*com

enRiseFramework Base
컴포넌트 개발 가이드

Version 1.0

2017.01.18

SN Technolodgy

개정 이력

버전	제/개정 일자	제/개정 페이지 및 수정 내용	제/개정자	승인자
1.0	2017.01.18	최초 작성	최승환	

목 차

1 enRiseFramework 실행환경

1.1 개요

온라인 개발절차 및 enRiseFramework 실행환경을 설명한다

1.2 코딩 원칙

- 주석은 모든 코드에 상세히 기술하는 것을 원칙으로 한다
- 소스 코드는 불가피한 사항을 제외하고 원칙적으로 중복을 금지한다
- 소스 코드에서 사용하는 모든 단어는 영어단어의 풀네임으로 작성하되 10 자 넘어가는 경우 약어를 사용해도 된다
- 소스 코드는 반드시 PMD 개발 원칙 및 보안 규격을 준수하여야 한다
- 파일은 2000 줄을 넘지 않는 것을 원칙으로 한다

1.3 Java 코딩 규칙

1.3.1 Package 문

- 반드시 자바파일의 첫번째 줄은 Package 문을 사용한다
- Package 는 다음 명명 규칙을 따른다

(회사구분).(제품구분).(기능구분).(세부유형 .. n)

ex) com.enrise.office.apvl.common

1) 제품구분

제품구분	설명
framework	enRiseFramework 공통 패키지
office	그룹웨어시스템 패키지
share	모든 제품의 공통 패키지
mobile	모바일 그룹웨어시스템 패키지
kms	지식관리시스템 패키지
test	테스트 패키지
config	설정파일 패키지

2) 기능구분

기능구분	설명
aars	문서관리 패키지
adbk	주소록 패키지
apvl	전자결재 패키지
comm	커뮤니티 패키지
infc	연계 패키지
mail	이메일 패키지
mes	메신지 연계 패키지
nboard	전자게시 패키지
nportal	포털 패키지
poll	설문조사 패키지
portal	구)포털 패키지
rsrc	자원예약 패키지
rule	규정집 패키지
scdl	일정관리 패키지
sop	구)업무편람 패키지
system	시스템관리 패키지
work	업무편람 패키지
alarm	알림 기능 패키지
cmmn	공통 기능 패키지
file	파일처리 패키지
orgn	조직관리 패키지
plugin	Plugin 공통 패키지
role	권한관리 패키지
util	유틸리티 패키지

3) 세부유형

세부유형	설명
common	공통 클래스 패키지
service.impl	서비스 클래스 패키지

dao	Data Access Object 클래스 패키지
vo	Value Object 클래스 패키지
web	Controller 클래스 패키지
filter	Filter 클래스 패키지
validator	Validator 클래스 패키지
interceptor	Interceptor 클래스 패키지
handler	Handler 클래스 패키지
view	View 클래스 패키지
bind	Bind 클래스 패키지
schedule	스케줄러 Job 클래스 패키지

1.3.2 Import 문

- import 문은 반드시 package 문 다음에 작성한다
- import 는 사용하는 클래스까지 정확하기 작성하며 * (모든클래스)는 사용하지 않는 것을 원칙으로 한다
- import 는 패키지명 회사구분순으로 정렬하고 엔터 공백으로 구분한다
- 동일한 회사 구분시 제품구분, 세부기능, 유형 순으로 정렬하여 작성하며 공백을 두지 않는다

ex)

```
import java.util.ArrayList;
import java.util.Map;
import java.io.File;   <---- (같은 회사구분 패키지인 경우는 공백을 두지 않는다)
                                <---- (회사구분 마다 엔터 공백을 통해 구분함)
import com.enrise.share.util.Globals;
import com.enrise.framework.db.AbstractDAO;
                                <---- (회사구분 마다 엔터 공백을 통해 구분함)
import org. springframework.stereotype.Repository;
```

1.3.3 주석문

1) 클래스 주석

- 클래스 주석은 반드시 클래스 바로 위에 작성한다. 단 사용되는 어노테이션이 있는 경우 어노테이션 위에 작성 한다
- 클래스 주석은 반드시 javadoc 주석을 사용한다

- @since 와 @author 는 반드시 작성한다

- author 는 반드시 본인 이름을 작성한다 (약어나 별칭 사용 금지)

- 주석안에 설명내용은 한줄이상인 경우 HTML 형식으로 작성한다

ex)

```
/**
 * <p>Exception 발생시 AOP(after-throwing) 에 의해 후처리로직 연결고리 역할 수행하는
클래스이다.</p>
 * <p>Exception 종류를 BizException, RuntimeException(DataAccessException 포함),</p>
 * @since JDK1.5
 * @author 이동하
 */
public class ExceptionTransfer {
```

2) 멤버변수 (Field) 주석

- Filed 주석은 반드시 javadoc 주석을 사용한다

ex)

```
public class ExceptionTransfer {
    /**
     * ExceptionTransfer 로그 객체
     */
    protected Log log = LogFactory.getLog(this.getClass());
}
```

3) 멤버함수 (Method) 주석

- Method 주석은 반드시 javadoc 주석을 사용한다

- @param 과 @return 은 반드시 작성한다 (@param 과 @return 사이에는 (Space)공백으로 구분한다

- 주석안에 설명내용은 한줄이상인 경우 HTML 형식으로 작성한다

ex)

```
public class ExceptionTransfer {
        /**
         * 발생한 Exception을 프레임워크에서 사용하는 Exception으로 변경하여 생성한다
         * @param clazz Exception을 생성할 클래스
         * @param msgKey messageSource Key
         * @param msgArgs messageSource에 전달할 파라미터 배열
         * @param e 발생한 Exception
         * @param locale 언어
```

```
         * @return 생성한 Exception
         */
        protected Exception processException(final Class clazz,
                final String msgKey, final String[] msgArgs,
                final Exception e, Locale locale) {
            return processException(clazz, msgKey, msgArgs, e, locale, null);
        }
}
```

4) 구현로직 주석

- 메서드 내의 구현로직 주석은 반드시 /* */ 또는 // 주석만을 사용하는 것을 원칙으로 한다

- 변수를 제외하고 구현로직 상단에 주석을 작성하는 것을 원칙으로 한다

ex) Single-Line 주석

```
/* 조건 문을 취급함 */
if(condition) {
    ...
}
```

ex) Trailing 과 End-Of-Line 주석

```
//Trailing 주석
if(condition) {
   return true; /* 조건 문을 취급함 */
}else {
   return false; /* 짧은 주석인 경우만 사용함 */
}
//End-Of-Line  주석
if(condition) {
   return true;  // 조건문을 취급함
}else {
   return false;  // 짧은 주석인 경우만 사용함
}
```

1.3.4 Class 와 Interface

- Class 와 Interface 는 Pascal Case 명명규칙을 따른다 (시작단어 대문자 중간에 이어지는 단어 대문자)

- 일반적인 약어(DB, IT 등)를 제외하고 영어단어 풀네임을 원칙으로 한다

- Interface 는 클래스 명명규칙이 동일하다

  Interface 를 통해 구현된 클래스는 반드시 Impl Subfix 를 작성한다

- VO 클래스는 DB 테이블명과 동일하게 작성한다 (단, join 을 통해 변경된 VO 인 경우는 클래스 subfix 에 VO 를 작성함)

- VO 와 Framework 클래스를 제외한 나머지 클래스는 반드시 다음과 같은 명명규칙을 따르는 것을 원칙으로 한다 (기능구분은 패키지 기능구분과 동일함)

  (기능구분) + (세부기능명) + Subfix

  ex) ApvlInterfaceServiceImpl

  4-1) Subfix

Subfix	설명
Controller	Controller 클래스 Subfix 명
DAO	DAO 클래스 Subfix 명
Service	Service 인터페이스 Subfix 명
ServiceImpl	Service 인터페이스 구현 클래스 Subfix 명
Handler	Handler 클래스 Subfix 명
Filter	Filter 클래스 Subfix 명
Util	Util 클래스 Subfix 명
CoGlobals	패키지 공통 클래스 Subfix 명
Interceptor	Interceptor 클래스 Subfix 명
Job	스케줄러 Job 클래스 Subfix 명

1.3.5 들여쓰기 (Indentation)

- Space 공백 4 개를 한 단위의 들여쓰기 표준으로 한다. 들여쓰기는 항상 탭(Tab)을 사용해야 하며, 모든 탭은 Space 공백 4 로 맞추어야 한다

1.3.6 줄의 길이 (Line Length)

- 한 줄에 80 글자 이상을 작성하지 않는 것을 원칙으로 한다

1.3.7 줄 나누기 (Wrapping Lines)

- 어떤 표현(expression)이 한 줄을 넘어갈 때는 다음의 일반적인 원칙에 따라서 줄을 나누도록 한다

  * 쉼표 뒤에 나눌 것

  * 연산자(operator)의 앞에서 나눌 것

  * 하위 단계에서 나누기 보다는 상위 단계에서 나눌 것

  * 새로운 줄의 시작부분을 윗줄의 같은 수준의 표현에 맞추어 정렬할 것

  * 위 규칙을 따랐는데 코드가 혼란스럽거나, 오른쪽 끝을 넘어가는 경우에는 8 글자 만큼만 들여쓸 것

ex) 메소드 호출을 나누는 예

```
someMehtod(longExpression1, longExpression2, longExpression3,  <---- 쉽표 뒤에 나눔
    longExpression4, longExpression5);  <---- 줄이 나눠진 경우 Indent 를 두어 다음 명령어와 혼동이 되지
                                            않도록 한다
```

ex) 산술 연산 표현을 나누는 예

```
longName1 = longName2 * (longName3 + longName4 – longName5)
    + 4 * longName6;  <---- 연산자 앞에서 나누며, 상위단계에서 나누어저 해석하기사 수월함
longName1 = longName2 * (longName3 + longName4     (잘못된 예시 – 피해야할 사항)
    - longName5) + 4 * longName6;  <---- 하위단계에서 나누어서 해석하기가 어려움
```

ex) 메소드 선언시 들여 쓰는 예

```
// 전통적인 들여쓰기
someMethod(int anArg, Object anotherArt, String yetAnotherArt,
            Object andStillAnother) {
      …
}
// 너무 깊게 들여 쓰는 것을 피하기 위해 8 칸만 들여 쓴다
private static synchronized horkingLongMethodName(int anArg
            Object anotherArg, String yetAnotherArg,
            Object andStillAnother) {
      …
}
```

ex) if 문에 대해서는 전통적인 4 칸 들여쓰기를 할 경우 본문을 알아보기가 어렵기 때문에 8 칸 들여쓰기를 한다

```
// 이렇게 쓰지 말 것
if ((condition1 && condition2)
        || (condition3 && condition4)
        || !(condition5 && condition6)) { // 안 좋은 줄 맞추기
        doSomethingAboutIt(); // 이 줄을 알아보기 어렵다
}
// 대신 이렇게 쓸 것
if ((condition1 && condition2)
                || (condition3 && condition4)
                || !(condition5 && condition6)) {
        doSomethingAboutIt();
}
// 또는 이렇게 쓸 것
if ((condition1 && condition2) || (condition3 && condition4)
                || !(condition5 && condition6)) {
        doSomethingAboutIt();
}
```

1.3.8 선언 (Declarations)

1) 줄당 선언 개수

- 한줄에 하나의 변수만 선언하는 것을 원칙으로 한다

ex)

```
int level, size; //잘못된 예시
//아래와 같이 한줄에 하나의 변수만을 선언
int level;
int size;
```

2) 초기화(Initialization)

- 메소드내에 지역변수로 선언된 경우는 반드시 초기화를 해야 한다

3) 배치(Placement)

- 선언은 블록의 시작부분에만 배치한다

ex)

```
void myMethod() {
    int int1 = 0; // 메소드 블록의 시작부분
    if (condition) {
        int int2 = 0; // "if"블록의 시작부분
        ...
    }
}
// 단, for 문의 인덱스에 대해서는 예외로 한다.
for (int i = 0; i < maxLoops; i++) { ... }
```

4) Class 와 Interface 선언

- 메소드 명과 매개변수 목록이 시작되는 괄호 "(" 사이에는 공백을 두지 않는다
- 블록 "{"의 시작은 선언문과 같은 줄의 끝에 놓는다
- 블록 "}"의 끝은 새로운 줄 시작부분에 작성하며, 시작 블록이 선언된 위치와 같은 수준에 작성한다
- 필드와 메소드, 메소드와 메소드 사이에는 엔터 공백을 둔다

ex)

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}
    ...
```

```
}
```

1.3.9 문(Statements)

1) 단순 문(Simple Statements)

- 각 줄에는 단 한 개의 문만 작성한다

ex)

```
argv++;
argc--;
argv++;argc--; //잘못된 예시 - 피해야 할 사항
```

2) 복합 문(Compound Statements)

- 복합 문은 블록 "{" 문 "}" 형식으로 여러 개의 문을 포함하는 문을 말한다

- 포함된 문들은 복합 문보다 한 단계 더 들여 써야 한다

- 블록 "{" 시작은 시작되는 줄의 끝에 위치하고 블록 "}" 끝은 새로운 줄의 시작부분에 작성하며 시작된 줄의 위치와 동일한 수준에 작성한다

- 모든 문은 괄호로 둘러싸여야 한다

3) 반환 문(Return Statements)

- 값이 있는 반환문의 경우 반환값을 더 명확하게 해주지 않는 이상 괄호를 사용하지 않는다

ex)

```
return;
return myDisk.size();
return (size ? size : defaultSize);
```

4) if, if-else, if else-if else 문

- if 문은 반드시 괄호를 사용한다

- if 문의 조건부 괄호와 블록 "{" 시작 사이에는 공백을 둔다

ex)

```
if(조건) {
    문;
}
if(조건) {
    문;
}else {
```

```
    문;
}
if(조건) {
    문;
}else if (조건) {
    문;
}else {
    문;
}
if(조건) 문; //잘못된 예시 – 피해야할 사항 (문법상 오류는 없지만 가독성이 떨어짐)
```

5) for 문

ex)

```
for (초기화; 조건; 갱신) {
    문;
}
```

6) while 문

ex)

```
while (조건) {
    문;
}
do {
    문;
} while (조건);
```

7) switch 문

- break 문을 사용하지 않는 경우 반드시 주석을 표시해야 한다
- default case 는 반드시 작성한다

ex)

```
switch (조건) {
case ABC:
    문;
```

```
    /* DEF Case 도 처리되어야 함 */
case DEF:
    문;
    break;
case XYZ:
    문;
    break;
default:
    문;
    break;  //default case 는 break 문을 생략해도 됨
}
```

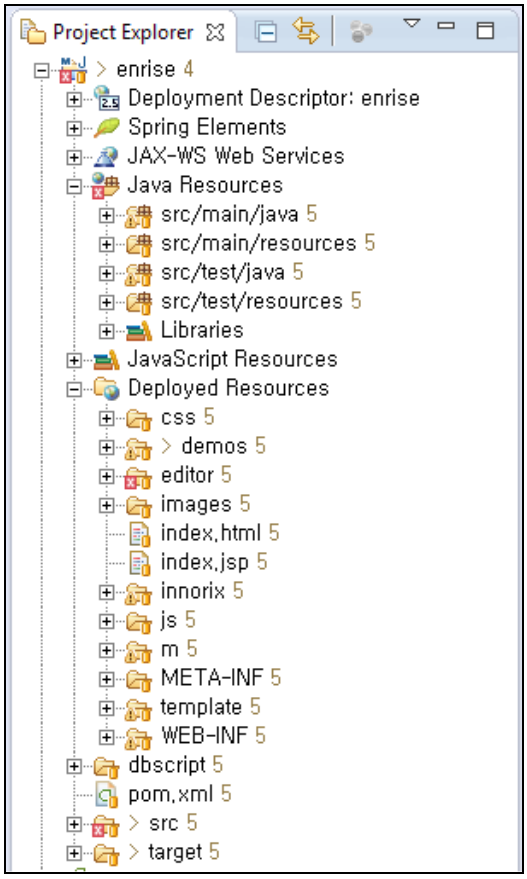
8) try-catch 문

ex)

```
try {
    문;
}catch (ExceptionClass e) {
    문;
}
try {
    문;
}catch (ExceptionClass e) {
    문;
}finally {
    문;
}
```

1.4 프로젝트 폴더 구조

- enRiseFramework 는 아래와 같은 폴더 구조를 구성되어 있다



폴더구분	폴더구조	설명
Java Resources	src/main/java	Java 클래스 파일
	src/main/resources	Ibatis, Spring. Properties, message 등 설정 파일
	src/test/java	Test (JUnit, JMock) 클래스 파일
	src/test/resources	Test 를 위한 Spring 설정 파일
Deployed Resources	css	Cascade Style Sheet 파일
	demos	웹 Client 단위 개발 및 Plugin Demo 파일
	editor	웹 에디터 (smart editor, namo editor, polaris editor) 파일
	images	각 종 이미지 파일
	Js	Javascript 파일
	m	모바일 UI 파일
	WEB-INF/jsp	Java Servlet Page 파일
	WEB-INF/tlds	Custom Tag library 설정 파일

	WEB-INF/tilesdef.xml	Tiles Layout 설정 파일
	WEB-INF/web.xml	Web Application 설정 파일
dbscript	mysqlUTF8	Mysql DDL, DML 파일
	oracleUTF8	Oracle DDL, DML 파일
maven deploy	target	Maven deploy 임시 폴더
	pom.xml	Maven 설정 파일

1.5 공통기반 핵심

1.5.1 IoC Container

- Spring Framework 에서 Bean 은 어플리케이션을 구성하고, IoC Container 에 의해 관리되어 지는 객체를 의미한다.

  enRiseFramework 에서 Bean 설정은 Java Resources/src/com/enrise/config/spring 폴더의 XML 파일로 관리되고 있으며 각 파일은 다음과 같은 의미를 가진다.

  (주의 XML 파일을 새로 추가하는 경우 반드시 context-그룹명.xml 형식으로 저장해야 한다)

- 각 XML 의 설정된 Bean 의 상세 설명 및 예제는 5. 업무처리 API 를 참고한다

파일명	설명
context-aspect.xml	후속처리 관련 AOP Bean 설정
context-asynctask.xml	비동기 처리 관련 Bean 설정
context-common.xml	공통으로 사용되는 Bean 설정
context-ekp-servlet.xml	ekp Context 에서만 사용되는 Bean 설정
context-mail.xml	이메일 처리관련 Bean 설정
context-orguserdetailshelper.xml	사용자 인증 방식관련 Bean 설정
context-restful.xml	RestFul 연계관련 Bean 설정
context-scheduler.xml	스케줄(배치) 관련 Bean 설정
context-sqlmap.xml	Ibatis DB 처리 관련 Bean 설정
context-transaction.xml	DB DataSource 및 트랜젝션 관련 Bean 설정
context-infc.xml	연계관련 Bean 설정
context-jms.xml	Java Message Service 관련 Bean 설정
context-mes.xml	구)메신저 연계 관련 Bean 설정

1.5.2 AOP

- enRiseFramework 의 AOP 는 @AspectJ 어노테이션 기반이 아닌 XML 설정 방식을 사용한다.

  예를 들어 트랜젝션 처리의 경우도 이와 같이 XML 설정을 통해 Point Cut, Advice 를 설정하도록 되어 있다

- 트랜젝션 처리와 관련된 AOP 설정은 아래와 같다

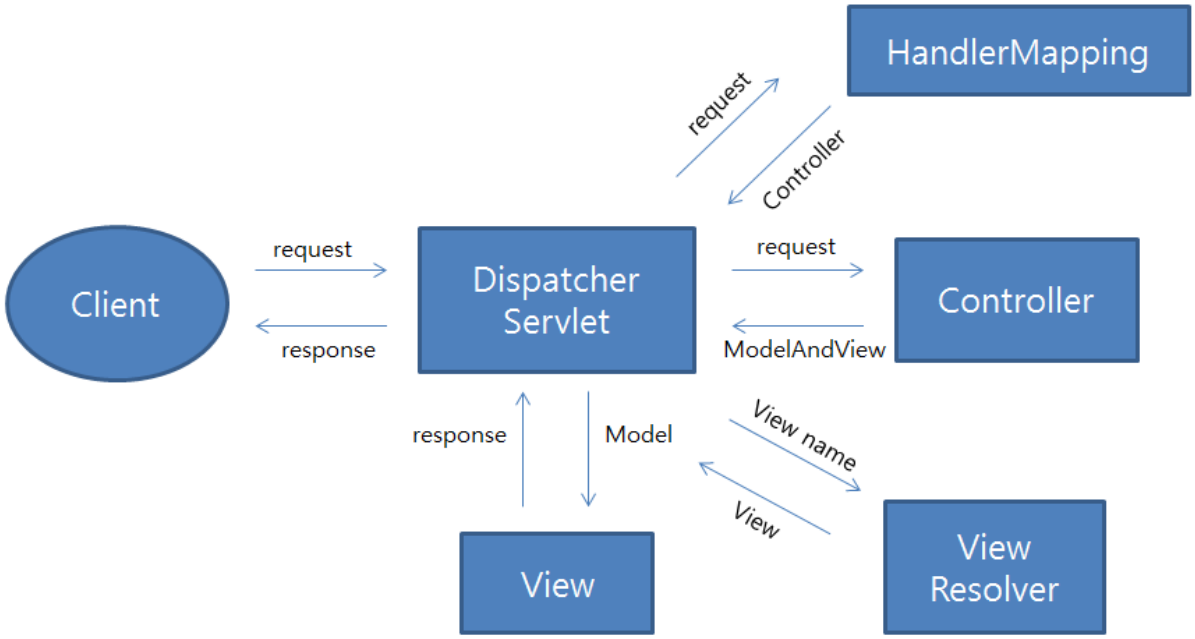
  작성된 클래스명이 Expression 과 일치하는 경우를 Point Cut 으로 설정하였고, Advice 가 실행되는 시점은 Exception 이 발생될 경우로 설정하여, com.enrise.office 하위의 모든 폴더 중 Impl 로 끝나는 클래스명의 모든 메서드가 실행될 때 Exception 이 발생될 경우 트랜젝션이 Rollback 이 되도록 구성되었다

```
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource" />
</bean>
<tx:advice id="txAdvice" transaction-manager="txManager">
        <tx:attributes>
                <tx:method name="*" propagation="REQUIRED" rollback-for="Exception" />
        /tx:attributes>
</tx:advice>
<aop:config>
        <aop:pointcut id="requiredTx"
                        expression="execution(* com.enrise.office..*Impl.*(..)) or
                                execution(* com.enrise.share..*Impl.*(..)) or
                                execution(* com.enrise.framework..*Impl.*(..))" />
        <aop:advisor advice-ref="txAdvice" pointcut-ref="requiredTx" />
</aop:config>
```

2 MVC

2.1 개요

enRiseFramework 는 Spring MVC 패턴으로 구성되어 있으며, 각각의 Model, View, Controller 의 개발방식을 설명한다



1) 클라이언트의 요청이 들어오면 Dispatcher Servlet 이 가장 먼저 요청을 받는다
2) HandlerMapping 이 요청에 해당하는 Controller 를 return 한다
3) Controller 는 비즈니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView 에 반영하여 return 한다
4) ViewResolver 는 view name 을 받아 해당하는 View 객체를 return 한다
5) View 는 Model 객체를 받아 rendering 한다

2.2 Annotation-based Controller

Controller 클래스를 생성하기 위해서는 com.enrise.framework.web.AbstractController 클래스를 상속받아야 하며 클래스에 @Controller 어노테이션을 붙혀주면 된다

```
package com.enrise.office.apvl;

@Controller

public class ApvlListController extends AbstractController {

    ...

}
```

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + web

  ex) com.enrise.office.apvl.web (그룹웨어시스템의 전자결재업무)

- 클래스명 : (기능구분) + (세부업무명) + Controller

  ex) ApvlListController (전자결재 리스트 관련 Controller 클래스)

2) @Request Mapping 어노테이션

- 요청에 대해 어떤 Controller, 어떤 메소드가 처리할지를 맵핑하기 위한 어노테이션이다.
  @RequestMapping 이 사용하는 속성은 아래와 같다

이름	타입	설명
value	String[]	URL 값을 맵핑되는 조건으로 부여한다 @RequestMapping(value="/sample.do") 또는 @RequestMapping(value={"/sample.do", "/sample_add.do"}) 와 같이 표기하며, 기본값이기 때문에 @RequestMapping("/sample.do")와 같이 표기할 수 도 있다. URL 은 "/sample/*.do" 와 같이 Ant-Style 의 패턴매칭을 이용할 수 도 있다. 단, 반드시 요청 URL 은 .do 를 사용해야 한다 (web.xml 에 .do 인 요청만 Dispacher Servlet 이 호출되도록 설정됨) **사용자 인증 방식으로 인해 ajax 처리 요청인 경우는 반드시 /ajax/sample.do 처럼 앞에 /ajax 를 반드시 지정해야 한다**
method	RequestMethod[]	HTTP 요청 방식을 맵핑되는 조건으로 부여한다 @RequetMapping(method = RequestMethod.POST) 와 같이 표기한다. 사용 가능한 Method 는 GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE 이다
params	String[]	HTTP 파라미터를 맵핑 조건으로 부여한다 @RequestMapping(params="myParam=myValue")와 같이 표기한다

ex)

```
@RequestMapping(value="/apvl/setApproval.do", method=RequestMethod.POST)
@ResponseBody
public ResultVO setApproval(@RequestBody ApprovalVO paramDatas, HttpServletRequest request) {
    ...
}
```

3) @RequestParam 어노테이션

- Controller 메소드의 파라미터와 웹요청 파라미터와 맵핑하기 위한 어노테이션이다
- @ RequestParam 이 사용하는 속성은 아래와 같다

이름	타입	설명
value	String	파라미터 이름
required	boolean	필수 여부이며 기본값은 true 이다
defaultValue	String	Null 인 경우 기본값

ex)

```
@RequestMapping("/apvl/list/getFormInfoList.do")
public ListVO getFormInfoList(HttpServletResponse response, HttpServletRequest request,
    @RequestParam("viewRowCnt") int viewRowCnt, @RequestParam("currentPage") int currentPage,
    @RequestParam(value="isSaveCookie", defaultValue="Y", required=false) String isSaveCookie)
        throws Exception {
    ...
}
```

4) @RequestBody 어노테이션

- RequestBody 전체를 변경하여 웹요청 파라미터와 맵핑하기 위한 어노테이션이다
- context-ekp-servlet.xml 에 AnnotationMethodHandlerAdapter 의 messageConverters 를 MappingJacksonHttpMessageConverter 로 설정하여 RequestBody 내용이 JSON 문자열로 요청되는 경우, 해당 JSON Object 와 일치하는 VO 객체로 파라미터를 자동 맵핑한다

5) @ResponseBody 어노테이션

- RequestBody 어노테이션과는 반대로 VO 객체를 JSON 문자열로 변환하여 Response 한다

2.3 View Resolver

Controller 에 실행된 비즈니스 처리 결과를 ModelAndView 에 담아 Response 할 View 를 호출해 주는 중간 역할을 해주는 객체로 Controller 와 View 의 분리(decoupling)하여, 코드의 확장 및 유연한 대처가 가능하게 한다

1) BeanNameViewResolver

- ModelAndView 객체에 담긴 View 이름으로 실제 View 객체를 반환한다

2) TilesViewResolver

- Tiles 로 구성된 jsp 파일로 forwarding 또는 redirect 하는 경우 사용한다. return 값을 String 문자열로 지정하면 되며, WebContent/WEB-INF/tilesdef.xml 에 설정된 패턴으로 작성하면 된다

3) InteranlResourceViewResolver

- 비즈니스 로직 처리가 완료된 후 jsp 파일로 forwarding 또는 redirect 하는 경우 사용한다. return 값을 String 문자열로 지정하면 되며, WebContent/WEB-INF/jsp/ 하위폴더부터 명명하고, subfix 는 .jsp 로 구성되어 있으며루 생략하면 된다
- 예제코드

```
return "orgn/ssoLogin";    // WebContent/WEB-INF/jsp/orgn/ssoLogin.jsp 로 forwarding 함
```

2.4 Model – VO 객체

VO 란 ValueObject 의 약자로 O/R Mapping 의 결과를 저장하여 Controller 까지 전달하기위해 추상화된 객체를 말한다.

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + vo

  ex) com.enrise.office.apvl.vo (그룹웨어시스템의 전자결재업무)

- 클래스명 : VO 명+ (VO) - (VO 는 보통 두개의 테이블에서 Join 된 결과를 저장하는 객체인 경우 지정함)

  ex) DraftDocument (결재문서)

2.5 Model – Ibatis 프레임워크

JDBC 를 사용한 Data Access 를 추상화하여 간편하고 쉽게 사용할 수 있는 Data Mapper Framework 이다. Ibatis 를 사용하면 관계형 데이터베이스에 Access 하기 위해 필요한 일련의 자바코드 사용을 현저히 줄일 수 있으며, 간단한 XML 에 SQL 문을 작성하여 Java 객체와 맵핑할 수 있다. 기존 Java 객체에 SQL 을 담아 사용하는 경우보다 SQL 문의 작성이나 수정, 관리가 용이하며, DBMS 가 변경되어도 프로그램을 수정할 필요 없어 XML 만 변경하면 된다는 큰 장점을 가지고 있다.

1) Ibatis Configuration XML 파일

- com.enrise.config.ibatis.(DBMS 벤더명).config 패키지에 작성한다
- 업무단위로 파일을 나눠서 작성하며, 관련 SQL Mapper XML 파일을 설정한다

2) Ibatis SQL Mapper XML 파일

- com.enrise.config.ibatis.(DBMS 벤더명).sqlmap.(업무명) 패키지에 작성한다
- 일반적으로 하나의 Mapper XML 파일에는 한테이블과 맵핑되며 해당 테이블의 Create, Read, Update, Delete SQL 문을 작성한다. 다른 테이블과 Join 을 통해 결과맵을 구성하는 경우에는 업무단위로 파일을 작성해도 무관하다
- 명명규칙 : DAO 클래스명 – SQL – (DBMS 벤터명).xml

  ex) ApvlDocListInfo-SQL-Oracle.xml

- 개발방법 및 자세한 사항은 W_SQL 작성가이드_교육용_v0.9 (패키지용).ppx 를 참고한다

3) Spring 프레임워크와 연계

Ibatis 프레임워크와 Spring 프레임워크와 연계하여 Bean 으로 등록하는 설정 파일은 context-sqlmap.xml 이다. 파일을 열어 설정파일을 확인하면 다음과 같다.

SQLMapClientFactoryBean 을 사용하여 sqlMapClient 객체를 IoC Container 에 생성한다

생성시 DI 되는 객체는 configLocations, dataSource 와 lobHandler 이다. globals.properties 에 DbType 은 현재는 oracle 로 되어 있으며, 추후 DBMS 가 변경될 경우는 이부분을 해당 벤더로 변경하여 SQL Mapper 를 새로 추가 개발해야 한다.

```
<bean id="lobHandler" class="org.springframework.jdbc.support.lob.DefaultLobHandler" lazy-init="true" />
<bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="configLocations">
                <list>
                        <value>classpath:/com/enrise/config/ibatis/${Globals.DbType}/config/*.xml</value>
                </list>
        </property>
        <property name="dataSource" ref="dataSource"/>
        <property name="lobHandler" ref="lobHandler"/>
</bean>
```

2.6 Model – DAO 객체

DAO 클래스를 생성하기 위해서는 com.enrise.framework.db.AbstractDAO 클래스를 상속받아야 하며, 클래스에 @Repository 어노테이션을 붙혀야 한다.

```
package com.enrise.office.apvl.service.impl;
@Repository("ApvlDocListInfoDAO")
public class ApvlDocListInfoDAO extends AbstractDAO {
    ...
}
```

1) 명명규칙

- 패키지 명 : com + enrise + (제품구분) + (기능구분) + service + impl

  ex) com.enrise.office.apvl.service.impl (오피스그룹의 전자결재업무)
- 클래스명 : (기능구분) + (세부업무명) + DAO
- ex) ApvlDocListInfoDAO (전자결재 문서 리스트 관련 DAO 클래스)

2) AbstractDAO

IBK 전자결재시스템 프레임워크에 추상화된 DAO 클래스로 아래와 같은 메서드를 지원한다

메소드명	파라미터	리턴값	설명
insert	queryId, 저장할 Object 정보	Ibatis 의 SelectKey 값	해당 queryId 의 insert SQL 문을 Object 정보로 저장한다
update	queryId, 수정할 Object 정보	수정된 row 수	해당 queryId 의 update SQL 문을 Object 정보로 수정한다
delete	queryId, 삭제할 Object 정보	삭제된 row 수	해당 queryId 의 delete SQL 문을 Object 정보로 삭제한다
selectByPk	queryId, 조회할 Object 정보	조회된 Object 정보	해당 queryId 의 select SQL 문을 Object 조건으로 조회한다
list	queryId, 조회할 Object 정보	조회된 Object 리스트 정보	해당 queryId 의 select SQL 문을 Object 조건으로 조회한다
listWithPaging	queryId, 조회할 Object 정보	조회된 Object 리스트 정보	deprecated 됨

ex)

```
@Repository("ApvlDraftdocumentDAO")
public class ApvlDraftdocumentDAO extends AbstractDAO {


    /**
     * 결재문서 정보를 가져온다.
     * @param String
     * @return ApvlDraftDocument
     * @throws Exception
     */
    public DraftDocument selectDraftdocument(String docId) throws Exception {
        return (DraftDocument) selectByPk("ApvlDraftdocument.selectDraftdocument",docId);
    }
```

```
/**
 * 결재문서 정보를 insert
 * @param DraftDocument
 * @return void
 * @throws Exception
 */
public void insertDraftdocument(DraftDocument info) throws Exception {
    insert("ApvlDraftdocument.insertDraftdocument", info);
}
 /**
 * 결재문서 정보를 update
 * @param DraftDocument
 * @return void
 * @throws Exception
 */
public boolean updateDraftdocument(DraftDocument info) throws Exception {
    return update("ApvlDraftdocument.updateDraftdocument", info) > 0;
}
/**
 * 결재할 문서를 삭제한다
 * @param docId
 * @return
 * @throws Exception
 */
public int deleteDocument(String docId) throws Exception {
    return delete("ApvlDraftdocument.deleteDocument", docId);
}
```

2.7 Model – Service 객체

Serivce 객체는 반드시 Service 인터페이스와 Service 인터페이스 구현 클래스로 구성되어야 한다. Service 인터페이스 구현 클래스는 com.enrise.framework.service.AbstractServiceImpl 클래스를 상속받아야 하며, @Service 어노테이션을 붙혀야 한다

```
/* Service Interface */
public interface ApvlApprovalInfoService {
    ...
}

/* Service 구현 클래스 */
@Service("ApvlApprovalInfoService")
public class ApvlApprovalInfoServiceImpl extends AbstractServiceImpl
            implements ApvlApprovalInfoService {
    ...
}
```

1) 명명규칙

- 인터페이스 패키지 명 : com + enrise + (제품구분) + (기능구분) + service

  ex) com.enrise.office.apvl.service (그룹웨어시스템의 전자결재업무)

- 인터페이스 클래스명 : (기능구분) + (세부업무명) + Service

  ex) ApvlDocListInfoService (전자결재 문서 리스트 관련 Service 인터페이스 클래스)

- 구현 클래스 패키지 명 : com + enrise + (제품구분) + (기능구분) + service + impl

  ex) com.enrise.office.apvl.service.impl (그룹웨어시스템의 전자결재업무)

- 구현 클래스명 : (기능구분) + (세부업무명) + Service + Impl **(Impl 은 반드시 작성해야 함)**

  ex) ApvlDocListInfoServiceImpl (전자결재 문서 리스트 관련 Service 구현 클래스)

2) @Resource 어노테이션

Ioc Container 에 등록된 Singtон Bean 을 가져와서 사용하기 위한 어노테이션이다

XML 에 지정된 id 명 또는 어노테이션 생성시 지정한 id 명과 일치해야 하며, id 가 없는 경우는 객체의 변수명과 name 속성값을 일치시키면 된다

ex) 다음 코드는 등록된 ApvlDocInfoDAO Bean 객체를 Service 에서 사용하는 예제이다

```
@Service("ApvlDocListInfoService")
public class ApvlDocListInfoServiceImpl extends AbstractServiceImpl  implements ApvlDocListInfoService {
        @Resource(name="ApvlDocListInfoDAO")
         private ApvlDocListInfoDAO docListInfoDAO;

         @Override
         public List<DocListVO> selectPendingDocList(DocListVO listInfo) throws Exception {
              ...
              List<DocListVO> resultPendingDocList =  docListInfoDAO.selectPendingDocList(listInfo);
              ...
         }
         ...
```

3) 트랜젝션

트랜젝션 처리는 3. 실행환경 > 3.3 공통기반 핵심의 AOP 를 참고한다

트랜젝션시 주의해야 할 사항은 다음과 같다

- 하나의 Service 메서드당 하나의 트랜젝션이다
- Service 내에서 실행되는 Service 는 같은 트랜젝션이다
- 처리 결과값과 상관없이 Exception 이 발생되는 경우 자동 Rollback 된다
- 트랜젝션과 관련없이 후처리를 하고자 하는 경우는 AbstractServiceImpl 의 leaveTrace 메소드를 사용한다

4) Exception 처리

Service 구현 클래스의 메서드는 반드시 throws Exception 을 감싸줘야 한다

특정 값에 의해 Exception 을 처리하는 경우는 아래의 예제코드와 같이 AbstractServiceImpl 의 processException 을 사용한다

processException 에서 생성된 Exception 은 BizException 으로 서비스에서 문제가 발생되어 예외처리했음을 의미하기 때문에 향후 오류의 추적이 용이하다

```
public void sampleMethod() throws Exception {
    boolean isSuccess = sampleDAO.insertSample(obj);
    if(!isSuccess) {
        throw super.processException("MSG_APVL_ERR_NO_EXIST_DOC_DEPT");
    }
```

```
}
```

2.8 JSP (Java Servlet Page)

View Resolver 를 통해 forward 되며 비즈니스 로직 처리 후 결과를 화면에 렌더링 하기 위한 Java Servlet Page 이다. 태그로 작성하기 때문에 코드의 가독성이 좋아 개발 및 유지보수가 용이하다

WebContent/WEB-INF/jsp 하위에 업무별로 저장되어 관리되며, JSP 구조는 아래와 같다

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="enrise" uri="http://www.esnt.co.kr/enrise-common"%>
<%@ taglib prefix="tiles" uri="http://tiles.apache.org/tags-tiles" %>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<c:set var="pageTitle" value="타이틀" scope="request" />
<jsp:include page="/WEB-INF/jsp/include/htmlHeader.jsp" flush="true" />
<script>
<!--
$(document).ready(function(){
      //onload시 처리
});
//-->
</script>
</head>
<body style="overflow:hidden">
.. HTML Custom Tag
</body>
</html>
```

1) 태그 라이브러리

<% %> Java 스크립틀릿로 개발하는 경우. 프로그램의 가독성이 떨어지며, 프로그램 소스량이 증가되어 유지보수가 어렵다. 태그 형태로 제어 및 공통처리, 유틸리티, UI 등을 쉽게 개발하기 위해 사용하는 것이 태그 라이브러리이며, enRiseFramework 에서는 jstl core, spring, enrise-common, tiles, form 등의 태그 라이브러리를 사용한다

2) htmlHeader.jsp

모든 JSP 에서 공통적으로 사용되는 CSS, Javascript, HTML Header 를 포함하는 JSP 반드시 지정해야 한다

3) javascript / HTML /CSS Framework 및 Plugin

- KRRI_GW_S06_enRise UI 개발 가이드_v1.0 를 참고한다