## Smart Green Platform 개발

# 엘리먼트별 개발가이드

## 문서번호 입력

**Ver. 1.0**

**관리부서 : 통합팀**

# 개 정 이 력

| 버전 | 작성일 | 변경내용[1] | 작성자 | 승인자 |
|---|---|---|---|---|
| 1.0.0 | 2013-12-02 | 기존 엘리먼트별 개발가이드 v1.0.0 에서 이관 및 플랫폼 개발 기준으로 수정 | 최윤종 | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

[1] 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장/절과 변경 내용을 기술한다.)

# 목 차

# 1. 개요

개발 뷰(Development View) 상에 존재하는 각 엘리먼트에 대한 개발방법을 가이드 한다. 엘리먼트 내역은 다음과 같다.

- View
- Controller
- Service/ServiceImpl
- Dao/DaoImpl
- SQL Map
- Action
- System Interface

본 문서에 제시된 내용 외에도 외부 문서 및 별도로 제공되는 문서를 참고하도록 한다. 참고할 문서는 다음과 같다.

- Java SE
  -http://www.oracle.com/technetwork/java/javase/documentation/index.html
- Java EE
  -http://www.oracle.com/technetwork/java/javaee/documentation/tutorials-137605.html
- Spring Framework
  -http://www.springsource.org/documentation
- iBATIS Framework
  -참고자료 'iBATIS-SqlMaps-2_en.pdf' 및 'iBATIS-SqlMaps-2-Tutorial_en.pdf'

# 2. View

## 2.1 역할

View 는 html 페이지와 같은 클라이언트에게 전송할 HTTP Response 데이터를 생성하는 역할을 한다.

- 사용자에게 시스템에 대한 Front-End Interface 의 역할을 수행한다.

- 사용자로부터 입력을 받아 데이터를 수집한다.
- 입력값 유효성 검증(Input Validation)을 수행한다.
- 사용자가 요청한 정보 또는 의사결정을 위한 정보를 제공한다.
- 다른 페이지로의 링크를 제공하여 화면의 네비게이션을 변화시킨다.
- 사용자가 입력한 값을 HTTP method 를 사용하여 서버단에 전달한다.

## 2.2 구성요소

| 구성요소 | 설명 | 비고 |
|---|---|---|
| jsp scriptlet | jsp 상에서 작성한 java 코드를 의미한다. 소스의 가독성과 유지보수성을 저해하므로 사용을 권장하지 않으며, 가급적 Tag Library 를 활용한다. | <%<br>// java code<br>.....<br>%> |
| tag library | jsp 에서 필요한 기능들을 태그 형태로 사용할 수 있게 한다. JSP Standard Tag Library 와 Spring Tag Library 등을 사용할 수 있다. | JSTL:<br><%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%><br>Spring Tag Library:<br><form:form>...</form:form> |
| html | 클라이언트 환경에 출력될 데이터의 구조를 정의한다. | <html>.....</html> |
| css | 클라이언트 환경에 출력될 데이터의 표현을 정의한다. | <link rel="stylesheet" type="text/css" href="/css/common.css" /><br><style type="text/css">.....</style> |
| javascript | 클라이언트 환경과 상호작용 하기 위한 동작을 정의한다. | <script type="text/javascript"></script> |

## 2.3 작성방법

SGP 의 기본적인 View 는 jsp 이므로 jsp 코딩 표준을 준수하여 작성하도록 한다.

- 모든 jsp 에서 페이지 인코딩을 반드시 선언한다.
  (<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>)

- <body> 태그 내의 컨텐츠는 다음과 같이 작성한다.
  -scriptlet 사용은 지양하고 tag library 위주로 작성한다.
  -tag library 태그 속성값은 single quotation(')으로 묶는다.
  -html 태그 속성값의 표현은 double quotation(")으로 묶는다.
  -링크와 같이 url 정보를 출력할 때는 tag library 의 <c:url> 태그를 사용한다.
  -데이터 건수에 따라 반복문 형태로 출력할 때는 <c:forEach> 태그를 사용한다.
  -사용자 입력양식(form) 작성시 Spring 에서 제공하는 <form:form> 태그를 사용한다.
  -기타 태그 요소 작성은 웹 표준 및 접근성 가이드에 따른다.
- html 문서의 구조는 응용공통 가이드를 참조하여 작성해야 한다.

## 2.4 예시

[목록조회]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ include file="/jspf/include/tagLib.jspf" %>
<script type="text/javascript" src="<%=jsPath %>/template/test.js"></script>
    <body>
        <h2>Template List</h2>
        <a href="<c:url value='/template/insert/form.sc'/> ">Add</a>
        <ul>
        <c:forEach items="${list}" var="templateModel">
            <li><a href="/template/get/${templateModel.id}.sc">${templateModel.name}</a></li>
        </c:forEach>
        </ul>
    </body>
```

[상세조회]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ include file="/jspf/include/tagLib.jspf" %>
<script type="text/javascript" src="<%=jsPath %>/template/test.js"></script>
    <body>
        <div>
            <a href="<c:url value='/template/list.sc'/> ">List</a>
            <a href="<c:url value='/template/insert/form.sc'/> ">Add</a>
            <a href="<c:url value='/template/update/${templateModel.id}/form.sc'/> ">Update</a>
        </div>
```

```
        <form:form action="/template/delete/${templateModel.id}.sc" method="delete">
            <input type="submit" value="DELETE" />
        </form:form>
        <h2>Template Model Information</h2>
        <ul>
            <li>${templateModel.id}</li>
            <li>${templateModel.name}</li>
        </ul>
    </body>
```

[입력]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ include file="/jspf/include/tagLib.jspf" %>
<script type="text/javascript" src="<%=jsPath %>/template/test.js"></script>
    <body>
        <a href="<c:url value='/template/list.sc'/> ">List</a>
        <form:form commandName="templateModel" action="/template/insert/add.sc"
method="post">
            <p>
                ID :
                <form:label path="id"></form:label>
                <form:input path="id" size="50" />
                <form:errors path="id" cssClass="smdis-error-message"/>
            </p>
            <p>
                Name :
                <form:label path="name"></form:label>
                <form:input path="name" size="50" />
                <form:errors path="name" cssClass="smdis-error-message" />
            </p>
            <input type="submit" value="Add" />
        </form:form>
    </body>
```

[수정]

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ include file="/jspf/include/tagLib.jspf" %>
<script type="text/javascript" src="<%=jsPath %>/template/test.js"></script>
    <body>
```

```
        <a href="<c:url value='/template/list.sc'/> ">List</a>
        <form:form commandName="templateModel"
action="/template/update/${templateModel.id}/update.sc"    method="put">
            <p>
                Name :
                <form:label path="name"></form:label>
                <form:input path="name" size="50" />
                <form:errors path="name" cssClass="smdis-error-message" />
            </p>
                <input type="submit" value="save" />
        </form:form>
    </body>
```

## 3. Controller

### 3.1 역할

각 메소드 단위로 특정 HTTP URL 과 매핑되어 클라이언트의 HTTP 요청을 처리하는 어플리케이션의 진입점으로서 파라미터를 준비하여 Service 를 호출하고 결과 데이터를 View 로 전달하는 역할을 한다.

### 3.2 구성요소

일반적인 java 클래스의 구성과 동일하다.

### 3.3 작성방법

- Spring Framework 의 어노테이션 기반 Controller 작성방법을 따른다.
- @RequestMapping 에 URI 를 지정하고, return 값으로 View(jsp)의 경로를 지정한다.
- 전달받을 값은 GET 방식일 경우 @RequestParam 어노테이션, POST 방식일 경우 @ModelAttribute 어노테이션을 사용한다.

- 결과를 표시할 View 로 전달되는 값은 Model 을 사용한다.

### 3.4 예시

```
@Controller
@SessionAttributes("templateModel")
public class TemplateController {

    @Autowired TemplateService templateService;

    @RequestMapping("template/list.do")
    public String list(Model model) {
        model.addAttribute("list", templateService.list());
        return "template/list";
    }

    @RequestMapping(value="template/get.do")
    public String view(@RequestParam String id, Model model){
        model.addAttribute("templateModel", templateService.get(id));
        return "template/view";
    }

    @RequestMapping("template/insert/form.do")
    public String form(Model model){
        model.addAttribute("templateModel", new TemplateModel());
        return "template/form";
    }

    @RequestMapping(value="template/insert/add.do")
    public String formSubmit(@ModelAttribute("templateModel") TemplateModel templateModel,
BindingResult result, SessionStatus status){
        if(result.hasErrors()) {
            return "template/insert/form";
        }
        templateService.add(templateModel);
        status.setComplete();
        return "redirect:/template/list.do";
    }
```

```
    @RequestMapping(value="template/update/form.do")
    public String updateForm(@RequestParam String id, Model model){
        model.addAttribute("templateModel", templateService.get(id));
        return "template/update";
    }

    @RequestMapping(value="template/update/update.do")
    public String updateSubmit(@ModelAttribute("templateModel") TemplateModel templateModel,
BindingResult result, SessionStatus status){
        if(result.hasErrors()) {
            return "template/update";
        }
        templateService.update(templateModel);
        status.setComplete();
        return "redirect:/template/list.do";
    }

    @RequestMapping(value="template/delete.do")
    public String delete(@RequestParam String id) {
        templateService.delete(id);
        return "redirect:/template/list.do";
    }

}
```

## 4. Service/ServiceImpl

### 4.1 역할

타 엘리먼트로부터 전달받은 파라미터를 토대로 업무에 필요한 비즈니스 로직을 처리하는 역할을 한다. DBIO 필요시 Dao 를 호출하고 트랜잭션 처리를 담당한다.

## 4.2 구성요소

| 구성요소 | 설명 | 비고 |
|---|---|---|
| PBI | Process Bean Interface<br>(비즈니스 서비스 Interface) | @Transactional 어노테이션 사용 |
| PBC | Process Bean Component<br>(비즈니스 서비스 Interface 구현체) | @Service 어노테이션 사용 |
| CBI | Common Bean Interface<br>(공통 서비스 Interface) | @Transactional 어노테이션 사용 |
| CBC | Common Bean Component<br>(공통 서비스 Interface 구현체) | @Service 어노테이션 사용 |

## 4.3 작성방법

- PBI/CBI 는 java 의 interface 로서 이름에 Service 라는 postfix 를 붙인다.
- PBI/CBI 는 트랜잭션 필요시 @Transactional 어노테이션을 선언한다.
  -기본적으로 클래스 레벨에 설정한다.
  -특정 메소드에서 설정이 다를 경우, 해당 메소드에서 어노테이션을 재설정 한다.
- PBC/CBC 는 PBI/CBI 의 구현체로서 이름에 ServiceImpl 이라는 postfix 를 붙인다.
- PBC/CBC 는 @Service 어노테이션을 선언한다.

## 4.4 예시 - PBI, CBI

```
@Transactional
public interface TemplateService {

    @Transactional(readOnly=true)
    public List<TemplateModel> list();

    @Transactional(readOnly=true)
    public TemplateModel get(String id);

    void add(TemplateModel templateModel);
```

```
    void update(TemplateModel templateModel);

    void delete(String id);

    void updateSome(String[] idList, String newName);

    void push(String parameter) throws Exception;

    void writeObject(String smartConnectId, String scdId, String controlPointId, String cpValue, String
transactionId, String responseActionId)
        throws SGPCoreException, SGPDeviceException;

    void createScd(String smartConnectId, Device scd)
        throws SGPCoreException, SGPDeviceException;
}
```

## 4.5 예시 - PBC, CBC

```
@Service
public class TemplateServiceImpl implements TemplateService {

    @Autowired TemplateDao templateDao;
    @Autowired SGPDeviceController deviceController;

    public void add(TemplateModel templateModel) {
        templateDao.addTemplate(templateModel);
    }

    public void delete(String id) {
        templateDao.deleteTemplate(id);
    }

    public TemplateModel get(String id) {
        return templateDao.getTemplate(id);
    }

    public List<TemplateModel> list() {
```

```
        return templateDao.listTemplate();
    }

    public void update(TemplateModel templateModel) {
        templateDao.updateTemplate(templateModel);
    }

    public void updateSome(String[] idList, String newName) {
        for (String id : idList) {
            TemplateModel modelGet = this.get(id);
            modelGet.setName(newName);
            this.update(modelGet);
        }
    }

    public void writeObject(String smartConnectId, String scdId, String controlPointId, String cpValue,
String transactionId, String responseActionId){
        deviceController.writeObject(smartConnectId, scdId, controlPointId, cpValue, transactionId,
        responseActionId);
    }

    public void createScd(String smartConnectId, Device scd) {
        deviceController.createScd(smartConnectId, scd);
    }

}
```

## 5. Dao/DaoImpl

### 5.1 역할

PBC/CBC 에 의해 호출되어 비즈니스에 필요한 DBIO 를 처리한다.

## 5.2 구성요소

| 구성요소 | 설명 | 비고 |
|---|---|---|
| EBI | Entity Bean Interface<br>(DBIO 처리 Interface) | |
| EBC | Entity Bean Component<br>(DBIO 처리 Interface 구현체) | AbstractSGPDao 상속<br>@Service 어노테이션 사용 |

## 5.3 작성방법

- EBI 는 java 의 interface 로서 이름에 Dao 라는 postfix 를 붙인다.
- EBC 는 EBI 의 구현체로서 이름에 DaoImpl 이라는 postfix 를 붙인다.
- EBC 는 @Service 어노테이션을 선언한다.
  -일반적인 Spring 기반 개발에서는 DaoImpl 에 @Repository 어노테이션을 사용하지만, SGP 에서는 @Service 를 사용한다.
- EBC 는 com.lgcns.sgp.framework.dao.AbstractSGPDao 를 상속받아 DBIO 처리를 위해 AbstractSGPDao 의 메소드를 호출하며 별도의 예외나 트랜잭션에 대한 처리는 수행하지 않는다.
- AbstractSGPDao 는 SQL 실행을 위한 다양한 API 를 제공한다.
  -insert/update/delete/select 구문별 메소드가 존재한다.
  -select 의 경우, 조회 건수에 따라 single/multi/paging 구문별 메소드가 존재한다.
  -insert/update/delete 의 경우, List 타입의 파라미터를 가지고 배치 실행하는 list 메소드가 존재한다.

## 5.4 예시 - EBI

```
public interface TemplateDao {

    List<TemplateModel> listTemplate();

    public TemplateModel getTemplate(String id);

    void addTemplate(TemplateModel templateModel);
```

```
    void updateTemplate(TemplateModel templateModel);

    void deleteTemplate(String id);

    void deleteTemplateAll();

    int getCount();

    List<TemplateModel> getPagingTemplate(SGPPagingModel model);

}
```

## 5.5 예시 - EBC

```
@Service
public class TemplateDaoImpl extends AbstractSGPDao implements TemplateDao {

    @SuppressWarnings("unchecked")
    public List<TemplateModel> listTemplate() {
        return selectMulti("template.list");
    }

    public TemplateModel getTemplate(String id) {
        return (TemplateModel) selectSingle("template.get", id);
    }

    public void addTemplate(TemplateModel templateModel) {
        insert("template.add", templateModel);
    }

    public void updateTemplate(TemplateModel templateModel) {
        update("template.update", templateModel);
    }

    public void deleteTemplate(String id) {
        delete("template.delete", id);
    }
```

```
    public void deleteTemplateAll() {
        delete("template.deleteAll");
    }

    public int getCount() {
        int result =  ((Integer) selectSingle("template.getCount")).intValue();
        return result;
    }

    @SuppressWarnings("unchecked")
    public List<TemplateModel> getPagingTemplate(SGPPagingModel model) {
        return selectPaging("template.list", model);
    }

}
```

# 6. SQL Map

## 6.1 역할

SQL Map 은 SGP Framework 에서 참조하여 수행할 SQL 정보를 담고 있다. 세부 항목으로는 query, type alias, parameter map, result map 등의 정보를 포함한다.

## 6.2 구성요소

| 구성요소 | 설명 | 비고 |
|---|---|---|
| SQL Map Config 파일 | iBATIS Framework 에 관련한 기본설정, SQL Map 파일 목록 정보를 담고 있다. | 전체 1 본 (META-INF/sqlmap_{DBMS 명}/sql-map-config.xml |
| SQL Map 파일 | DB 에 대해 수행할 query, type alias, parameter map, result map 정보를 담고 | 업무별 1 본 (META-INF/sqlmap_{DBMS 명} |

| 구성요소 | 설명 | 비고 |
|---|---|---|
| | 있다. | 폴더에 업무별로 하위 폴더를 만들어서 작성) |

## 6.3 작성방법

SQL Map Config 파일은 공통으로 사용되므로 미리 작성하여 제공된다. 또한, 업무별로 작성하는 SQL Map 파일은 자동으로 로딩되도록 설정되어 있으므로 개발시 특별한 예외상황이 아니라면 SQL Map Config 파일을 수정할 필요가 없다.

SQL Map 파일은 개발표준을 준수하도록 하며 다음과 같이 작성한다.

- 최상위 태그인 <sqlMap> 태그에 반드시 namespace 를 선언한다.
  -namesapce 는 SQL Map 파일을 고유하게 구분하기 위한 명칭으로 Dao 에서 특정 query 를 지정할 때, '[namespace].[query id]'와 같은 형태로 사용된다.
- typeAlias, parameterMap, resultMap 태그는 query 보다 위쪽에 순서대로 작성한다.
  -가장 위에 typeAlias 정보를 일괄적으로 선언하고, 그 뒤로 parameterMap, resultMap 순으로 선언한다.
- query 정보는 SQL 문장에 따라 select, insert, update, delete 등의 태그를 사용하여 선언한다.
  - query 내에 xml 금칙문자(<, >, &, ", ')가 있을 경우, xml 파싱오류가 발생하므로 반드시 CDATA 태그(<![CDATA[와 ]]>)로 감싸주도록 한다.
  - query 태그의 id 속성값은 해당 query 를 참조하는 메소드명과 동일하게 사용하고, 여러 메소드에서 동일 query 를 참조하는 경우에는 해당 query 의 기능을 쉽게 표현하는 '동사+명사' 형태의 명칭으로 id 를 선언한다.
- 같은 파일 내의 다른 query 에서 참조하기 위한 query 조각(sql 태그)을 작성할 경우, 반드시 다른 query 보다 먼저 작성되어야 한다.
  -파일 내에서 선언 순서가 뒤바뀌면 메모리 로딩 및 파싱할 때 오류가 발생할 수 있다.
  -다른 파일의 query 를 참조할 경우 반드시 참조대상 query 조각이 먼저 로딩되도록 해야 한다.
- SQL 구문의 작성은 DBA 의 SQL 가이드에 따르도록 한다.

SGP Framework 의 SQL Map 은 Spring Framework 와 연동된 iBATIS Framework 를 사용하므로 SQL Map 작성과 관련한 세부 문법은 iBATIS 매뉴얼을 참조하도록 한다.

## 6.4 예시

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

<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN" "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="template">

  <typeAlias alias="TemplateModel" type="com.lgcns.sgp.template.model.TemplateModel" />

  <parameterMap id="empParam" class="com.lgcns.sgp.test.empVO">
    <parameter property="empNo" javaType="decimal" jdbcType="NUMERIC" />
    <parameter property="empName" javaType="string" jdbcType="VARCHAR" nullValue="blank" />
    <parameter property="job" javaType="string" jdbcType="VARCHAR" nullValue="" />
    <parameter property="mgr" javaType="decimal" jdbcType="NUMERIC" />
    <parameter property="hireDate" javaType="date" jdbcType="DATE" />
    <parameter property="sal" javaType="decimal" jdbcType="NUMERIC" />
    <parameter property="comm" javaType="decimal" jdbcType="NUMERIC" nullValue="-99999" />
    <parameter property="deptNo" javaType="decimal" jdbcType="NUMERIC" />
  </parameterMap>

  <resultMap id="deptResult" class="com.lgcns.sgp.test.deptVO">
    <result property="deptNo" column="DEPT_NO" />
    <result property="deptName" column="DEPT_NAME" />
    <result property="loc" column="LOC" />
  </resultMap>

  <select id="listTemplates" resultClass="TemplateModel">
  <![CDATA[
    select * from sgp_template order by id
  ]]>
  </select>

  <select id="getTemplate" parameterClass="java.lang.String" resultClass="TemplateModel">
  <![CDATA[
    select * from sgp_template where id = #id#
  ]]>
  </select>

  <insert id="addTemplate" parameterClass="TemplateModel">
```

```
  <![CDATA[
    insert into sgp_template (id, name) values(#id#, #name#)
  ]]>
  </insert>

  <update id="updateTemplate" parameterClass="TemplateModel">
  <![CDATA[
    update sgp_template set name = #name# where id = #id#
  ]]>
  </update>

  <delete id="deleteTemplate" parameterClass="java.lang.String">
  <![CDATA[
    delete from sgp_template where id = #id#
  ]]>
  </delete>

  <delete id="deleteAllTemplates">
  <![CDATA[
    delete from sgp_template
  ]]>
  </delete>

  <select id="getTemplateCount" resultClass="java.lang.Integer">
  <![CDATA[
    select count(*) from sgp_template
  ]]>
  </select>

</sqlMap>
```

# 7. Action

## 7.1 역할

SGP Framework 에서 EDA 가 이벤트라고 판단한 경우 다음으로 실행해야 하는 대상 비즈니스로서 EDA 의 Input Data 인 ActionParam 을 이용해 PBI 가 필요한 Input Data 형태로 변환한 뒤 PBI 를 호출하는 역할을 수행한다.

## 7.2 구성요소

N/A

## 7.3 작성방법

- com.lgcns.sgp.framework.service.action.AbstractSGPAction 클래스를 상속받아 execute() 메소드를 구현한다.
- @SGPAction 어노테이션을 클래스 상단에 선언한다.
  -어노테이션에 지정할 Action 명은 개발표준의 명명규칙을 준수한다.
- Action 은 action description 파일에 Action 명과 함께 설명을 등록한다. 설명은 한글과 영문으로 각각의 파일에 등록한다.
  -한글 파일: resources/META-INF/action/sgp-action-description_ko.properties
  -영문 파일: resources/META-INF/action/sgp-action-description_en.properties
- 디바이스로부터 올라온 정보와 이벤트 처리시 가공된 정보가 ActionParam 에 저장되어 execute() 메소드의 인자로 전달되므로 PBI 호출에 필요한 데이터를 생성한 후, 호출하도록 한다.

## 7.4 예시

```
@SGPAction ("xxx.aaa.bbb.templateAddAction")
public class TemplateAddAction extends AbstractSGPAction {

    @Autowired TemplateService templateService;

    @Override
    public Object execute(ActionParam actionParam) {
        // 필요할 경우 Smart Connect 로부터 올라온 디바이스 정보를 Get
        String scdId = actionParam.getScdId();
        List<CovMessage> covList = actionParam.getCovList();

        // 필요할 경우 Event Pro 가 이벤트 처리 중 추가/가공 된 정보를 Get
```

```
        int eventProcessType = actionParam.getEventProcessType();
        EventMessage eventResult = actionParam.getEventMessage();
        String eventYN = eventResult.getEventYN();
        String sgpEventId = eventResult.getSgpEventId();
        String etc = eventResult.getEtc();

        // 필요할 경우 Service 실행에 필요한 Model Set
        TemplateModel templateModel = new TemplateModel();
        templateModel.setId(scdId);

        templateService.add(templateModel);

        // 특별히 리턴할 값이 없는 경우 actionParam 그대로 리턴한다.
        return actionParam;
    }

}
```