|  |  |
| --- | --- |
| **개발표준 정의서** | |
| **차세대 OOO시스템 구축사업 (공공 분야)**  업무명  **Version 0.1**  발주사명 | |
|  |  |
| SK주식회사 | |

**제.개정 이력**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **버전** | **제.개정 일자** | **제.개정 내용** | **작성자** | **승인자** |
| 0.1 | YYYY-MM-DD | 최초 등록 | OOO | OOO |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Index / 목차

[1. 개요 5](#_Toc102134280)

[1.1. 개발 표준 정의서의 필요성 5](#_Toc102134281)

[1.2. 개발 표준 정의서 적용의 예외 5](#_Toc102134282)

[2. 소프트웨어 아키텍처 6](#_Toc102134283)

[2.1. 적용 환경 6](#_Toc102134284)

[2.2. 도입 솔루션 6](#_Toc102134285)

[2.3. 적용 아키텍처 7](#_Toc102134286)

[2.4. Layer별 아키텍처 8](#_Toc102134287)

[2.4.1. UI 9](#_Toc102134288)

[2.4.2. Controller 9](#_Toc102134289)

[2.4.3. Service 9](#_Toc102134290)

[2.4.4. Mapper 9](#_Toc102134291)

[2.4.5. SqlMap 9](#_Toc102134292)

[2.4.6. DataSource 9](#_Toc102134293)

[2.5. 어플리케이션 구조 10](#_Toc102134294)

[3. 개발 표준 14](#_Toc102134295)

[3.1. 내부 호출 URI 작성 표준 14](#_Toc102134296)

[3.2. OPEN API 표준 18](#_Toc102134297)

[3.2.1. 구성 요소 18](#_Toc102134298)

[3.2.2. 기본 구조 20](#_Toc102134299)

[3.2.3. URL 기본 형식 20](#_Toc102134300)

[3.2.4. URI 작성 표준 21](#_Toc102134301)

[3.2.5. 기본 처리 유형별 Request Method 21](#_Toc102134302)

[3.3. 명명 STYLE 22](#_Toc102134303)

[3.3.1. Pascal Case 22](#_Toc102134304)

[3.3.2. Camel Case 22](#_Toc102134305)

[3.3.3. Lower Case 22](#_Toc102134306)

[3.3.4. Upper Case 22](#_Toc102134307)

[3.3.5. Hungarian Notation 23](#_Toc102134308)

[3.4. 명명 규칙 23](#_Toc102134309)

[3.4.1. 업무 구분 및 주제 영역 23](#_Toc102134310)

[3.4.2. 공통 사항 23](#_Toc102134311)

[3.4.3. 클래스 파일명 24](#_Toc102134312)

[3.4.4. 화면단 파일명 24](#_Toc102134313)

[3.4.5. 함수(Method) 명 25](#_Toc102134314)

[3.4.6. 변수(Variable) 명 26](#_Toc102134315)

[3.4.7. 상수(Constant) 명 27](#_Toc102134316)

[3.5. 주석 처리 30](#_Toc102134317)

[3.5.1. 문서 주석 30](#_Toc102134318)

[3.5.2. 함수 주석 31](#_Toc102134319)

[3.5.3. 변수 주석 32](#_Toc102134320)

[3.5.4. 제어 구조 주석 32](#_Toc102134321)

[3.5.5. MyBatis SQL 주석 33](#_Toc102134322)

[3.5.6. 기타 주석 처리 34](#_Toc102134323)

[3.6. Annotation 적용 35](#_Toc102134324)

[3.6.1. 어노테이션 사용 순서 35](#_Toc102134325)

[3.6.2. Reflection 이란 35](#_Toc102134326)

[3.6.3. 주요 어노테이션 태그 설명 및 예시 35](#_Toc102134327)

[3.7. CODING STYLE 40](#_Toc102134328)

[3.7.1. Indent (들여쓰기) 40](#_Toc102134329)

[3.7.2. Space 41](#_Toc102134330)

[3.7.3. Brace (중괄호) 43](#_Toc102134331)

[3.7.4. 용어 44](#_Toc102134332)

[3.7.5. Logging 44](#_Toc102134333)

[3.7.6. File 명명 및 Line 44](#_Toc102134334)

[3.8. UI 구조 46](#_Toc102134335)

[3.8.1. 프로젝트 구조 46](#_Toc102134336)

[3.8.2. WebSquare 구성 46](#_Toc102134337)

[3.8.3. 화면 구조 46](#_Toc102134338)

[3.8.4. DataCollection 구조 46](#_Toc102134339)

[3.9. UI 개발 표준 47](#_Toc102134340)

[3.9.1. WebSquare 47](#_Toc102134341)

[3.9.2. Rexpert 48](#_Toc102134342)

[3.10. Presentation 계층 개발 49](#_Toc102134343)

[3.10.1. Controller 개발 표준 49](#_Toc102134344)

[3.10.2. UI-Controller Data 처리 표준 52](#_Toc102134345)

[3.11. Business계층 개발 53](#_Toc102134346)

[3.11.1. Service 개발 표준 54](#_Toc102134347)

[3.11.2. VO(Value Object) 개발 표준 58](#_Toc102134348)

[3.12. Persistence 계층 개발 74](#_Toc102134349)

[3.12.1. Mapper 개발 표준 74](#_Toc102134350)

[3.12.2. Mapper 작성 표준 79](#_Toc102134351)

[3.13. 그 외 개발 표준 89](#_Toc102134352)

[3.13.1. Logging 표준 89](#_Toc102134353)

[3.13.2. 보안 로깅 91](#_Toc102134354)

[3.13.3. Exception 표준 92](#_Toc102134355)

[3.13.4. Mapper Exception 처리 92](#_Toc102134356)

[3.13.5. Service Exception 처리 92](#_Toc102134357)

[3.13.6. Controller Exception 처리 92](#_Toc102134358)

[3.13.7. Exception Resolver 93](#_Toc102134359)

[3.13.8. Transaction 표준 93](#_Toc102134360)

[3.13.9. Property표준 96](#_Toc102134361)

[3.13.10. Message 표준 96](#_Toc102134362)

[3.13.11. Interceptor 정책 100](#_Toc102134363)

[3.13.12. 세션 처리 정책 104](#_Toc102134364)

[3.13.13. Session 정의 104](#_Toc102134365)

[3.13.14. 페이징 처리 표준 105](#_Toc102134366)

[3.13.15. 파일 업로드/다운로드 표준 118](#_Toc102134367)

[3.13.16. Chart 표준 118](#_Toc102134368)

[3.13.17. SMS 표준 118](#_Toc102134369)

[3.13.18. E-Mail 표준 118](#_Toc102134370)

# 개요

본 문서는 OOO 차세대 시스템 구축 개발에 대한 개발 표준을 정의한 가이드로써, 시스템 개발자를 위한 개발 절차 및 방안을 정의하고 있다. 개발자는 본 문서에 정의된 개발환경을 이해하고 개발 표준을 준수하여 코딩을 수행하여야 한다.

## 개발 표준 정의서의 필요성

개발 표준 정의서는 내부에서 작성되는 응용 프로그램에 대하여 공통된 구조와 형태를 제시함으로써

* 개발된 소스 코드의 가독성을 높여 개발자들의 소스 코드에 대한 이해도를 증가시킨다.
* 개발자들 간의 의사소통을 원활하게 한다.
* 유지보수 시점의 생산성을 증가시킨다.

## 개발 표준 정의서 적용의 예외

개발이 진행되는 과정에서 개발 표준 정의서 내 표준의 범위를 벗어나거나 개발 표준을 적용하기 어려운 상황이 발생할 수 있다. 이런 경우 불가피하게 예외를 인정할 수는 있지만, 이에 대해 개발 표준 담당자와 논의하여 대응 방안을 마련하고, 예외 상황에 대한 공유가 이루어질 수 있도록 한다.

\* 본 문서는 참고용으로 실제 공공 분야 차세대 시스템 구축시 필수적으로 지켜야할 사항은 아니며, **기본적인 개발 표준정의 가이드**에 대한 설명을 하는 문서 이다.

프로젝트 수행시 개발 표준 정의를 위한 참고 용도로만 사용하고, 실제로는 고객의 요구사항 및 아키텍처 구조에 맞게 개발 표준정의서를 작성해야 한다.

페이지의 마지막 줄

# 소프트웨어 아키텍처

## 적용 환경

|  |  |
| --- | --- |
| 구분 | 버전 정보 |
| JDK | Open JDK 11.0.12 (LTS) 이상 |
| Servlet | 3.0 이상 |
| 프레임워크 | SPRING 5.3.14.RELEASE (Spring Boot 2.6.2 Release) |
| IDE | Eclipse 4.21 (2021-09) 이상 또는 eGovFrame 3.10이상 |
| WAS | Spring Boot 내장 WAS (Tomcat 9.0.52 or Undertow)  **Sample** |
| DBMS | Oracle Database 19c |
| UI | Websquare 5.0 (SP5) |
| Reporting | Clip Report (예정 재사용 소프트웨어) |

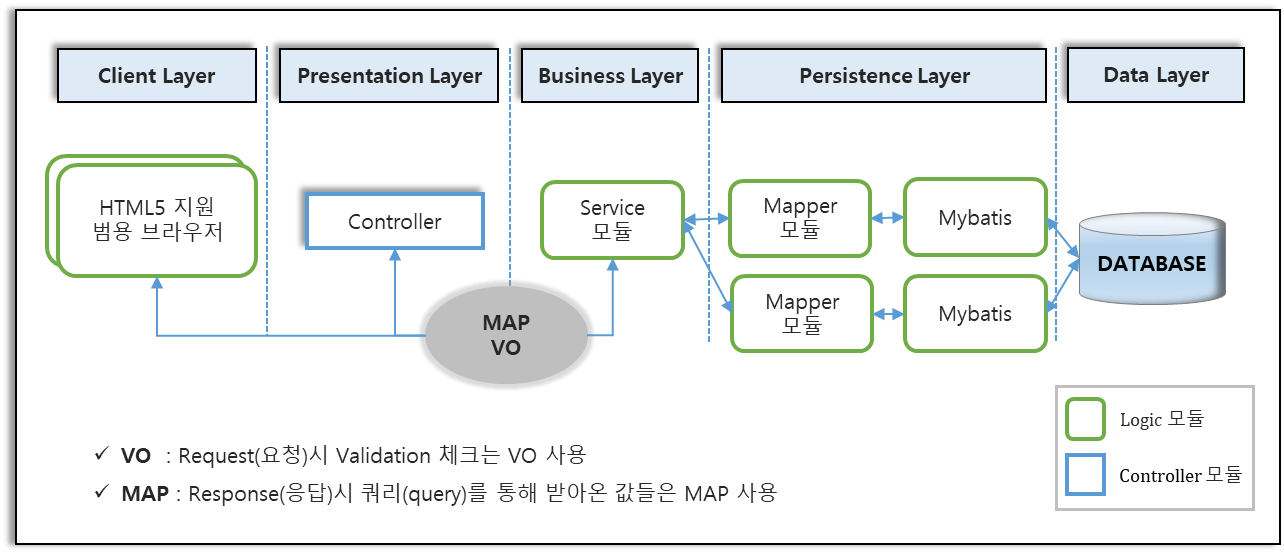
## 도입 솔루션

OOO 차세대 시스템 구축에 신규로 도입되는 솔루션은 글로벌 웹 표준기술(HTML5) 기반의 제품을 적용하여 프로세스를 선진화하고 고객서비스 편리성을 제공한다.

|  |  |  |
| --- | --- | --- |
| 구분 | 제품명(제조사) | 주요 규격 및 기능 |
| UI Platform | WebSquare  (인스웨이브시스템) | * Non-ActiveX 방식의 웹 표준 HTML5 표준 기반 제품 * RIA 기반의 UI/UX 개발 도구 제공 및 국제표준준수 * 다양한 제품 및 프레임워크 연계 지원 * Multi Browser 등 웹호환성 / 접근성 준수 |
| Reporting 도구 | Clip Report | * Non-ActiveX 방식의 웹 표준 HTML5 표준 기반 제품   **Sample**   * 자체 Chart, 바코드, Cross Tab, Drilling 등의 기능 제공 * 섹션 기반 리포팅 및 표 컨트롤 지원 |

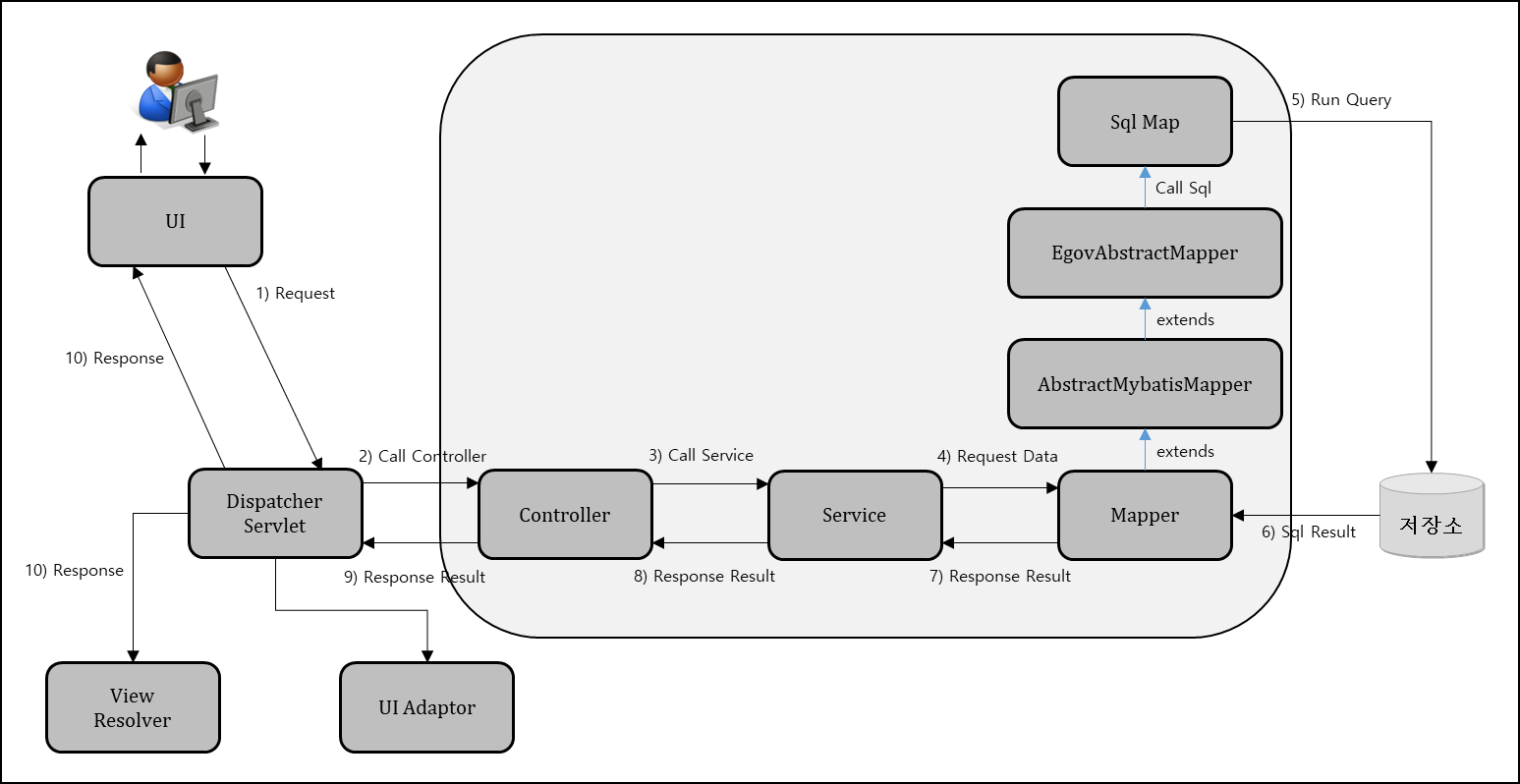
## 적용 아키텍처

OOO 차세대 시스템 구축은 Spring boot 2.6.2 를 활용한 통합 프레임워크를 기반으로 개발하며 다음과 같은 어플리케이션 아키텍처를 적용한다.



**Sample**

* 온라인 서비스는 UI에서 URL 호출을 시작으로 Controller, Service, Mapper 및 SQL 호출까지의 일련의 과정으로 진행된다. 현행 아키텍처에 따라 서비스가 호출/응답 되는 구조는 다음과 같다.

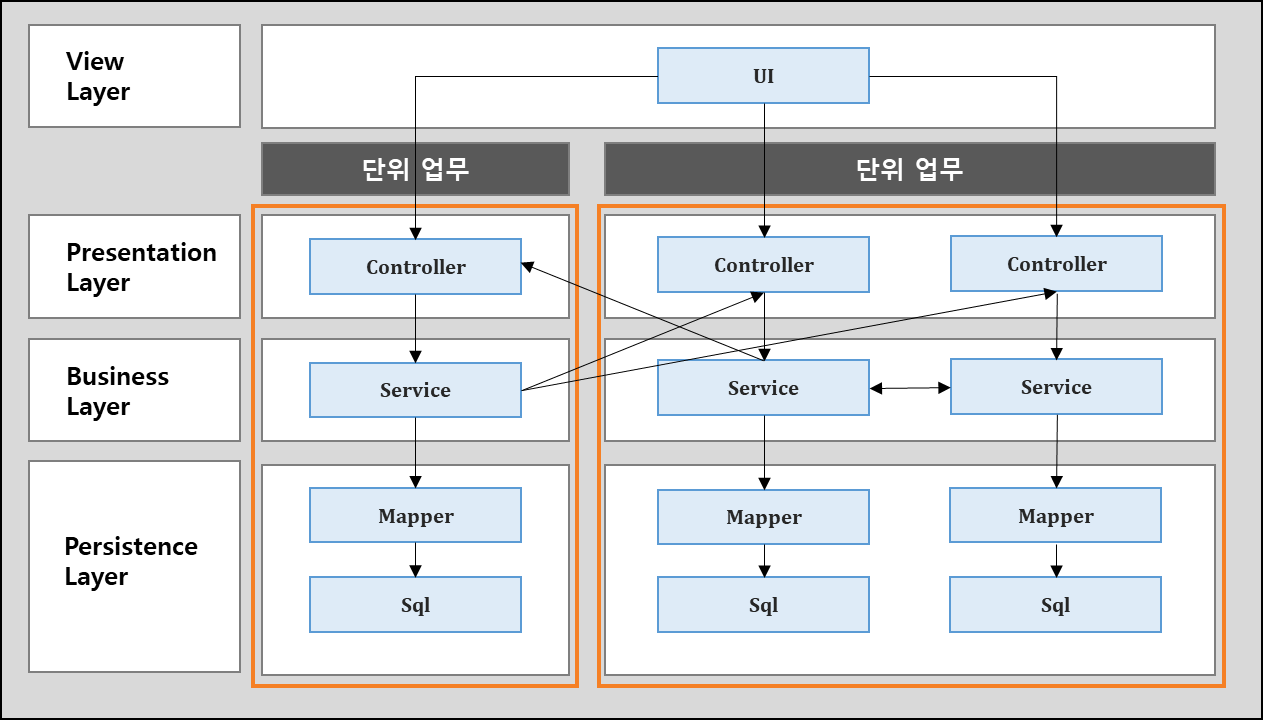


1. 사용자의 요청에 따라 UI에서 생성한 request URL을 Dispatcher Servlet이 받는다.
2. 요청을 전달할 Controller를 호출한다. 이때 UI에서 생성한 dataset 또는 parameter를 UI Adaptor를 거쳐 프레임워크에서 사용하기 적합한 data 형태로 변환하여 Controller에 전달한다.
3. Business logic 처리를 위해서 Controller에서 Service 를 호출한다.
4. Service 에서 Data Access 처리를 위한 Mapper를 호출한다.
5. Mapper는 mapper에 구현된 sql 결과를 Service를 거쳐 Controller에 전달한다. Controller는 결과 Object와 결과 Object를 전달할 view 이름을 Dispatcher Servlet에 전달한다. 이때 Controller는 처리 결과를 UI에서 사용할 수 있는 dataset으로 가공하여 전달한다.
6. Dispatcher Servlet은 View Resolver를 통해서 응답할 view 객체를 찾는다.
7. Dispatcher Servlet은 찾아낸 View 객체에 응답을 전달하고, UI는 전달된 데이터를 이용하여 화면을 구성하여 사용자에게 보여준다.

## Layer별 아키텍처

아래 그림은 개발 시 각 Layer 별 호출 패턴을 정의한 것이다. 검정색 화살표로 표시된 부분이 호출 가능한 패턴을 나타낸 것이며, 주황색 박스로 표시된 부분은 단위 업무의 컴포넌트(Controller, Service, Mapper) 기준 이다.

* Controller, Mapper는 다른 단위업무(컴포넌트) 서비스 호출 금지
* 동일 단위업무(컴포넌트) 내 Service끼리는 호출 가능
* 단위업무 Service에서 다른 단위업무(컴포넌트) 의 Controller는 호출 가능



**Sample**

### UI

* UI 부분은 WebSquare, Clip Report, JSP 등 모든 UI에 해당한다.
* 필드 또는 변수의 Validation Check 는 UI 에서 모두 처리한다.
* UI 에서 서비스 호출은 Controller 호출을 통해서만 할 수 있으며 Controller에 해당하는 URL을 호출 할 수 있다.
* Mapper 또는 SQL을 직접 호출 할 수 없다.

### Controller

* Controller 에서는 1개의 Service를 호출 할 수 있다.
* 실제 업무관련 로직은 Service 에서 처리되며, Controller는 UI 에서 받은 파라미터를 Service에 전달하고, Service에서 리턴 받은 결과를 UI로 전달하는 부분을 처리한다. Service에서 구현되어야 할 업무 로직이 Controller에 구현되는 것은 지양한다.

### Service

* Service 는 실제 업무와 관련된 로직을 구현한다.
* Service 자신의 Mapper를 호출하거나 다른 Service를 호출할 수 있다.
* Service 는 Data처리 관련 메소드에 대한 구체적인 로직을 구현 한다.

### Mapper

* SQL 이 있는 SqlMap을 호출한다.
* Mapper에서는 타 Layer뿐만 아니라, 다른 Mapper에 대한 호출을 하지 않는다.

### SqlMap

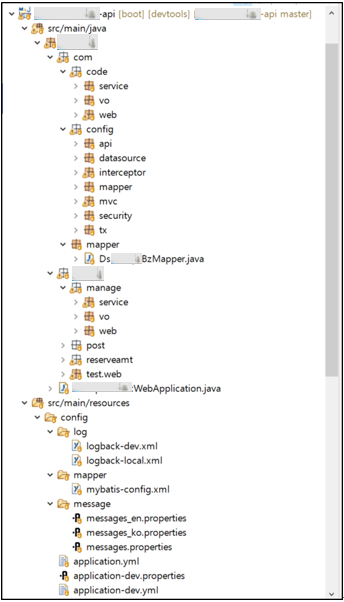
* 원하는 데이터를 얻거나 조회/등록/수정/삭제 처리를 수행하기 위한 SQL이 작성 되어있다.
* SQL 문을 정의하는 xml문서로서 DB처리에 필요한 모든 Query는 이 파일에 구현한다.

### DataSource

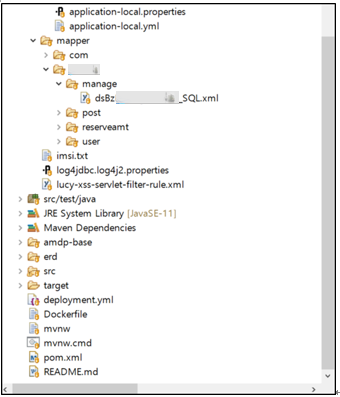
* SQL이 적용되기 위한 데이터 접속 정보를 지정한다.
* 본 프로젝트에서는 MybatisMapper 를 상속 받아 시스템별 공통 Mapper를 구성하여 사용한다

## 어플리케이션 구조

본 프로젝트는 다음과 같은 어플리케이션 구조로 구성되어 있으며, OOO 프로젝트 (xxx-xxx-api) 샘플로 가이드 한다. (다른 프로젝트도 유사한 구조로 되어 있음)



**Sample**



**Sample**

- **src/main/java** : 프로젝트 자바 소스

.xxxWebApplication.java: 어플리케이션을 시작하는 Main()메서드가 있는 클래스

(최상위 패키지에 위치하도록 한다.)

* **Xxxxxx(패키지명) > com > code : 공통코드 그룹**

> service : Service 클래스 파일로 비즈니스 로직 서비스

> vo : VO 클래스 파일로 Value Object 객체

> web : Controller 클래스 파일로 자바에서 일반적인 Controller

(웹 처리를 위한 데이터 송수신 및 담당 서비스 호출)

* **Xxxxxx(패키지명) > com > config : 공통 설정**

> api : api테스트를 위한 swagger config 설정

> datasource : jdbc properties, Configuration properties 설정

> interceptor : 로그인 session 처리를 위한 인터셉터 설정 (URL 체크등 포함)

> mapper : SqlSessionConfig 설정

> mvc : WebMvcConfig 설정, error 페이지 및 redis 구동 설정

> security : API 접근에 대한 인가 정보 확인 객체, Controller에 접근 권한을 설정하기 위한

객체, Spring Security 설정

> tx : 트랜잭션 처리 설정

* **Xxxxxx(패키지명) > 업무영역1depth(2자리) > 업무영역2depth(3자리) > 단위업무 코드(4자리)**

> service : 비즈니스 로직 service

> vo : Value Object 객체

> web : 자바에서 일반적인 Controller (웹 처리를 위한 데이터 송수신, 담당 service 호출)

**Sample**

- **src/main/resources** : 프로젝트 리소스

**> Config** : logback설정, mybatis설정 정보 파일(mybatis-config.xml), 메시지 설정,

JDBC 연결설정 및 서버설정 등 구성정보 파일 (application.yml)

> log : logback 설정 xml

> mapper : mybatis-config.xml 설정

> message : 메시지 Properties 설정

> application.yml : 자바에서 일반적인 Spring설정 파일 (jdbc, 세션, context-path 설정 등)

**> mapper** : SQL Mapping XML 파일 (SQL.xml)

**> 업무영역 1depth(2자리) > 업무영역 2depth(3자리) > 단위업무 코드(4자리)**

: mybatis Mapper 인터페이스를 활용한 업무 쿼리문 작성 파일

- **src/test/java** : 테스트 클래스

(스프링 자동 구성을 이용하여 구성한 Junit 테스트 클래스)

- **target** : Maven 빌드 결과 경로

- deployment.yml : 컨테이너 배포를 위한 설정 파일

- Dockerfile : Docker 컨테이너를 실행시키기 위한 파일

- mvnw : 메이븐 래퍼, 메이븐 미설치시 mvn 명령 대신 사용

- mvnw.cmd : 메이븐 시작 배치 스크립트

- pom.xml : 메이븐 의존성 설정 파일

페이지의 마지막 줄

# 개발 표준

## 내부 호출 URI 작성 표준

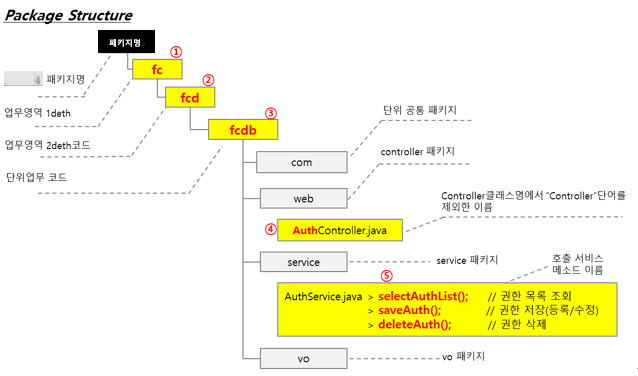
내부 호출 URI 표현을 위한 작성 표준 기본 가이드는 다음과 같다.

* 모든 Resource는 유일한 URI로 정의해야 한다.
* URI는 슬래시(/)로 hierarchical하도록 구성해야 한다.
* URI 의 작성 표준은 다음과 같다.

. /업무영역 1depth 코드(2자리)/업무영역 2depth 코드(3자리)/단위업무 코드(4자리)

/Controller클래스명에서 “Controller”단어를 제외한 이름/호출 서비스 메소드 이름 + .do

**[샘플 예시]**

****

**Sample**

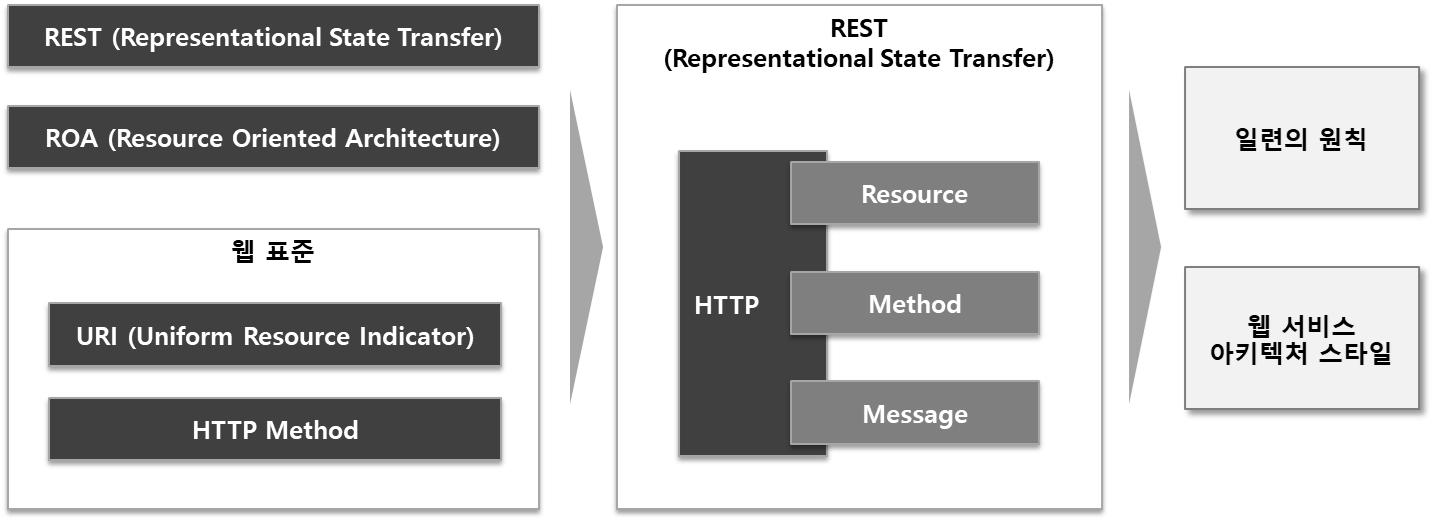
|  |
| --- |
| **<< AuthController.Class 소스 예시>>**  @RestController  @RequestMapping("/api")  **public** **class** AuthController {  @Autowired  AuthService authService;  /\*\*  \* **TODO** 검색조건에 의한 권한 목록을 조회한다.  \* **@param** AuthVO  \* **@param** model  \* **@return**  \*/  @Auth(role = Role.***ADMIN***)   1. **② ③ ④ ⑤**   @PostMapping("/fc/fcd/fcdb/Auth/selectAuthList.do")  **public** ModelAndView selectAuthList(@Param(name = "params") AuthVO authVO, ModelAndView model) {    model.addObject("result", authService.selectAuthList(authVO));  **return** model;  }  /\*\*  \* **TODO** [추가]권한을 저장(등록/수정) 한다.  \* **@param** paramMap  \* **@param** session  \* **@param** model  \* **@return**  \*/  **① ② ③ ④ ⑤**  @PostMapping("/fc/fcd/fcdb/Auth/saveAuth.do")  **public** ModelAndView saveAuth(@Param Map<String, String> paramMap, ModelAndView model) {  authService.saveAuth(paramMap);  **return** model;  }  /\*\*  \* **TODO** [추가]권한을 삭제 한다.  \* **@param** paramMap  \* **@param** session  \* **@param** model  \* **@return**  \*/  **① ② ③ ④ ⑤**  @PostMapping("/fc/fcd/fcdb/Auth/deleteAuth.do")  **public** ModelAndView deleteAuth(@Param Map<String, String> paramMap, ModelAndView model) {  authService.deleteAuth(paramMap);  **return** model;  }  }   1. /fc : 업무영역 1depth 코드(2자리) 2. /fcd : 업무영역 2depth 코드(3자리) 3. /fcdb : 단위업무 코드(4자리) 4. /Auth : “AuthController” 클래스명에서 “Controller”단어를 제외한 이름 5. 호출 서비스 메소드 이름 + .do   > /selectAuthList.do  > /saveAuth.do  > /deleteAuth.do |

페이지의 마지막 줄

## OPEN API 표준

본 프로젝트에서 사용되는 Open API는 일반적인 REST API 기준을 따르고 있다. REST(Representational State Transfer)는 ROA(Resource Oriented Architecture)를 따르는 웹 서비스 아키텍처 스타일의 일종이다. URI와 HTTP Method와 같은 웹 표준을 통해 객체화된 서비스에 어떻게 사용되어야 하는지에 관한 일련의 원칙을 정의한 것이다.

즉, REST는 어떤 자원(Resource)에 어떤 행위(Method)를 어떻게(Message) 할지를 HTTP 기반으로 정해 놓은 아키텍처 스타일이라고 할 수 있다.



**Sample**

### 구성 요소

Open API(REST API) 를 구성하는 Resource(자원), Method(행위), Message(내용)는 다음과 같음

* **Resource (자원) :** 접근할 자원(URI)을 정의
* 예시

|  |  |
| --- | --- |
| URI | 의미 |
| http://api.domain.com/codes | 코드정보 Collection |
| http://api. domain.com/codes/100 | 100번 코드정보 |
| http://api. domain.com/codes/100/name | 100번 코드의 명칭 |

* **Method (행위) :** 자원에 대한 처리 행위를 정의

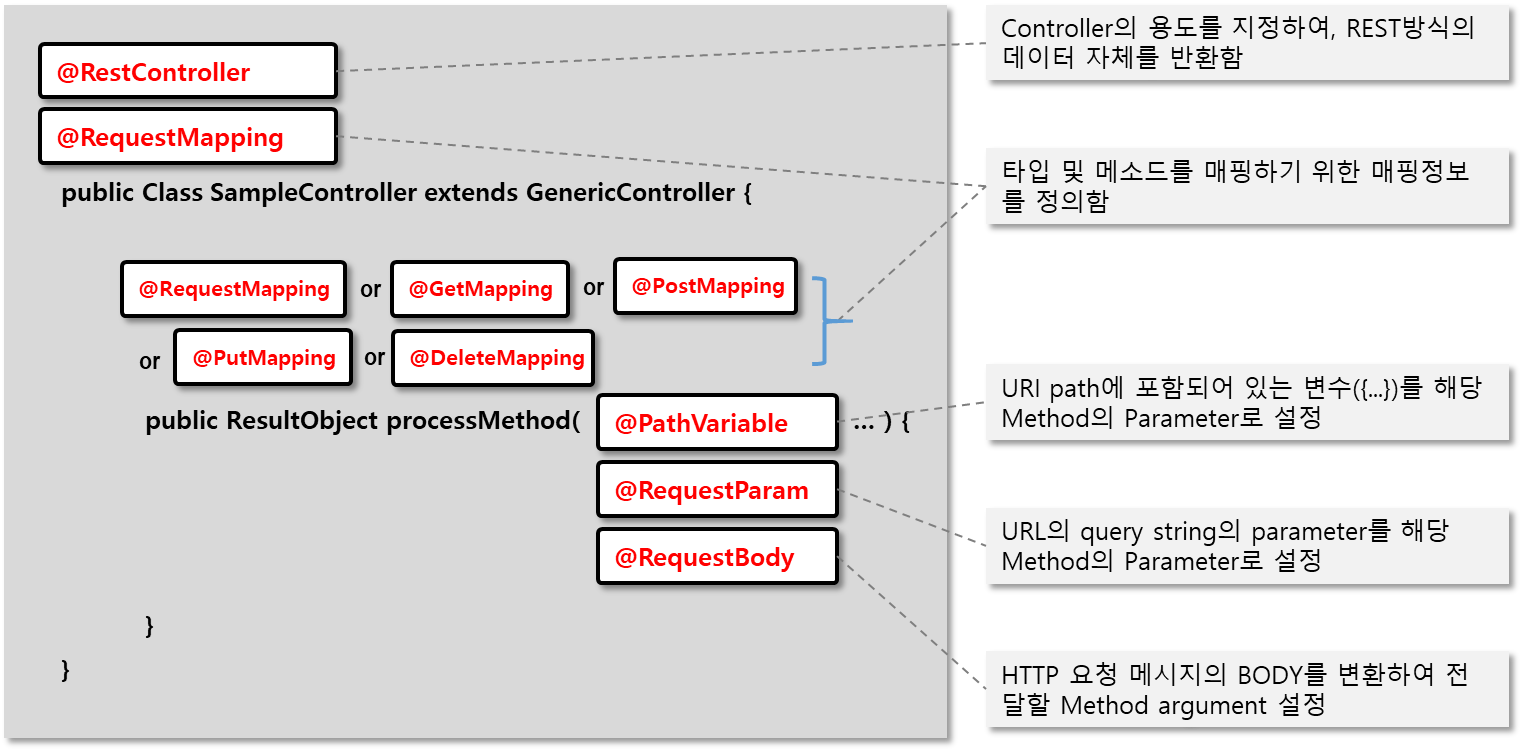
|  |  |  |  |
| --- | --- | --- | --- |
| Method | Operation | Description | Qualities |
| GET | Resource 조회 | 명시된 Resource에 대한 정보를 가져(read)온다. | Idempotent  (멱등성) |
| POST | Resource 생성/변경/삭제 | 처리할 데이터를 특정 Resource에 전송하여 생성(Create)한다. | Not  Idempotent |

* **Message (내용) :** 자원에 대한 처리행위의 내용(Payload)을 정의

|  |  |  |
| --- | --- | --- |
| Item | Description | Remark |
| HTTP Header | Body의 Content type을 명시한다. | Content-type: application/json  Accept:application/json |
| HTTP Body | Body에 포함된 데이터를 통해 정보를 전달한다. | JSON 포맷 사용 |
| Status code | 리소스 요청에 대한 응답 상태를 나타낸다. | 200 : 정상  201 : Request가 처리되었고, 새로운 자원  생성이 됨을 의미  202 : Request가 수락되었으나,Response  메시지를 전달할 때 까지 해당  프로세스가 완료되지 못하였을 때  204 : Request를 처리 했지만,Client에게  전달할 새로운 정보가 없는 경우  301 : Client가 요청한 자원의URI가  변경되었을 경우  302 : Client가 요청한 자원의URI가 임시  변경된 경우   1. : 요청 부적절   401 : Request가 User 인증이 필요함을  Client에게 알려주기 위해 사용  403 : Request가 거절됨  404 : URI에 해당하는 자원을 찾을 수 없는  경우  405 : 잘못된 HTTP Method를 사용한 경우  406 : 비즈니스 로직 에러인 경우  (Request데이터 오류)  409 : Client의 Request를 수행중 충돌이  발생한 경우  411 : Client가 Content-Length를 보내지  않아 처리할 수 없는 경우  412 : Request Header에 설정되어 있는  조건이 올바르지 않은 경우  415 : Platform이 지원하지 않는Media  Type을 요청한 경우  500 : 서버에 예기치 않은 오류로 Request를  처리할 수 없는 경우  504 : 과부하 or Maintenance로 인해 응답할  수 없는 경우 |

### 기본 구조

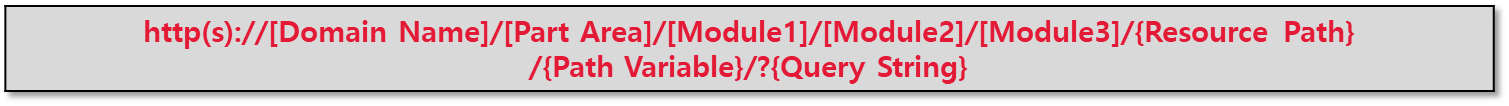
* REST API 원칙을 따르는 Controller Class 및 Method의 기본 구조와 역할은 아래와 같다.



**Sample**

### URL 기본 형식

* REST API 를 표현하는 URL의 기본 형식은 다음과 같다.



|  |  |  |
| --- | --- | --- |
| Attribute | Description | Example |
| Domain Name | 도메인 | http(s)://**www.g2b.go.kr** |
| Part Area | API 서버 구분  (업무영역 1depth) 식별명 | http(s)://www.g2b.go.kr/**com** |
| Module1 | 모듈 구분 1  (업무영역 2depth) 식별명 | http(s)://www.g2b.go.kr/com/**co** |
| Module2 | 모듈 구분 2  (단위업무) 식별명 | http(s)://www.g2b.go.kr/com/co/**coa** |
| Module3 | 모듈 구분 3  (Controller명칭) 식별명 | http(s)://www.g2b.go.kr/com/co/coa/**authcontroller** |
| Resource Path | API 제공을 위해 필요한 식별자 | http(s)://www.g2b.go.kr/com/co/coa/authcontroller/**codes** |
| Path Variable | API 제공을 위해 필요한 기본적인 Parameter | http(s)://www.g2b.go.kr/com/co/coa/authcontroller/codes  /**{codeId}** |
| Query String | API 제공을 위해 보조적인 Parameter가 필요한 경우에 사용 | http(s)://www.g2b.go.kr/com/co/coa/authcontroller/  codes**?limit=10&offset=1** |

### URI 작성 표준

RESTFUL한 URI 표현을 위한 기본 가이드는 다음과 같다.

* 모든 Resource는 유일한 URI로 정의해야 한다.
* 가능하면 짧고 의미가 있는 단어로 표현하여 URI만으로 직관적으로 이해할 수 있도록 한다.
* URI는 정보의 자원을 표현하는 데 집중해야 한다. 따라서, Resource는 동사보다는 명사로 표현해야 하며 행위가 URI 표현으로 들어가지 않도록 한다.
* URI는 소문자만을 사용한다. 특히, 대소문자를 혼합해서 사용하지 않도록 한다. 대소문자에 따라 서로 다른 Resource로 인식되며, 사용 시에 혼란을 줄 수 있다.

**Sample**

* 확장자를 사용하지 않도록 한다. 특정 확장자로 제한하지 않음으로써 Resource를 다루는 데 더 유연해질 수 있다. 필요하면 확장자 대신 Accept Header를 사용하여 문제를 해결한다.
* URI는 슬래시(/)로 hierarchical하도록 구성해야 한다. 상위 경로는 하위 경로의 집합을 의미하는 단어로 구성한다
* Collection을 나타내는 경우 복수로 표현한다.
* Resource와 직접적인 관련이 없는 정보는 Query String으로 처리한다.
* URL 인코딩이 필요한 문자는 사용하지 않는다.
* Underscore(\_) 문자는 가독성이 떨어지므로 사용하지 않는다.
* Non-Query/Non-Search Resource에 대한 Query String 사용을 지양해야 한다.

### 기본 처리 유형별 Request Method

* 기본 업무처리 유형별로 사용하는 Request Method 및 Parameter는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| 처리 유형 | Request Method | Params |
| 조회 (get) | GET | - |
| 생성 (create) | POST | - |
| 변경 (update) | POST ***PUT(X)*** | method=put |
| 삭제 (delete) | POST ***DELETE(X)*** | method=delete |

**\* Request Method 중 PUT과 DELETE는 사용을 제한하기 때문에 POST와 params를 이용하여 처리한다.**

**Sample**

## 명명 STYLE

### Pascal Case

* 첫 번째 단어를 포함하여 모든 단어의 첫 문자를 대문자로 표기하는 방식
* 식별자의 각 시작 단어를 대문자로 조합 한다. (aka Upper Camel Case)

ex) 자바 클래스명 : class MyName

### Camel Case

* 여러 단어를 조합할 때 각 단어를 쉽게 구분하기 위해 첫 단어를 제외하고 첫 문자를 대문자로 표기하는 방식
* 식별자의 첫 글자는 소문자로 시작하며, 첫 단어 이외의 나머지 단어는 대문자로 시작한다.

(aka Lower Camel Case)

ex) 자바 변수명 : String myName;

자바 메소드 : public String createMyName() {} / public void setMember() {}

### Lower Case

* 식별자의 모든 단어를 소문자로 조합한다.

ex) 자바 패키지명 : package koneps.pblanc.manage.web;

### Upper Case

* 식별자의 모든 단어를 대문자로 조합한다. 단어들간의 구분자는 underscore(‘\_’)로 구분한다.

ex) ERROR\_CODE\_SUCC\_VALUE

### Hungarian Notation

* 변수 선언 시 접두어를 붙여 변수의 의미나 타입을 명확하게 인지하기 위해 만든 표기법으로 각 식별자의 데이터 형식을 이름의 앞에 붙인다. 주로 지역변수 명에 적용한다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 순번 | Prefix | Type | Example | 비고 |
| 1 | b | boolean | bStatCheck | 1 byte |
| 2 | c | char | cPrdtCd | 2 byte |
| 3 | i | int | iLoopCnt | 4 byte |
| 4 | l | long | lTotalSum | 8 byte |
| 5 | f | float | fSalaryAverage | 4 byte |
| 6 | d | double | dIncomeStats | 8 byte |
| 7 | str | String | strName | - |

## 명명 규칙

### 업무 구분 및 주제 영역

* 업무 분류표

. 명명규칙\_운영계 단위업무별 분류코드표\_메트릭스\_통합(통합).xls 를 준용한다.

* 주제 영역

. 명명규칙\_운영계 단위업무별 분류코드표\_메트릭스\_통합(통합).xls 를 준용한다.

### 공통 사항

* 분류코드 : 프로그램 소스는 업무 분류 구분을 기준으로, DB 관련내용은 주제영역 분류를 기준으로 한다.
* 업무 분류구분과 주제영역 분류코드는 틀릴 수 있다.
* 절대 Path 사용을 원칙으로 한다.
* 파일명은 확장자를 포함하여 최대 50자까지만 허용한다.
* 경로 및 확장자를 포함하여 최대 100자까지만 허용한다.
* 표준사전 사용시에는 단어 첫 글자만 대문자로 하며, 특성을 가장 잘 반영할 수 있는 약어를 조합하여 가급적 짧게 표현하되 유일한 이름을 갖도록 한다.
* 한 Directory 내의 파일 개수는 200개 이하로 한다.
* 파일은 최하위 Directory에만 위치한다.
* File Encoding은 UTF-8을, Java File의 경우에는 UTF-8을 기본으로 한다.
* 기본 해상도 화면크기는 가로 영역만 정의한다.

. 최적화면 : 1440

. 최소화면 : 1024

* 표준단어 및 영문: 엑셀파일 제공 (위치는 추후 작성)

### 클래스 파일명

* 클래스 파일명은 단어 + 클래스유형의 형태로 명명하며 단어는 명사 약어로 구성한다.
* 클래스 파일명은 최대 5개 이하의 표준단어를 조합하여 명칭을 정한다.
* 여러 단어로 구성되는 경우 첫 글자는 대문자를(**Pascal Case**) 사용하여 구별하며, 확장자를 포함하여 총 50 자를 넘지 않도록 한다.
* 파일명은 다른 파일명과 겹치지 않게 한다.

|  |  |  |
| --- | --- | --- |
| 유형 | 명명 규칙 | Example |
| Controller | 업무단어 + Controller | UserMgrController |
| Service | 업무단어 + Service | UserMgrService |
| Mapper | 단어(Ds:데이터소스 + 프로젝트명 +Bz) + Mapper | DsXxxxxxBzMapper |

### 화면단 파일명

* 파일명은 단어 + 화면유형의 형태로 명명하며 단어는 명사 약어로 구성한다.
* 화면단 파일명은 최대 5개 이하의 표준단어를 조합하여 명칭을 정한다.
* 여러 단어로 구성되는 경우 첫 글자는 대문자로(**Pascal Case**) 사용하여 구별하며, 총 50 자를 넘지 않도록 한다.

|  |  |  |
| --- | --- | --- |
| 유형 | 명명 규칙 | Example |
| 목록 조회 화면 | 단어 + L | UserMgrL |
| 상세 조회 화면 | 단어 + V | UserMgrV |
| 등록 화면 | 단어 + I | UserMgrI |
| 수정 화면 | 단어 + U | UserMgrU |
| 팝업 화면 | 단어 + P + L/V/I/U | UserMgrPV |
| 리포트 화면 | 단어 + R | UserMgrR |
| Hidden | 단어 + H | UserMgrH |
| 화면에 include 되는 화면 | 단어 + D + L/V/I/U | UserMgrDV |
| 복합(등록, 수정, 조회 등) | 단어 + M | UserMgrM |

### 함수(Method) 명

* 함수는 작업유형 + 명사 약어의 형태로 명명한다.
* 함수는 최대 5개 이하의 표준단어를 조합하여 명칭을 정한다.
* 작업유형은 소문자로 시작하고, 여러 단어로 구성되는 경우 첫 글자는 대문자로 사용하여 구별한다. (**Camel Case**)

|  |  |
| --- | --- |
| 작업 유형 | 표준 예시(Sample) |
| 조회 | **(단건) select**Method()  **(다건) select**Method**List**() |
| 입력 | **insert**Method() |
| 수정 | **update**Method()  **Sample** |
| 삭제 | **delete**Method() |
| 저장 | **save**Method() |
| get | **get**Method() |
| set | **set**Method() |
| boolean | **is**Method() |
| 복합처리 | **process**Method() |
| 검증 | **validate**Method() |
| 연계 | **interface**Method() |
| 쓰기 | **write**Method() |
| 등록/수정 | **merge**Method() |
| 등록/수정/삭제 | **multi**Method() |

* 함수명은 총 50 자를 넘지 않도록 한다.
* 함수명은 파일 내 다른 함수명과 겹치지 않게 한다.

### 변수(Variable) 명

* 변수는 prefix + 단어조합으로 구성하며, prefix는 다음과 같다.

|  |  |
| --- | --- |
| 변수 유형(Variable Type)별 Prefix | Prefix 예시(Example) |
| String str | **String** **str**MyName; |
| Char c | **Char** **c**CopyCode; |
| Int i | **Int** **i**MyWidth; |
| Float f | **Float** **f**MyWidth; |
| Long l | **Long** **l**MyWidth; |
| Double d | **Double** **d**MyWidth; |
| Boolean b | **Boolean bStatCheck** |

* 다수의 데이터를 다루는 배열형 변수를 정의하는 경우 변수명 뒤에 복수를 의미할 수있도록 변수명 뒤에 식별자 s를 붙히고, 리스트형 변수를 정의하는 경우 변수명 뒤에 식별자 List를 붙인다.

|  |
| --- |
| String[] names  List nameList; |

* 주로 제어문에서 사용되는 임시 변수들은 다음과 같이 한 글자로 사용한다.

|  |
| --- |
| i,j,k,m,n : 정수형 변수  c,d : 문자형 임시변수  x,y,z : 좌표  e, e1, ... : Exception  o : 객체  s : 문자열(String) |
| 사용 예시) for (int i = 0; i < MAX; i++) |

### 상수(Constant) 명

* 변수명 전체를 대문자로 하고 단어 연결에는 underscore(‘\_’) 를 사용한다.

ex) MINIMUM\_BALANCE

* 각 유형별 표준/가이드 및 예시(Sample)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 대분류 | 소분류 | 표준/가이드 | 예시(Sample) |
| Java | Package | - | 2~15자 내외의 소문자만 사용, Full Name으로 명명 | xxx.xxxxx.manage.web; |
| Class | General Rule | ‘\_’ 사용 금지 | class MyName |
| Abstract | Abstract+업무기능.java | AbstractMemberAction.java |
| Controller | 서비스명+Controller.java | MemberController.java |
| Service | 서비스명+Service.java | MemberService.java |
| Java | Class | VO | 모델명+VO.java | MemberVO.java |
| Exception | 명사+Exception.java | SystemNotAvailableException.java |
| Method | General Rule | 동사의 첫문자는 소문자 명사의 첫문자는 대문자 대부분 동사로 시작 | deposit()  addInterest() |
| Controller & Service class | General Rule | selectMethod()  selectMethodList()  insertMethod()  updateMethod()  deleteMethod()  saveMethod()  getMethod()  setMethod()  isMethod()  processMethod()  validateMethod()  interfaceMethod()  writeMethod()  mergeMethod()  multiMethod() |
| VO class  **Sample** | General Rule | setMethod()  addMethod()  removeMethod()  nextMethod()  getMethod() |
| Variable | General Rule | 약어 사용 자제  변수 명 만으로 의미 파악 가능하도록 함  첫 글자는 반드시 영문 소문자 사용  변수 중간에 '\_' 사용하지 말 것(상수만 허용)  여러 단어 조합시 두 번째 이후 단어는 대문자로 시작 | List<Map<String, String>> list  Map<String, String> param  String filePathNm |
| Member variable | *General Naming Rule과 동일* |
| Java | Variable | Local variable | General Naming Rule과 동일 Member variable과 동일한 이름은 자제 배열 변수는 되도록 복수형으로 사용  List변수는 List접미사 사용 Map변수는 Map접미사 사용 | int[] testScores  List<String>testScoreList  Map<String,Adress> userMap |
| Parameter variable | *Local variable와 동일* |
| Constant | General Rule | 상수 선언은 반드시 static final 사용  대문자만 사용  단어 사이는 '\_'사용 첫 글자에 '\_'나 '$' 사용 금지  누구나 이해할 수 있는 영문 사용 | DEFAULT\_DOWNLOAD\_URL =”http://repo.maven.apache.org/” |
| Others | SqlMap | General | SQL-Map XML은 기본적으로 Service 당 1개 생성(필요시 예외 허용)  **Sample**  파일명은 Service class명 앞에 dsBz+업무 단어를 붙혀 주고, 뒤에 Service 단어 삭제 후, \_SQL접미사 사용 | Service Class 명:  XxxxManageService  SqlMapper파일명:  dsBz + XxxxManage + \_SQL  (dsBzXxxxManage\_SQL.xml) |
| XML ID | 기본적으로 Method부분과 동일 namespace 지정 필수 namespace명은 Service Class에서 지정된 SQL\_PREFIX 명칭을 사용  ID는 실제 호출 Service 클래스의 메소드 명과 동일하게 작성 | namespace=”xxxx.manage.”  Service Class에서 지정된 SQL\_PREFIX=”xxxx.manage”  id="selectXxxxList" |
| XML Tag:  <parameterType> | 객체의 프로퍼티 값이 SQL문의 입력 파라미터에 지정,  패키지 이름을 포함한 클래스 이름 | parameterType=”xxx.xxxx.  manage.vo.XxxxManageListVO”  또는,  parameterType=”egovMap” |
| Others | SqlMap | XML Tag:  <resultType> | SQL 실행 결과를 담을 객체,  패키지 이름을 포함한 클래스 이름 | resultType=”xxx.xxxx.  manage.vo.XxxxManageListVO”  또는,  resultType=”egovMap” |
| yml | - | 단어 사이 hyphen('-') 사용 | application-local.yml |
| xml | - | 단어 사이 hyphen('-') 사용 | mybatis-config.xml |

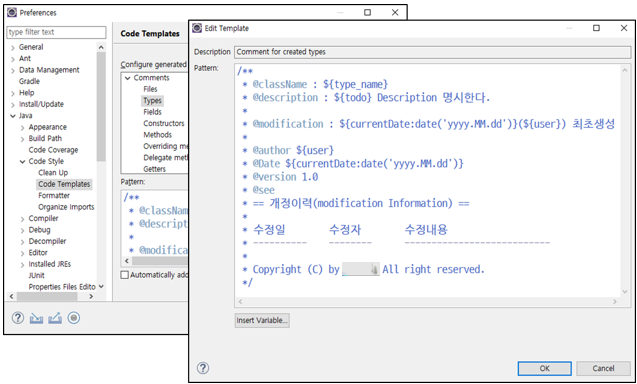
## 주석 처리

### 문서 주석

* 모든 클래스와 UI 관련 파일은 상단에 반드시 주석을 정의한다.
* 클래스에 대해 기술하고 확장 클래스에 대해서는 @see 항목을 반드시 정의한다.

|  |
| --- |
| **/\*\***  **\***  **\* @className** : **${type\_name}**  **\* @description : ${todo} Description 명시한다.**  **\***  **\* @modification : ${currentDate:date('yyyy.MM.dd')}(${user}) 최초생성**  **\***  **\* @author** **${user}**  **\*** **@Date** **${currentDate:date('yyyy.MM.dd')}**  **\* @version 1.0**  **\* @see**  **\***  **== 개정이력(Modification Information) ==**  **\***  **\* 수정일**▨▨▨▨▨▨▨▨**수정자**▨▨▨▨▨▨▨▨**수정내용**  **\* ----------**▨▨▨▨**--------**▨▨▨▨▨▨**---------------------------**  **\***  **\* Copyright (C) by 시스템명 All right reserved.**  **\* /**  **※ 수정일 ~ 수정자 ~ 수정내용 사이 간격은 2TAP(8칸) 이고, 좌측 정렬 이다.** |

* 주석 표준 적용 방법 : 다음 내용을 Eclipse 설정을 통해서 적용한다
* Window > Preferences > Java > Code Style > Code Templates 선택 후 > 오른쪽 창에서 Comments > Types 선택 > Edit... 버튼 클릭 > Edit Template 창의 Pattern에 위의 내용 입력 후 OK 버튼 클릭



* 적용된 주석 코드를 파일에 적용하려면 클래스 명을 선택하고 alt + shift + J 누른다.

### 함수 주석

* 모든 함수에도 반드시 함수에 대한 주석 정보를 입력한다.
* 함수 주석 표준 적용 방법 : Window > Preferences > Java > Code Templates 선택 후 > 오른쪽 창에서 Comments > Methods 선택 > Edit... 버튼 클릭 > Edit Template 창의 Pattern에 다음 내용 입력 후 OK 버튼 클릭

|  |
| --- |
| /\*\*  \* ${todo} Description 명시한다.  \* ${tags}  \*/ |

* 함수 명을 선택한 후 alt + shift + J 를 누르면 기본 주석이 입력된다.

|  |
| --- |
| /\*\*  \* 페이징을 위한 총 목록 수를 조회한다.  \* @param searchMap - 검색조건  \* @return int - 총갯수  \* @throws Exception  \*/ |

### 변수 주석

* 변수 선언 시 해당 변수에 대한 설명을 다음과 같이 Single Line Comment로 처리한다.

|  |
| --- |
| int iPageCount = 0; //페이지 개수  String strUserName = “”; // 사용자명 |

### 제어 구조 주석

* If, for, while 등의 제어문의 첫 line에 Single Line Comment로 간단한 설명을 주석 처리하며, 2가지 이상의 상이한 제어문이 중첩될 경우 각 제어문의 마지막 line에 Single Line Comment로 제어문이 끝남을 주석 처리한다.

|  |
| --- |
| for(...) { //사용자 id 가 계속 증가한다  for (...) { //사용자 방문자수가 하나씩 증가한다  while (...) { //포인트가 0이 될때까지  if(...) { //상담 신청을 했다면..  switch (...) { //담당자라면..  ...  } // end switch  } // end if  } // end while  } // end for j  } // end for |

### MyBatis SQL 주석

* 모든 쿼리 id별로 주석을 정의한다.

|  |
| --- |
| <!--  쿼리명 : "selectAuthList" (권한목록 조회)  설 명 : 사용자별 권한 목록을 조회한다.  parameterType = xxxxx.com.co.coa.vo.AuthVO (검색조건)  resultType = xxxxx.core.util.egovMap  수정일▨▨▨▨▨▨▨▨수정자▨▨▨▨▨▨▨▨수정내용  ==========▨▨▨▨========▨▨▨▨▨▨==============  2021.12.02▨▨▨▨홍길동▨▨▨▨▨▨▨▨최초 생성  -->  <select id=*"selectAuthList"* parameterType=*"xxxxx.com.co.coa.vo.AuthVO"* resultType=*"egovMap"*> |

* 쿼리 실행시 디버깅 정보를 필수도 정의한다.

|  |
| --- |
| <mapper namespace=*"co.coa.auth"*>  <select id=*"selectAuthList"* parameterType=*"xxxxx.com.co.coa.vo.AuthVO"* resultType=*"egovMap"*>   * 작성 방법 : xml 별 namespace + sql Id   예시)  /\* co.coa.auth.selectAuthList \*/  SELECT A.AUTH\_CD |

### 기타 주석 처리

* 업무 로직 구현 시 클래스 내 다른 클래스 호출 시 또는 함수 내에서 다른 함수 호출 시에는 반드시 Single Line Comment 처리를 한다
* 업무 로직 구현 시 중요한 소스 코드나 복잡한 소스 코드는 Single Line Comment로 각 로직에 설명을 추가한다. 그러나 너무 상세한 주석 처리 보다는 코드를 최적화하여 직관적으로 이해할 수 있는 코드를 작성하는 것이 원칙이다.
* 사용자 요청 또는 업무 로직 변경으로 인하여 프로그램 내의 일부 로직을 삭제할 경우 가급적 직접적으로 삭제하지 않고 블록 주석을 달아 사용자에게 보여지지 않도록 처리한다

페이지의 마지막 줄

## Annotation 적용

* Annotation(@)의 사전적 의미는 주석이라는 뜻이다.
* 코드 사이에 주석처럼 쓰이며, 특별한 의미 기능을 수행하도록 하는 기술 이다.
* Spring Framework는 각각의 빈(Bean)들을 관리하기 위한 설정을 XML, JAVA CONFIG 혹은 Annotation으로 한다. XML로 관리할 경우 때때로 방대하고 복잡한 속성 파일들로 인해 시스템 개발 및 유지보수의 지연을 초래할 가능성이 높아진다.
* 따라서 본 프로젝트에서는 Spring Framework 개개의 빈(Bean) 설정을 Annotation으로 관리한다.
* Annotation은 클래스, 메서드 등의 상단에 필요한 정보를 설정함으로써 Spring Framework Container에서 기동할 때 해당 정보를 설정 정보가 아닌 클래스 정보에서 파싱해서 활용한다.
* Annotation 태그는 org.springframework.beans.factory.annotation 패키지에 정의되어 있다.

### 어노테이션 사용 순서

* 어노테이션을 정의 한다.
* 클래스에 어노테이션을 배치 한다.
* 코드가 실행되는 중에 Reflection을 이용하여 추가 정보를 획득하여 기능을 실행 한다.

### Reflection 이란

* 프로그램 실행 중에 자신의 구조와 동작을 검사, 조사하고 수정하는 것 이다.
* 개발자가 데이터를 보여주고, 다른 포멧의 데이터를 처리하고 통신을 위해 Serialization(직렬화)를 수행하고, Bundling(하나의 파일로 모아줌)을 하기위해 일반 소프트웨어 라이브러리를 만들도록 도와 준다.
* Reflection을 이용하면 Annotation 지정만으로 원하는 클래스를 주입할 수 있다.

### 주요 어노테이션 태그 설명 및 예시

|  |  |
| --- | --- |
| Annotation 태그명 | 설명 / 사용 예시(Example) |
| @Service  (클래스 단위선언) | Service 클래스의 구현체 정의 시 사용된다. 즉, @Service를 적용한 클래스는 비지니스 로직이 들어가는 Service로 등록된다.  - 사용 예)  @Service("sampleService")  public class SampleService extends HiraAbstractService { |
| @Override  (함수 단위선언) | @Override annotation은 해당 요소가 가장 상위 클래스에서 선언된 한 요소를 오버라이드 할 것임을 컴파일러에게 알려준다. @Override로 표시된 메서드가 상위 클래스에 있는 메서드를 오버라이드 하는데 실패할 경우 컴파일러는 에러를 발생시킨다.  - 사용 예)  @Override  public List selectSamples(Map searchMap) throws Exception{  List list = sampleMapper.selectList(“xxx.xxxxx”,searchMap);  return list;  } |
| @Controller  (클래스 단위선언) | 해당 클래스가 Controller임을 나타내기 위해 사용된다. @Controller Annotation을 붙여주면 스프링이 controller 클래스로 등록한다.  - 사용 예)  @Controller  public class SampleController extends HiraController{ |
| @Resource  (클래스 단위선언) | 어플리케이션에서 필요로 하는 자원을 자동 연결 할 때 사용한다. 스프링에서는 의존하는 빈 객체를 전달할 때 사용된다. (Bean ID를 이용한 DI(Dependency Injection) 지원) 즉, @Resource로 선언해서 사용하면 기존의 new로 선언해서 사용할 필요가 없다.  - 사용 예)  @Resource (name = "sampleService")  private SampleService sampleService;  List list = sampleService.selectSample(); |
| @Autowired  (클래스 단위선언) | 어플리케이션에서 필요로 하는 자원을 자동 연결 할 때 사용한다. 스프링에서는 의존하는 빈 객체를 전달할 때 사용된다.(Bean ID를 이용한 DI 지원) 즉, @Autowired로 선언해서 사용하면 기존의 new로 선언해서 사용할 필요가 없다.  - 사용 예)  @Autowired  private SampleService sampleService;  List list = sampleService.selectSample(); |
| @RequestMapping  (함수 단위선언) | HTTP Request에 대한 매핑 정보를 받아서 바로 해당 Controller 클래스의 해당 Method로 지정한다.  - 사용 예)  @RequestMapping(“/co/com/Noti/selectSample.do")  public String selectSample() throws Exception {  → UI에서 /sample/selectSample.do URL이 호출될 때 selectSample(...) 함수가 실행 된다. |
| @SuppressWarnings  (함수 단위선언) | List, Map, Entry, HashMap 등에서 불필요한 캐스팅을 막기 위해 Generics에서 parametric types를 지정해 주도록 하기 위해서 Warning을 띄우는데, 이 경고를 없애기 위해 사용한다  - 사용 예)  @SuppressWarnings("unchecked")  public Map<String, Object> selectAuthByAuthCd(String authCd) throws Exception {  ...  }  → 기본 해결책은 명시적 타입 사용을 원칙으로 한다.  예) HashMap<String, Object>  List<HashMap<String, Object>> |
| @Deprecated  (함수 단위선언) | 특정 클래스나 인터페이스, 메서드, 필드 등이 앞으로 더 이상 사용되지 말아야 한다는 것을 경고하기 위해서 사용한다. 컴파일러는 deprecated된 메서드나 클래스 혹은 변수를 사용할 때마다 경고를 발생 시킨다. |
| @RequestParam | HTTP 요청 파라미터를 Controller 메서드의 파라미터와 매핑하기 위해서 사용된다.  - 사용 예)  public String hello(@RequestParam("name") String name) throws Exception {  → UI dataset 형태의 파라미터를 주고 받아야 하기 때문에 이번 프로젝트에서는 사용하지 않는다. |
| @Param  (Controller 함수 단위선언) | HTTP 요청 파라미터를 Controller 메서드의 파라미터와 매핑하기 위해서 사용된다.  - 사용 예)  @RequestMapping("/test.do")  **Public** ModelAndView test(@Param Map<String, Object> paramMap,ModelAndView model) { |
| @PathVariable | HTTP 요청에 대해 매핑되는 request parameter 값이 자동으로 바인딩되며, URI에서 각 구분자에 들어오는 값을 처리해야 할 때 사용된다.  - 사용 예)  Public boolean deletePost(@PathVariable("idx") int postNum) {  return postService.deletePost(postNum);  }  → method parameter 앞에 사용하면서 해당 URL에서 {특정값}을 변수로 받아 올 수 있다.  → REST API에서 값을 호출할 때 주로 많이 사용한다. |
| @RequestBody | 요청이 온 데이터를 바로 Class나 Model로 매핑하기 위해서 사용된다.  - 사용 예)  @PostMapping(“/pblancs”)  public ModelAndView insert(@RequestBody PblancManageVO PblancManageVO, ModelAndView model) {  PblancManageVO resultVO = pblancManageService.insertPblanc(PblancManageVO)  → HTTP POST 요청에 대해 request body에 있는 RequestData를 바로 Model 이나 클래스로 매핑 한다. |
| @Auth  (Controller 함수 단위선언) | 세션 체크를 위해 사용된다.  @Auth(role = Role.***NONE***) //세션 없이 접근  - 사용 예)  @RequestMapping("/test.do")  **Public** ModelAndView test(@Param Map<String, Object> paramMap,ModelAndView model) {  Role 속성에 따라 세션 및 권한 체크가 다르게 동작 한다.  Role {***ADMIN***, ***USER***, ***NORMAL***, ***NONE***}  @Auth가 없을 경우 Role. ***NORMAL*** 과 동일하게 동작한다.  **Sample**  **※ 참고 사항**  1) @Auth 어노테이션은 필수로 선언해야 한다.  2) Role 속성은 다음과 같다.  ① ADMIN : 개인정보 또는 관리자만 접근해야 하는 경우  ② USER : 사용자 인터페이스(사용자가 실행)로 접근해야 하는 경우  ③ NORMAL : 코드성(공통코드 조회 등) 데이터 접근을 해야 하는 경우  ④ NONE : 세션 없이 접근하는 경우 |

* @Service, @Repository의 annotation명은 해당 클래스명의 첫 글자를 소문자로 하여 지정한다.

페이지의 마지막 줄

## CODING STYLE

### Indent (들여쓰기)

* Indent 는 tab을 사용한다.
* indent의 크기는 4로 한다. (4개의 공백문자를 들여쓰기의 기본 단위로 사용)
* 한 Line은 160문자 내에서 작성하길 권장 한다.

(예시)

public class Sample

{

▨▨▨▨private int size = 100;

▨▨▨▨//Statements

}

* 한 Line을 초과하는 경우에는 콤마 후 또는 연산자 앞에서 line을 변경하며, 새로운 line의 들여쓰기는 이전 line의 expression 부분과 같은 레벨로 들여쓰기

(예시)





int result = a + b + c + d + e + f + g + h + i + j + k + l

+ n + o + p;

someMethod(int anArg, Object anotherArg,

String yetAnotherArg, Object andStillanother)

{

▨▨▨▨//Statements

}

* 같은 레벨로 들여쓰기가 어려울 경우, 8칸 들여쓰기

(예시)

private static synchronized horkingLongMethodName (int anArg,

▨▨▨▨▨▨▨▨Object anotherArg, String yetAnotherArg,

▨▨▨▨▨▨▨▨Object andStillAnother)

{

▨▨▨▨//Statements

}

* If 문장의 조건절 line 변경 시, 첫 문장부터 8칸 들여쓰기. 8칸이 어려울 경우에는 4칸 들여쓰기

(예시)

if ((condition1 && condition2)

▨▨▨▨▨▨▨▨|| (condition3 && condition4)

▨▨▨▨▨▨▨▨|| ! (condition5 && condition6)) {

▨▨▨▨doSomethingAboutIt();

}

* “result (condition) ? value1 : value2” 문장이 한 line을 초과하는 경우, “:”를 “?”에 맞추어 들여쓰기

(예시)



alpha = (aLongBooleanExpression) ? beta

: gamma;

### Space

* 한 줄에 하나의 Statement만 기술한다.
* semicolon, comma, 예약어 뒤에는 space를 둔다.

(예시)

someMethod(param1,▨param2,▨param3);

* unary(단항) operation은 space를 두지 않는다. (ex, i++;)
* binary operation은 양쪽에 space를 둔다. (ex, i = i + 1;)
* .을 제외한 모든 binary 연산자는 연산자들로부터 space로 분리되어야 한다.

(예시)

a▨+=▨c▨+▨d;

a▨=▨(a▨+▨b)▨/▨(c▨\*▨d);

while▨(isStop) {

▨▨▨▨n++;

}

* 괄호 안에 괄호가 있는 경우 괄호 사이에 space를 두지 않는다.
* 괄호와 함께 나타나는 키워드는 space에 의해 나누어져야 한다.

(예시)

while▨(true) {

//Statements

}

* 메소드 이름과 메소드의 여는 괄호 사이에 빈 space가 사용되어서는 안 된다.

(예시)

public static String getDefaultCurrentDateTime() {

return FastDateFormat.getInstance("yyyyMMddHHmmss").format(new Date());

}

* for 문장 내의 표현식은 space로 분리 한다.

(예시)

for▨(expr1;▨expr2;▨expr3)

### Brace (중괄호)

* ‘{‘ 기존의 '{'와 비교해서 indent(tab)를 준다.
* '}'는 짝이 되는 '{'와 동일하게 indent(tab)를 준다.
* brace에 주석을 기입하는 경우 '//' 주석을 사용한다.

(예시)

Class Sample extends Object {

▨▨▨▨int var1;

Int var2;

Sample(int i, int j) { //Statements

var1 = i;

var2 = j;

▨▨▨▨}

…

}

### 용어

* 용어는 원칙적으로 메타데이터 관리시스템의 표준용어를 준수하며 표준단어의 단어영문명을 사용한다.
* 하나의 단어를 사용하는 경우에는 용어에 해당하는 영문명을 사용한다.
* 여러 개의 단어를 조합하여 사용하는 경우에는 단어영문명을 조합해서 사용한다.
* 용어를 사용할 때 동음이의어에 주의한다.

### Logging

* Log는 반드시 Framework에서 제공한 Logger만 사용한다.
* Log는 debug, info, warn, error로 구별하여 사용한다.
* Log는 반드시 발생 시간과 위치, 그리고 내용을 포함한다.
* Log는 한 줄만 출력한다. (debug log 제외)
* debug log는 개발자가 개발 시에만 사용하고, 운영 중에는 사용하지 않는다.
* info log는 운영자에게 도움이 될 내용을 기록한다.
* warn log는 error는 아니나 잠재적인 error의 발생이 가능한 내용을 기록한다.
* error log는 error code와 함께 error에 대한 내용을 기록한다.

**Sample**

**[logback 사용시 주의 사항]**

* logback을 사용하여 로깅 하는 경우에는 아래와 같이 로그 레벨을 확인하는 구문을 반드시 입력하도록 한다.

(예시)

List<BoardVO> boardList = (List<BoardVO>)sqlMapClientTemplate.queryForList("getBoardList");

log.debug(“Ouput데이터 출력 : ”+boardList);

### File 명명 및 Line

* File 이름은 확장자를 포함하여 50자를 넘지 않도록 한다.
* File 명은 명명규칙을 참고하여 메타데이터 관리시스템의 표준용어를 사용하여 작성하며, 한글에 해당하는 영문 표기나 영어 약어 이외의 특수 기호 및 숫자는 사용하지 않는다.
* File의 크기는 1000 Line을 넘지 않는다. (주석 제외)
* function(Method)의 크기는 100 Line을 넘지 않는다. (주석 제외)
* Statement의 크기는 160 Character를 넘지 않는다.

페이지의 마지막 줄

## UI 구조

### 프로젝트 구조

* 프로젝트는 시스템 별로 하나로 생성, 관리된다. 즉, 시스템 별 xml 파일이 하나씩 생성, 관리된다. 따라서 각 시스템 별 개발자들은 같은 WebSquare5 프로젝트 파일(xxx.xml) 파일을 열어서 작업하게 된다.
* 프로젝트의 UI 구조는 다음과 같으며, \프로젝트명\src\main\main\ 폴더 하위에 sm폴더가 소스 폴더이며, 화면을 구성하는 파일구조는 Java Package와 동일하다.

**Sample**

### WebSquare 구성

* 화면 구조는 다음과 같다.

### 화면 구조

* 일반 웹 브라우저로 구성되는 프레임 구조

### DataCollection 구조

* WebSquare HTML5에서 제공하는 DataCollection은 두 종류로 분류되는데, 하나는 key=value의 쌍으로 이루어져 있는 parameter 형태와 key 값의 헤더와 Value와 data 를 리스트 형태로 가지고 있는 DataCollection으로 구성된다.

페이지의 마지막 줄

## UI 개발 표준

### WebSquare

* XXXXX\_COM\_UI개발표준 정의서 문서를 참고 한다.

페이지의 마지막 줄

### Rexpert

* Report 구성에 필요한 OOO을 사용한다.

페이지의 마지막 줄

## Presentation 계층 개발

* 클라이언트의 요청에 대해 직접적인 서비스를 수행하는 Presentation Logic, 권한, 인증, 보안, 세션관리, User Interface 등을 제공하며 Business 서비스로의 접근을 제어한다.
* 본 절에서는 Controller의 개발 방법과 UI로 데이터 전송을 위한 개발 방법을 가이드 한다.

### Controller 개발 표준

* Controller는 사용자가 웹 브라우저를 통해 전송한 Request를 받아서 명령을 처리하는 역할을 수행하며 주요 처리는 service로 한다.
* 사용자의 Request는 DispatcherServlet으로 전달되고, DispatcherServlet에서 서블릿을 통해 Request를 전달 받아서 해당 Request를 처리할 Controller를 생성하여 호출한다. 처리된 결과는 Controller에서 사용자에게 전달한다.
* 업무 유스케이스 당 이를 처리하는 Controller 클래스를 정의한다.
* 업무 특성상 분리가 필요한 경우는 Controller를 추가로 작성한다.
* Controller는 Service 를 하나 이상 호출 할 수 있으며, Mapper를 직접 호출 할 수 없다.

(특별한 경우가 아니면 Controller에서 Service는 한번만 사용하는 것을 원칙으로 한다. Controller에서 여러 개의 Service를 사용할 경우 Connection Pool을 Service당 한 개씩 사용하여 과도하게 Connection Pool을 사용하게됨)

* Controller 클래스는 src/main/java/프로젝트명/업무영역1depth(2자리)/업무영역2depth(3자리)/단위업무 코드(4자리)

**Sample**

/web 패키지 하위에 생성한다.

ex) src/main/java/xxxxx/fc/fcd/fcdb/web/….java

. xxxxx – OOO시스템 (프로젝트명)

. 업무영역 1depth(2자리): fc – 업무공통

. 업무영역 2depth(3자리): fcd – 서비스운영관리

. 단위업무 코드(4자리): fcdb – 사용자/권한관리

* 클래스 명은 xxxController.java로 하며 대문자로 시작한다.

ex) CodeMgrController.java

* Controller 클래스는 다음과 같은 순서로 작성 한다.

. 패키지 선언

. import문

. 클래스 주석

. Controller Annotation 선언

. Controller 클래스 시작

. Resource Annotation 선언 (@Autowired 등)

. 함수(Method) 주석

. Auth(역할 속성) 및 PostMapping Annotation 선언

. Request 처리 Method 작성

. Controller 클래스 종료

**[샘플 예시]**

|  |
| --- |
| **package** xxxxx.fc.fcd.fcdb.web;  **import** java.util.List;  **import** java.util.Map;  **import** javax.servlet.http.HttpSession;  **import** javax.validation.Valid;  ①annotation 사용을 위해 관련 클래스를 import 한다.  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.GetMapping;  **import** org.springframework.web.bind.annotation.PostMapping;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.servlet.ModelAndView;  ② Controller에서 호출할 Service와 상속해서 사용할 클래스를 import 한다.  **import** xxxxx.com.co.coa.service.AuthService;  **import** xxxxx.com.co.coa.vo.AuthVO;  **import** xxxxx.core.config.annotation.Param;  **import** xxxxx.core.util.MessageUtils;  ③ 반드시 클래스에 대한 주석 정보를 작성한다.(클래스 명 선택 후, alt + shift + J)  /\*\*  \*  \* **@className** : AuthController  \* **@description** : **TODO** 권한 관리를 위한 Controller  \*  \* **@modification** : 2022.02.10(홍길동) 최초생성  \*  \* **@author** 홍길동  \* **@Date** 2022.02.10  \* **@version** 1.0  \* **@see**  \* == 개정이력(Modification Information) ==  \*  \* 수정일 수정자 수정내용  \* ---------- -------- ---------------------------  \*  \* Copyright (C) by xxxxx All right reserved.  \*/  ④ RestController로 선언한다. WAS 동작 시 해당 Class가 RestController로 동작하도록 메모리에 할당한다.  @RestController  @RequestMapping("/api")  **public** **class** AuthController {  ⑤ 사용할 Service 객체(클래스)를 자동으로 와이어링 해준다.  @Autowired  AuthService authService;  ⑥ 모든 함수에 대해서도 반드시 주석 정보를 작성한다.  /\*\*  \* TODO 검색조건에 의한 권한 목록을 조회한다.  \* **@param** AuthVO  \* **@param** model  \* **@return**  \*/  ⑦@PostMapping을 사용하여 해당 함수 호출을 위한 URI를 설정하고, request 정보가 담긴 @Param과 Result 정보가 담긴 ModelAndView를 사용하여 함수를 작성한다. 추가로 Role 속성에 맞게 @Auth 어노테이션을 필수로 선언하여 준다.  @Auth(role = Role.***ADMIN***)  @PostMapping("/co/coa/Auth/selectAuthList.do")  **public** ModelAndView selectAuthList(@Param(name = "params") AuthVO authVO, ModelAndView model) {  ⑧ 위에서 선언한 service를 호출하여 처리한 결과 데이터를 리턴 받고, 결과 데이터를 UI에 전달하기 위해 Response Data에 PUT 한다.  model.addObject("result", authService.selectAuthList(authVO));  ⑨ UI로 리턴하기 위한 View Resolver를 설정 한다.  **return** model;  }  } |

### UI-Controller Data 처리 표준

* @RequestMapping에 의해 호출되는 함수는 @Param과 ModelAndView의 인자를 가진다. @Param은 UI에서 전송된 데이터 정보를 포함하며, ModelAndView는 UI로 전송할 return data 정보이다.

|  |
| --- |
| @RequestMapping(value = "/co/coa/Noti/selectSamples.do")  **public** ModelAndView selectSamples(@Param Map<String, String> paramMap, ModelAndView model) {  ...  } |

* UI에서 받은 파라미터 데이터 추출 :

UI에서 전송한 parameter 데이터들을 @Param의 get() 함수를 이용해서 추출한다. 추출한 데이터는 ModelAndView 형태로 return 된다.

|  |
| --- |
| 1. Map<String, String> paramMapTemp = paramMap.get ("dsParam"); 2. String wrkId = (String) paramMapTemp.get("pWrkId"); 3. List<Map<String, String>> list = paramMap.get("dsList");   **for** (Map<String, String> map : list){  } |

* UI로 return할 단건 DataCollection 설정 :

addObject ("UI에서 받을 DataCollection 명", "return할 Map 명") 함수를 이용해서 UI로 return할 DataCollection 을 설정한다.

|  |
| --- |
| model.addObject("key1", paramMapTemp); |

* UI로 return할 다건 DataCollection 설정 :

addObject ("UI에서 받을 DataCollection 명", "return할 Object 명") 함수를 이용해서 UI로 return할 DataCollection 을 설정한다.

|  |
| --- |
| model.addObject("key1", list); |

## Business계층 개발

* 클라이언트로부터 요청 받은 business logic을 처리하는 layer로써, Persistence Layer에 데이터를 요청하고 결과를 받아서 Presentatoin Layer에 보내는 역할을 수행한다.
* 본 절에서는 Service 개발 방법을 가이드 한다.

### Service 개발 표준

* Service 클래스로써, 실제 업무관련 로직을 구현한다.
* Service 에서는 Mapper 또는 다른 Service 를 호출할 수 있다.
* Service 클래스는

src/main/java/koneps/업무영역 1depth/업무영역 2depth(2자리)/단위업무(3자리)/service패키지 하위에 생성한다.

ex) src/main/java/xxxxx/com/co/coa/service/…Service.java

. xxxxx – OOO시스템 (프로젝트명)

. 업무영역 1depth: com – 공통영역

. 업무영역 2depth(2자리): co – 공통관리

. 단위업무(3자리): coa – 공통권한관리

* 클래스 명은 xxxService.java로 하며 각 단어는 대문자로 시작한다.

**Sample**

ex) CodeMgrService.java

* transaction은 Service의 method 단위로 관리된다. method 명에 따라 transaction이 관리 되므로 method 명 작성시 반드시 표준을 준수해야 한다.

(\*3.13.8. Transaction 표준 > Transaction 기준표 참조)

* Service 클래스는 다음과 같은 순서로 작성 한다.

. 패키지 선언

. import문

. 클래스 주석

. 데이터 유효성 검증 어노테이션인 @Validated 와 @Service 어노테이션 선언

. Service 클래스 시작

. Resource Annotation 선언 (@Autowired 등)

. Business 처리 Method 작성

. Service 클래스 종료

**[샘플 예시]**

|  |
| --- |
| **package** xxxxx.com.co.coa.service;  **import** java.util.HashMap;  **import** java.util.List;  **import** java.util.Map;  **import** javax.validation.Valid; ① annotation 사용 및 Mapper 사용을 위해 관련 클래스를 import 한다. **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** org.springframework.validation.annotation.Validated;  **import** xxxxx.com.co.coa.vo.AuthVO;  **import** xxxxx.com.mapper.DsXxxxxBzMapper;  **import** xxxxx.core.constants.Constants;  **import** xxxxx.core.util.StringUtil;  ② 반드시 클래스에 대한 주석 정보를 작성한다. (클래스 명 선택 후, alt + shift + J)  /\*\*  \*  \* **@className** : AuthService  \* **@description** : **TODO** 권한 관리를 위한 Service  \*  \* **@modification** : 2022.02.10(홍길동) 최초생성  \*  \* **@author** 홍길동  \* **@Date** 2022.02.10  \* **@version** 1.0  \* **@see**  \* == 개정이력(Modification Information) ==  \*  \* 수정일 수정자 수정내용  \* ---------- -------- ---------------------------  \*  \* Copyright (C) by xxxxx All right reserved.  \*/  ③ 데이터 유효성 검증을 위한 Validated 어노테이션 과 Service 어노테이션을 선언한다.  @Validated  @Service  **public** **class** AuthService {  ④ Service에서 사용할 Mapper를 정의한다.  @Autowired  DsXxxxxBzMapper dsXxxxxBzMapper;  ⑤ SQL\_PREFIX 변수를 정의한다. (\*SQL\_PREFIX는 SqlMap에서 사용할 namespace명이 된다.)  **private static final** String **SQL\_PREFIX =** “co.coa.auth.”**;**  ⑥ 모든 함수에 반드시 주석을 작성한다.  /\*\*  \* **TODO** 권한 목록을 조회한다.  \* **@param** paramMap  \* **@param** session  \* **@return**  \*/  ⑦ Controller에서 호출해서 사용할 Service 함수를 구현한다. 이 함수에서 모든 Business Logic을 구현한다. (샘플 예시는 AuthVO 데이터에 대한 데이터 유효성 검증을 위해 @Valid 어노테이션을 추가 선언 하였다.)  **public** List<Map<String, String>> selectAuthList(@Valid AuthVO authVO) {  **return** dsXxxxxBzMapper.selectListForPaging(***SQL\_PREFIX***+"selectAuthList", authVO);  }  } |

* 복합(insert/update/delete) 작업 수행 시 :

1. List 크기를 확인 한다.
2. List의 크기만큼 반복문을 수행한다.
3. 각 row의 ROW\_TYPE 값을 통해서 작업 유형(INSERT, UPDATE, DELETE)을 확인한 후 해당하는 Mapper 함수를 호출한다.

|  |
| --- |
| /\*\*  \* 샘플 목록을 저장한다.  \* **@param** stdCdList 샘플목록  \*/  **private static final** String ***SQL\_PREFIX* =** “co.coa.auth.”**;**  **public** Map<String, String> multiStdCds(List<Map<String, String>> list) {  int insertCnt = 0; //등록 성공 수  int updateCnt = 0; //수정 성공 수  int deleteCnt = 0; //삭제 성공 수   1. **if** (list != **null** && list.size()>0) { 2. **for** (Map<String, String> map : list) { 3. **if** (map.get(Constants.KEY\_ROW\_TYPE)==Constants.ROW\_TYPE\_INSERT) {   insertCnt += dsXxxxxBzMapper.insert(***SQL\_PREFIX***+ “쿼리ID”, map);  }  **else** **if**(map.get(Constants.*KEY\_ROW\_TYPE*)==Constants.*ROW\_TYPE\_UPDATE*) {  updateCnt += dsXxxxxBzMapper.update(***SQL\_PREFIX***+“쿼리ID”, map);  }  **else** **if**(map.get(Constants.*KEY\_ROW\_TYPE*)==Constants.*ROW\_TYPE\_DELETE*) {  deleteCnt += dsXxxxxBzMapper.delete(***SQL\_PREFIX***+ “쿼리ID”, map);  }  }  }  Map<String, String> resultMap = **new** HashMap<String, String>(); //등록,수정,삭제 성공 수 담아서 넘길 map  resultMap.put("insert", Integer.*toString*(insertCnt));  resultMap.put("update", Integer.*toString*(updateCnt));  resultMap.put("delete", Integer.*toString*(deleteCnt));  } |

### VO(Value Object) 개발 표준

* VO 클래스로써, BaseObject를 상속받아서 Data Transfer를 위한 Value Object를 구현한다.
* 각 업무 프로젝트에서는 제공되는 Basic VO를 extends하여 기본이 되는 VO attribute를 상속받고 추가로 관리하는 attribute를 선언하여 사용하도록 한다.
* DB Entity와 매핑되는 Basic VO 클래스는 테이블명 및 컬럼명과 동일한 클래스명 과 attribute를 갖는다.
* VO 클래스는

src/main/java/koneps/업무영역 1depth/업무영역 2depth(2자리)/단위업무(3자리)/vo 패키지 하위에 생성한다.

ex) src/main/java/xxxxx/com/co/coa/vo/…VO.java

. xxxxx – OOO시스템 (프로젝트명)

**Sample**

. 업무영역 1depth: com – 공통영역

. 업무영역 2depth(2자리): co – 공통관리

. 단위업무(3자리): coa – 공통권한관리

* 클래스 명은 xxxVO.java로 하며 각 단어는 대문자로 시작한다.

ex) AuthVO.java

* VO 클래스는 다음과 같은 순서로 작성 한다.

. 패키지 선언

. import문

. 클래스 주석

. lombok Annotation 선언 (@Data 등)

. VO 클래스 시작

. Validation Annotation 선언 (@NotNull, @NotEmpty, @NotBlank 등)

. 필요한 Attribute 선언

. VO 클래스 종료

**[샘플 예시]**

|  |
| --- |
| **package** xxxxx.com.co.coa.vo;  ① Validation Annotation 사용을 위해 관련 클래스를 import 한다.  **import** java.io.Serializable;  **import** javax.validation.constraints.NotBlank;  **import** org.apache.poi.util.NotImplemented;  **import** com.drew.lang.annotations.NotNull;  ② Extends 하여 상속 받을 Basic VO 클래스를 import 한다.  **import** xxxxx.core.vo.PageVO;  ③ Lombok 의 @Data 어노테이션 사용을 위해 관련 클래스를 import 한다.   * 클래스명 위에 @Data 어노테이션을 명시해줌으로써 getter, setter, equals 와 같은 메소드를   따로 작성해야 하는 번거로움을 덜어준다.  **import** lombok.Data;  ④ 반드시 클래스에 대한 주석 정보를 작성한다. (클래스 명 선택 후, alt + shift + J)  /\*\*  \*  \* **@className** : AuthVO  \* **@description** : **TODO** 권한 관리를 위한 VO  \*  \* **@modification** : 2022.02.10(홍길동) 최초생성  \*  \* **@author** 홍길동  \* **@Date** 2022.02.10  \* **@version** 1.0  \* **@see**  \* == 개정이력(Modification Information) ==  \*  \* 수정일 수정자 수정내용  \* ---------- -------- ---------------------------  \*  \* Copyright (C) by xxxxx All right reserved.  \*/  ⑤ 클래스명 위에 @Data 어노테이션을 선언해 준다.  @Data  **public** **class** AuthVO **extends** PageVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;    ⑥ 권한코드 attribute는 null, “”(empty), “ “(blank)를 허용하지 않음을 명시하는 @NotBlank validation 어노테이션을 선언하고, 유효성 검증 메시지를 설정 한다. (\*유효성 검증 메시지는 TB\_MSG(\*메시지 유효성 관리 테이블) DB에 저장된 코드화된 메시지를 사용해야 한다.)  3)  //유효성 검증 메시지: “권한코드 데이터가 없습니다.”  1) 2)  @NotBlank(message = ”{msg.common.nodata1,권한코드}”)  String authCd; //권한코드  String authNm; //권한명  String authDivCd; //권한구분코드  String userDivCd; //사용자구분코드  String sysCd; //시스템코드  String useAt; //사용여부  String authDesc; //권한설명  } |

1) TB\_MSG(\*메시지 유효성 관리 테이블) DB에 저장된 코드 Key값

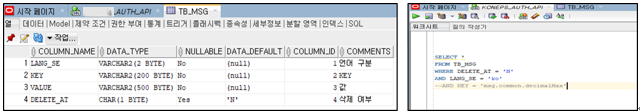
2) TB\_MSG(\*메시지 유효성 관리 테이블) DB에 저장된 코드 Key에 넘겨주는 인자 값

- ex) 배열 [0],[1],[2] 인 경우, 넘겨주는 인자 값은 3개

3) 유효성 검증 메시지 (출력되는 메시지)

**※ TB\_MSG(\*메시지 유효성 관리 테이블) DB에 추가로 필요한 유효성 검증 메시지는 개발자가 추가하여 사용한다.**

* TB\_MSG(\*메시지 유효성 관리 테이블) 설명
* 테이블 컬럼 조회 속성



**Sample**

. LANG\_SE : 언어 구분 (ko – 한국어)

. KEY : 메시지 코드 KEY 값 (메시지 유형에 따른 유니크한 키 값)

→ 일반 메시지는 “메시지유형.작업유형.발생원인” 3자리로 구성하며, 메시지 유형은 다음 4가지

유형을 사용 한다.

① msg: 일반적인 안내를 위해 사용하는 메시지 (ex: msg.common.nodata 등)

② confirm: 확인이 필요한 경우 사용하는 메시지 (ex: msg.confirm.do 등)

③ error: 오류 발생시 사용하는 메시지 (ex: error.common.save 등)

④ warn: 경고 또는 공지를 위해 사용하는 메시지 (ex: warn.common.required 등)

→ 업무 메시지는 “메시지유형.업무영역1depth(2자리).업무영역2depth(3자리).단위업무 코드(4자리)

.발생원인” 의 5자리로 구성한다. 메시지 유형은 일반 메시지와 동일하며, 발생원인은 최대 3개의

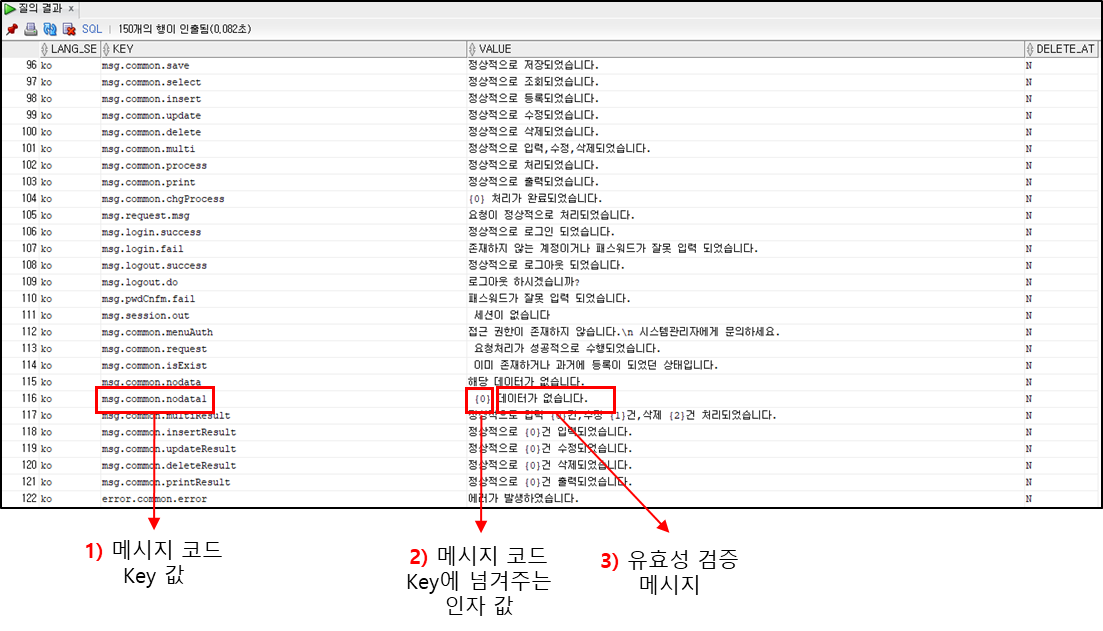
표준단어의 단어 영문명을 조합해서 사용 한다.

(※ 자세한 Message 표준은 3.13.10. Message표준을 참고 한다.)

. VALUE : 유효성 검증 메시지 (출력되는 메시지)

. DELETE\_AT : 삭제 여부 (Y/N)

* 유효성 검증 메시지 샘플



* VO에서 사용되는 Validation 주요 어노테이션 사용법을 살펴보면 다음과 같다.
* VO Validation 주요 어노테이션 목록

|  |  |  |
| --- | --- | --- |
| 유형 | VO 클래스 Validation 어노테이션 사용 예시 | 설명 |
| @NotNull | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;    //유효성 검증 메시지: “사용자명 데이터가 없습니다.”  @NotNull(message = ”{msg.common.nodata1,사용자명}”)  String userName; //사용자명  } | 해당 Attribute(userName)는 Null 을 허용하지 않는다. |
| @NotEmpty | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;    //유효성 검증 메시지: “사용자명 데이터가 없습니다.”  @NotEmpty(message = ”{msg.common.nodata1,사용자명}”)  String userName; //사용자명  } | 해당 Attribute(userName)는 Null과 “”(empty: 빈공백)을 허용하지 않는다. |
| @NotBlank | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “사용자명 데이터가 없습니다.”  @NotBlank(message = ”{msg.common.nodata1,사용자명 }”)  String userName; //사용자명  } | 해당 Attribute(userName)는 Null, “”(empty: 빈공백), “ ”(blank: 빈공백 문자열) 을 허용하지 않는다. |
| @Null | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “사용자명은 Null 값이어야 합니다.”  @Null(message = ”{msg.common.nulldata1,사용자명 }”)  String userName; //사용자명  } | 해당 Attribute(userName)는 Null 만 허용합니다. |
| @AssertFalse | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  String userName; //사용자명  //유효성 검증 메시지: “사용자명은 null이 아니어야 합니다”  @AssertFalse(message = ”{msg.common.assertFalsedata1,사용자명 }”)  **Public boolean** isValidName() {  **return** userName == null  }  } | @AssertFalse 어노테이션이 선언된 요소는 거짓이어야 합니다.  - 지원되는 타입은 Boolean 이다.  - null 요소는 유효한 것으로 간주 된다. |
| @AssertTure | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  int age; //나이  //유효성 검증 메시지: “나이는 0보다 크고, 150보다 작아야 합니다.”  @AssertTrue(message = ”{msg.common.assertTruedata2, 0,150 }”)  **Public boolean** isValidAge() {  **return** age > 0 && age < 150; //나이는 0보다 크고, 150보다 작다.  }  } | @AssertTrue 어노테이션이 선언된 요소는 참이어야 합니다.  - 지원되는 타입은 Boolean 이다.  - null 요소는 유효한 것으로 간주 된다. |
| @DecimalMax | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “180 이하여야 합니다.”  @DecimalMax(value = “180”, inclusive = **true**)  int userHeight;  } | @DecimalMax 어노테이션이 선언된 요소는 value값이 최대값 이다.  - 같은 값 허용  - Null 값 허용  - 어노테이션이 선언된 요소(attribute)는 value보다 작거나 같아야 함  - inclusive가 true인 경우 value보다 크거나 같아야 하고, false인 경우 value보다 커야함  - 지원 타입은 BigDecimal, BigInteger, byte, short, int, long  (반올림 오류로 인해 double, float은 지원x) |
| @DecimalMin | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “180 이상이어야 합니다.”  @DecimalMin(value = “180”, inclusive = **true**)  int userHeight;  } | @DecimalMin 어노테이션이 선언된 요소는 value값이 최소값 이다.  - 같은 값 허용  - Null 값 허용  - 어노테이션이 선언된 요소(attribute)는 value보다 작거나 같아야 함  - inclusive가 true인 경우value보다 크거나 같아야 하고, false인 경우 value보다 커야함  - 지원 타입은 BigDecimal, BigInteger, byte, short, int, long  (반올림 오류로 인해 double, float은 지원x) |
| @Digits | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “키 입력 자리 수는 정수 3자리, 소수 2자리로 입력해야 합니다.”  @Digits(integer = 3, fraction = 2, message = ”{msg.common.digitsdata1, 키}”)  int userHeight;  } | @Digits 어노테이션이 선언된 요소는 허용되는 범위내 숫자이어야 한다.  - integer = 정수 허용 자리수 (3이면, 3자리까지 허용)  - fraction = 소수점 허용 자리수 (0이면, 소수점을 허용하지 않음)  - Null 값 허용  - 지원 타입은 BigDecimal, BigInteger, byte, short, int, long |
| @Future | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “출시날짜는 미래 날짜여야 합니다.”  @Future(message =”{msg.common.futuredata1, 출시날짜}”)  Date releaseDate;  } | @Future 어노테이션이 선언된 요소는 현재날짜 보다 미래 날짜만 허용 한다.  - Null 값 허용  - 시간까지는 체크 안됨  - 지원 타입은  Date, Calender, Instant, LocalDate 등 |
| @FutureOrPresent | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “출시날짜는 현재 또는 미래 날짜여야 합니다.”  @FutureOrPresent(message =”{msg.common.futureOrPresentdata1, 출시날짜}”)  Date releaseDate;  } | @FutureOrPresent 어노테이션이 선언된 요소는 현재날짜 또는 미래날짜만 허용 한다.  - Null 값 허용  - 시간까지는 체크 안됨  - 지원 타입은  Date, Calender, Instant, LocalDate 등 |
| @Past | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “출시날짜는 과거 날짜여야 합니다.”  @Past(message =”{msg.common.pastdata1, 출시날짜}”)  Date releaseDate;  } | @Past 어노테이션이 선언된 요소는 현재날짜 보다 과거 날짜만 허용 한다.  - Null 값 허용  - 시간까지는 체크 안됨  - 지원 타입은  Date, Calender, Instant, LocalDate 등 |
| @PastOrPresent | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “출시날짜는 과거 또는 현재 날짜여야 합니다.”  @PastOrPresent(message =”{msg.common.pastOrPresentdata1, 출시날짜}”)  Date releaseDate;  } |  |
| @Max | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “나이는 25 이하여야 합니다.”  @Max(value = 25, message =”{msg.common.maxdata2, 나이}”)  int age;  } | @Max 어노테이션이 선언된 요소(attribute)는 value값보다 작거나 같아야 한다.  - 같은 값 허용  - Null 값 허용  - 지원 타입은 BigDecimal, BigInteger, byte, short, int, long  (반올림 오류로 인해 double, float은 지원x) |
| @Min | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “나이는 25 이상이어야 합니다.”  @Min(value = 25, message =”{msg.common.mindata2, 나이}”)  int age;  } | @Min 어노테이션이 선언된 요소(attribute)는 value값보다 작거나 같아야 한다.  - 같은 값 허용  - Null 값 허용  - 지원 타입은 BigDecimal, BigInteger, byte, short, int, long  (반올림 오류로 인해 double, float은 지원x) |
| @Pattern | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “사용자 ID는 반드시 영문 또는 숫자만 입력 가능 합니다.”  @Pattern(regexp = ”[0-9a-zA-Z]", message =”{msg.common.patterndata2, 사용자 ID}”)  int userId;  } | @Pattern 어노테이션이 선언된 요소(attribute)는 regexp= 에 기술된 정규식을 만족해야 한다.  - Null 값 허용  - 정규식은 Java 정규식 규칙을 따름 |
| @Size | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “사용자명은 4글자 이상 100글자 이하이어야 합니다.”  @Size(min = 4, max = 100, message =”{msg.common.sizedata3, 사용자명}”)  String userName;  } | @Size 어노테이션이 선언된 요소(attribute)는 min 값이상 이고, max값이하 이어야 한다.  - Null 값 허용  - min의 Default값은 0  - max 의 Default 값은 2147483647 |
| @Email | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “올바른 형식의 이메일 주소여야 합니다.”  @Email  String userEmail;  } | @Email 어노테이션이 선언된 요소(attribute)는 올바른 형식의 이메일 주소이어야 한다. |
| @Positive | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “나이는 0 보다 커야 합니다.”  @Positive(message =”{msg.common.positivedata1, 나이}”)  int age;  } | @Positive 어노테이션이 선언된 요소(attribute)는 양수이어야 합니다. (0은 잘못된 값)  - 양수만 허용  - Null 값 허용  - 지원 타입은  BigDecimal, BigInteger, byte, short, int, long, float, double |
| @PositiveOrZero | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “나이는 0 이상이어야 합니다.”  @PositiveOrZero(message =”{msg.common.positiveOrZerodata1, 나이}”)  int age;  } | @PositiveOrZero 어노테이션이 선언된 요소(attribute)는 양수 또는 0 이어야 합니다.  - 양수 또는 0 허용  - Null 값 허용  - 지원 타입은  BigDecimal, BigInteger, byte, short, int, long, float, double |
| @Negative | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “금액은 0 미만이어야 합니다.”  @Negative(message =”{msg.common.negativedata1, 금액}”)  double amt;  } | @Negative 어노테이션이 선언된 요소(attribute)는 음수이어야 합니다.  (0은 잘못된 값)  - 음수만 허용  - Null 값 허용  - 지원 타입은  BigDecimal, BigInteger, byte, short, int, long, float, double |
| @NegativeOrZero | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “금액은 0 이하여야 합니다.”  @Negative(message =”{msg.common.negativeOrZerodata1, 금액}”)  double amt;  } | @NegativeOrZero 어노테이션이 선언된 요소(attribute)는 음수 또는 0 이어야 합니다.  - Null 값 허용  - 지원 타입은  BigDecimal, BigInteger, byte, short, int, long, float, double |
| @CreditCardNumber | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “올바르지 않은 신용카드 번호 입니다.”  @CreditCardNumber(ignoreNonDigitCharacters = **true**)  String creditCardNumber; //신용카드 번호  } | @CreditCardNumber 어노테이션이 선언된 요소(attribute)는 유효한 신용 카드 번호이어야 합니다.  -ignoreNonDigitCharacters  가 true인 경우 입력에서  숫자가 아니 문자를  무시하고, false인 경우  오류가 발생  (defulat 값은 false) |
| @Length | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “길이가 2에서 5사이여야 합니다.”  @Length(min = 2, max = 5)  String length;  } | @Length 어노테이션이 선언된 요소(attribute)는 문자열의 길이가 min과 max 사이인지 확인 합니다.  - min의 default 값은 0  - max의 default 값은 2147483647 |
| @UniqueElements | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “고유 요소만 포함해야 합니다.”  @UniqueElements  List<String> uniqueElements;  } | @UniqueElements 어노테이션이 선언된 요소(attribute)는 Collection의 모든 객체가 고유한지 확인 합니다.  - list에서 중복을 확인 |
| @URL | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “올바른 URL이어야 합니다.”  @URL  String url;  } | @URL 어노테이션이 선언된 요소(attribute)는 올바른 URL인지 확인 합니다.  - @URL(protocol=, host=, port=)  매개변수 protocol, host, port는 URL의 해당 부분과 매칭 |
| @JuminNo | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “잘못된 주민등록번호 입니다.”  @JuminNo(allowedDash = **true**)  String juminNo; //주민등록번호  } | @JuminNo 어노테이션이 선언된 요소(attribute)는 올바른 주민등록번호 인지 확인 합니다.  - 매개변수 allowedDash가 추가 되면 dash(-) 사용을 허용 (기본값: false)  - null 요소는 유효한 것으로 간주 |
| @BizNo | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “잘못된 사업자등록번호 입니다.”  @BizNo(allowedDash = **true**)  String bizNo; //사업자등록번호  } | @BizNo 어노테이션이 선언된 요소(attribute)는 올바른 사업자등록번호 인지 확인 합니다.  - 매개변수 allowedDash가 추가 되면 dash(-) 사용을 허용 (기본값: false)  - null 요소는 유효한 것으로 간주 |
| @CorpNo | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “잘못된 법인번호 입니다.”  @CorpNo(allowedDash = **true**)  String corpNo; //법인번호  } | @CorpNo 어노테이션이 선언된 요소(attribute)는 올바른 법인번호 인지 확인 합니다.  - 매개변수 allowedDash가 추가 되면 dash(-) 사용을 허용 (기본값: false)  - null 요소는 유효한 것으로 간주 |
| @MaxBytes | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “10 bytes 이하여야 합니다. (입력값: 27 bytes)”  @MaxBytes(value = “10”, inclusive = **true**)  String maxBytes;  } | @MaxBytes 어노테이션이 선언된 요소(attribute)는 문자열의 최대 byte값 이하인지 확인 합니다.  - 매개변수 value는 최대 크기, inclusive(기본값)는 포함 여부 |
| @DecimalRange | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “5.1에서 9.9 사이여야 합니다.”  @DecimalRange(min = “5.1”, max = “9.9”)  double decimalRange;  } | @DecimalRange 어노테이션이 선언된 요소(attribute)는 {min}에서 {max} 사이여야 합니다  - Null 값 허용  - 지원 타입은  BigDecimal, BigInteger, CharSequence, byte, short, int, long  (반올림 오류로 인해 double, float은 지원x) |
| @DateFormat | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “잘못된 날짜 형식입니다. (형식: yyyy-mm-dd)”  @DateFormat(value = “yyyy-MM-dd”)  String dataFormat;  } | @DataFormat 어노테이션이 선언된 요소(attribute)는 올바른 날짜 형식인지 확인 합니다.  - 매개변수 value는 날짜형식 (예, yyyy-MM-dd HH:mm:ss)  - Null 값 허용 |
| @DateFormatFuture | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “미래 날짜여야 합니다. (현재: 2022-02-21 17:11:14)”  @DateFormatFuture(value = “yyyy-MM-dd HH:mm:ss” , inclusive = **false**)  String dateFormatFuture;  } | @ DateFormatFuture 어노테이션이 선언된 요소(attribute)는 현재 또는 미래의 날짜인지 확인 합니다.  - 매개변수 value는 날짜형식(예, yyyy-MM-dd HH:mm:ss), inclusive는 현재 포함 여부  - Null 값 허용 |
| @DateFormatPast | @Data  **public** **class** SampleUserVO **extends** UserVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = -1384171486694302968L;  //유효성 검증 메시지: “현재 또는 과거의 날짜여야 합니다. (현재: 2022-02-21)”  @DateFormatPast(value = “yyyy-MM-dd” , inclusive = **true**)  String dateFormatPast;  } | @ DateFormatPast 어노테이션이 선언된 요소(attribute)는 현재 또는 과거의 날짜인지 확인 합니다.  - 매개변수 value는 날짜형식(예, yyyy-MM-dd HH:mm:ss), inclusive는 현재 포함 여부  - Null 값 허용 |

페이지의 마지막 줄

## Persistence 계층 개발

* Persistence 계층은 데이터에 접근하거나 Mapping과 같은 역할을 담당하는 계층이다. 본 프로젝트에서는 MyBatis mapper을 사용하여 이러한 역할들을 처리하며 Spring에서 제공하는 mapper과의 연동 방법을 사용하여 비즈니스 계층의 객체와 의존 관계를 구성한다.
* 본 절에서는 Mapper 와 SQL 작성 방법을 가이드 한다.

### Mapper 개발 표준

* Mapper는 MyBatis의 프레임워크의 기능을 이용하여 Data Access를 처리하는 클래스이다.
* Mapper는 Service 에서만 호출할 수 있다.
* Service 마다 Mapper 를 생성하지 않고 공통 Mapper를 만들어 여러 Service 에서 사용 할 수 있도록 F/W에서 제공한다(dataSource별 공통사용 Mapper 생성)
* Mapper 클래스는 src/main/java/xxxxx/com/mapper/패키지 하위에 생성한다.

ex) src/main/java/xxxxx/com/mapper/….java

. xxxxx – OOO시스템 (프로젝트명)

. 업무영역 1depth: com – 공통영역

* Mapper에서는 Spring FW의 관련 Exception을 자동으로 Throw 하므로 별도의 Exception 처리가 필요한 경우를 제외하고는 데이터 처리 오류 관련 Exception을 처리하지 않는다.
* framework에서 Datasource와 Connection 생성, 관련 자원의 해제 등의 처리를 모두 담당하므로 Mapper에서 이와 관련된 별도의 로직을 사용하지 않는다.
* DataAcess 작업별 주체

|  |  |  |
| --- | --- | --- |
| 작업 | myBatis FW | 개발자 |
| Connection 관리 | O | X |
| SQL 작성 | X | O |
| Statement 관리 | O | X |
| ResultSet 관리 | O  **Sample** | X |
| Row 데이터 추출 | X | O |
| Parameter 선언 | X | O |
| Parameter 셋팅 | O | X |
| Transaction 관리 | O | X |

* 공통 Mapper 클래스에서 사용하는 method별 사용법을 살펴보면 다음과 같다.
* Mapper Method 목록

|  |  |  |
| --- | --- | --- |
| 유형 | Mapper Method | 설명 |
| 단건 조회 | Object **selectOne**(String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한 파라미터 객체(보통 Map)  - return) Object : 결과 Object 객체(SQL mapping 파일에서 지정한 resultClass/resultMap 에 의한 결과 객체)  - 사용 예시 1)  **public** String selectSampleYn(Map<String, Object> paramMap) {  String rtnStr = "N";  **int** cnt = dsXxxxxBzMapper.selectOne(***SQL\_PREFIX*** + "selectSampleYn", paramMap);  **if**(cnt > 0) {  rtnStr = "Y";  }  **return** rtnStr;  }  - 사용 예시 2)  **public** Map<String, Object> selectSampleId(Map<String, Object> paramMap) {  **return** dsXxxxxBzMapper.selectOne(***SQL\_PREFIX*** + "selectSampleId", paramMap);  } | 단일 row 또는 단일 column을 조회하며 결과를 Object로 반환한다. |
| 다건 조회 | List<Map<String,Object>>**selectList**(String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한 파라미터  객체(보통 Map)  - return) List : 결과 객체(보통 Map)의 List(SQL mapping  파일에서 지정한 resultClass/resultMap 에 의한 결과 객체)  - 사용 예시)  **public** List<Map<String, Object>> selectSampleList(Map<String, Object> paramMap) {  **return** dsXxxxxBzMapper.selectList(***SQL\_PREFIX*** + "selectSampleList", paramMap);  } | 리스트를 조회하여 결과를 List로 반환한다. |
| 페이징 조회 | List<Map> **selectListForPaging** (String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한 파라미터 객체(보통 Map)  파라미터에는 currentPage, recordCountPerPage 포함되어야 함  - return) List : 결과 객체(보통 Map)의 List(SQL mapping 파일에서 지정한 resultClass/resultMap 에 의한 결과 객체)  - 사용 예시)  **public** List<Map<String, Object>> selectSamples(Map<String, Object> paramMap) {  dsXxxxxBzMapper.selectListForPaging (***SQL\_PREFIX*** + "selectSamples", paramMap);  } | 리스트를 조회하여 페이징 정보 결과 객체를 반환 한다. |
| 입력 | Object **insert**(String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한 파라미터 객체(보통 Map)  - return) Object : 입력 시 selectKey를 사용하여 key를 딴 경우 해당 key  - 사용 예시)  **public** **int** insertSample(String sqlId, Map<String, String> paramMap) {  **return** dsXxxxxBzMapper.insert(***SQL\_PREFIX*** + "insertSample", paramMap);  } | 데이터 입력 후 입력 시 작성한 selectkey에 의한 key값을 반환한다. |
| 수정 | int **update**(String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한(key 조건 및 변경 데이터를 세팅한) 파라미터 객체(보통 Map)  - return) Object : DBMS가 지원하는 경우 update 적용 결과 count  - 사용 예시)  **public** **int** updateSample(String sqlId, Map<String, String> paramMap){  **return** dsXxxxxBzMapper.update(***SQL\_PREFIX*** + "updateSample",paramMap);  } | 데이터 수정 후 수정한 count 수를 반환한다. |
| 삭제 | int **delete**(String queryId, Object parameterObject)  - param) queryId : 실행할 SQL mapping query Id  - param) parameterObject: 쿼리 실행에 필요한(일반적으로 key 조건을 세팅한) 파라미터 객체(보통 Map)  - return) Object : DBMS가 지원하는 경우 delete 적용 결과 count  - 사용 예시)  **public** **int** deleteSample(String sqlId, Map<String, String> paramMap){  **return** dsXxxxxBzMapper.delete(***SQL\_PREFIX*** + "deleteSample",paramMap);  } | 데이터 삭제 후 삭제한 count 수를 반환한다. |

* Insert 시에 Object 로 반환하는 이유는 insert시에 생성한 sequence 를 반환할 수 있기 때문이며, 이것은 일반적인 ORM의 특성이다. 물론 mybatis는 ORM은 아니지만 DB library 로서의 최신 트렌드를 반영한다.
* insert 시 반환을 위해서는 SqlMap 파일 내에 다음과 같이 selectKey 쿼리를 작성해야 한다. 물론 사용되는 parameterType(or parameterClass)가 해당 key를 지원해야 한다.

|  |
| --- |
| <insert id=*"insertSample"* parameterType=*"egovMap"*>  <selectKey keyProperty=*"sampleSeq"* resultType=*"int"* order=*"before"*>  SELECT SQ\_TBQBM004.NEXTVAL FROM DUAL  </selectKey>  ...  </insert> |

### Mapper 작성 표준

* Mapper 에서 호출하는 SQL들을 XML 파일에 작성한다.
* 기본적으로 하나의 Mapper 에 대해 하나의 xml 파일을 만든다. (Mapper : SqlMap = 1 : 1)
* SqlMap 파일은 src/main/resources/mapper/시스템 구분/업무영역(2자리)/업무구분(3자리)/ 패키지 하위에 생성한다.

ex) src/main/resources/mapper/com/co/coa/…\_sql.xml

. 시스템 구분: com – 공통관리

. 업무영역(2자리): co – 공통영역

. 업무구분(3자리): coa – 공통권한관리

* 파일 명은 datasource명-업무영역(2자리).업무구분(3자리)명.업무단어(Service클래스명에서 Service단어 제외)\_SQL.xml로 하며 소문자로 시작한다.

ex) dsBz-co.coa.Sample\_SQL.xml

**Sample**

. datasource 명 : 업무관리-dsBz

. 업무영역(2자리): co – 공통영역

. 업무구분(3자리): coa – 공통권한관리

. 업무단어: Sample - SampleService 클래스명에서 Serivce단어를 제외

* SqlMap 파일 유형을 다음의 두 가지로 정의한다.

1. 유스케이스 단위 조회 SqlMap : 여러 테이블 간 join이 필요한 조회 query를 위한 파일이며, 특정 테이블에 대한 입력/수정/삭제를 위한 query를 정의하지 않는다.

query id는 업무영역(2자리).업무구분(3자리)명.prefix[단어]로 작성하며, 다른 query id와 중복되지 않도록 한다. prefix는 service method의 prefix와 동일하다.

ex) co.coa.Template.selectUserInfo

1. 테이블 단위 CRUD : 한 테이블에 대한 조회/입력/수정/삭제 query를 위한 파일이며, join이 필요한 조회를 위한 query를 정의하지 않는다.

query id는 업무영역(2자리).업무구분(3자리)명.prefix[테이블명] 또는

업무영역(2자리).업무구분(3자리)명.update[주요컬럼]으로 작성하며, 다른 query id와 중복되지 않도록 한다. prefix는 select/insert/update/delete를 사용하며, 주요 컬럼은 3개 이하로 구성한다.

ex) co.coa.Sample.selectTBca001

**[샘플 예시]**

|  |
| --- |
| ① 반드시 SqlMap 파일에 대한 주석을 작성한다. (파일명, 설명, 수정 정보)  <!--  SQL File Name : dsBz-co.coa.Auth\_SQL.xml  Description : 단위업무 분류 관리를 위한 Mapper  Modification Information  수정일 수정자 수정내용  ======= ====== =================  2022.02.10 홍길동 최초 생성  -->  **※ 수정일 ~ 수정자 ~ 수정내용 사이 간격은 2TAP(8칸) 이고, 좌측 정렬 이다.**  ② sqlMap 파일의 namespace를 유일하게 작성한다.  (\*namespace는 Service클래스에서 정의한 SQL\_PREFIX와 동일)  <mapper namespace=*"co.coa.auth"*>  ③ 모든 query에 주석을 작성한다.  <!--  쿼리명 : “selectAuthList” (권한목록 조회)  설 명 : 사용자별 권한 목록을 조회한다.  parameterType = xxxxx.com.co.coa.vo.AuthVO (검색조건)  resultType = egovMap  수정일 수정자 수정내용  ======= ====== =================  2022.02.10 홍길동 최초 생성  -->  **※ 수정일 ~ 수정자 ~ 수정내용 사이 간격은 2TAP(8칸) 이고, 좌측 정렬 이다.**  <select id=*"selectAuthList"* parameterType=*"xxxxx.com.co.coa.vo.AuthVO"* resultType=*"egovMap"*>  ④ 모든 쿼리는 <![CDATA[...]]> 안에 작성하며 dynamic SQL중 ${} 는 사용하지 않는 것을 표준으로 한다. 사용이 필요한 경우 공통기술팀과 협의한다. (#{} 는 사용 가능)  <![CDATA[  ⑤ 쿼리 작성 시 첫 라인에 해당 query id 주석을 기입한다.  /\* co.coa.auth.selectAuthList \*/  SELECT TB.\* FROM (  SELECT A.AUTH\_CD  , A.AUTH\_NM  , A.AUTH\_DIV\_CD  , A.USE\_AT  , A.AUTH\_DESC  , A.SYS\_CD  , A.ROW\_STAT\_CD  , NVL(B.SCREEN\_CNT, 0) AS SCREEN\_CNT  , COUNT(1) OVER(ORDER BY A.AUTH\_CD) AS ROW\_NUM  , COUNT(1) OVER() AS TOT\_CNT  FROM  (  SELECT A.AUTH\_CD  , A.AUTH\_NM  , A.AUTH\_DIV\_CD  , A.USE\_AT  , A.AUTH\_DESC  , A.SYS\_CD  , A.ROW\_STAT\_CD  FROM TB\_CA\_AUTH A  WHERE A.ROW\_STAT\_CD <![CDATA[<>]]> 'D'  <if test=*'authCd != null and authCd != ""'*>  AND A.AUTH\_CD LIKE '%' || #{authCd} ||'%'  </if>  <if test=*'authNm != null and authNm != ""'*>  AND A.AUTH\_NM LIKE '%' || #{authNm} ||'%'  </if>  <if test=*'authDivCd != null and authDivCd != ""'*>  AND A.AUTH\_DIV\_CD = #{authDivCd}  </if>  <if test=*'userDivCd != null and userDivCd != ""'*>  AND A.USER\_DIV\_CD = #{userDivCd}  </if>  <if test=*'authClCd != null and authClCd != ""'*>  AND A.AUTH\_CL\_CD = #{authClCd}  </if>  <if test=*'wkgdCd != null and wkgdCd != ""'*>  AND A.WKGD\_CD = #{wkgdCd}  </if>  <if test=*'sysCd != null and sysCd != ""'*>  AND A.SYS\_CD = #{sysCd}  </if>  <if test=*'useAt != null and useAt != ""'*>  AND A.USE\_AT = #{useAt}  </if>  <if test=*'authDesc != null and authDesc != ""'*>  AND UPPER(A.AUTH\_DESC) LIKE '%'||UPPER(#{authDesc})||'%'  </if>  )A  ,(  SELECT COUNT(MENU\_ID) SCREEN\_CNT  , AUTH\_CD  FROM  (  SELECT D.MENU\_ID  , A.AUTH\_CD  FROM TB\_CA\_AUTH\_EVENT A  , TB\_CS\_SCN\_EVENT B  , TB\_CS\_UNIT\_TASK\_SCN C  , TB\_CA\_MENU D  WHERE A.EVENT\_ID = B.EVENT\_ID  AND A.SCN\_ID = B.SCN\_ID  AND A.SCN\_ID = C.SCN\_ID  AND A.MENU\_ID = D.MENU\_ID  GROUP BY D.MENU\_ID,A.AUTH\_CD  )  GROUP BY AUTH\_CD  ) B  WHERE A.AUTH\_CD = B.AUTH\_CD(+)  ORDER BY AUTH\_CD DESC  ) TB  <![CDATA[  WHERE TB.ROW\_NUM >= #{firstIndex} AND TB.ROW\_NUM <= #{lastIndex}  ]]>  </select> |

* Select 조회 구문은 select statement에 작성하는 것을 표준으로 한다.

1. select statement는 id, parameterType(or parameterMap), resultType(or resultMap)를 기본 요소로 가진다.
2. query id : 필수 요소이며, 명명 규칙은 위의 내용을 참조한다.
3. parameterType : 해당 쿼리에 전달할 파라미터가 있을 경우 사용한다. java.util.Map을 parameterType으로 사용하는 것을 기본으로 하지만 필요 시 별도의 parameterClass나 parameterMap을 정의해서 사용할 수 있다. (parameterClass 사용 시 query에 들어 갈 parameter의 동적 바인딩을 위해 login=#{login}의 형식으로 사용하고, parameterMap 사용 시 login=?의 형식으로 사용한다.)
4. resultType : 결과값이 있을 때 사용한다. egovMap을 resultType로 사용하는 것을 표준으로 하지만 필요 시 별도의 resultClass나 resultMap을 정의해서 사용할 수 있다.(연계 요청 처리 시 resultMap을 정의해서 사용한다.) resultClass로 egovMap을 사용하는 경우 컬럼 매핑은 다음과 같다.

ex) db coulum : POST\_NO → map key : postNo

|  |
| --- |
| **① ② ③ ④**  <select id=*"selectSampleList"* parameterType=*"java.util.Map"* resultType=*"egovMap"*>  <![CDATA[  SELECT /\* co.coa.postNo.selectPostNoByCondition \*/  post\_no  , post\_no\_sno  , TRIM (sggu\_nm || ' ' || emdong\_nm || ' ' || apt\_nm) AS whl\_addr  , post\_no\_whl\_addr  FROM tbjca130  WHERE post\_no LIKE #{postNo}||'%'  AND post\_no\_whl\_addr LIKE '%'||#{postNoWhlAddr}||'%'  ORDER BY post\_no  ]]>  </select> |

* insert/ update/ delete : 입력/수정/삭제 구문은 각각 insert/update/delete statement에 작성하는 것을 표준으로 한다.

1. insert/update/delete statement는 id, parameterType(or parameterMap)를 기본 요소로 가진다.
2. query id : 필수 요소이며, 명명 규칙은 위의 내용을 참조한다.
3. parameterType : 해당 쿼리에 전달할 파라미터가 있을 경우 사용한다. java.util.Map을 parameterType으로 사용하는 것을 기본으로 하지만 필요 시 별도의 parameterClass나 parameterMap을 정의해서 사용할 수 있다.

|  |
| --- |
| **① ② ③**  <insert id=*"insertSample"* parameterType=*"egovMap"*>  <![CDATA[  INSERT INTO /\*sample.DevSampleDAO.insertSample\*/ 테이블명  (  WRK\_ID  , WRK\_TYPE\_CD  , WRK\_NM  , SERVICE\_ID  , WRK\_TYPE  , SEMAPHORE\_PREFIX  )  VALUES  (  #{wrkId}  , #{wrkTypeCd}  , #{wrkNm}  , #{serviceId}  , #{wrkType}  , #{semaphorePrefix}  )  ]]>  </insert>  **① ② ③**  <update id=*"updateSample"* parameterType=*"egovMap"*>  <![CDATA[  ...  ]]>  </update>  **① ② ③**  <delete id=*" deleteSample"* parameterType=*"egovMap"*>  <![CDATA[  ...  ]]>  </delete> |

* CLOB(Character Large Object) / BLOB(Binary Large Object) 조회

CLOB은 큰 문자 객체, BLOB은 큰 이진 객체를 이야기 하며, JDBC를 이용하여 데이터를 가져올 때 고급 데이터 타입인 BLOB, CLOB을 지원 한다. 이처럼 많은 양의 데이터를 저장하도록 설계된 타입이다. BLOB은 파일과 같은 많은 양의 이진 데이터를 저장하고, CLOB은 많은 양의 텍스트를 저장한다.

1. CLOB 또는 BLOB 조회를 위해서 "resultClass" 대신 "resultType(or resultMap)"을 사용한다.
2. longdesc 가 CLOB이며, String으로 변환할 것을 명시한다. userImage 가 BLOB이며, byte로 변환할 것을 명시한다.
3. select 결과를 resultType(or resultMap)에 담는다. 여기서 LONGDESC는 CLOB 형태이며 USER\_IMAGE는 BLOB 형태이다.

|  |
| --- |
| **①**  <resultMap id=*"egovLobMap"* class=*"xxxxx.core.util.egovLobMap"*>  <result property=*"id"* column=*"ID"* />  <result property=*"name"* column=*"NAME"* />  <result property=*"description"* column=*"DESCRIPTION"* />  **②**  <result property=*"longdesc"* column=*"LONGDESC"* jdbcType=*"CLOB"* javaType=*"java.lang.String"* />  <result property=*"userImage"* column=*"USER\_IMAGE"* jdbcType=*"BLOB"* javaType=*"[B"* />  <result property=*"useYn"* column=*"USE\_YN"* />  <result property=*"regUser"* column=*"REG\_USER"* />  </resultMap>  **③**  <select id=*"selectSample"* parameterType=*"java.lang.String"* resultType=*"egovLobMap"*>  <![CDATA[  SELECT /\*sample.selectSampleById\*/  ID, NAME, DESCRIPTION  , **LONGDESC**  , **USER\_IMAGE**  , USE\_YN, REG\_USER  FROM SAMPLE  WHERE ID = #{id}  ]]>  </select> |

* CLOB, BLOB 입력/수정 : 일반적으로 String형의 입력 데이터를 CLOB에 저장할 수 있으나, 명시적으로 CLOB, BLOB을 지정해서 사용한다.

1. insert, update 구문 작성 시 다음과 예와 같이 해당 field가 CLOB, BLOB임을 명시하여 처리한다.

ex) #{longDesc:CLOB}, #{userImage:BLOB}

|  |
| --- |
| <insert id=*"insertSample"* parameterType=*"egovMap"*>  <![CDATA[  INSERT INTO /\* sample.insertSample \*/ SAMPLE  ( ID  , NAME  , DESCRIPTION  , LONGDESC  , USER\_IMAGE  , USE\_YN  , REG\_USER  )  VALUES  ( #{id}  , #{name}  , #{description}  **①**  , **#{longDesc:CLOB}**  , **#{userImage:BLOB}**  , #{useYn}  , #{regUser}  )  ]]>  </insert> |

페이지의 마지막 줄

## 그 외 개발 표준

### Logging 표준

* 기본적으로 slf4j 라이브러리를 사용한다.

. SLF4J(Simple Logging Façade for Java)의 약자로 Log4J 또는 Logback과 같은 백엔드 Logger

Framework의 façade pattern을 가지고 있다.

. SLF4J API를 사용하면 구현체 종류에 상관없이 일관된 로깅 코드를 작성할 수 있다.

. 여러 개의 클래스가 하나의 역할을 수행할 때, 대표적인 인터페이스만을 다루는 클래스를 두어

원하는 기능을 처리할 수 있게 도와주는 패턴이다.

* Unchecked Exception이 발생한 경우 Stack Trace 내용을 로그에 남긴다.
* log는 Framework에서 제공된 Logger만 사용한다
* log는 debug, info, warn, error로 구별하여 사용한다.(로그레벨은 debug < info < warn < error < fatal)

① debug: 개발자가 개발 시에만 사용하고, 운영 중에는 사용하지 않는다.

② Info: 운영자에게 도움이 될 내용을 기록한다.

③ warn: error는 아니나 잠재적인 error의 발생이 가능한 내용을 기록한다.

④ error: error code와 함께 error에 대한 내용을 기록한다.

**Sample**

* log는 한 줄만 출력하는 것을 표준으로 하며(debug log 제외), 반드시 발생 시간과 위치, 내용을 포함한다.
* Logging 시 성능을 위하여 현재 Logging 레벨을 체크하는 구문을 반드시 추가한다.

|  |
| --- |
| log.debug(로그내용);  - 사용 예시)  ***log***.debug("redis cache setCacheVal err"+ e.getMessage()); |

* mybatis를 통해서 수행되는 SQL 로그를 남기기 위해서 log4jdbc 라이브러리를 사용할 수 있다. 개발 중 JDBC 호출이나 SQL문에 대한 로그를 남기려면 다음과 같이 설정을 수정한 후 이용한다.
* src/main/resources/config/log/logback.xml 파일의 logger 내용을 수정한다.

. driverClassName : net.sf.log4jdbc.DriverSpy

. url : jdbc:log4jdbc:oracle:thin:

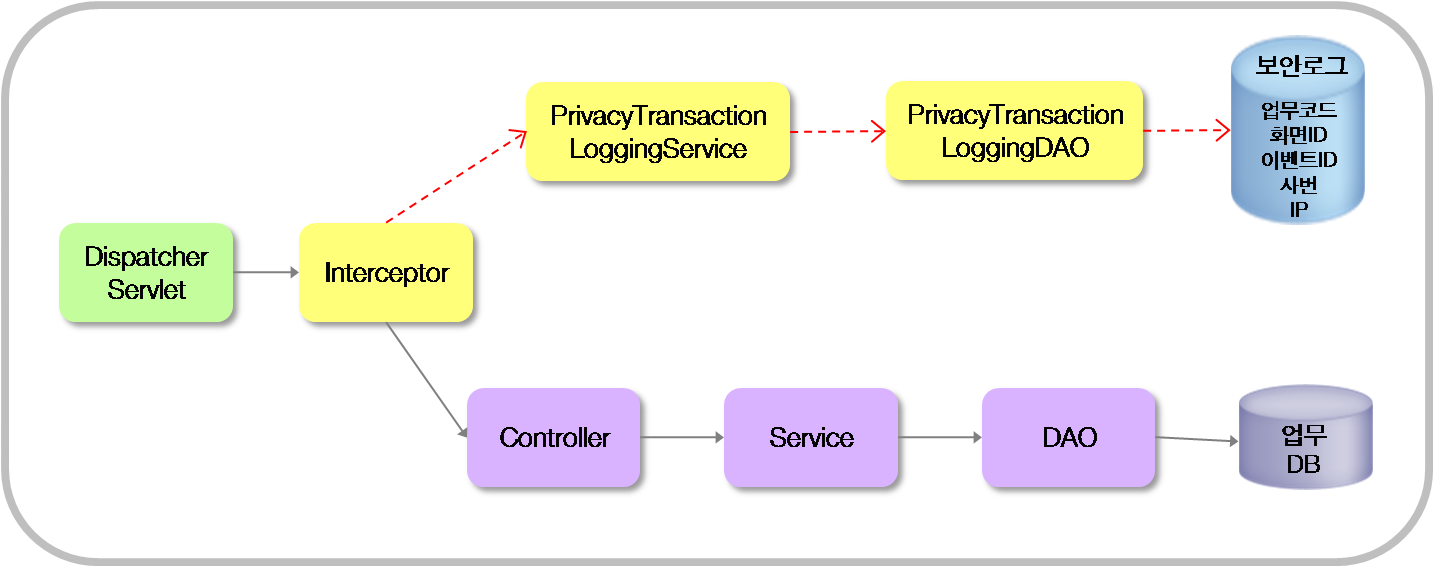
|  |
| --- |
| <logger name=*"xxxxx"* level=*"DEBUG"*>  <appender-ref ref=*"logstash"* />  </logger>  <logger name=*"jdbc"* level=*"OFF"*/>  <logger name="jdbc.sqlonly" level="OFF"/>  <logger name=*"jdbc.sqltiming"* level=*"DEBUG"*/>  <logger name=*"jdbc.audit"* level=*"OFF"*/>  <logger name=*"jdbc.resultset"* level=*"OFF"*/>  <logger name=*"jdbc.resultsettable"* level=*"DEBUG"*/>  <logger name=*"jdbc.connection"* level=*"OFF"*/>  <root level=*"INFO"*>  <appender-ref ref=*"CONSOLE"* />  </root> |

* log4jdbc에서 제공하는 로깅 유형 목록

|  |  |
| --- | --- |
| 로깅 유형 | 설명 |
| jdbc.sqlonly | SQL문만을 로그로 남기며, PreparedStatement일 경우 관련된 argument 값으로 대체된 SQL문이 보여진다. |
| jdbc.sqltiming | SQL문과 해당 SQL을 실행시키는데 수행된 시간 정보(milliseconds)를 포함한다. |
| jdbc.audit | ResultSet을 제외한 모든 JDBC 호출 정보를 로그로 남긴다. 많은 양의 로그가 생성되므로 특별히 JDBC 문제를 추적해야 할 필요가 있는 경우를 제외하고는 사용을 권장하지 않는다. |
| jdbc.resultset | ResultSet을 포함한 모든 JDBC 호출 정보를 로그로 남기므로 매우 방대한 양의 로그가 생성된다. |

### 보안 로깅

* 웹을 통한 특정 업무 요청/처리 시 해당 요청에 대한 사용 기록을 남겨 관리할 필요가 있을 수 있다. 이런 업무 처리시 PrivacyTransactionLoggingService를 이용하여 보안 로깅을 저장하도록 한다. 이에 대한 처리 메커니즘은 다음과 같다.



* 차세대 OOO시스템은 화면의 내용에 다음과 같은 항목이 있을 경우 보안로깅 대상 화면이 된다.

**Sample**

. 주민등록번호

. 카드번호

* 보안 로깅 대상 화면인 경우 조회 조건에 다음 표의 로깅 대상이 있을 경우 해당 내용에 대한 보안 로깅을 남긴다. 보안 로깅은 framework에서 공통으로 처리한다. 다만 공통 처리를 위해서 다음 항목을 조회 조건으로 사용하는 경우 개발자는 반드시 dsParam이라는 DataCollection에 로깅 대상 항목의 UI Property 명을 준수하여 server단에 전달해야 한다.

. 보안 로깅 dataset명 : dsParam

. 보안 로깅 대상 항목

* 보안로깅 대상 항목

|  |  |  |  |
| --- | --- | --- | --- |
| NO | 로깅 대상 | DB Column 명 | UI Property 명 |
| 1 | **주민등록번호** | JNO | jno |
| 2 | **접속IP** | IP\_ADDR | ipAddr |
| 3 | **성명** | NM | nm |
| 4 | **법인등록번호** | CORP\_RNO | corpRno |
| 5 | **주소** | ADDR | Addr |

### Exception 표준

* 예외 처리는 시스템의 비정상적인 동작(Exception)에 어떻게 대응하는 가를 정의하는 것으로 Exception을 정의하지 않으면 시스템 장애 발생 시 사용자가 원하는 결과를 얻지 못한다. 따라서 Exception을 처리하는 로직을 두고, Exception 발생 시 적절한 처리를 함으로써 원하는 결과를 얻을 수 있다.
* Exception 발생 시 기본적으로 스프링 프레임워크에서 각 유형별로 정의된 Exception을 받아서 상위 레이어로 리턴하고 UI에 에러페이지 또는 alert으로 표시하도록 한다.

### Mapper Exception 처리

* Mapper 클래스는 스프링에서 제공하는 MybatisMapper 를 상속받아 사용하며, 별도의 SQL관련 Exception 로직을 작성하지 않는다.
* Exception이 발생되면 Service로 throw되고 Service에서 Controller로 throw 한다.

### Service Exception 처리

* Exception을 강제로 throw하고자 할 경우에는 BizException의 processException("메시지 코드") 함수를 사용해서 Controller로 throw 한다. 메시지 코드를 넘길 경우 메시지 코드에 해당하는 메시지가 적용된다.

### Controller Exception 처리

* Controller 에서는 Service에서 throw 된 Exception을 catch하여 관련 처리(메시지 설정)를 하거나, 처리를 하지 않고 ExceptionResolver를 통해 해당 Exception에 대한 default 메시지 처리를 할 수 있다. 기본적으로는 try-catch문을 사용하여 UI에 전달할 성공,실패 메시지를 설정한다.
* 다음은 Controller에서 Exception을 catch 하여 UI에 보낼 메시지를 설정하는 코드이다

1. service 클래스에서 설정한 BizException 메시지가 있을 경우 다음과 같이 해당 메시지를 전달할 수 있다. 다음과 같이 catch한 bizException에 대한 에러 메시지 코드를 설정한다.
2. 다음과 같이 Exception에 대한 기본 메시지를 설정한다.

setMessageFail 함수는 다음과 같은 Parameter를 가지며, 메시지 키와 Exception을 반드시 넘겨 줘야 한다.

. setMessageFail(String msgKey, String[] args, Exception e)

. msgKey : 오류 메시지 키(BusinessException 외 필수, messagesmessage-common.properties

파일에 정의)

. args : msgKey 구성에 필요한 parameter

. e : 발생한 Exception(필수)

|  |
| --- |
| **try** {  ...  model.addObject(“msg”, MessageUtils.getMessage("s"));  } **catch** (BizException bize) {   1. throw new BusinessException("msg.login.fail");   } **catch** (Exception e) {   1. model.addObject(“msg”, MessageUtils.getMessage("x"));   model.addObject(“statCd”,”-1”)  } |

### Exception Resolver

* Exception Resolver를 정의해서 사용하고 있다.
* Exception이 발생했을 경우 web root 밑의 jsp 경로에 있는 com/Error.jsp 파일이 실행된다
* 아래 xml 파일을 통해서 Exception 종류별 커스터마이징 또한 가능하다.

### Transaction 표준

* Transaction 은 AOP로 관리되기 때문에 개발자는 Transaction 처리에 대해 고민할 필요가 없다. 다만 공통 Mapper를 사용할 때 어떤 Mapper사용 할지에 대한 부분만 선택해주면 된다. 설정 변경 및 추가적인 Mapper 설정이 필요하면 공통기술팀에 요청해서 처리한다.

1. txMAP 값 부분은 어떤 함수에 대해서 transaction을 적용할지에 대한 설정이며, 아래 설정의 requiredTx 설정된 method 부분은 Exception이 발생했을 경우 Rollback 하도록 정의한 것이고 readOnlyTx 부분은 transaction이 적용되지 않는다.
2. aop 설정 부분은 프로젝트 패키지 하위의 모든 Service로 정의된 클래스에 대해 transaction 처리를 정의한 부분이다.

|  |
| --- |
| **①**  Map<String, TransactionAttribute> txMap = new HashMap<>();  txMap.put("multi\*", requiredTx);  txMap.put("save\*", requiredTx);  txMap.put("insert\*", requiredTx);  txMap.put("update\*", requiredTx);  txMap.put("delete\*", requiredTx);  txMap.put("merge\*", requiredTx);  txMap.put("process\*", requiredTx);  txMap.put("trSelect\*", requiredTx);  txMap.put("select\*", readOnlyTx);  txMap.put("get\*", readOnlyTx);  **②**  **private** **static** **final** String ***AOP\_POINTCUT\_EXPRESSION*** = "execution(\* koneps..service.\*Service.\*(..))";  **public** Advisor txAdviceAdvisor() {  AspectJExpressionPointcut pointcut = **new** AspectJExpressionPointcut();  pointcut.setExpression(***AOP\_POINTCUT\_EXPRESSION***);  **return** **new** DefaultPointcutAdvisor(pointcut, txAdvice());  } |

* Transaction 기준표

|  |  |
| --- | --- |
| 메소드 명 | Transaction 여부 |
| select\* | X |
| get\* | X |
| multi\* | O |
| save\* | O |
| insert\* | O |
| Update\* | O |
| Delete\* | O  **Sample** |
| merge\* | O |
| process\* | O |
| trSelect\* | O |

페이지의 마지막 줄

### Property표준

* Properties로 관리되어야 하는 값들은 src/main/resource/config/ 에서 .yml파일로 관리한다. 개발자가 임의로 수정할 수 있으나, 개발자 PC, 개발 서버, 운영 서버에 대한 설정 정보가 상이할 수 있으니 개발자 PC의 정보 수정 시 공통기술팀과 공유하고 전체 개발자에게 반드시 공지한다.
* property는 yml 파일에 속성값을 등록한 후 사용한다. Key와 value 의 쌍의 정보를 entry로 등록하며, WAS가 구동 시 메모리에 올려 놓는다.

|  |
| --- |
| test.test: aaaa  base.file.path\_co: d:/XXXXX\_PROJECT/temp/xxxxx/ |

* Controller 및 Service에서 설정한 property를 조회한다.
* Controller에서 propertyService 를 정의하여 사용 가능하다.

|  |
| --- |
| @Autowired  PropertyService propertyService;  propertyService.getString("base.file.path\_co"); |

### Message 표준

* 프로젝트에서 사용하는 모든 메시지는 message.properties 파일에 설정한 message 파일에서 관리한다.

1. 메시지 추가, 삭제, 수정이 필요한 경우 src/main/resources/messages/message.properties 파일을 수정한다.
2. 일반 메시지는 메시지유형.작업유형.발생원인의 3자리로 구성한다. 메시지 유형은 아래의 4가지 유형을 사용한다.

**Sample**

. msg : 일반적인 안내를 위해 사용하는 메시지

. confirm : 확인이 필요한 경우 사용하는 메시지

. error : 오류 발생 시 사용하는 메시지

. warn : 경고 또는 공지를 위해 사용하는 메시지

1. 업무 메시지는 메시지유형.업무영역2depth(2자리).단위업무(3자리).발생원인의 4자리로 구성한다. 메시지 유형은 일반 메시지와 동일하며, 발생원인은 최대 3개의 표준단어의 단어영문명을 조합해서 사용한다.

|  |
| --- |
| **②**  **#일반 메세지 처리#**  msg.common.save=정상적으로 저장되었습니다.  msg.common.select=정상적으로 조회되었습니다.  msg.login.success=정상적으로 로그인 되었습니다.  **#일반 확인 처리#**  confirm.common.delete=삭제하시겠습니까?  confirm.common.select=조회하시겠습니까?  confirm.common.insert=등록하시겠습니까?  **#일반 에러 발생 시 처리#**  error.common.error=에러가 발생하였습니다.  error.common.save=저장하는데 에러가 발생하였습니다.  error.common.insert=등록하는데 에러가 발생하였습니다.  error.common.update=수정하는데 에러가 발생하였습니다.  **#경고 또는 공지 성 메세지 처리#**  warn.common.required={0}은(는) 필수 입력값입니다.  warn.common.minlength={0}은(는) {1}자 이상 입력해야 합니다.  warn.common.maxlength={0}은(는) {1}자 이상 입력할수 없습니다.  **③**  **#업무 메시지 처리(예시)#**  msg.sm.sfa.saveFail=업무내역 저장이 비정상적으로 종료되었습니다.  msg.sm.sfa.saveSuccess=업무내역 저장이 정상적으로 종료되었습니다.  **#업무 경고 또는 공지성 메시지 처리(예시)#**  warn.sm.sfa.lcsGvdpl=업무 내역이 존재합니다.  warn.sm.sfa.vac01=업무 대체자를 확인해주세요.  **#업무 확인 처리(예시)#**  confirm.sm.sfa.slfChk01={0} 부적정 {1} 건 보완요청\n\n {2} 담당자 미확인. \n\n {2} 담당자가 부적정 확인할 경우를 제외하고 [보완요청]이 진행됩니다. \n\n 처리하시겠습니까?  confirm.sm.sfa.slfChk02={0} 담당자가 보완 요청을 했습니다. 보완 요청하시겠습니까?  **#업무 에러 발생 시 처리(예시)#**  error.sm.sfa.saveStep1Error101=에러발생-[업무현황]업무현황 반영중 오류 0 또는 2개이상의 데이터가 반영되었습니다.[Error-No 101]  error.sm.sfa.saveStep1Error102=에러발생-[업무현황]업무현황 반영중 오류 0 또는 2개이상의 데이터가 반영되었습니다.[Error-No 102] |

* Controller/Service에서 상속하는 AstractService에 messageSource가 이미 정의되어 있기 때문에 별도의 선언 없이 사용할 수 있다.

1. Controller에서 UI로 보낼 성공, 실패 메시지 코드를 다음과 같이 사용할 수 있다.

|  |
| --- |
| **① model.addObject("statCd", -1); //프로세스 상태값 -1 비정상동작 , 0 정상동작**  **model.addObject("msg", MessageUtils.getMessage(code, args));** |

* UI에서는 Global Data Collection에 기 정의된 message를 사용한다.
* 메시지 : gdsMsgList

메시지 코드 사용을 위한 스크립트를 선언한다.

UI에서 다음과 같이 메시지 코드를 사용할 수 있다.

|  |
| --- |
| \*-----------------------------------------------------------------------------  \* DESC : 외부 참조 선언  \*---------------------------------------------------------------------------\*/  **①** include "lib::common\_msg.js";  ...  **②**  //일반 메시지  gfn\_gdsMessage("msg.common.select");  //파라미터 메시지  gfn\_gdsMsgParam("warn.common.minlength",“param1,param2,param3…”); |

* UI에서 사용할 메시지는 반드시 GlobalVariables의 gdsMsgList Dataset에 정의해서 사용한다. 메시지 정의 형식은 message.properties 파일과 동일하다.

페이지의 마지막 줄

### Interceptor 정책

* Interceptor는 xxxxx(프로젝트명)/com/config/mvc/WebMvcConfig.java 파일에 설정되어 있다.
* 사용자의 모든 요청에 대해 Controller 수행 전 interceptor를 적용한다.
* interceptor 파일은 src/main/java/xxxxx/com/config/interceptor/ 하위에 위치한다.
* Controller 수행 전, 후 필요한 처리가 Interceptor의 preHandle 메서드와 afterCompletion 메서드에 정의되어 있다.(postHandle 메서드는 에러 발생 시 실행되지 않으므로 후처리는 afterCompletion 메서드에 구현하였다.)
* preHandel : Controller 실행 전 필요한 전처리를 수행한다.(세션 체크)

1. 세션을 선언한다.
2. 세션을 체크한다.

2-1) 세션이 NULL이고 요청 들어온 URI가 .do인 경우, 요청 받은 Controller를 수행하지 않고

해당 UI에 세션 오류 메시지를 보낸다.

2-2) 세션이 NULL이고 요청 들어온 URI가 .rest인 경우, 별도의 전처리 없이 요청 받은

Controller를 수행한다.

2-3) 세션이 존재하면, 세션에서 필요한 정보를 추출하여 HttpServletRequest에 담은 후 요청

받은 Controller를 수행한다.

**Sample**

* afterCompletion : Controller 수행 후 필요한 후 처리를 수행한다. (보안 로깅)

1. 로그 기록 여부 값이 Y인 경우, 로깅에 필요한 값들을 추출한다.
2. privacyTransactionLoggingService를 이용해 보안 로깅을 한다.

|  |
| --- |
| @Controller("bizIntraInterceptor")  **public** **class** BizIntraInterceptor **extends** HandlerInterceptorAdapter{  ...  **public** **boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) **throws** ServletException {  SSO sso = **null**;  **try** {  **①**  HttpSession session = request.getSession(**true**);  RequestContextHolder.getRequestAttributes().setAttribute(*LOCAL\_SESSION*, session, RequestAttributes.SCOPE\_SESSION);    **2-1)**  **if** (session.getAttribute(Constants.USER\_SESSION) == **null** || ((UserSessionVO)session.getAttribute(Constants.USER\_SESSION)).getEmpNo() == **null**){  String uri = request.getRequestURI();  **if** (uri.indexOf(".do") > -1 && !isSimsaRequest(uri)) { // .ndo는 bypass 대상임.  PlatformData xData = **new** PlatformData();  VariableList xVars = **new** VariableList();  xVars.add("ErrorMsg", "장시간 사용하지 않았거나 일시적인 오류로 로그인 정보가 없습니다. 다시 로그인 하시기 바랍니다.");  xVars.add("ErrorCode", "-801");  xData.setVariableList(xVars);  HttpPlatformResponse xResData = **new** HttpPlatformResponse(response);  xResData.setCharset(PlatformType.DEFAULT\_CHAR\_SET);  xResData.setData(xData);  xResData.sendData();    **return** **false**;  }  } **else** {  **2-2)**  // 세션정보  UserSessionVO userSessionVO = (UserSessionVO)session.getAttribute(Constants.USER\_SESSION);    request.setAttribute("empNo", userSessionVO.getEmpNo());  request.setAttribute("adminYn", userSessionVO.getAdminYn());  request.setAttribute("eamToken", userSessionVO.getEamToken());    //log.debug(">>>>>>>>> ssEmpNo : " + userSessionVO.getEmpNo());  //log.debug(">>>>>>>>> ssAdminYn : " + userSessionVO.getAdminYn());  //log.debug(">>>>>>>>> ssEamToken : " + userSessionVO.getEamToken());  }  } **catch**(Exception e) {  log.error("ERROR OCCURRED", e);  }  **return** **true**;  }  ...  **public** **void** afterCompletion(HttpServletRequest httpservletrequest, HttpServletResponse httpservletresponse, Object handler, Exception exception) **throws** Exception {  **if** (**null** != httpservletrequest.getAttribute(Constants.XDATA\_REQUEST)) {  XDataRequest reqData = (XDataRequest) httpservletrequest.getAttribute (Constants.XDATA\_REQUEST);  String unitBizCd = reqData.getParam("eamUnitBizCd");  String scnId = reqData.getParam("eamScnId");  String evId = reqData.getParam("eamEvId");  String eamAdtYn = reqData.getParam("eamAdtYn");  String prslRs = reqData.getParam("prslRs");    **②**  **If** (reqData.containsParam("logWrtYn")) {  String logWrtYn = reqData.getParam("logWrtYn");    **if** ("Y".equals(logWrtYn)) {  Map map = **new** HashMap();  map.put("unitBizCd", unitBizCd);  map.put("scnId", scnId);  map.put("evId", evId);  map.put("cnnEno", httpservletrequest.getAttribute("empNo"));  map.put("cnnIpAddr", httpservletrequest.getHeader("Proxy-Client-IP")); // real client IP  map.put("cnnParaTxt", StringUtils.left(reqData.toLog(""), 450)); // 한글이 50자 미만으로 가정 차세대 OOO시스템 DB는 한글과 일부 특수문자는 3자로 처리 함  map.put("prslRs", prslRs); // 한글이 50자 미만으로 가정 차세대 OOO시스템 DB는 한글과 일부 특수문자는 3자로 처리 함  privacyTransactionLoggingService.insertLog(map);  }  }  }  } |

페이지의 마지막 줄

### 세션 처리 정책

* 정상적인 로그인 시 로그인 사용자 별로 session 키에 대응되는 session 값이 생성되며 Controller에서 Session 키를 이용해 세션 값을 획득하여 사용할 수 있다.

### Session 정의

* Session 키와 Session 값은 다음과 같다.

|  |  |
| --- | --- |
| Session 키 | Session 값 |
| Constants.USER\_SESSION | UserSessionVO |

* xxx.com.login.service.vo.UserSessionVO의 속성 값은 다음과 같다.

|  |  |
| --- | --- |
| Session 키 | Session 값 |
| userId | 사용자 ID |
| userName | 사용자 NAME |
| userGroup | 할당된 사용자 그룹 |
| empNo | 사원 번호 |
| deptCd | 부서 코드 |
| deptNm | 부서명 |
| dutyCd | 직책 코드  **Sample** |
| dutyNm | 직책 명 |
| dtgrdCd | 직급 코드 |
| dtgrdNm | 직급 명 |
| otel | 사무실전화 |
| htel | 휴대전화 |
| src | MESSENGER |
| adminYn | admin여부 |
| eamToken | eamToken |

### 페이징 처리 표준

* 차세대 OOO시스템은 다음과 같은 세 가지 페이징 정책이 존재하며, 스크롤을 통한 데이터 조회를 표준으로 한다.

. 스크롤을 통한 데이터 조회(scroll paging)

. 그리드 하단의 페이지 번호 출력(number paging)

. Auto 페이징(auto paging)

* 스크롤을 통한 페이징 처리를 위해서 필요한 작업은 다음과 같다.
* UI 개발 시 추가해야 할 내용을 살펴보면 다음과 같다.

1. 페이지 처리에 필요한 공통 스크립트를 include 한다.
2. 스크롤 페이지 처리를 위한 전역변수를 선언하고 초기화 한다.

. new ExtLib.Paging (objForm, objGrid, strSearchFunction, strPageType, recordCountPerPage, nPageScale, strPageDivNm, strBindDataSet)

. objForm: 대상 폼

. objGrid : 조회 데이터를 매핑할 그리드 컴포넌트

**Sample**

. strSearchFunction : 페이징 처리 시 호출할 함수명

. strPageType : 페이징 타입(스크롤 페이징 처리 시 별도 선언 필요 없음, S:Scroll(default), D:Number, A:Auto)

. recordCountPerPage: 한 번에 보여줄 row 수(필요 시 기술, default:15)

. nPageScale : number paging 시 그리드 하단에 보여 줄 페이지 번호 수(스크롤 페이징 처리 시 별도 선언 필요 없음, default:10)

. strPageDivNm : number paging 시 하단 Div ID

. strBindDataSet : input dataset name(default : dsParam)

1. 조회 버튼 클릭 시 발생하는 이벤트에서 조회 함수를 호출한다.
2. 조회 함수는 반드시 pagingInfo object를 설정한다. Transaction 호출 전 Controller에 전달할 page 정보를 설정한다. (this.SVC\_SEARCH.pagingInfo = this.fs\_objPagingInfo00) – sample::Sam66DL01.xml 참조
3. gfn\_transaction 함수를 호출한다. 스크롤 페이징 처리 시 UI에서 수신할 DataCollection명을 반드시 결과 리스트는 ~Paging 으로 설정한다.
4. 그리드 상단에 총 건수 표시가 필요한 경우 call back 함수에서 해당 내용을 출력한다.

|  |
| --- |
| **①**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Script Include  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 변수 선언부  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **②** scwin.fs\_objPagingInfo00 = null;//페이징 관련 object 변수  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 트랜잭션 객체 선언  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*-------------------------------------------------------------------------------  \* 설명 : 메세지 조회  \*------------------------------------------------------------------------------\*/  scwin.SVC\_SEARCH = { // Transaction 요청전 validation 체크 [옵션]  preSvc : function(){  // 전처리  }  , submissionId : "sbmLogin " // Transaction 서비스ID  // Transaction 요청 경로(controller @RequestMapping value) [필수]  , action : "/sample/paging.do"  // 송신 Dataset(복수일 경우 " "으로 구분)[옵션]  , ref : [{"id":"dmLogin","key":"dlUser"}]  **④**, target : [{"id":"dmLoginResult","key":"dlUserInfo"}] "  // Callback 함수[옵션]  , callback : function (nErrorCode, strErrorMsg){  //서비스 이후 처리할 로직 처리  **⑥** this.gfn\_trace("총 " + this.gfn\_numFormat(this.dsParam.getColumn(0, "totCnt")) +  " 건");  }  };  ...  scwin.fn\_init = function ()  {  /\*\*  \* 페이징을 사용하기 위해 초기화 함수를 호출.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* this - 현재 scope  \* this.grdList - 페이징 처리 할 대상 그리드 콤포넌트  \* this.fn\_search - 공통에서 호출될 서비스 함수  \* "S" - scroll 사용시 구분문자  \* 50 - 페이지당 보여줄 리스트 수  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  this.fs\_objPagingInfo00 = new ExtLib.Paging(this, this.grdList, this.fn\_search, "S", 50);    //초기 조회  **③** this.fn\_search();  }  scwin.fn\_search = function(objButton)  {  //서비스 호출전 object에 페이징 info 지정 처리.  scwin.SVC\_SEARCH.pagingInfo = this.fs\_objPagingInfo00;    scwin.gfn\_transaction(this.SVC\_SEARCH);  }} |

* 그리드 하단에 페이지 출력을 통한 조회를 위해서 필요한 작업은 다음과 같다.
* UI 개발 시 추가해야 할 내용을 살펴보면 다음과 같다.

1. 페이지 처리에 필요한 공통 스크립트를 include 한다.
2. 스크롤 페이지 처리를 위한 전역변수를 선언하고 초기화 한다.
3. 페이징 object를 생성한다. new ExtLib.Paging(this, this.grdList, this.fn\_search, "S", 50);

. objForm (대상 폼) - 현재 scope

. this.grdList - 페이징 처리 할 대상 그리드 콤포넌트

. this.fn\_search - 공통에서 호출될 서비스 함수

. "D" - number타입 paging 사용시 구분문자

. 20 - 페이지당 보여줄 리스트 수

. 10 - 보여줄 페이지수

. this.divPaging - 페이징 영역 division 콤포넌트

. "dsParam" - 'totCnt' 를 가지고 다닐 데이타셋명칭 default : dsParam

1. 수신데이타셋은 그리드에 바인드된 데이터셋을 기입한다. (outDatasets : "dsList=dsResult dsParam=dsParam")
2. Transaction 전 넘겨줄 pagingInfo object 를 셋팅한다. (pagingInfo : this.fs\_objPagingInfo00)
3. 조회함수를 호출한다.
4. 그리드 상단에 총건수 표시가 필요한 경우 call back 함수에서 해당 내용을 출력한다.

|  |
| --- |
| **①**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Script Include  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  include "lib::ComLib.js";// 공통함수 호출  **②**  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 변수 선언부  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*\*  \* 페이징을 사용하기 위해 초기화 함수를 호출.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* this - 현재 scope  \* this.grdList - 페이징 처리 할 대상 그리드 콤포넌트  \* this.fn\_search - 공통에서 호출될 서비스 함수  \* "D" - number타입 paging 사용시 구분문자  \* 20 - 페이지당 보여줄 리스트 수  \* 10 - 보여줄 페이지수  \* this.divPaging - 페이징 영역 division 콤포넌트  \* "dsParam" - 'totCnt' 를 가지고 다닐 데이타셋명칭 default : dsParam  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  //페이징 관련 object 변수  **③**this.fs\_objPagingInfo00 = new ExtLib.Paging(this, this.grdList, "fn\_search", "D", 20, 10, this.divPaging);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 트랜잭션 객체 선언  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*-------------------------------------------------------------------------------  \* 설명 : 메세지 조회  \*------------------------------------------------------------------------------\*/  scwin.SVC\_SEARCH = {  // Transaction 요청전 validation 체크 [옵션]  preSvc : function(){  // Transaction 서비스ID  , svcId : "SVC\_SEARCH"  // Transaction 요청 경로(controller @RequestMapping value) [필수]  , controller : "/sample/paging.do"  // 송신 Dataset(복수일 경우 " "으로 구분)[옵션]  , inDatasets : "dsParam=dsParam"  // 수신 Dataset(복수일 경우 " "으로 구분)[옵션]  **④**, outDatasets : "dsList=dsResult dsParam=dsParam"  // Dataset 외의 Transaction을 위한 인자값 a='b'의 형태로 입력  , args : ""  // Callback 함수[옵션]  , callback : function (nErrorCode, strErrorMsg){  //서비스 이후 처리할 로직 처리  **⑦**this.gfn\_trace("총 " + this.gfn\_numFormat(this.dsParam.getColumn(0, "totCnt")) + " 건");  }  //페이징 object 추가  **⑤** ,pagingInfo : this.fs\_objPagingInfo00  };  ...  /\*-------------------------------------------------------------------------------  \* 설명 : 폼 초기화 처리  \*------------------------------------------------------------------------------\*/  this.fn\_init = function ()  {  //초기 조회  this.fn\_search();  }  /\*-------------------------------------------------------------------------------  \* 설명 : 조회 버튼 클릭시  \*------------------------------------------------------------------------------\*/  this.fn\_search = function(objButton)  {  **⑥**this.gfn\_transaction(this.SVC\_SEARCH);  } |

* 페이징 처리를 위해서 서버단에서 필요한 작업은 다음과 같다.
* Controller 클래스의 해당 메서드에서 다음과 같이 UI로부터 수신한 Dataset을 추출해서 서비스 호출 시 인자로 전달한다.

|  |
| --- |
| // UI에서 정의한 Param Dataset을 가져와서 검색조건을 위한 정보를 생성한다.  List sampleList = sampleMapper.selectListForPaging(sqlId,searchMap) |

* SQL 작성 : UI로부터 전달받은 페이지 파라미터 값을 이용하여 SQL 조건문을 작성하고 총 건수는 쿼리에 포함되는 것을 기본 규칙을 한다.

|  |
| --- |
| SELECT AA.\* /\* aaa.bbb.pagingSample \*/  FROM(  SELECT /\*+ index(a PK\_TBCIB31) \*/  ROW\_NUMBER() OVER(ORDER BY a.IDX ) ROW\_NUM  , COUNT(1) OVER() AS TOT\_CNT  , IDX  , OBJECT\_NAME  , TEST\_TYPE  , TEST\_VALUE  FROM PAGING\_TEST a) AA  WHERE ROW\_NUM >= #{firstIndex} AND ROW\_NUM <= #{lastIndex} |

* 차세대 시스템은 대량 데이터 조회 시 발생할 수 있는 시스템 오류를 방지하기 위해 최대 조회 건수 (OOOO건)가 설정되어 있다. 최대 조회 건수 보다 많은 데이터 조회 시 UI에 오류 메시지가 전달된다.
* 데이터(list) 조회 시 사용자에게 보여지는 페이징 처리 없이 전체 데이터 조회가 필요한 경우 다음 내용을 준수하여 개발하면 공통 모듈에서 최대 건수만큼 반복 조회하여 요청한 dataset을 리턴한다.

(페이징 기능 이용)

* UI 개발 시 추가해야 할 내용을 살펴보면 다음과 같다.

1. 전체 조회 처리에 필요한 공통 스크립트를 include 한다. (페이징 스크립트 이용)
2. 스크롤 페이지 처리를 위한 전역변수를 선언 한다.
3. Transaction object에서 현재페이지와 토탈페이지를 셋팅한다.
4. Outdataset 으로 ~Paging으로 생성한 데이터셋을 기술한다.
5. 조회 함수에 페이징 처리를 위한 기본 설정 값을 선언한다.

. this.fs\_objPagingInfo00 = new ExtLib.Paging(this, this.grdList, this.fn\_search, "A", 700);

. this - 현재 scope

. this.grdList - 페이징 처리 할 대상 그리드 콤포넌트

. this.fn\_search - 공통에서 호출될 서비스 함수

. A" - 자동조회

1. 자동페이징 처리 시 페이징 건의 마지막 자료를 취득하여 NEXT를 처리하기 위한 파라미터를 설정하고 계속 서비스를 수행한다.
2. 서비스 호출전 object에 페이징 지정 처리
3. Transaction 서비스를 호출한다.

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Script Include  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **①**include "lib::ComLib.js";// 공통함수 호출  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 변수 선언부  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **②**this.fs\_objPagingInfo00 = null;//페이징 관련 object 변수    /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 화면 트랜잭션 객체 선언  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*-------------------------------------------------------------------------------  \* 설명 : 메세지 조회  \*------------------------------------------------------------------------------\*/  this.SVC\_SEARCH = {  preSvc : function(){ // Transaction 요청전 validation 체크 [옵션]    /\*\*  \* 조회 조건에 페이지 정보를 담아서 전달.  \* getCurrentPage - 현재페이지 (default 1)  \* getRecordCountPerPage - 페이지당 보여질 리스트수 \*/  this.dsParam.clearData(); this.dsParam.addRow(); **③**this.dsParam.setColumn(0,"currentPage", this.fs\_objPagingInfo00.getCurrentPage()); this.dsParam.setColumn(0,"recordCountPerPage", this.fs\_objPagingInfo00.getRecordCountPerPage());  }  // Transaction 서비스ID  , svcId : "SVC\_SEARCH"  // Transaction 요청 경로(controller @RequestMapping value) [필수]  , controller : "/sample/paging.do"  // 송신 Dataset(복수일 경우 " "으로 구분)[옵션]  , inDatasets : "dsParam=dsParam"  // 수신 Dataset(복수일 경우 " "으로 구분)[옵션]  **④**, outDatasets : "dsListPaging=dsResult dsParam=dsParam"  // Dataset 외의 Transaction을 위한 인자값 a='b'의 형태로 입력하고, 빈칸으로 구분[옵션]  , args : ""  // Callback 함수[옵션]  , callback : function (nErrorCode, strErrorMsg){  }  };  /\*-------------------------------------------------------------------------------  \* 설명 : 폼 초기화 처리  \*------------------------------------------------------------------------------\*/  this.fn\_init = function ()  {  /\*\*  \* 페이징을 사용하기 위해 초기화 함수를 호출.  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* this - 현재 scope  \* this.grdList - 페이징 처리 할 대상 그리드 콤포넌트  \* this.fn\_search - 공통에서 호출될 서비스 함수  \* "A" - 자동조회  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \*/  **⑤**this.fs\_objPagingInfo00 = new ExtLib.Paging(this, this.grdList, this.fn\_search, "A", 700);    //초기 조회  this.fn\_search();  }  **⑥**  /\*----------------------------------------------------------------------------  \* 함수명 : fn\_nextPage()  \* Desc : 자동페이징 처리 시 페이징 건의 마지막 자료를 취득하여  \* NEXT를 처리하기 위한 파라미터를 설정하고 계속 서비스를  \* 수행한다.  \* Argument : 없음  \* return Type : boolean  \* return 내용 : true / false  \*--------------------------------------------------------------------------\*/  this.fn\_nextPage = function ()  {  if( this.gfn\_trim(this.dsParam.getColumn(0, "nextRowYn")) == "N" ) {  return false;  } else {  return true;  }  }  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  \* 각 COMPONENT 별 EVENT 영역 (필수)  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\*-------------------------------------------------------------------------------  \* 설명 : 조회 버튼 클릭시  \*------------------------------------------------------------------------------\*/  this.fn\_search = function(objButton)  {  //서비스 호출전 object에 페이징 지정 처리.  **⑦**this.SVC\_SEARCH.pagingInfo = this.fs\_objPagingInfo00;    **⑧**this.gfn\_transaction(this.SVC\_SEARCH);  } |

### 파일 업로드/다운로드 표준

* 파일 첨부는 공통 기능을 이용하여 구현하는 것을 표준으로 한다. 첨부된 파일은 공통 테이블을 통해 그 내역이 관리되며 업무 테이블에서는 문서번호만 가지면 된다. /XXXXX\_PROJECT/temp/xxxxx/
* 파일 저장 경로는 다음과 같다.
* 업무별 파일 저장 경로 (로컬 PC)

|  |  |  |
| --- | --- | --- |
| 업무 | 저장 경로 | 임시 저장 경로 |
| 업무관리 | D:\xxxxx\_project\temp\xxxxx\년\월\일 | D:\attach\_tmp\xxxxx |

* 업무별 파일 저장 경로 (개발 서버)

**Sample**

|  |  |  |
| --- | --- | --- |
| 업무 | 저장 경로 | 임시 저장 경로 |
| 업무관리 | D:\xxxxx\_project\temp\xxxxx\년\월\일\[1,2,3…]  (최대 100개) | /attach\_tmp/xxxxx/ |

### Chart 표준

* Chart 작성은 공통 기능을 이용하여 구현하는 것을 표준으로 한다.

src/main/webapp/xpf/lib/common\_chart.js의 gfn\_InitChart 함수를 사용하여 그린다.

### SMS 표준

* SMS 전송 화면이 필요한 경우 공통에서 작성한 SMS 화면을 호출해서 사용한다. 별도의 화면 없이 로직을 구현하고자 하는 경우 SMS API를 호출해서 사용한다.

### E-Mail 표준

* E-Mail 전송 화면이 필요한 경우 공통에서 작성한 E-Mail 화면을 호출해서 사용한다. 별도의 화면 없이 로직을 구현하고자 하는 경우 E-Mail API를 호출해서 사용한다.

페이지의 마지막 줄