throws Exception {
    pushMessageService.sendTelegramMessage
        ("메세지 테스트", "649017344");
}
```

파일 핸들링

개요

FileHandler Util은 시스템 내의 용량 체크, 파일 핸들링, 권한 제어 등의기능을 담당한다.

권한 제어는 리눅스 환경에서 Owner의 권한만 제어가 가능하다.

dependency

```
<dependency>
      <groupId>commons-io</groupId>
      <artifactId>commons-io</artifactId>
</dependency>
```

FileHandlerUtil api목록

메서드명	파라미터	반환값	설명
getStorageInfo	String path	long	해당 디스크의 가용 용량을 반환.(MB단위)
getStorageInfo	String path, FileSizeUnit unit	long	해당 디스크의 가용용량을 반환(byte단위)
makeFile	String path, String filename, byte[] data		파일을 지정한 경로에 생성
removeFile	String path	boolean	파일을 삭제
copyFileToDir	String src, String destPath		파일을 디렉토리로 복사
copyDir	String src, String desc		디렉토리 복사
makeDir	String path	boolean	디렉토리 생성
removeDirForce	String path		디렉토리 강제삭제. 디렉토리가 비어있지 않아도 모든 디렉토리 삭제
removeDir	String path		디렉토리 삭제. 디렉토리가 비어있지 않으면 Exception
moveFile	String src, String desc		파일이동
moveDir	String src, String desc		디렉토리 이동
changeFileAuthReadable	File file, boolean readable		파일의 읽기권한 제어
changeFileAuthWriteable	File file, boolean writable		파일의 쓰기권한 제어
changeFileAuthExecutable	File file, boolean executable		파일의 실행권한 제어

예제

[가용용량 체크]
localPath: 원하는 경로의 String을 입력

```
assertTrue(FileHandlerUtil.getStorageInfo(localPath) > 0);
```

[파일생성 및 삭제]
임시파일을과 임시디렉토리를 생성한다.

```
File file = new File(localfilePath);
try (BufferedWriter bw = new BufferedWriter(new FileWriter(file))) {
    bw.write("This is test");
}
```

생성한 임시파일의 byte데이터를 makeFile API에 넘긴다.

```
byte[] byt = Files.readAllBytes(Paths.get(file.getAbsolutePath()));
FileHandlerUtil.makeFile(dirPath, "테스트.txt", byt);
```

[디렉토리 생성 및 삭제]

현재 path에 dirTest라는 디렉토리를 만들고 삭제한다.

```
assertTrue(FileHandlerUtil.makeDir("./dirTest"));
try {
    FileHandlerUtil.removeDirForce("./dirTest");
} catch (IOException e) {
    fail(e.getMessage());
}
```

[파일 및 디렉토리 복사]
첫번째 임시 디렉토리와 임시 파일을 생성한다.

```
FileHandlerUtil.copyFileToDir(localfilePath, localDirPath);
/* 첫번째 임시디렉토리를 두번째 임시디렉토리로 복사 */
FileHandlerUtil.copyDir(localDirPath, "./test2");
FileHandlerUtil.removeDirForce("./test2");
FileHandlerUtil.removeDirForce(localDirPath);
```

[파일 및 디렉토리 이동]

```
String moveDirPath = localPath + "이동디렉토리";
FileHandlerUtil.moveFile(localfilePath, moveDirPath);
assertTrue(new File(moveDirPath).listFiles().length > 0);
FileHandlerUtil.removeDirForce(moveDirPath);
```

파일 업로드

개요

FileUpload Util은 웹 서비스 요청으로 파일 업로드를 수행한다.

업로드 가능한 확장자, 파일 용량 체크, 파일 이름 난독화 등의 기능을 수행한다.

설정

어드민을 통한 설정

업로드 가능한 확장자 및 파일 사이즈 등은 **[어드민 – 시스템 설정 -시스템 환경관리]**를 통해 DB Table(CHMM_SYSTEM_DEFAULT_INFO)에 저장되어 있다.

properties 통한 설정(application.properties)

```
##FILE
chmm.file.dir=C:\\files
chmm.file.upload.tableName=CHMM_SYSTEM_DEFAULT_INFO
chmm.file.upload.allowExtensionKey=FILEUPLOAD_ALLOWED_EXTENSION
chmm.file.upload.maxSizeKey=FILEUPLOAD_MAX_SIZE
chmm.file.upload.column=ENV_VALUE
```

context-file.xml설정

```
<bean id="fileUploadUtil"
class="net.lotte.chamomile.commons.file.FileUploadUtil">
    <property name="dataSource" ref="dataSource"/>
    <!-- file size의 단위(byte, kb, mb 등) -->
    <property name="fileSizeUnit">
        <value type="net.lotte.chamomile.commons.file.FileSizeUnit">
            BY
        </value>
    </property>
    <!-- file upload 설정 참조할 table 명 -->
    <property name="tableName" value="${chmm.file.upload.tableName}"/>
    <!-- table에 들어있는 파일업로드 확장자 리스트 -->
    <property name="allowExtensionKey"
value="${chmm.file.upload.allowExtensionKey}"/>
    <!-- table에 들어있는 파일 최대 사이즈 -->
    <property name="maxSizeKey" value="${chmm.file.upload.maxSizeKey}"/>
    <!-- table이 key value 로 되어있기때문에 column값 명시 -->
    <property name="column" value="${chmm.file.upload.column}"/>
</bean>
```

사용방법

요청으로 들어온 file은 FileUploadUtil에 넘겨준다.
업로드 수행된 후 관련 정보가 반환되며 DB에 추가로 저장하는 작업을 한다면 FileUploadVo[] fileVo 를 이용하면 된다.

```
@Resource FileUploadUtil fileUploadUtil;
@RequestMapping(value = "/upload", method = RequestMethod.POST)
public void fileUpload(MultipartFile file) throws Exception {
    FileUploadVo[] fileVo = fileUploadUtil.fileUpload(
        new MultipartFile[] {file});
}
```

FileUploadVo 객체 제공 데이터

메서드명	데이터 타입	설명
getUploadPath()	String	파일이 업로드된 절대경로를 반환한다.
getOriginalFileName()	String	업로드된 파일의 원본 파일명
getUploadFileName()	String	업로드 후 hash값으로 변환된 파일명
getFileSize()	long	업로드된 파일의 크기

파일 다운로드

개요

FileDownload Util은 웹 서비스상에서 파일을 클라이언트에게전송하는 역할을 한다.

FileUpload Util과 상관없이 사용이 가능하며 FileDownload Util내에서 직접 Http Response 를 수행한다.

설정

[루트 디렉토리 설정(application.properties)]

```
chmm.file.dir=C:\\files
```

[FileDirectoryManagement Bean 등록]

```
<!-- properties 에 적혀있는 file서버의 경로를 저장하고있다. -->
<bean id="fileDirectoryManagement"
class="net.lotte.chamomile.commons.file.FileDirectoryManagement">
    <!-- file server path -->
    <constructor-arg name="dirPath" value="${chmm.file.dir}"/>
</bean>
```

[FileDownloadUtil Bean 등록]

```
<!-- file download util -->
<bean id="fileDownloadUtil"
class="net.lotte.chamomile.commons.file.FileDownloadUtil" />
```

사용법

서버에 있는 파일명에 “/”또는 “\” 문자가 포함될경우 SecurityException이
발생한다.

1) 루트 디렉토리에서 파일 다운로드 하는 경우

```
@Resource FileDownloadUtil fileDownloadUtil;

@RequestMapping(value = "/download", method = RequestMethod.GET)
protected void download(HttpServletRequest req,
                        HttpServletResponse res){
    fileDownloadUtil.fileDownload(
        "[서버에 있는 파일명]",
        "[클라이언트에 노출할 파일명]",
        res);
```

2) 루트의 하위 디렉토리에서 파일 다운로드 하는 경우

```
@Resource FileDownloadUtil fileDownloadUtil;

@RequestMapping(value = "/download", method = RequestMethod.GET)
protected void download(HttpServletRequest req,
                        HttpServletResponse res){
    fileDownloadUtil.fileDownload(
        “[하위 디렉토리명]”,
        "[서버에 있는 파일명]",
        "[클라이언트에 노출할 파일명]",
        res);
```

FTP

개요

구조

FTP 유틸은 FtpUtil과 SftpUtil 클래스로 제공된다.

FtpUtil은 org.apache.commons.net.ftp.FTPClient 클래스를Wrapping하여 업로드, 다운로드, 리스트 조회 등의 기능을 수행한다.

SftpUtil은 com.jcraft.jsch.ChannelSftp 클래스를 Wrapping하여 마찬가지로 업로드, 다운로드, 리스트 조회 등의 기능을 수행한다.

FTP는 Encryption, Firewalls, Vulnerabilities 등의 이유로 SFTP 를 이용하는 것이 바람직하며 Util자체의 기능은 동일하므로 SftpUtil을 위주로 설명한다.

dependency

```
FTP (Apache License 2.0)
<dependency>
    <groupId>commons-net</groupId>
    <artifactId>commons-net</artifactId>
</dependency>
SFTP (GNU LGPL License)
<dependency>
    <groupId>com.jcraft</groupId>
    <artifactId>jsch</artifactId>
</dependency>
```

설정

인증정보를 담고있는 CommonAuth생성

```
<bean id="commonAuth" class="net.lotte.chamomile.commons.cm.CommonAuth">
    <constructor-arg name="address" value="${chmm.smtp.address}" />
    <constructor-arg name="port" value="${chmm.smtp.port}" />
    <constructor-arg name="id" value="${chmm.smtp.id}" />
    <constructor-arg name="password" value="${chmm.smtp.pw}" />
</bean>
```

SFTP Util bean등록

```
<bean id="sftpUtil" class="net.lotte.chamomile.commons.ftp.SftpUtil">
    <constructor-arg name="auth" ref="sftpAuth" />
    <constructor-arg name="hostKey" ref="${sftp.hostKey}" />
</bean>
```

사용법

[연결]

```
CommonAuth commonAuth = new CommonAuth([주소(ip, domain등)], [ID], [PASSWORD]);
ftpUtil = new SftpUtil(commonAuth, [호스트 키]);
ftpUtil.connect();
ftpUtil.setEncoding([인코딩정보(UTF-8등)]);
```

[Direct remote control]

SftpUtil의 method를 사용할 수도 있지만 직접적인 컨트롤을 원한다면 다음과 같이 사용이 가능하다

```
ChannelSftp sftp = ftpUtil.getSftp();
Sftp.mkdir(“test”);
Sftp.chmod(774, “/someFile”);
```

[Check Working Directory]

SftpUtil의 현재 working directory를 다음과 같이 가져올 수 있다.

```
ftpUtil.changePath(“/someDirectory”);
Final String workingDirectoryPath = ftpUtil.getWorkingDirectory();
```

[List of Files]

```
Public LsEntry[] listOfFiles(): 모든 파일리스트를 가져온다.
Public LSEntry[] listOfFiles(final String regex): 파일이름이
regex패턴에 일치하는 파일만 가져온다.

모든 파일 가져오기
LsEntry[] files = ftpUtil.listOfFiles();

txt 확장자 파일만 가져오기
LsEntry[] files = ftpUtil.listOfFiles("^\\S+.(?i)(txt)$");
```

[List of Directories]

현재 경로(Working Directory 또는 pwd)에서 디렉토리 리스트를가져오는 API는 2개로 구성되어 있다.

```
public LsEntry[] listOfDirs(): 모든 디렉토리 리스트를 가져온다.
Public LSEntry[] listOfDirs(final String regex): 파일이름이
regex패턴에 일치하는 파일만 가져온다.

모든 디렉토리 가져오기
LsEntry[] files = ftpUtil.listOfDirs();
```

[File Upload]

파일업로드 API는 업로드될 디렉토리경로와 업로드할 파일경로를 입력한다.

```
final String uploadDirPath = "…"; (FTP 서버내부의 디렉토리 경로)
final String uploadFilePath = "…"; (웹서버내부의 파일 경로)
ftpUtil.uploadFile(uploadDirPath, uploadFilePath);
```

[File & Directory Delete]

삭제할 파일의 경로를 입력한다.

```
ftpUtil.deleteFile("/someDirectory/someFile.txt")
```

삭제할 디렉토리의 경로를 입력한다. 해당 디렉토리의 내부가 비어있지 않으면 삭제할 수 없다.

```
ftpUtil.deleteDir("/someDirectory")
```

[make Directory]

디렉토리를 생성한다.

```
ftpUtil.makeDirectory("/someDirectory")
```

[Download File]

다운로드 받을 파일경로와 로컬상에 저장할 위치를 입력한다.

```
// FTP서버의 someFilePath에 해당하는 파일을 로컬 C:\Directory 경로로 다운로드한다.
ftpUtil.downloadFile("someFilePath", "C:\Directory");
```

Json Util

개요

마샬링은 객체의 메모리 구조를 저장이나 전송을 위해서 적당한 자료 형태로 변형하는 것을 의미한다.

마샬링은 통신을 위해 데이터가 이동되어야 할 경우 사용되며, 다른 컴퓨터나 다른 프로그램 간에 데이터를 전송해야 할 때 사용한다.

마샬링과 반대되는 의미로 언마샬링을 사용하며, 내용은 다르지만 유사한 의미로 직렬화(Serialization)와 역직렬화(Deserialization)가 있다.

동작방식

JsonUtil은 Jackson library 의 ObjectMapper를 기반으로 동작하며, 제공되는 기능은 마샬링, 언마샬링이 있다.

해당 기능은 static method를 직접 호출하여 결과를 얻을 수 있다. 자세한 사용방법은 테스트 코드를 통해 샘플을 제공한다.

dependency

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.woodstox</groupId>
    <artifactId>woodstox-core</artifactId>
</dependency>
```

사용법

마샬링

마샬링을 하기 위해 먼저 마샬링을 하기 위한 클래스를 만들어 둔다.

아래는 id, name, skills의 멤버 변수를 가진 클래스이다.

```
@JsonTypeName(value = "Employee")
@JsonTypeInfo(include = JsonTypeInfo.As.WRAPPER_OBJECT, use =
JsonTypeInfo.Id.NAME)
public class Employee {
```

```
    private int id;
    private String name;
    private List<String> skills;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<String> getSkills() {
        return skills;
    }
    public void setSkills(List<String> skills) {
        this.skills = skills;
    }
}
```

변환을 위해 해당 클래스에 위와 같은 Type annotation을 선언해야 한다. 명시적으로 선언하지 않으면 타입을 추정하여 변환을 하게 된다.

@JsonTypeName은 클래스가 가지는 논리명을 바인딩하기 위한 정보로 JsonTypeInfo를 설정하는데 사용된다.

@JsonTypeInfo는 Json의 직렬화 및 해제에 대한 형식 정보를 기술하며 실제 클래스에 대한 정보를 보존하는데 사용된다. 위 기술된 내용 중 상세 항목의 내역은 다음과 같다.

- include 항목의 WRAPPER_OBJECT 는 실제 json의 유형 식별자 값을 포함한다는 설정이다.
- use의 NAME 항목은 논리 타입 이름이 타입 정보로 사용된다는 것을 의미한다.
- 여기서는 앞서 만든 클래스에서 선언한 @JsonTypeName항목을 참조하게 된다.

상세한 annotation의 설명은 Jackson-annotation javadoc 웹페이지에서 확인할 수 있다.
https://fasterxml.github.io/Jackson-annotations/javadoc/2.4/com/fasterxml/jackson/annotation/package-summary.html

아래 테스트 코드는 앞에서 만든 클래스의 객체를 생성한 뒤 마샬링을 통해 jsonString을 얻는 과정이다.

```
@Test
public void testJsonMarshaller() {
    Employee employee = new Employee();
    List<String> skills = new ArrayList<String>();
    skills.add("test1");
    skills.add("test2");
    employee.setId(101);
    employee.setName(“test_name");
    employee.setSkills(skills);
    try {
        System.out.println(JsonUtil.toJson(employee));
    } catch (JsonProcessingException e) {
```

```
        fail(e.getMessage());
    }
}
```

아래와 같이 객체를 생성하고 값을 설정한다.

```
Employee employee = new Employee();
List<String> skills = new ArrayList<String>();
skills.add("test1");
skills.add("test2");
employee.setId(101);
employee.setName(“test_name");
employee.setSkills(skills);
```

생성한 객체를 아래와 같이 JsonUtil의 toJson 함수를 호출하여 마샬링을 수행한다.

```
try {
    System.out.println(JsonUtil.toJson(employee));
} catch (JsonProcessingException e) {
```

언마샬링

반대로 아래 테스트 코드는 앞에서 만든 클래스의 객체를 얻기 위해, jsonString을 언마샬링하여 객체를 얻는 과정이다.

```
@Test
public void testJsonUnMarshaller() {
    try {
        Employee employee = JsonUtil.fromJson(Employee.class, str);

        System.out.println("ID: " + employee.getId());
        System.out.println("Name: " + employee.getName());
        System.out.println("Skills: " +
StringUtils.join(employee.getSkills(), ','));
    } catch (JsonParseException e) {
        fail(e.getMessage());
    } catch (JsonMappingException e) {
        fail(e.getMessage());
    } catch (IOException e) {
        fail(e.getMessage());
    }
}
```

아래 string은 객체 변환을 위해 만든 jsonString이다.

```
String str=
    "{"+
    "    \"Employee\" : {"+
    "    \"id\" : 123,"+
    "    \"name\" : \"abcd\","+
    "    \"skills\" : [ \"test1\", \"test2\" ]"+
    "    }"+
    "}";
```

생성한 string을 아래와 같이 JsonUtil의 fromJson 함수를 호출하여 언마샬링을 수행한다.

```
try {
    Employee employee = JsonUtil.fromJson(Employee.class, str);

    System.out.println("ID: " + employee.getId());
    System.out.println("Name: " + employee.getName());
    System.out.println("Skills: " + StringUtils.join(employee.getSkills(),
','));
} catch (JsonParseException e) {
```

API

JsonUtil은 마샬링, 언마샬링을 위한 유틸로서 다음과 같은 static method로 구성되어 있다.

```
public static String toJson(Object target)
public static <T> T fromJson(Class<T> classpath, String target)
```

Object target : 마샬링 할 객체를 의미한다.

Class<T> classpath : 언마샬링 대상이 되는 객체를 의미한다.

String target : 언마샬링 할 json String을 의미한다.

XML Util

개요

동작방식

XmlUtil은 Jackson library 의 XmlMapper를 기반으로 동작하며, 제공되는 기능은 3가지로, 마샬링, 언마샬링, Document 변환이 있다.

해당 기능은 static method를 직접 호출하여 결과를 얻을 수 있다. 자세한 사용방법은 테스트 코드를 통해 샘플을 제공한다.

12.1.2. dependency

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
<dependency>
    <groupId>com.fasterxml.woodstox</groupId>
    <artifactId>woodstox-core</artifactId>
</dependency>
```

사용법

마샬링

마샬링을 하기 위해 먼저 마샬링을 하기 위한 클래스를 만들어 둔다.

아래는 id, name, skills의 멤버 변수를 가진 클래스이다.

```
public class Employee {
    private int id;
    private String name;
    private List<String> skills;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public List<String> getSkills() {
        return skills;
    }
    public void setSkills(List<String> skills) {
        this.skills = skills;
    }
}
```

변환을 위해 해당 클래스에 @XmlRootElement 등의 annotation을 선언해야 하나, 명시적으로 선언하지 않아도 타입을 추정하여 변환이 가능하다.

아래 테스트 코드는 앞에서 만든 클래스의 객체를 생성한 뒤 마샬링을 통해 xmlString을 얻는 과정이다.

```
@Test
public void testXmlMarshaller() {
```

```
    Employee employee = new Employee();
    List<String> skills = new ArrayList<String>();
    skills.add("test1");
    skills.add("test2");
    employee.setId(101);
    employee.setName(“test_name");
    employee.setSkills(skills);
    try {
        XmlUtil.toXml(employee);
    } catch (JsonProcessingException e) {
        fail(e.getMessage());
    }
}
```

아래 와 같이 객체를 생성하고 값을 설정한다.

```
Employee employee = new Employee();
List<String> skills = new ArrayList<String>();
skills.add("test1");
skills.add("test2");
employee.setId(101);
employee.setName(“test_name");
employee.setSkills(skills);
```

생성한 객체를 아래와 같이 XmlUtil의 toXml 함수를 호출하여 마샬링을 수행한다.

```
try {
    System.out.println(XmlUtil.toXml(employee));
} catch (JsonProcessingException e) {
```

언마샬링

반대로 아래 테스트 코드는 앞에서 만든 클래스의 객체를 얻기 위해, xmlString을 언마샬링하여 객체를 얻는 과정이다.

```
@Test
public void testXmlUnMarshaller() {
    try {
        Employee employee = XmlUtil.fromXml(Employee.class, str);

        System.out.println("ID: " + employee.getId());
        System.out.println("Name: " + employee.getName());
        System.out.println("Skills: " + StringUtils.join(employee.getSkills(),
','));
    } catch (JsonParseException e) {
        fail(e.getMessage());
    } catch (JsonMappingException e) {
        fail(e.getMessage());
    } catch (IOException e) {
        fail(e.getMessage());
    }
}
```

아래 string은 객체 변환을 위해 만든 xmlString이다.

```
String str =
    "<Employee>\n" +
    "   <id>101</id>\n" +
    "   <name>test_name</name>\n" +
    "   <skills>"+
    "       <skills>test1</skills>\n" +
    "       <skills>test2</skills>\n" +
    "   </skills>"+
    "</Employee>";
```

생성한 string을 아래와 같이 XmlUtil의 fromXml 함수를 호출하여 언마샬링을 수행한다.

```
try {
    Employee employee = XmlUtil.fromXml(Employee.class, str);

    System.out.println("ID: " + employee.getId());
    System.out.println("Name: " + employee.getName());
    System.out.println("Skills: " + StringUtils.join(employee.getSkills(),
','));
} catch (JsonParseException e) {
```

Document변환

아래 테스트 코드는 String 형태의 xml을 Document타입으로 변환하는 과정이다.

```
@Test
public void testXmltoDocument() {
    try {
        assertEquals("Employee",

XmlUtil.toDocument(str).getDocumentElement().getNodeName());
    } catch (ParserConfigurationException e) {
        fail(e.getMessage());
    } catch (SAXException e) {
        fail(e.getMessage());
    } catch (IOException e) {
        fail(e.getMessage());
    }
}
```

API

XmlUtil은 마샬링, 언마샬링을 위한 유틸로서 다음과 같은 static method로 구성되어 있다.

```
public static String toXml(Object target)
public static <T> T fromXml(Class<T> classpath, String target)
public static Document toDocument(String xmlString)
```

Object target : 마샬링 할 객체를 의미한다.

Class<T> classpath : 언마샬링 대상이 되는 객체를 의미한다.

String target : 언마샬링 할 xml String을 의미한다.

String xmlString : Document 변환을 할 xmlString을 의미한다.

이메일

개요

Mail Util은 SMTP서버를 접속하고 발신자, 수신자, 제목, 내용 등의 데이터를 넣고 메일을 발신하는 유틸이다.

사전에 SMTP 서버가 별도로 구축되어있어야 하며 SMTP 서버의 id, password, port 등을 알고있어야 한다.

관련 클래스

CommonAuth: SMTP 서버에 대한 정보를 갖고 있는 클래스이다.

MailVo: 하나의 메일에 대한 수신자, 발신자, 제목, 내용 등의 데이터를 담고 있는 클래스이다. 내부 static 클래스 Builder를 통해 생성할 수 있다.

MailUtil: CommonAuth와 MailVo를 이용하여 직접 메일을 발신하는 기능을 하는 유틸 클래스이다.

dependency

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-email</artifactId>
</dependency>
```

사용법

설정

SMTP서버 정보를 application.properties파일에 입력한다.

```
##SMTP
chmm.smtp.address=111.222.333.444
chmm.smtp.port=1234
chmm.smtp.id=lotte
chmm.smtp.pw=1234
```

메일서버 인증을 위한 CommonAuth bean을 생성한다.

```
<bean id="commonAuth" class="net.lotte.chamomile.commons.cm.CommonAuth">
    <constructor-arg name="address" value="${chmm.smtp.address}" />
    <constructor-arg name="port" value="${chmm.smtp.port}" />
    <constructor-arg name="id" value="${chmm.smtp.id}" />
    <constructor-arg name="password" value="${chmm.smtp.pw}" />
</bean>
```

CommonAuth bean을 이용해 MailUtil Bean을 생성한다.

```
<bean id="mailUtil" class="net.lotte.chamomile.commons.mail.MailUtil">
    <constructor-arg name="auth" ref="commonAuth" />
    <constructor-arg name="ssl" value="false" />
</bean>
```

예제

MailUtil bean을 이용해 메일을 발신한다.

```
@Resource MailUtil mailUtil;
...
MailVo mail = new MailVo.MailVoBuilder()
            .setFrom(userVo.getUserEmail())
            .setCharset("UTF-8")
            .setSubject(vo.getTitle())
            .setMsg(vo.getBody())
            .addTo(vo.getEmail())
            .build();
try {
    mailUtil.send(mail);
} catch(EmailException e) {
    logger.error(e.getMessage());
}
```

API

빌더를 통해 만들어지는 MailVo구조는 아래와 같다.

필드명	메서드	설명
from	setFrom	발신자지정
cc	addCc(String[] cc)	참조자 추가
	addCc(String cc)	참조자 추가
	subCc(String cc)	참조자 삭제
bcc	addBcc(String[] bcc)	숨은참조자 추가
	addBcc(String bcc)	숨은참조자 추가
	subBcc(String bcc)	숨은참조자 삭제
to	addTo(String[] to)	수신자 추가
	addTo(String to)	수신자 추가
	subTo(String to)	수신자 삭제
subject	setSubject(String subject)	제목 입력
msg	setMsg(String msg)	메시지 입력
charset	setCharset(STring charset)	인코딩 정보 입력(기본UTF-8)
attachPath	addAttachPath(String path)	첨부파일경로 추가
	subAttachPath(String path)	첨부파일경로 삭제
attachName	addAttachName(String name)	첨부파일명 추가
	subAttachName(String name)	첨부파일 경로 삭제
isAttach		자동 입력
htmlMsg	setHtmlMsg(String htmlMsg)	html 메시지 입력

메뉴

개요

캐모마일 프레임워크의 Admin시스템을 통해 등록된 메뉴에 권한을 부여하고 이 메뉴리스트를 불러서 활용할 때 사용하는 API를 말한다.

메뉴 API를 만들게된 목적은 복잡한 권한시스템과 맞물려 메뉴의 계증구조까지 가져와야되는 어려움이 있기때문에 캐모마일 프레임워크를 사용하여 개발을 하는 모든 사용자는 메뉴 API만 호출하면 메뉴리스트를 얻을 수 있어서 개발시간 단축과 번거로움을 해소할 수 있다.

사용법

설정

context-common.xml파일

```
<bean id="menuService" class="net.lotte.chamomile.commons.menu.MenuService">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

예제

```
@Autowired
private MenuService menuService;
…
List<Map<String, Object>> resultMapList =
menuService.findMyMenu(MenuService.ADMIN_MENU, null); //locale이 null이면 현재 설정된 언어로 데이터를 얻어온다.
```

json형태로 변환시 아래와 데이터는 배열로 순서대로 반환되므로 화면단에서 파싱한다.

```
[
    {
        "useYnNm":"사용",
        "useYn":"1",
        "menuSeq":"10",
        "menuHelpUri":null,
        "adminMenuYn":"1",
        "menuId":"menu00000001",     -> 메뉴아이디
        "menuLvlNm":"대메뉴",
        "menuUri":null,              -> 메뉴에 지정된 url
        "upperMenuId":"root",        -> 부모 메뉴아이디
        "leftMenuYn":"1",
        "menuLvl":"0",               -> 메뉴레벨(깊이)
"menuScript":"<i class='fa fa-cubes ...></b></i>", -> 메뉴를 꾸며줄 부가정보(스크립트)
        "menuDesc":null,
        "menuName":"자원관리"         -> 메뉴명
},
{
    "useYnNm":"사용",
    "useYn":"1",
    "menuSeq":"10",
    "menuHelpUri":null,
    "adminMenuYn":"1",
    "menuId":"menu00000002",
    "menuLvlNm":"레벨1",
    ...
```

API

[메뉴목록 조회]

메서드명	파라미터	반환값	설명
findMyMenu	String adminYn, Locale locale	List<Map<String, Object>>	adminYn : MenuService.ADMIN_MENU 또는 MenuService.SERVICE_MENU locale : 적용하고자하는 언어 정보(null일경우 현재 설정언어)

다국어

개요

chamomile프레임워크는 properties파일 외에 데이터베이스 기반의 다국어 처리기능을 제공한다.

기존의 properties파일을 사용하는 방식과 혼용하여 사용이 가능하며 이를 통합하는 유틸리티를 제공한다.

사용자가 직접 만들어놓은 테이블을 이용 할 수 도 있으며 이 경우 어드민의 다국어 관리는 사용할 수 없다.

사용법

설정

[파일기반 다국어 메시지 설정]

```
<bean id="fileMessageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource
">
    <property name="basenames">
        <list>
<!-- 메시지 properties 파일을 등록한다. 등록 하지 않은 경우의 기본값은
message.properties 이다. -->
            <value>classpath:/message/message-common</value>
            <value>classpath:/message/message-user</value>
        </list>
    </property>
    <property name="cacheSeconds">
        <value>60</value>
    </property>
</bean>
```

[데이터베이스 기반 다국어 메시지 설정]

```
<bean id="databaseMessageSource"
        class="net.lotte.chamomile.commons.message.DatabaseMessageSource">
    <property name="dataSource" ref="dataSource"/>
    <property name="tableInfo">
        <props><!--다국어 테이블정보. 입력하지 않을경우 default값으로 처리됨-->
            <!-- 다국어 테이블명-->
            <prop key="table">CHMM_MESSAGE_SOURCE_INFO</prop>
            <!--다국어 코드 컬럼-->
```

```
            <prop key="key">CODE</prop>
            <!--언어코드 컬럼-->
            <prop key="language">LANGUAGE_CODE</prop>
            <!--언어코드 컬럼-->
            <prop key="country">COUNTRY_CODE</prop>
            <!--메시지 컬럼-->
            <prop key="message">MESSAGE</prop>
        </props>
    </property>
    <!--캐시 정보(지정하지 않을 경우 호출할 때 마다 DB조회)-->
    <property name="cacheManager" ref="cacheManager"/>
    <property name="cacheName" value="commonMessageSource"/>
</bean>
```

[다국어 메시지소스 통합]

```
<bean id="messageSource"
class="net.lotte.chamomile.commons.message.IntegratingMessageSource">
    <property name="messageSources"><!-- 설정된 메시지 소스 목록 -->
        <list><!-- 메시지 소스 목록을 나열한다. -->
            <ref bean="databaseMessageSource"/>
            <ref bean="fileMessageSource"/>
        </list>
    </property>
</bean>
```

예제

[java소스에서의 사용]

```
@Autowired
private MessageSource messageSource;
...
messageSource.getMessage("[다국어 코드]", null, null);
//arguments가 존재할경우 (e.g. 총 {0}건이 등록되었습니다.)
messageSource.getMessage("[다국어 코드]", new Integer[]{3}, null);
```

[jsp소스에서의 사용]

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
...
<spring:message code="[다국어 코드]"/>
<!--arguments가 존재 할 경우 (arguments : 적용할 값 들, argumentSeparator :
arguments가 여러 개일 경우 구분해줄 문자) -->
<spring:message code='alert.msg.norequiredmessage' arguments='${alertCnt }'
argumentSeparator=";"/>
```

[js파일, 또는 html파일에서의 다국어사용]

서버에서 다국어 목록 조회

```
jQuery.i18n.properties({
            name:'message-common',
            path:admin.context + '/i18n/all/',
            mode:'map',
            language:'NA',
            callback: function () {
                init();
            }
        });
```

다국어 사용

```
jQuery.i18n.prop("[다국어 코드]");
//arguments가 존재할경우. 다국어 코드 다음 파라미터로 해당하는 개수만큼 추가한다.
jQuery.i18n.prop("[다국어 코드]", 1, 50);
```

자동 페이징

개요

페이징 모듈은 페이징이 필요한 리스트 조회를 별도의 페이징 쿼리 없이 사용 가능하게 하는 모듈이다.

MyBatis의 Interceptor를 사용하여 페이징 쿼리를 생성한다.

페이징을 위해 전체 count를 조회하는 쿼리가 포함되어 있어 대용량 데이터 조회에는 적합하지 않다.

성능저하가 우려되는 쿼리에는 전체 count를 조회하지 않고 페이지 번호 없이 사용하는 방법도 지원한다.

5개의 DB(Oracle, Tibero, MySQL, MariaDB, MSSQL)를 지원한다.

사용법

설정

context-*.xml파일(e.g. context-datasource.xml)에서mybatis설정파일 위치를
지정한다.

```
<bean id="sqlSessionFactory"
class="net.lotte.chamomile.core.mybatis.RefreshableSqlSessionFactoryBean">
...
<property name="configLocation" value="classpath:sql/config/sqlMapConfig.xml" />
...
</bean>
```

mybatis 설정파일에서 plugin항목을 추가한다.

```
<plugins>
    <plugin
interceptor="net.lotte.chamomile.commons.pageable.MybatisPageableInterceptor"/>
</plugins>
```

사용법

[Pageable 객체 생성]

Pageable 객체를 생성하는 방법은 2가지로 다음과 같다.

```
Pageable(int page, int size)
Pageable(int page, int size, boolean countable)
```

생성자의 인자는 다음과 같다.

- int page : 요청하고자하는 페이지의 번호를 의미한다. 0부터 시작한다.
- int size : 한 페이지가 가지는 row의 개수를 의미한다.
  key-value 쌍에서 해당 key에 대한 value를 의미.
- boolean countable : total count의 쿼리 여부. True시 total count를 질의함.

객체를 countable 인자 없이 생성시 내부적으로 true값이 부여된다.

countable을 false값으로 생성시 total count는 질의하지 않고 값은 0이 되며, 전체 쿼리 내용 중 size만큼만 조회하게 된다.

Pageable객체 생성은 아래와 같이 한다.

```
Pageable pageRequest = new Pageable(0, 10);
```

[DAO에 인터페이스 추가]

DAO에 select 쿼리를 호출하는 함수에 pageable 인자를 받을 수 있도록 추가 한다.
리턴 타입은 Page<T>형태로 내부에 리스트형태로 T객체를 저장 한다.

```
원형: List<TestVO> testListData(TestVO vo) throws Exception;
추가: Page<TestVO> testListData(TestVO vo, Pageable pageRequest) throws
Exception;
```

[데이터 조회]

DAO를 호출 하는 쪽에서 pageable객체를 인자로 전달한다.

```
Pageable pageRequest = new Pageable(0, 10);
Page<TestVO> resultpage = testDAO.testListData(vo, pageRequest);
```

[조회된 데이터 추출]

반환된 객체에서 전체 데이터 개수와 조회된 목록을 추출한다.

```
resultpage.getTotalElements();
resultpage.getContent();
```

API

Page객체에서 사용가능한 메서드는 아래와 같다.

메서드명	반환값	설명
getNumber()	Int	요청한 페이지숫자를 리턴
getSize()	Int	페이지당 레코드 수를 리턴
getTotalPages()	Int	총 페이지 수를 리턴
getNumberOfElements()	Int	현재 페이지의 레코드 수를 리턴
getTotalElements()	Int	총 레코드 수를 리턴
hasPreviousPage()	Boolean	이전 페이지가 있는지 여부
isFirstPage()	Boolean	첫 페이지인지 여부
hasNextPage()	boolean	다음 페이지가 있는지 여부
isLastPage()	boolean	마지막 페이지 여부
nextPageable()	Pageable	다음 요청 Pageable 객체 리턴
previousPageable()	Pageable	이전 요청 Pageable 객체 리턴
getContent()	List<T>	조회 결과 리턴
hasContent()	boolean	조회 결과가 있는지 여부

프로퍼티

개요

.properties는 응용 프로그램의 구성 가능한 파라미터들을 저장하기 위해 사용하는 파일로, key=value 형식의 데이터를 저장한다.

DatabaseProperties는 파일로 관리되는 property를 DB기반으로 관리하기 위해 제공된다. Table에 key와 value 데이터를 입력 후 API 호출 시 key에 맞는 value를 반환하도록 구성되어 있다.

DatabasePropertyUtil은 Apache 에서 제공하는 Configuration을 사용하여 DB기반의 프로퍼티를 처리한다. 또한 캐싱 처리를 위해 CacheManager의 설정이 필요하다.(캐시 가이드 참조)

dependency

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-configuration2</artifactId>
</dependency>
```

사용법

설정

```
<bean id="databasePropertyUtil"
class="net.lotte.chamomile.commons.property.DatabasePropertyUtil">
    <constructor-arg>
        <ref bean="databaseProperties"/>
    </constructor-arg>
</bean>
<bean id="databaseProperties"
class="net.lotte.chamomile.commons.property.DatabaseProperties">
    <property name="dataSource" ref="dataSource"/>
    <property name="table" value="${chmm.property.table}"/>
    <property name="keyColumn" value="${chmm.property.key.column}"/>
    <property name="valueColumn" value="${chmm.property.value.column}"/>
    <property name="cacheManager" ref="cacheManager"/>
    <property name="cacheName" value="commonProperty"/>
</bean>
```

DatabaseProperties 의 설정 항목은 다음과 같다.

- dataSource : database 설정 빈을 주입
- table : 프로퍼티 테이블 명을 지정
- keyColumn : 프로퍼티 테이블의 key값을 저장할 컬럼을 지정
- valueColumn : 프로퍼티 테이블의 value값을 저장할 컬럼을 지정
- cacheManager : 캐시처리를 위한 cacheManager를 주입
- cacheName : 구성된 캐시를 가져오기 위해 cacheName을 지정

* 캐시 구성을 위한 방법은 캐시 개발가이드를 참조한다.

예제

```
@Resource
private DatabasePropertyUtil databaseProprertyUtil;
...
databaseProprertyUtil.getProperty("test.property");
```

API

메서드명	파라미터	반환값	설명
getProperty	String key	String	key에 해당하는 값을 가져온다.
addProperty	String key, String value		DB에 key/value를 추가한다.
setProperty	String key, String value		key에 해당 하는 값을 수정한다.
clearProperty	String key		key에 해당하는 값을 삭제한다.

Encryption

개요

평문에대해 단방향, 양방향 암호화 알고리즘을 제공하고 비밀키, 공개키알고리즘을 제공한다.

OWASP 10대 위험요소중에 하나인 크로스사이트 스크립팅을 방지 해준다.

암호화 제공 방식 : AES256(양방향 대칭키), ARIA(양방향 대칭키), RSA-OAEP(양방향 비대칭키), SHA256(단방향)

사용법

XSS필터

[설정]

web.xml

```
<filter>
    <filter-name>XSS</filter-name>
    <filter-class>net.lotte.chamomile.commons.security.XSSFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>XSS</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

파일업로드시 multipart외에 다른 항목들에 대해 필터링을 적용하고자 하는경우 아래의 설정을 xss 필터 설정 전에 추가해준다.

```
<filter>
    <filter-name>multipartFilter</filter-name>
    <filter-
class>org.springframework.web.multipart.support.MultipartFilter</filter-class>
    <!-- multipartResolverBeanName을 별도로 셋팅해주지 않고 기본값인
filterMultipartResolver 으로 사용하도록 한다.
인터넷예제에는 init param태그에 별도로 셋팅해주는 부분이 있는데 이 경우 업로드가 정상적으로
되지 않는 케이스가 발생할 수 도 있다. -->
</filter>
<filter-mapping>
    <filter-name>multipartFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

tomcat 사용시 위와 같은 설정에도 파일 업로드가 정상적으로 되지 않거나 multipart 외에 다른 필드의 xss처리가 정상적으로 되지 않는 다면 아래 설정을 추가한다.

서버쪽 context.xml파일이나 META-INF 내의 context.xml파일에서 아래 부분을 추가한다.

(allowCasualMultipartParsing="true")

```
...
<Context ... allowCasualMultipartParsing="true" >
...
```

Servlet 3.0에서는 apache commons에서 많이 사용되던 파일업로드를 @MultipartConfig를 통해 업로드를 지원하고 있다. 그에 따라 HttpServletRequest.getPart나 HttpServletRequest.getParameter 에 대한 분석이 제한됨에따라 tomcat이를 분석할 수 있도록 설정해주는 옵션이다.

암호화

[AES256]

key의 길이는 총 16자(32bytes)이다.

키값이 길다면 16자까지만 값을 사용하지만 길이가 짧다면 exception이 발생한다.

```
private String key = "1234567890!@#$%^";
private String str = "ldcc롯데정보통신";
private String encoded;;
private AES256Util aes;
...
aes = new AES256Util(key);
//암호화
encoded = aes.encode(str);
//복호화
String decoded = aes.decode(encoded);
```

[ARIA]

```
private String key = "1234567890!@#$%^";
private String str = "ldcc롯데정보통신";
private String encoded;;
private ARIACryptoUtil aria;
...
aria = new ARIACryptoUtil(key);
//암호화
encoded = aria.encode(str);
//복호화
String decoded = aria.decode(encoded);
```

[RSA-OAEP]

비대칭키 알고리즘의 경우 암호화는 공개키로 하고 복호화는 개인키로 하게된다.

```
private String key = "1234567890!@#$%^";
private String str = "ldcc롯데정보통신";
private String encoded;
private RSAOAEPCryptoUtil rsa;
private byte[] publicKey;
private byte[] privateKey;
...
rsa = new RSAOAEPCryptoUtil();
//개인키(비밀키) 획득
privateKey = rsa.getPrivateKey();
//공개키 획득
publicKey = rsa.getPublicKey();
//암호화
encoded = RSAOAEPCryptoUtil.encode(str, publicKey);
//복호화
String decoded = RSAOAEPCryptoUtil.decode(encoded, privateKey));
```

[SHA256]

단방향 암호화의 경우 복호화는 하지 않는다. 일반적으로 암호화된 값이 같은지 다른지 따라 데이터 유효성을 검증한다.

```
String str = "ldcc롯데정보통신";
//암호화된 해시값을 유추하기 어렵도록 salt값을 추가한다.(16bytes이면 충분하다)
String decoded = SHA256Util.encode(str,"salt");
```

String Util

개요

StringUtil은 String 제어, XSS Encoding & Decoding 등의 기능을
수행한다.

dependency

```
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
</dependency>
```

API

StringUtil

메서드명	파라미터	반환값	설명
allIndexOf	String source, String target	List<Integer>	source의 substring 중 target과 일치하는 substring 의 index들을 반환
indexOf	String source, String target	int	source의 substring 중 target과 앞에서부터 처음 일치하는 index를 반환
ellipsis	String source, int MAXLEN	String	source의 길이가 MAXLEN을 넘어가는 경우 자름
ellipsis	String source, int MAXLEN, String limit	String	str의 길이가 MAXLEN 을 넘어가는 경우 limit 으로 치환
encodeXSS	String source	String	XSS Encoding (< 와 > 를 치환)

메서드명	파라미터	반환값	설명
decodeXSS	String source	String	XSS Decoding(< 와 > 를 치환)
camelToUnderscore	String str	String	camel 양식의 문자열을 underscore(_) 양식으로 변환.
splitByLength	String str, int cutLength	List<String>	주어진 문자열을 지정된 길이 만큼 잘라서 List로 반환한다.
appendToString	String... args	String	String들을 하나의 String으로 반환한다.
match	String src, String regex	boolean	정규식패턴과 일치하는지 판단한다.
ellipsisByByte	String src, int byteLen, String suffix	String	String을 원하는 크기(byte 단위)로 줄여 마지막 접미사를 붙여 반환한다. (접미사도 length에 포함) 한글의 경우 중간에 절단이 이루어 졌을경우 깨지는 것을 방지하기위해 완성형으로 제공한다.
abbreviate	String src, int maxWidth, String enc, String suffix	String	String을 원하는 길이(byte 단위)로 줄여 마지막 접미사를 붙여 반환한다. 접미사의 길이도 길이에 포함되며, 원하는 크기가 접미사의 길이보다 작으면 BaseRuntimeException 던진다. 주어진 인코딩 기준으로 byte[]를 얻은 후 이를 byte 단위로 나누며, 나누는 과정에 깨지는 문자는 버린다.

UUID

개요

UUID Util은 범용 고유 식별자를 의미하여 Java API 에서128bit로 구성되어있다.

주로 난수 생성이 필요할 때 사용하며, 입력값을 기준으로UUID 를 생성하게 되면 매번 동일한 UUID 가 생성된다.

API

```
UuidUtil
```

메서드명	파라미터	반환값	설명
getUuid		String	UUID 반환
getUuidOnlyString		String	UUID 반환('-'제거)

메서드명	파라미터	반환값	설명
getUuid	String input	String	input을 토대로 UUID 생성
getUuidOnlyString	String input	String	input을 토대로 UUID 생성 ('-'제거)

Cache

개요

어플리케이션이나 시스템에서 클라이언트에 대한 요청을 처리 할 때 자주 사용되는 데이터가 있을 수 있다. 이러한 데이터가 데이터베이스에서 가져올 경우 시스템 성능에 영향을 미칠 수 있다.

Chamomile Framework는 캐시를 지원하므로 어플리케이션에서 자주 사용되는 데이터를 캐싱하여 성능을 개선 할 수 있다.

주로 Update가 자주 일어나지 않으며 Read가 빈번하게 일어나는 데이터를 캐싱하면 어플리케이션의 속도를 개선 할 수 있다.

무분별하고 적절하지 못한 데이터 캐싱하면 예상 외의 결과가 발생 할 수 있기 때문에 캐싱할 데이터를 신중하게 선별해서 개발해야 한다. Chamomile Framework는 어플리케이션에 캐시를 투명하게 추가할 수 있도록 지원한다.

스프링에서 제공하는 CacheManager를 사용하며 추상화를 지원한다.

실제 캐싱한 데이터의 저장이 일어나는 캐시 구현체는 별도의 설정을 통해 변경 가능하며, Chamomile Cache에서는 캐시 구현체를Ehcache와 Redis로 선정하여 가이드 한다. (기본 Ehcache)

CacheManager 객체를 직접 사용하는 방법과 CacheManager Abstraction을 사용하는 방법 두 가지 개발 방법을 제공한다.

추상화를 지원하기 때문에 위의 두가지 방법 중 어떤 방법으로 개발하더라도 캐시 구현체의 변경에 따른 소스 변경은 일어나지 않는다.

Ehcache

Ehcache는 성능을 향상시키고 데이터베이스를 off-load하며 확장 성을 단순화하는 오픈 소스 표준 기반 캐시이다.

강력하고 검증 된 기능을 갖추고 널리 사용되는 다른 라이브러리 및 프레임워크와 통합되므로 가장 널리 사용되는 Java 기반 캐시이다.

다양한 캐시 정책 설정이 가능하다. (xml)

기본적으로 JVM Heap 메모리에 캐싱한 데이터가 적재된다.

때문에 별도의 네트워크 트래픽이 발생하지 않는다. (Clustering 제외)

Redis

메모리 위에서 동작하는 Key/value 저장소인 Redis는 NoSQL DBMS로 분류되며 동시에 Memcached와 같은 인메모리 솔루션으로 분리된다.

다중 서버 구성 가능하다. (샤딩, 레플리케이션 등)

데이터베이스로 사용될 수 있으며, Cache로도 사용될 수 있는 기술이다.

성능은 머신 스펙에 따라 다르나 초당 2만 ~ 10만회 수행가능하다.

Ehcache VS Redis

어플리케이션 특징과 인프라의 구성 및 캐싱 데이터의 크기에 따라 적절한 캐시 구현체를 선택한다. (Ehcache or Redis)

Java Heap 영역에서 데이터 조회가 가능한 Local Cache(이하 Ehcache)가 네트워트 트래픽이 발생하는 Global Cache(이하 Redis)에 비해 상대적으로 성능이 좋을 수 있다.

하지만 서비스 확장을 위해 WAS 인스턴스의 수가 증가할수록, 혹은 캐시에 저장되는 데이터 크기가 커질수록 Redis가 Ehcache에 비해 효과적이다.

Ehcache와 Redis 모두 이중화 구성에 대한 캐시 Clustering을 지원한다.

Chamomile Framework는 캐시 추상화를 제공하기 때문에 어떠한구현체를 선택하더라도 캐시를 사용한 개발 방법은 같다. 즉, 캐시 구현체를 변경하더라도 소스 변경은 없다.

캐시 구현체	특 징	적합한 어플리케이션
Ehcache	데이터 캐싱 속도가 빠르다 캐싱된 데이터의 크기에 따라 서버 인스턴스의 메모리 사용량이 커진다.	WAS 인스턴스 수가 많지 않은 시스템 WAS의 Heap 메모리가 많이 확보된 시스템
Redis	상대적으로 큰 데이터를 캐싱 할 수 있다. 네트워크 트래픽이 발생해 속도가 저하 될 수 있다. 이중화를 통해 장애에 유연하게 대응 가능하다.	Redis가 구동될 수 있는 인프라가 확보된 시스템 캐시 동기화에 대한 데이터 싱크가 중요한 시스템

dependency

```
<!-- ehcache -->
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache</artifactId>
</dependency>
<!-- spring data redis -->
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-redis</artifactId>
</dependency>
<!-- redis clients jedis -->
<dependency>
    <groupId>redis.clients</groupId>
    <artifactId>jedis</artifactId>
</dependency>
```

사용법

설정

[context-*.xml파일 설정]

e.g. context-cache.xml

```
xmlns:cache="http://www.springframework.org/schema/cache"
xsi:schemaLocation="...
http://www.springframework.org/schema/cache
http://www.springframework.org/schema/cache/spring-cache-4.3.xsd"
...
<!-- Annotation 기능 활성화 -->
<cache:annotation-driven cache-manager="cacheManager" />
```

ehcache 설정

스프링에서 제공하는 CacheManager를 통해 구현체(EhCacheCacheManager)를 등록한다.(location을 지정하지 않으면 기본적으로 classpath의 ehache.xml)

```
<!-- EhCache -->
<bean id="cacheManager"
class="org.springframework.cache.ehcache.EhCacheCacheManager" p:cache-manager-
ref="ehcache"/>
<bean id="ehcache"
class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean" p:config-
location="classpath:config/cache/ehcache.xml"/>
```

[ehcache.xml]

ehcache.xml 파일을 생성하여 캐시에 대한 정책을 세운다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="ehcache.xsd" updateCheck="true"
        monitoring="autodetect"
        dynamicConfig="true">

<diskStore path="java.io.tmpdir" />
<!-- 공통 코드 캐시 -->
<cache name="commonCodeCache"
    maxEntriesLocalHeap="100000"
    maxEntriesLocalDisk="10000"
    eternal="false"
    diskSpoolBufferSizeMB="20"
    timeToIdleSeconds="300"
    timeToLiveSeconds="600"
    memoryStoreEvictionPolicy="LFU"
    transactionalMode="off">
    <persistence strategy="localTempSwap" />
</cache>
```

위의 설정에 대한 내용을 간략히 확인해보면 아래와 같다.

- samplecache라는 캐시저장소를 생성
- 메모리에 생성될 최대 객체수는 100,000개
- 디스크에 저장될 객체의 수는 10,000개
- 저장된 캐시를 제거한다.
- 300초 동안 캐시를 사용하지 않으면 삭제
- 600초 뒤에 캐시를 삭제

[Ehcache의 세부(정책) 설정 방법]

설 정	내 용
name	캐시 이름을 설정한다. 캐시를 식별하는데 사용되며 고유해야 한다.
maxEntriesLocalHeap	메모리에 생성 될 최대 객체 수를 설정한다. 0 = 제한 없음.
maxEntriesLocalDisk	디스크(DiskStore)에서 유지 관리 할 최대 객체 수를 설정한다. 기본값은 0이며 제한 없음.
eternal	캐싱 데이터가 영속성 여부를 설정한다. true이면 timeToIdle과 timeToLive 속성은 무시된다.
timeToIdleSeconds	캐싱 데이터의 유휴 시간을 설정한다. 해당시간동안 캐시를 사용하지 않으면 삭제된다. eternal 속성이 false이여야 한다. 기본값은 0이며 삭제되지 않는다.
timeToLiveSeconds	캐싱 데이터의 만료 시간을 설정한다. 해당시간이 지나면 캐시간이 삭제된다. eternal 속성이 false이여야 한다. 기본값은 0이며 삭제되지 않는다.
diskExpiryThreadIntervalSeconds	디스크(DiskStore)에 저장된 캐시들을 정리하기 위한 작업의 실행 간격 시간을 설정한다. 기본값은 120초이다.
diskSpoolBufferSizeMB	버퍼 영역에 캐시를 저장 한 다음 디스크에 비동기 적으로 기록한다. 기본 크기는 30MB이다. OutOfMemory 오류가 발생하면 설정한 크기를 낮추는 것이 좋다.
clearOnFlush	캐시에서 flush ()가 호출 될 때 MemoryStore가 지워 져야 하는지 여부를 설정한다. 기본값은 true이며 즉 MemoryStore는 삭제된다.
memoryStoreEvictionPolicy	maxEntriesLocalHeap 설정값에 실행된다. 기본 설정은 LRU이다. LRU - 가장 오랫동안 사용하지 않은 데이터 부터 제거 FIFO - 먼저 입력된 것 부터 제거 LFU - 사용빈도가 가장 적은 데이터부터 제거

***Redis*설정**

스프링에서 제공하는 CacheManager에 대한 통해 구현체(RedisCacheManager)를 등록한다.

```
<bean id="cacheManager"
class="org.springframework.data.redis.cache.RedisCacheManager">
        <constructor-arg name="template" ref="redisTemplate"/>
```

jedisConnFactory와 redisTemplate를 통해서 Redis 접속 정보를 등록한다. (별도의 접속 정보를 등록하지 않으면 기본적으로 127.0.0.1:6379 이다.)

```
<!-- Jedis Connection Factory -->
<bean id="jedisConnFactory"
class="org.springframework.data.redis.connection.jedis.JedisConnectionFactory"
      p:usePool="${chmm.redis.usePool}" p:hostName="${chmm.redis.hostName}"
p:port="${chmm.redis.port}" />

<!-- String Serializer -->
<bean id="stringRedisSerializer"
class="org.springframework.data.redis.serializer.StringRedisSerializer"/>

<!-- Redis Template -->
<bean id="redisTemplate"
class="org.springframework.data.redis.core.RedisTemplate" p:connectionFactory-
ref="jedisConnFactory" p:keySerializer-ref="stringRedisSerializer" />
```

그 외 제공되는 캐시 구현체

Chamomile Cache에서는 Spring에서 제공하는 CacheManager추상화를 사용하기 때문에 앞서 설명한 Ehcache와 Redis 외에 추가적인 캐시 구현체 설정이 가능하다. (SimpleCache, Guava)

시스템의 특정 이슈로 인해 Ehcache와 Redis의 사용이 불가한 경우 아래와 같은 구현체로 캐시 사용이 가능하지만 권장하지 않는다.

SimpleCache와 Guava는 별도의 라이브러리를 요구하지 않는다.

SimpleCache 캐시 사용시 bean 등록 예시

```
<bean id=”cacheManager”
      class=”org.springframework.cache.support.SimpleCacheManager”>
    <property name=”caches”>
        <set>
            <bean
 class=”org.springframework.cache.concurrent.ConcurrentMapCacheFactoryBean”
             name=”samplecache”/>
        </set>
    </property>
</bean>
```

Guava 캐시 사용시 bean 등록 예시

```
<bean id="cacheManager"
class="org.springframework.cache.guava.GuavaCacheManager" />
```

Annotaion기반 사용법

Chamomile Cache에서는 Spring에서 제공하는 CacheManager Abstraction을 통한 어노테이션 방법과 CacheManager 직접사용 방법으로 개발이 가능하다.

Chamomile Cache에서는 어노테이션 방법을 권장하며 Spring의 트랜잭션 어노테이션과 같이 코드의 변화를 최소화하며 메서드 레벨로 캐시 처리가 가능하다.

어노테이션 목록은 아래와 같다.

- **@Cacheable** : 캐시 저장 및 엑세스.

- **@CacheEvict** : 캐시를 제거한다.
- **@CachePut** : 메서드 실행을 방해하지 않고 캐시를 업데이트한다.
- **@Caching** : 메서드에 적용될 다중 캐시를 적용한다.

@Cacheable

@Cacheable 어노테이션은 캐시 할 메서드를 지정하는데 사용한다.

@Cacheable 어노테이션이 있는 메소드가 호출되면, 이미 실행된 메서드라면 여러 번 실행하지 않고 캐시 된 스토리지에서 결과를 반환한다.

아래의 코드는 가장 간단한 형식 @Cacheable을 통한 캐시 사용 방법이다. 메서드에 어노테이션과 함께 연결한 캐시의 이름을 지정한다.

```
@Cacheable(value = " samplecache " )
Public Sample getById(int id) {...}
```

위의 코드에서 메서드 getById는 samplecache 캐시와 연결된다. 메서드가 호출 될 때마다 캐시를 검사하여 이미 실행되었고 반복 될 필요가 없는지 확인한다.

하나의 캐시만 선언하지 않고 어노테이션에서는 여러 개의 이름을 지정하여 두 개 이상의 캐시를 사용하면 메서드를 실행하기 전에 각 캐시를 검사한다. 적어도 하나의 캐시에 히트가 발생하면 연결된 캐시의 값이 반환된다.

```
@Cacheable(value = "samplecache1, samplecache2")
public Sample getById(int id) {...}
```

캐시는 기본적으로 key – value 저장소이기 때문에 캐시 된 메서드를 호출 할 때마다 캐시 액세스에 적합한 key로 변환 해야한다.

key의 명시적 선언은 SpEL 사용하여 가능하며, 명시적으로 key를 지정하지 않았다면 메서드의 인자를 key로 사용한다.

아래의 코드는 SpEL를 사용하여 명시적인 key선언의 예이다.

```
@Cacheable(value = "samplecache", key = "#sample")
public Sample getById(Sample sample) {...}

@Cacheable(value = "samplecache", key = "#sample.id")
public Sample getById(Sample sample) {...}

@Cacheable(value = "samplecache", key = "#sample.name")
public Sample getById(Sample sample) {...}
```

개발 요건에 따라 메서드에 항상 캐싱되는 것이 적합하지 않을 수 있다. 이럴 경우 condition 속성을 통해 true나 false가 되는 SpEL 표현식을 받는 conditional 파라미터로 캐시 적용 유무를 선택 할 수 있다. SpEL 표현식에 의해 true이면 메서드를 캐시하고 false이면 메서드가 캐시 되지 않는다.

아래의 코드는 파라미터인 id가 200일 경우 캐시하지 않는다.

```
@Cacheable(value = "samplecache", key = "#id", condition="#id!=200")
public Sample getById(int id) {...}
```

아래의 코드는 파라미터인 id가 300 이상일 경우 캐시한다.

```
@Cacheable(value = "samplecache", key = "#id", condition="#id>300")
public Sample getById(int id) {...}
```

@CachePut

메서드의 실행에 영향을 주지 않고 캐시를 업데이트해야하는 경우 @CachePut 어노테이션을 사용할 수 있다.

메서드는 항상 실행(조건이 맞으면)되며 그 결과는 캐시에 저장한다.

@Cacheable와 같은 옵션을 지원하며 메서드 흐름 최적화가 아닌 캐시 생성에 사용한다.

아래의 코드는 id를 키로 메서드의 결과를 캐시에 저장하며, 같은 key(id)로 호출되더라도 @Cacheable과는 다르게 메서드가 실행된다.

```
@CachePut(value = "samplecache", key = “3_4_5")
public Sample updateSample(Sample sample) {...}
```

> 주의 : @CachePut과 @Cacheable서로 다른 동작을 수행하기 때문에 같은 방법으로 어노테이션을 설정하는 것은 일반적으로 권장하지 않는다. @Cacheable은 캐시를 사용하여 메서드 실행을 건너 뛰게 되지만 @CachePut은 캐시 업데이트를 실행하기 위해 실행을 강제한다. 이로 인해 예상하지 못한 동작이 발생할 수 있다.

@CacheEvict

@CacheEvict 어노테이션은 캐시를 제거한다. 즉, 캐시에서 데이터를 제거하는 트리거로 동작하는 메서드다.

다른 캐시 어노테이션과 마찬가지로 @CacheEvict 또한 동작할 때 영향을 미칠 하나 이상의 캐시를 지정해야 한다.

또한 @CacheEvict에서도 key나 condition을 지정해야 할 수 있지만 key에 매핑된 하나의 엔트리 가 아니라 전체를 제거하기 위해서는 allEntries 속성을 추가로 사용할 수 있다.

아래의 코드는 samplecache 캐시의 모든 엔트리를 삭제의 예이다

```
@CacheEvict(value = "samplecache", allEntries = true)
public void clearCache() {...}
```

@CacheEvict를 다른 어노테이션과 다르게 void를 메서드에 적용 할 수 있다.
메서드가 트리거로 동작하므로 리턴값은 무시한다. 이는 캐시에 데이터를 넣거나 갱신해서 그 결과가 필요한 @Cacheable과 다른 점이다.

@Caching

@Caching 동일한 메소드에서 @CacheEvict 또는 @CachePut과 같은 유형의 여러 어노테이션을 지정하려는 경우 유용하게 사용 할 수 있다.

같은 key를 사용하여 같은 아이템을 포함하는 두 개의 캐시가 있다고 가정 해 보자.
두 캐시에서 모두 아이템을 제거하려는 경우는 간단하다.

```
@CacheEvict(value = {"cache1", "cache2"}, key = "#sample.id")
public void refreshSample(Sample sample) {...}
```

만약 다른 key를 사용한다면 아래와 같이 @Caching 어노테이션을 통해서 여러유형의 어노테이션을 지정할 수 있다.

```
@Caching(evict = {
@CacheEvict(value = "cache1", key = "#sample.id"),
@CacheEvict(value = "cache2", key = "#sample.name")
})
public void refreshSample(Sample sample) {...}
```

ChacheManager를 이용한 사용법

CacheManager을 직접 사용하면 일반적인 컬렉션(HashMap 등)과 같이 개발이 가능하다.

CacheManager와 Cache 클래스에서 제공하는 주요 API는 아래와 같다.

- CacheManager.getCache : 지정한 캐시를 리턴
- CacheManager.getCacheNames : 등록된 모든 캐시의 이름을 리턴
- Cache.put : 지정한 키로 Object를 캐시에 적재
- Cache.get : 지정한 키에 매핑하는 캐시아이템을 리턴
- Cache.evict : 지정한 키에 매핑된 캐시아이템을 삭제
- Cache.clear : 적재된 캐시아이템을 모두 삭제
- Cache.getName : 캐시명을 리턴

> 참고 : Ehcache와 Redis의 캐시구현체와 관계 없이 같은 인터페이스로 작동하기 위해서는 Ehcache패키지가 아닌 Spring패키지 제공하는 CacheManager와 Cache 클래스를 임포트 해야 한다.

```
import org.springframework.cache.CacheManager;
import org.springframework.cache.Cache;
```

CacheManager를 직접 사용하기 위해서는 의존성주입(@Autowired)을 통해 CacheManager 객체를 가져와야 한다.

```
@Autowired
private CacheManager cacheManager;
```

CacheManager.getCache(String name)

cacheManager의 getCache 메서드를 통해 캐시를 가져온다. 여기서 말하는 캐시는 ehcache.xml에 정의한 캐시의 이름이다.

Redis의 경우 별도의 설정 파일이 없으므로 선언한 이름으로 캐시가 생성된다.

CacheManager의 직접 사용을 위해 반드시 호출해야 하는 API이다.

```
Cache cache = cacheManager.getCache("samplecache");
```

CacheManager.getCacheNames()

Ehcache의 경우 ehcache.xml에 등록되어 있는 캐시의 이름을 가져온다.

Redis는 별도의 지정한 캐시가 없으므로 getCache를 통해 생성된 모든 캐시의 이름을 가져온다.

```
cacheManager.getCacheNames();
```

Cache.put(Object key, Object value)

앞서 호출한 getCache를 통해 리턴된 Cache 객체로 캐시아이템을 캐시에 저장 할 수 있다.

아래의 코드는 put을 통한 캐시아이템 저장 예시 이다.

```
// String “key1”를 키로 Sample 객체를 캐시에 저장
cache.put("key1", new Sample(100, "Developer_1", "java"));
// String “key2”를 키로 “Devleoper_2”를 캐시에 저장
cache.put("key2", "Developer_2");
// Integer 100을 키로 현재 시간을 캐시에 저장
cache.put(100, new Date().toString());
```

> 참고 : 캐시 업데이트는 별도의 API가 있지 않으며, 업데이트를 위한 새로운 캐시아이템을 같은 key로 put을 하게 되면 기존의 캐시는 삭제되고 새로 등록한 캐시아이템으로 업데이트 된다.

Cache.get(Object key)

캐시에 저장된 캐시아이템을 key를 통해 가져올 수 있다.

get을 통해 호출되는 리턴 타입은 ValueWrapper클래스이며,get을 한번 더 호출하여 실제 반환 값을 리턴 받는다.

아래의 코드는 get을 통한 캐시아이템 호출 예시이다.

```
cache.get("key1").get();
cache.get("key2").get();
cache.get(100).get();
```

만약 호출하는 key에 대해 저장된 캐시가 없다면 NullPointerException이 발생한다.

Cache.evict(Object key)

키에 매핑된 캐시아이템을 삭제한다.

해당 키에 저장된 캐시아이템이 없더라도 에러를 발생하지 않는다.

아래의 코드는 evict을 통한 특정 키에 대한 캐시아이템 삭제 예시이다.

```
cache.evict("key1");
cache.evict("key2");
cache.evict(100);
```

Cache.clear()

캐시에 있는 모든 캐시아이템을 삭제한다.

아래의 코드는 clear를 통한 캐시아이템 전체 삭제 예시이다.

```
cache.clear();
```

Cache.getName()

현재 연결된 캐시의 이름을 가져온다.

아래의 코드는 getName을 통한 캐시 이름 호출 예시이다.

```
cache.getName();
```

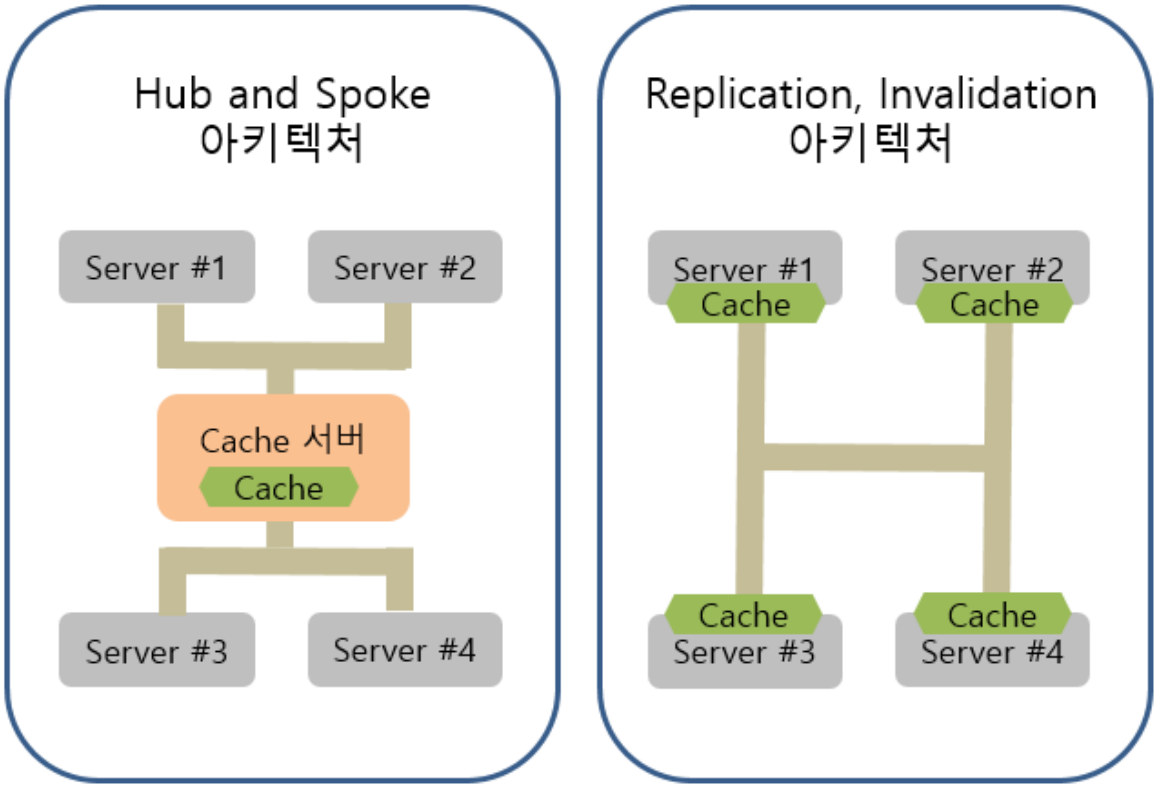
Cache Clustering

캐시 클러스터링은 이중화 된 서버간의 캐시 동기화를 의미한다.

Chamomile Cache에서는 캐시 구현체(Ehcache/Redis)에 상관 없이 캐시 클러스터링을 지원하고 있다.

캐시 클러스터링을 구성하면 연결된 모든 서버간의 캐시 및 엘리먼트가 동기화 되며 요건에 따라 보다 효율적으로 시스템을 구성 할 수 있다.

Chamomile Cache에서는 클러스터링을 알아보기 이전에 아래와 같은 두가지 형식의 분산 캐시 아키텍처 모델을 확인한다.

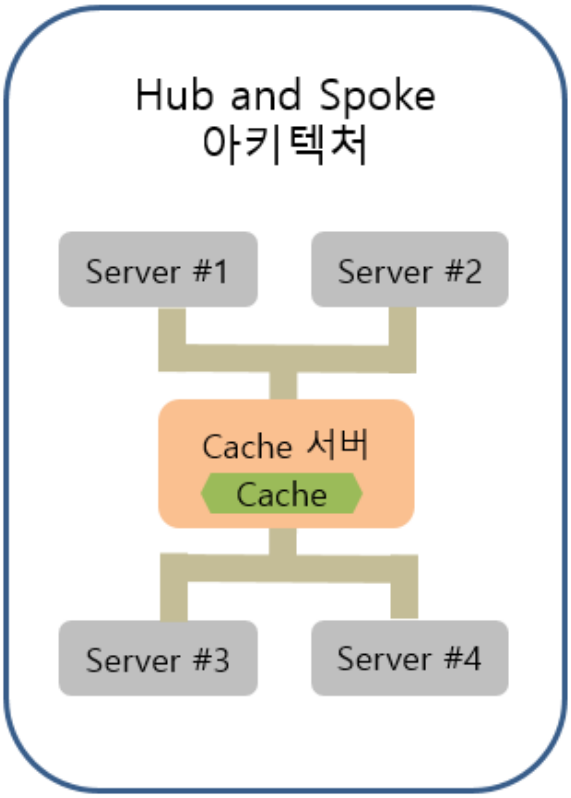


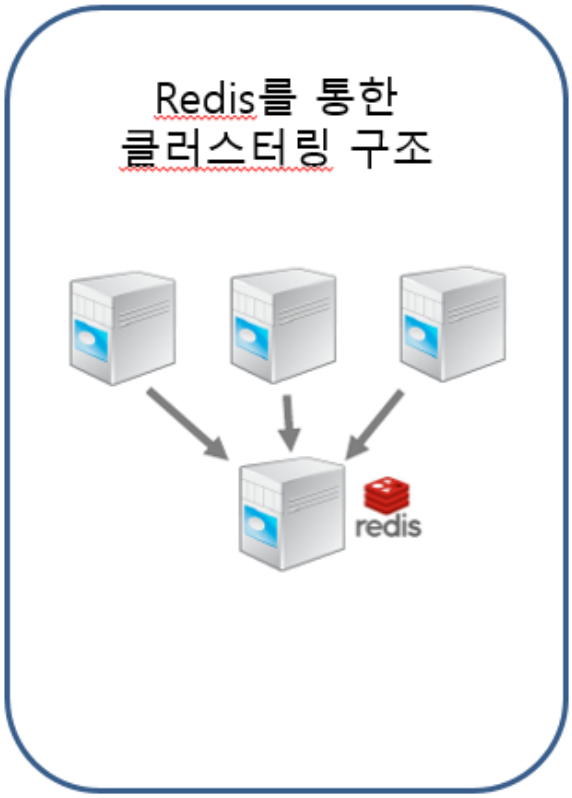
Redis

Redis는 앞서 살펴본 분산 캐시 아키텍처 모델 중 Hub and Spoke 모델 구조이다.

각 서버에서 접근하는 모든 캐시는 중앙 서버(Redis)에서 관리하며 이를 통해 모든 서버간의 캐시가 클러스터링 된다.

때문에 캐시 구현체를 Redis로 하였다면 별도의 캐시 클러스터링 설정 없이 클러스터링에 참여하는 모든 서버는 같은 Redis 서버와 연결되면 된다.





Ehcache

Ehcache는 앞서 살펴본 분산 캐시 아키텍처 모델 중 Replication, Invalidation 모델 구조이다.

이는 각 캐시 서버들간에 자체적으로 클러스터링 하는 구조이다.

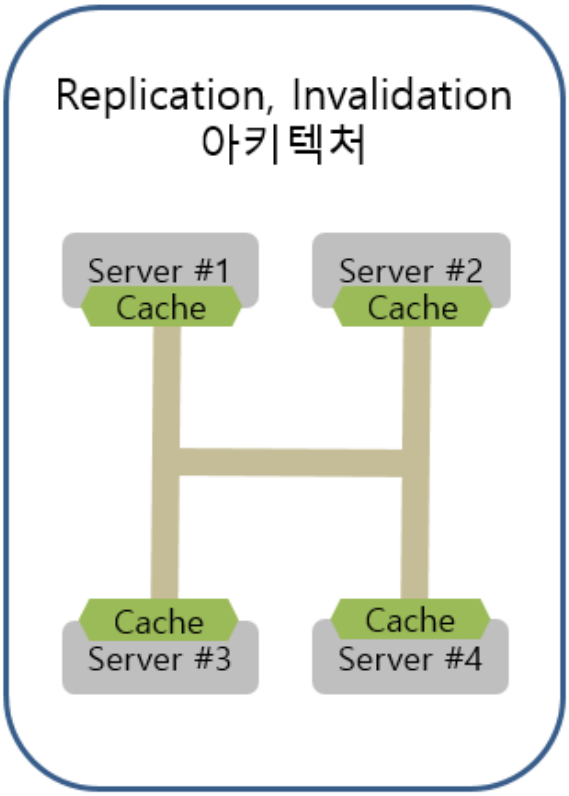
한 노드에서 변경이 발생하면 클러스터링에 참여하는 모든 서버들에게 변경을 알리는 방식으로 동작한다.

즉, 클러스터에 있는 서버가 n개인 경우 한번 변경에 n-1개의 통지가 발생

별도의 서버 구축 필요없이 설정만으로 캐시 클러스터링이 가능하다.

Chamomile Cache는 피어 자동 발견 및 RMI를 이용한 클러스터간 데이터 전송의 신뢰성 등 분산 캐시를 위한 기능을 제공하고 있다.

또한 다양한 옵션을 통해 클러스터링 상황에 맞게 설정이 가능하다.



노드 관리(ehcache)

Automatic Peer Discovery : Ehcache에서의 클러스터링은 새로운 노드(서버)가 추가 될 경우 자동적으로 추가된 노드를 발견하는 방식

Manual Peer Discovery : 그리고 지정된 노드 목록에 대해서만 클러스터의 노드로 사용하는 방식이 있다.

클러스터링 설정을 위해서는 다음의 세개의 정보를 지정해줘야 한다.

CacheManager와 Cache 클래스에서 제공하는 주요 API는 아래와 같다.

- CacheManagerPeerProvider : 피어 발견 관련 설정
- CacheManagerPeerListener : 메시지 수신 관련 설정
- CacheReplicator : 캐시 별 메시지 생성 규칙 설정

피어발견

CacheManagerPeerProvider – Automatic

자동으로 추가된 노드를 발견하는 Automatic 방식은 지정한 멀티캐스트 IP(224.0.0.1~239.255.255.255)와 포트에 참여하는 노드를 자동적으로 발견하게 된다.

지정한 IP와 포트에 참여한 노드는 자신을 다른 서버들에게 통지한다.

이 방식을 사용하면 ehcache.xml을 설정 파일이 서버별로 다르지 않고 통일된 설정을 가져갈 수 있는 장점이 있다.

해당 설정의 예시는 다음과 같으며, ehcache.xml에 작성한다.

```
<cacheManagerPeerProviderFactory
 class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
 properties="peerDiscovery=automatic,
multicastGroupAddress=230.0.0.100, multicastGroupPort=1234" />
```

properties설명

설정	내용
peerDiscovery	Automatic로 지정하면 멀티캐스트 방식을 사용한다.
multicastGroupAddress	멀티캐스트 IP 지정
multicastGroupport	멀티캐스트 Port 지정

CacheManagerPeerProvider – Manual

지정된 노드 목록에 대해서만 클러스터의 노드로 사용하는 방식의 설정은 다음과 같다.

```
<cacheManagerPeerProviderFactory
 class="net.sf.ehcache.distribution.RMICacheManagerPeerProviderFactory"
 properties="peerDiscovery=manual,
rmiUrls=//server1:1234/cache1|//server2:1234/cache2" />
```

properties설명

설정	내용
peerDiscovery	Manual로 지정하면 직접 노드를 지정한다.
rmiUrls	분산 노드에 참여할 서버 및 캐시 목록을 지정한다. 현재 노드의 정보는 포함하지 않는다.

피어정보 수신

CacheManagerPeerListener

노드 지정 방식(Manual)으로 설정했다면, 노드에서 발생한 변경 정보를 수신할 때 사용할 포트번호를 지정한다.

```
<cacheManagerPeerListenerFactory
class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"
 properties="port=1234, socketTimeoutMillis=120000" />
```

노드 자동 발견 방식(Automatic)으로 설정했다면, 별도의 properties 설정을 하지 않는다.

```
<cacheManagerPeerListenerFactory
 class="net.sf.ehcache.distribution.RMICacheManagerPeerListenerFactory"/>
```

properties설명

설정	내용
port	메시지를 수신할 때 사용되는 포트
socketTimeoutMills	이 노드에 메시지를 보냈을 때 메시지 전송을 기다리는 시간. 기본값 2000ms

동기화

CacheReplicator

캐시별로 다른 노드에 변경 내역을 알려주기 위해 <cacheEventListenerFactory> 태그를 통해 언제 어떻게 캐시의 변경 내역을 통지할지 여부를 지정 할 수 있다.

아래 코드는 설정의 예시이다.

```
<cache name="samplecache"
       maxEntriesLocalHeap="100000"
       maxEntriesLocalDisk="10000"
       eternal="false"
       diskSpoolBufferSizeMB="20"
       timeToIdleSeconds="300"
       timeToLiveSeconds="600"
       memoryStoreEvictionPolicy="LFU"
       transactionalMode="off">

    <cacheEventListenerFactory
class="net.sf.ehcache.distribution.RMICacheReplicatorFactory"
properties="replicateUpdatesViaCopy=true,replicateUpdates=true" />

    <persistence strategy="localTempSwap" />
</cache>
```

위 코드와 같이 <cacheEventListenerFactory>의 구현 클래스로 RMICacheReplicatorFactory를 지정하면 캐시에 변경이 생길 때 마다 해당 변경 내역을 클러스터에 참여하고 있는 노드의 캐시에 통지하게 된다.

properties설명

설정	내용
replicatePuts	캐시에 새로운 요소가 추가 됐을 때 다른 노드에 복사 여부 설정
relplicateUpdates	캐시 요소의 값이 변경되었을 때 다른 노드에 값을 복사 여부 설정
relplicateRemovals	캐시 요소가 삭제되었을 때 다른 노드에 반영 할지의 여부 설정
relplicateAsynchronously	비동기로 값을 복사 할 지의 여부 설정
relplicateUpdatesViaCopy	새로운 요소를 다른 노드에 복사 혹은 삭제 여부 설정
asynchronousReplication IntervalMills	비동기 방식을 사용할 때 변경 내역을 다른 노드에 통지하는 주기. 기본값 1000

> 참고 : 위 속성의 기본값은 모두 true이다.

CacheManager가 초기화 될 때 클러스터에 참여한 다른 노드로부터 데이터를 로딩 할 수 있다. 때문에 서버가 shutdown 이후 다시 구동 되었을 때 바로 서비스를 제공 할 수 있다.

<bootstrapCacheLoaderFactory>의 구현클래스를 RMIBootstrapCache LoaderFactory로 지정한다.

```
<cache name="samplecache"
       maxEntriesLocalHeap="100000"
       maxEntriesLocalDisk="10000"
       eternal="false"
```

```
        diskSpoolBufferSizeMB="20"
        timeToIdleSeconds="300"
        timeToLiveSeconds="600"
        memoryStoreEvictionPolicy="LFU"
        transactionalMode="off">

    <bootstrapCacheLoaderFactory
class="net.sf.ehcache.distribution.RMIBootstrapCacheLoaderFactory"
properties="bootstrapAsynchronously=true, maximumChunkSizeBytes=5000000" />

    <cacheEventListenerFactory
 class="net.sf.ehcache.distribution.RMICacheReplicatorFactory" />

    <persistence strategy="localTempSwap" />
</cache>
```

properties설명

설정	내용
bootstrapAsynchronously	비동기적으로 수행할지 여부 설정
maximumChunkSizeBytes	클러스터에 참여한 다른 노드로 부터 로딩 가능한 데이터의 최대 크기

샘플프로젝트 확인

앞서 알아본 Cache Clustering 기능은 제공되는 샘플 프로젝트로 Ehcache, Reids 모두 확인 가능하다.

아래와 같은 시나리오로 Cache Clustering 작동을 확인한다.

1. 제공되는 chamomile-samples-cache 프로젝트를 준비한다.(개발도구를 통해 생성가능)
   (초기 설정은 Ehcache로 되어있으며 context-cache.xml을 통해 확인한다.)
2. 포트가 다른 두 개의 WAS를 준비하고 프로젝트를 배포하여 구동한다.
3. 두 개의 WAS가 모두 구동 되면 아래의 1번 WAS 테스트 URL을 호출하여 현재 시간을 가져온다.
   URL : http://IP:PORT1/(컨텍스트명)/clustering

1. URL을 재호출해도 캐시에 저장된 시간을 가져오기 때문에 현재 시간이 바뀌지 않는 것을 확인한다.
2. 2번 WAS의 테스트 URL을 호출한다.
   URL : http://IP:PORT2/(컨텍스트명)/clustering
3. 마찬가지로 1번 WAS의 캐시에 저장된 시간을 가져오는 것을 확인 할 수 있다.
4. Context-cache.xml에서 CacheManager의 구현체를 Ehcache에서 Redis로 바꾼다. (주석 해제)
5. 1번부터 6번까지의 시나리오를 반복하여 Redis에서도 마찬가지로 캐시클러스터링 기능을 확인한다.

모니터링

##Scouter

> Web 어드민 설정 - 모니터링 설정 섹션에서 설명한다.

Chamomile Actuator

캐모마일에는 애플리케이션을 프로덕션 환경으로 푸시 할 때 모니터링하고 관리하는데 도움이되는 캐모마일 액츄에이터 기능을 제공한다. 관련 기능은 아래와 같이 설정을 추가하여 기능을 사용할 수 있다.

적용방법

의존성 추가

```
<dependencies>
  <dependency>
    <groupId>net.lotte.chamomile</groupId>
    <artifactId>chamomile-actuator</artifactId>
  </dependency>
</dependencies>
```

일반 캐모마일 프로젝트에서는 아래와 같이 관련 빈을 xml에 추가로 등록하여야 한다.

```
<!-- 스프링 부트 어드민 연결을 위한 Bean -->
<beans:bean class="net.lotte.chamomile.actuator.ActuatorConfiguration" >
</beans:bean>
```

end points

제공 되는 end point목록은 아래와 같다.

actuator : HATEOAS 형태로 정보를(enable되어있는 것들만) 표시

auditevents : application 에서 수행되는 인증을 이벤트로 감지한다.

autoconfig : auto-configuration이 적용/미적용 이유를 보여준다.

beans : 빈 정보들을 보여준다.

configprops : @ConfigurationProperties 로 조합된 목록 표시(property가 설정된 정보 표시)

env : 환경 프로퍼티를 보여준다.

flyway : Flyway 데이터베이스 마이그레이션을 표시. Flyway Bean필요 (flyway : 오픈소스 데이터베이스 마이그레이션 도구)

health : application의 상태 정보를 표시

trace : http 추적정보 표시

info : application의 정보 표시(임의의?)

loggers : logger설정을 보여주고 수정할 수 있도록 해준다.

liquibase : liquibase의 데이터베이스 마이그레이션을 표시

> liquibase : database schama변경을 tracking하여 관리할 수 있게 해주는 오픈소스(쿼리들을 직접 수행하는것이 아닌 liquibase를 통해 실행하면 그 이력이 표시된다.)

metrics : 메모리, 클래스 로더, http 요청횟수, 마지막 요청의 걸린 시간 등 매트릭 정보를 보여준다.

mappings : @RequestMapping으로 설정된 정보들을 보여준다.

shutdown : application을 shutdown시킨다.(기본설정 : disabled)

dump : thread dump를 수행한다.

애플리케이션이 웹 애플리케이션 (Spring MVC) 인 경우 다음 추가 엔드 포인트를 사용할 수 있다.

docs : spring-boot-actuator-docs를 추가하면 Actuator의 endpoints의 문서로 확인 할 수 있다.

heapdump : hprof를 이용한 heap dump 압축 파일을 다운받는다.

jolokia : Jolokia를 사용할 경우 HTTP를 통해 JMX를 확인할 수 있다.

> jolokia : HTTP 프로토콜을 이용해 손쉽게 JMX 값을 JSON 형식으로 받아볼 수 있게 해주는 일종의 JMX-HTTP 커넥터

logfile : 로그파일의 내용을 리턴한다.

end point 설정

접근 제어

- 설정시 사용되는 property
  - endpoint.{name}.sensitive : spring security에의해 인증을 수행할지 여부를 설정한다.(true/false)
- endpoint.{name}.enabled : 해당 정보를 활성화 할지 여부를 설정한다.(true/false)
  - endpoint별로 추가적인 항목들의 설정이 가능하다.(id, time-to-live 등)
- 모든 endpoints를 비활성화 하고 info endpoint 만 활성화하고자 할 경우 아래와 같이 설정한다.
- endpoints.enabled=false
- endpoints.info.enabled=true
- 마찬가지로 모든 endpoints들의 인증을 활성화 하고 비활성화 하고 info endpoint 만 활성화하고자 할 경우 아래와 같이 설정한다.
- endpoints.sensitive=true
- endpoints.info.sensitive=false

CORS

- Cross-origin resource sharing (CORS) 표준에 의해 허용할 URL을 별도로 지정할 수 있다. 아래와 같이 요청하는 주소와 method를 적어준다.
- endpoints.cors.allowed-origins=http://example.com
- endpoints.cors.allowed-methods=GET,POST

사용자 정의 endpoint 추가

- 제공되는 인터페이스인 Endpoint<T> 를 구현하여 bean으로 생성하면 getId인터페이스에서 반환하는 이름으로 주소 입력시(/customEndpoint) 내용을 확인 할 수 있다.

```
@Component
public class CustomEndpoint implements Endpoint<List<String>> {
```

```
  @Override
  public String getId() {
    return "customEndpoint";
  }

  @Override
  public boolean isEnabled() {
    return true;
  }

  @Override
  public boolean isSensitive() {
    return true;
  }

  @Override
  public List<String> invoke() {
    // Custom logic to build the output
    List<String> messages = new ArrayList<String>();
    messages.add("This is message 1");
    messages.add("This is message 2");
    return messages;
  }
}
```

출력 결과 : ["This is message 1", "This is message 2"]

health indecator

- health정보는 ApplciationContext에 정의된 모든 HealthIndicator bean에서 수집된다.
- Spring boot 에는 auto config를 통해 다양한 HealthIndicators가 포함되어 있고 직접 작성할 수도 있다.
- health정보는 기본적으로 인증을 통해 접근할 수 있도록 sensitive설정이 true로 설정되어있다.
- 서비스 거부(Dos) 공격을 방지하기위해 상태 응답도 캐싱된다. 기본 캐시시간은 endpoints.health.time-to-live 으로 설정한다.(milli second)
- 기본적으로 제공되는 indicator는 아래와 같다.

  CassandraHealthIndicator

  DiskSpaceHealthIndicator

  DataSourceHealthIndicator

  ElasticsearchHealthIndicator

  JmsHealthIndicator

  MailHealthIndicator

  MongoHealthIndicator

  RabbitHealthIndicator

  RedisHealthIndicator

  SolrHealthIndicator

  위와 같은 설정은 management.health.defaults.enabled 를 통해 활성/비활성화 할 수 있다.
- 사용자 정의 indicator

  HealthIndicator interface를 구현함으로써 사용자 정의 indicator를 생성할 수 있다.

```
import org.springframework.boot.actuate.health.Health;
import org.springframework.boot.actuate.health.HealthIndicator;
import org.springframework.stereotype.Component;

@Component
public class MyHealthIndicator implements HealthIndicator {

  @Override
  public Health health() {
    int errorCode = check(); // perform some specific health check
    if (errorCode != 0) {
      return Health.down().withDetail("Error Code", errorCode).build();
    }
    return Health.up().build();
  }
}
```

위 예제에서 HealthIndicator가 접미어로 설정되며 indicator사용시 "my"라는 항목에 표시가 된다.

infoContributors

- info endpoint에의해 제공되는 정보는 ApplicationContext에 정의된 모든 Infocontributor Bean에서 수집된 다양한 정보를 노출한다. Spring boot 에는 auto config를 통해 다양한 infoContributors 포함되어 있고 직접 작성할 수도 있다.
- 기본적으로 제공되는 InfoContributor는 아래와 같다.

  EnvironmentInfoContributor : Environment에 정의된 모든 info키 노출

  GitInfoContributor : git.properties을 사용하는 경우 git정보 노출

  BuildInfoContributor : META-INF/build-info.properties 파일을 사용하는 경우 build정보 노출
- management.info.defaults.enabled 값을 통해 활성/비활성 여부를 설정할 수 있다.
- 사용자 정의 contributor작성

InfoContributor interface를 구현하여 생성할 수 있다.

```
import java.util.Collections;

import org.springframework.boot.actuate.info.Info;
import org.springframework.boot.actuate.info.InfoContributor;
import org.springframework.stereotype.Component;

@Component
public class ExampleInfoContributor implements InfoContributor {

  @Override
  public void contribute(Info.Builder builder) {
    builder.withDetail("example",
                       Collections.singletonMap("key", "value"));
  }
}
```

이후 "info" endpoint요청시 아래와 같이 출력된다.

```
{
  "example": {
    "key" : "value"
  }
}
```

Auditing

- Spring security를 사용하고 있다면 AbstractAuthenticationAuditListener와 AbstractAuthorizationAuditListener를 구현하여 인증 상태 리포팅, 잠금정책등을 구현할 수 있다.

Tracing

- 기본적으로 request와 response정보들을 확인할 수 있다. timestamp를 이용하여 처리 시간을 구할 수도 있다.
- custom tracing
    - 추가 정보를 추적하고자 할 경우 TraceRepository를 구현하여 Bean으로 등록한다.

Process monitoring

- ApplicationPidFileWriter : application.pid 인 application의 pid를 포함하는 파일을 만들어 준다.
- EmbeddedServerPortFileWriter : application.port인 파일명으로 applciation의 포트를 포함하는 파일을 만들어준다.
- 위 기능을 사용하기 위해서는 META-INF/spring.factories 파일에 아래와 같이 정의한다.

```
org.springframework.context.ApplicationListener=\
org.springframework.boot.system.ApplicationPidFileWriter,\
org.springframework.boot.actuate.system.EmbeddedServerPortFileWriter
```

- SpringApplication.addListeners(...) 를 통해 적절한 Writer객체를 전달하여 리스너를 활성화 할 수 있다. 이를 통해 사용자정의 파일을 생성할 수 있다.

Cloud Foundry support

- 클라우드 인스턴스에 배포할 경우 /cloudfoundryapplication 경로를 통해 간접적인 NamedMvcEndpoint Bean에 대한 대체 보안 경로를 제공한다. 여기에는 application의 각종 endpoint정보들이 포함된다. 예시로 /cloudfoundryapplication/health등으로 사용한다.
- 이 후 spring security설정을 통해 "mvcMatchers("/cloudfoundryapplication/**").permitAll()" 과 같이 설정 한 후 사용한다.

end points를 활용한 모니터링

chamomile actuator는 기본적으로 JMX를 통해 application정보들을 확인할 수 있다.

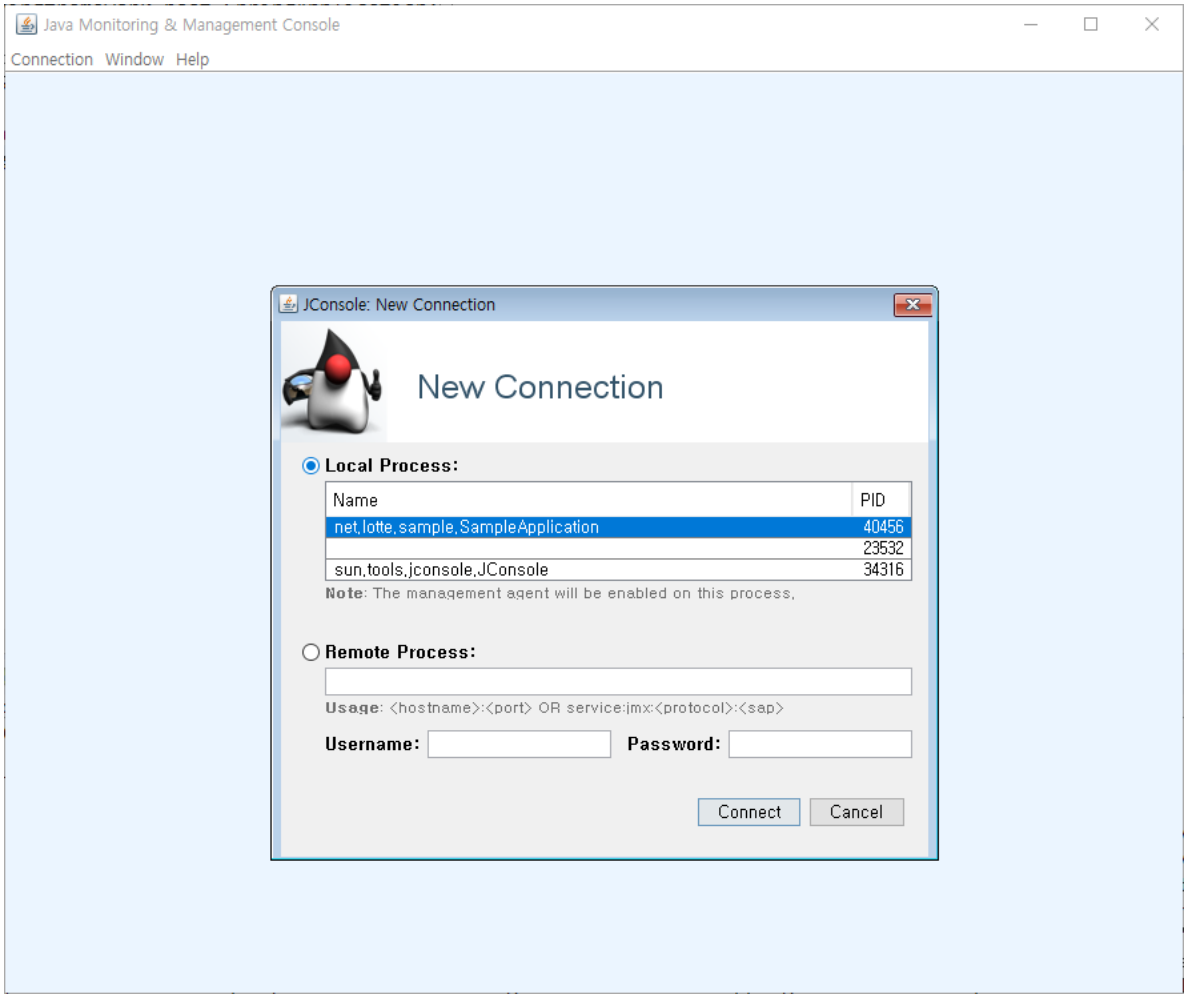
> *JMX(Java Management eXtensions) : jdk 1.5부터 포함된 사양으로 application의 상태를 모니터링 하고 설정하는데 사용하는 기술로 JMX의 핵심 구성요소는 MBean(Managed Bean) 이다.*

Spring에서는 여러가지 MBean유형중 모델 MBeand으로 Spring bean들을 export하여 사용되고 있음.

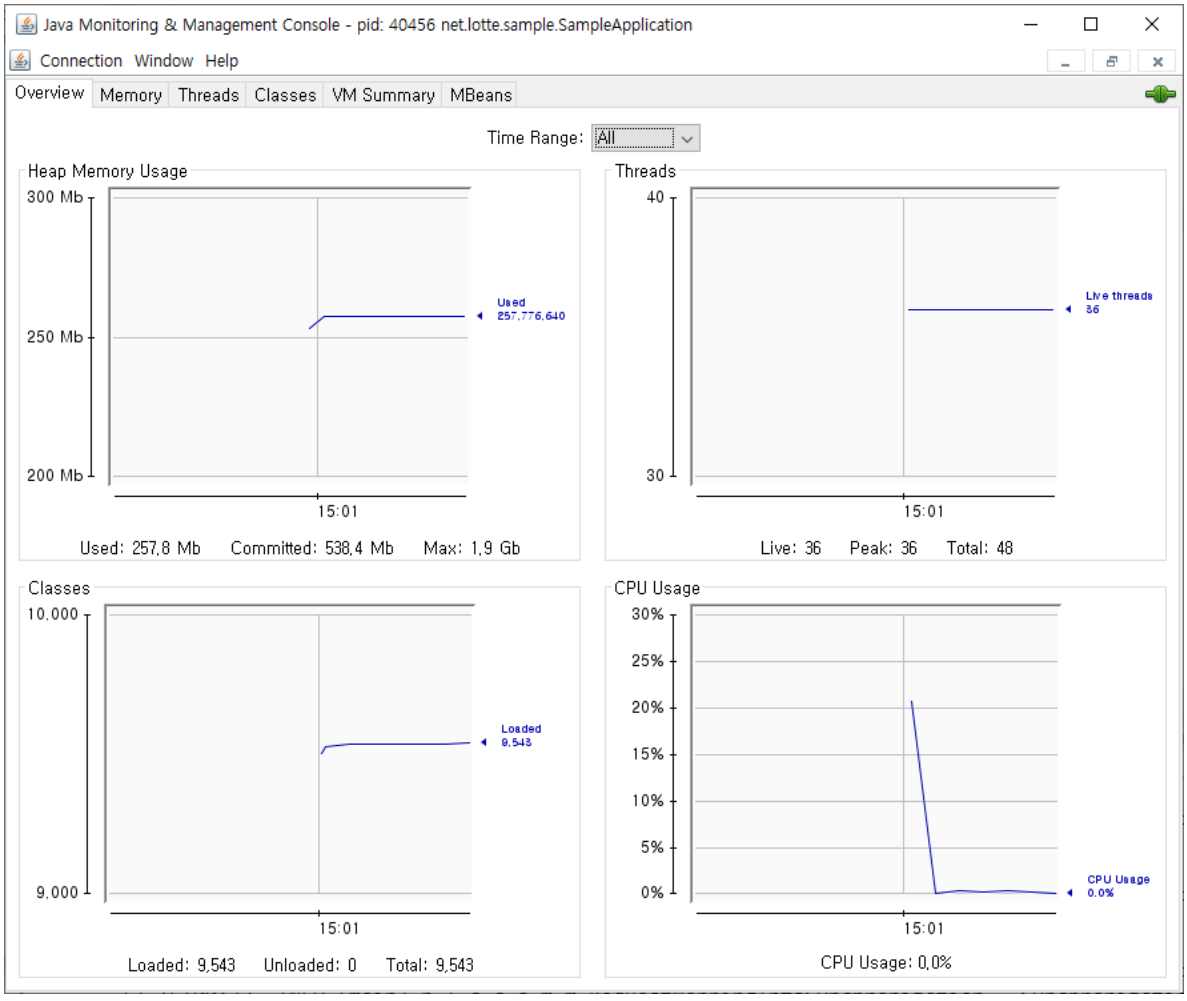
windows기준으로 작성한다.

jconsole

- cmd창에서 jconsole입력
- 실행된 JConsole 창에서 Local Process항목에서 구동중인 application을 선택한다.

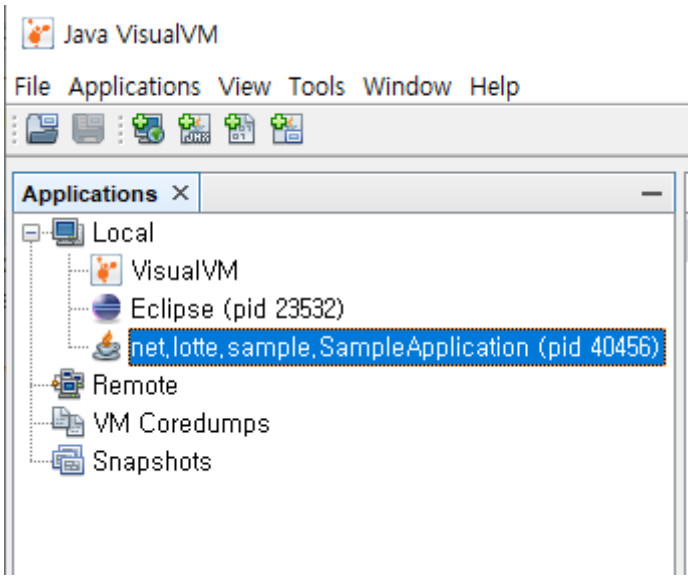


- 아래 화면 처럼 모니터링을 수행할 수 있다.

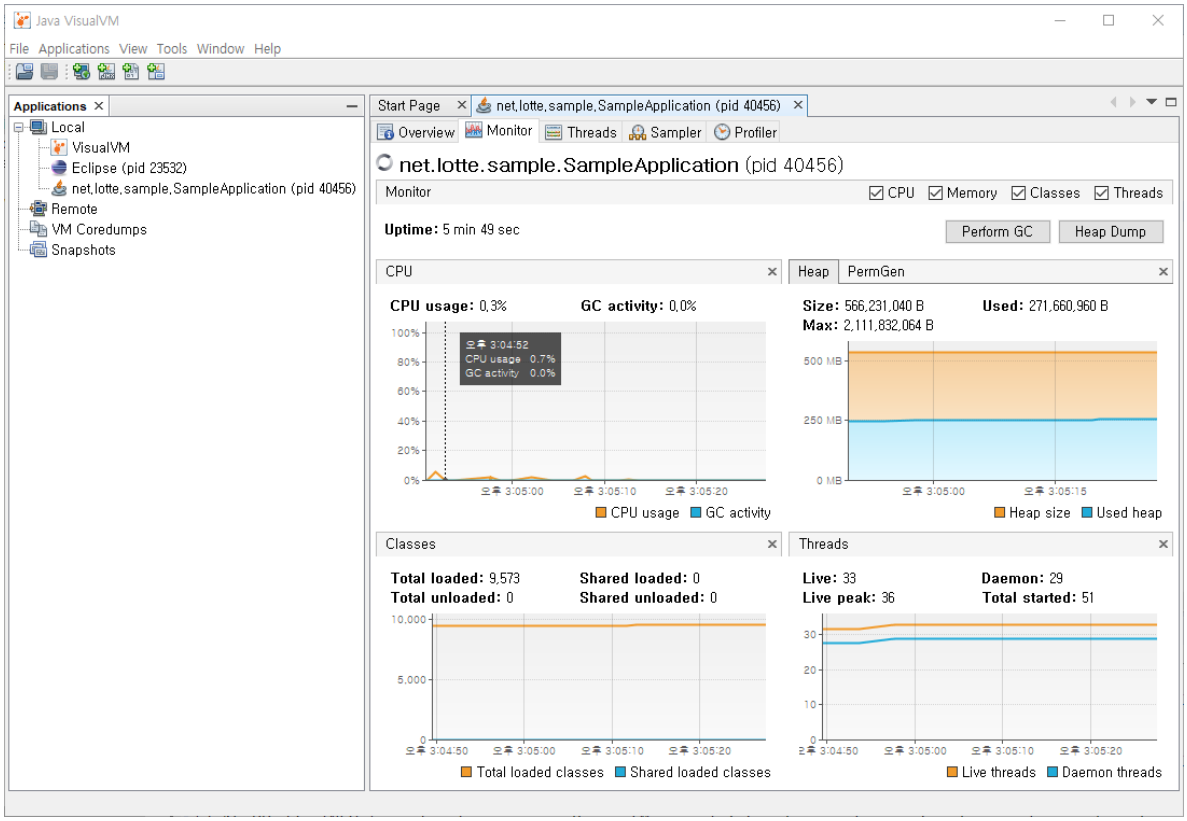


Java Visual VM

- oracle jdk사용시 jdk 6~8버전에서는 함께 배포 되었으나 jdk 9부터는 별도로 다운받아 사용해야 한다.
- https://visualvm.github.io/download.html 에서 다운로드 한다.
- cmd창에서 jvisualvm입력
- 실행된 Java VisualVM 창에서 Application - Local 항목에서 구동중인 application을 선택한다.



- 아래 화면 처럼 모니터링을 수행할 수 있다.(jconsole보다 좀더 향상된 UI의 모니터링을 제공한다.)



캐모마일 어드민과 연동

- 캐모마일 액츄에이터는 캐모마일 어드민과 연동이 가능하다.
- 캐모마일 어드민에는 spring boot admin이라는 모니터링 어플리케이션을 내장하고 있다.
- 캐모마일 어드민과 연동하기 위해서 위에서 설명한 방법대로 pom.xml에 추가한후 ActuatorConfiguration 대신 SpringBootAdminClientConfiguration 빈을 등록한다.

의존성 추가

```
<!-- 스프링 부트 어드민 연결을 위한 Bean -->
<beans:bean class="net.lotte.chamomile.actuator.ActuatorConfiguration" >
</beans:bean>
```

그리고 나서 application.properties에 아래와 같이 설정을 추가한다.

```
# 어플리케이션 이름
spring.application.name=demo
...

# 캐모마일 어드민 주소
spring.boot.admin.client.url=http://127.0.0.1:23626/sba
# 캐모마일 어드민 관리자 아이디
spring.boot.admin.client.username=admin
# 캐모마일 어드민 관리자 비밀번호
spring.boot.admin.client.password=1111
# 현재 앱 서비스 주소
spring.boot.admin.client.instance.service-base-url=http://127.0.0.1:8080
# 현재 앱 유저 아이디
spring.boot.admin.client.instance.metadata.user.name=admin
# 현재 앱 유저 비밀번호
spring.boot.admin.client.instance.metadata.user.password=1111
```