IV. SKIAF Front-End 개발 가이드

Index of Contents

IV. SKIAF Front-End 개발 가이드

IV.1 개요

[목적]

본 문서는 프로젝트에서 SKIAF Framework를 활용하여 Front-End어플리케이션을 개발하는 절차 및 방법을 기술한다. 웹 표준 준수를 목표로 한다. 본 프로젝트의 개발 절차 및 방법을 가이드 함으로써, 개발의 생산성과 효율성을 향상하고, 개발된 결과의 통일성을 유지하여 향후 유지보수의 편의성, 높은 가독성을 제공하기 위함이다.

[문서의 범위]

본 문서에서는 다음의 사항을 기술한다.

- Framework 기반 기본 아키텍처
- HTML 표준
- Thymeleaf
- Script 작성원칙

본 문서는 기본 Javascript 프로그래밍 관련 내용은 다루지 않는다(개발자가 Javascript 언어에 대한 기본적인 지식과 내용은 이미 갖추었다고 가정)

[대상]

본문서는 Front-End어플리케이션을 개발하는 개발자를 대상으로 한다.

IV.2 Front-End 개발

IV.2.1 SKIAF Front-End 구조

[디렉토리 구조]

SKIAF 의 Front-end 소스는 skiaf-template 프로젝트의 src/main/resources/ 하위에 존재하며, 간략하게 보면 다음과 같은 구조를 갖는다.

```
src/main/resources
 +- static
 |   +- dtd
 |   +- skiaf
 |       +- alopex
 |       |   +- grid
 |       |   +- grid-plugin-search
 |       |   +- ui
```

```
|       +- crypto
|       +- css
|       |   +- common
|       |       +- skiaf-common.css
|       |       +- skiaf-layout.css
|       +- favicon
|       +- images
|       +- script
|       |   +- common
|       |   |   +- common-const.js
|       |   |   +- common-element.js
|       |   |   +- common-program.js
|       |   |   +- common-ui.js
|       |   |   +- common-util.js
|       |   |   +- common.js
|       |   +- login
|       |   |   +- login-form.js
|       |   +- jquery
|       |   |   +- jquery.min.js
|       |   +- index.js
+- templates
    +- skiaf
        +- fragments
        |   +- body.html
        |   +- config.html
        |   +- footer.html
        |   +- header.html
        +- layout
        |   +- default-layout.html
        |   +- error-layout.html
        |   +- login-layout.html
        +- view
            +- login
            |   +- login-form.html
            +- index.html
```

중요 디렉토리와 파일에 대한 설명은 아래 표를 참고하면 된다.

- static/skiaf/alopex 하위의 폴더와 파일들은 Alopex에서 다운로드 받은 파일을 그대로 오버라이드 해서 사용하기 때문에, 수정하면 안된다.
- templates/skiaf/view 하위의 컨텐츠 html에서 쓰이는 js 파일들은 그 경로와 동일한 위치로 사용중이다. (login-form.js 와 login-form.html 참고)
- 실제 SKIAF 를 사용해서 프로젝트를 진행할 때, SKIAF 의 파일과 구분되도록 static/skiaf , templates/skiaf 와 동일하게 폴더를 따로 구성해서 진행하는 것을 권고한다.

디렉토리					설명
static					정적인 css, js, html 파일들을 포함하고있다.
	dtd				mybatis dtd 파일을 포함하고있다.
	skiaf				
		alopex			Alopex 관련 파일을 포함하고있다.
			grid		Alopex Grid 관련 파일을 포함하고있다.
			grid-plugin-search		Alopex Grid Plugin 중, Search Plugin 관련 파일을 포함하고있다. 추가로 사용할 플러그인이 있을 경우, 이처럼 alopex 폴더 하위에 추가하면 된다.
			ui		Alopex UI 관련 파일을 포함하고있다.
		crypto			keystore 파일을 포함하고있다.
		css			css 파일을 포함하고있다.
			common		
				skiaf-common.css	
				skiaf-layout.css	
		favicon			
		images			image 파일을 포함하고있다.
		script			js 파일을 포함하고있다.
			common		공통으로 사용하고 있는 js 파일들을 포함하고있다.
				common-const.js	공통으로 사용하는 상수 오브젝트가 정의되어있다.
				common-element.js	프로그램 요소를 권한에 따라 공통으로 처리한다.
				common-program.js	프로그램 도움말(매뉴얼)에 대해 공통으로 처리한다.
				common-ui.js	ui 관련 함수가 정의되어있다.
				common-util.js	공통으로 사용하는 유틸리티 함수들을 포함하고있다.
				common.js	Alopex, AlopexGrid setup 관련 설정을 한다.
			login		
				login-form.js	login/login-form.html 에서 사용하는 js 파일이다.
			jquery		
				jquery.min.js	

			index.js		
	{myProject}				진행중인 프로젝트의 static 파일들을 이런식으로 구성하는 것을 권고한다.
		css			
		images			
		script			
templates					Thymeleaf 에서 사용할 template html 파일들을 포함하고있다.
	skiaf				
		fragments			Thymeleaf 의 layout 을 구성하는 fragments html 파일들을 포함하고있다.
			body.html		default-layout.html 레이아웃의 body 태그를 구성하는 fragments html 이다.
			common-js.html		body.html에 include 되어있는 html 이다. 업무관련 공통 업무를 담당하는 js 파일들을 선언한다.
			config.html		head 태그를 구성하는 fragments html 이다. meta정보, 공통으로 사용하는 css, js 파일들을 선언한다.
			data.html		config.html에 include 되어있는 html 이다. 서버데이타를 javascript 전역 객체로 바인딩해주는 로직이 포함되어있다.
			error.html		error-layout.html 레이아웃의 body 태그를 구성하는 fragments html 이다.
			footer.html		footer 태그를 구성하는 fragments html 이다.
			header.html		header 태그를 구성하는 fragments html 이다. gnb 메뉴를 포함하고있다.
			lnb.html		lnb 메뉴를 구성하는 html 이다.
			login.html		login-layout.html 레이아웃의 body 태그를 구성하는 fragments html 이다.
			popup-data.html		window.SKIAF.programPopup 을 생성하고 팝업 도움말을 출력한다.

		layout			Thymeleaf 의 layout decorator html 파일들을 포함하고으며, 디자인별로 레이아웃 파일을 분리해서 사용한다.
			default-layout.html		에러페이지와 로그인페이지를 제외한 모든 컨텐츠 페이지에서 사용하는 layout decorator html 이다.
			error-layout.html		에러페이지를 구성하는 layout decorator html 이다.
			login-layout.html		로그인페이지를 구성하는 layout decorator html 이다.
		view			controller return 값의 경로인 실제 컨텐츠 페이지 html 파일들을 포함하고 있다.
			login		
				login-form.html	
			index.html		
	{myProject}				진행중인 프로젝트의 template 파일들을 이런식으로 구성하는 것을 권고한다.
		fragments			
		layout			
		view			

[window.SKIAF 파일 명명규칙]

SKIAF 에서 권고하는 static 파일들의 명명규칙이다.

파일 형태	확장자	명명규칙	예제
자바 스크립트	.js	소문자. 명칭이 길어질 경우 dash(-)로 구분	alopex-ui.min.js common-ui.js
스타일시트	.css	소문자. 명칭이 길어질 경우 dash(-)로 구분	alopex-ui-default.css skiaf-common.css
HTML	.html	소문자. 명칭이 길어질 경우 dash(-)로 구분	default-layout.html index.html

[window.SKIAF 오브젝트]

window.SKIAF는 common-*.js 파일 내에 정의되어 있으며, 공통업무 관련 로직을 수행하는 script 와, 업무 처리용 페이지에서 필요로 하는 기능들을 window 객체로 선언하여 사용된다.

팝업 등 및 각 페이지 처리에서 필요한 기능들에 대하여 window.SKIAF 내에 선언되어 있는 아래 내용을 참고하여 호출해서 사용하도록 한다.

객체명	설명	예시
window.SKIAF.CONSTATNT	• 공통 상수 오브젝트이다. • CommonConstant.java와 동일하며 서버와 프론트에서 사용하는 공통 상수 값들이 정의 되어있다.	SKIAF.CONSTATNT.TIME_PATTERN
window.SKIAF.ENUM	• 서버에서 enum으로 관리하는 객체가 정의되어있다.	SKIAF.ENUM.MENU_TYPE
window.SKIAF.JS_CONSTANT	• javascript에서 사용하는 상수 오브젝트이다. • common-const.js 에 정의 되어있다.	SKIAF.JS_CONSTANT.SYSTEM_MESSAGE
window.SKIAF.PATH	• SKIAF 의 모든 request 요청시의 path 가 정의되어있다. (view, api) • Path.java 와 동일하며 서버와 프론트의 path를 일치시키기 위해 사용한다.	SKIAF.PATH.VIEW_BOARDS
window.SKIAF.bcmHomeUrl	• application.properties 에 정의되어있는 bcm.home.url 값을 출력한다. • 로그인 후 이동하는 home 페이지 url	SKIAF.bcmHomeUrl
window.SKIAF.bcmUIHomeUrl	• application.properties 에 정의되어있는 bcm.ui.home.url 값을 출력한다. • UI 가이드 home 페이지 url	SKIAF.bcmUIHomeUrl
window.SKIAF.console	• console.log 대신 사용하는 SKIAF 의 console 메세지 출력 함수이다. • trace, debug, info, warn, error 가 정의되어있다.	SKIAF.console.info('loading...');
window.SKIAF.elementRole	• 호출하는 페이지에 설정되어있는 프로그램 요소 권한 목록을 출력한다. • 프로그램 요소 권한 목록이 있을 경우, 해당 권한이 있는 사용자에게만 프로그램 요소가 노출되거나 작동된다.	SKIAF.elementRole
window.SKIAF.excelUtil	• 공통으로 사용하는 엑셀다운로드 관련 함수가 정의되어있다. • AlopexGrid 의 excel 다운로드 기능을 활용한다.	SKIAF.excelUtil.excelExportUrl(excelParm);

window.SKIAF.i18n	• SKIAF 의 다국어 관련 오브젝트가 정의되어있다.
	  • langCurrentCode : 현재 사용중인 언어 코드
	  • langCurrentCodeName : 현재 사용중인 언어명
	  • langDefaultCode : 기본 사용 언어 코드
	  • langSupportedCodes : 다국어 지원하는 언어 코드 목록 (최대 3개)
	  • messages : 다국어 처리된 메세지
	• messages
	  • 서버 Controller 에서 ModelAndView 를 return 할 때 addObject 의 key로 MessageComponent.JS_MESSAGES_KEY_PATTERN 을 사용한 경우, value 값으로 필터링 된 메세지를 갖고 있다.
	  • bcm.common.* 의 패턴을 갖는 메세지의 경우, Controller에서 필터링 하지 않더라도 사용할 수 있도록, ControllerAspect.java 에서 기본적으로 추가하고있다.
		SKIAF.i18n.langCurrentCode

window.SKIAF.loginUserInfo	• 로그인한 사용자 관련 정보가 포함되어있다.
	  • roleList : 사용자가 갖고 있는 권한 목록
	  • user : 사용자 정보
	  • userGroupList : 사용자가 포함 된 사용자그룹 목록
		SKIAF.loginUserInfo.user.userName

window.SKIAF.loginUtil	• 로그인 시에 사용하는 함수가 정의되어있다.	SKIAF.loginUtil.loginSuccess(SKIAF.returnUrlAfterLogin);
window.SKIAF.returnUrlAfterLogin	• 로그인 후 이동할 페이지 url	SKIAF.returnUrlAfterLogin
window.SKIAF.passwordUtil	• SKI 보안가이드에서 정의하는 패스워드 조합 정책에 따라 패스워드를 체크하는 스크립트 함수가 정의 되어있다. • 서버 로직(UserServiceImpl.java)과 일치한다.	SKIAF.passwordUtil.unionRule(password)
window.SKIAF.popup	• Alopex popup api를 활용한 공통 팝업 호출 함수가 정의 되어있다. • 팝업 종류는 alert, confirm, exception 이 있다. • 호출한 팝업에서 확인 버튼을 클릭했을 때 실행 되는 callback 함수를 매개 변수로 추가해서 사용할 수 있다.	SKIAF.popup.alert('이름을 입력해주세요.'); SKIAF.popup.alert('ID를 입력해주세요.', function(){ alert('확인 버튼을 누르셨습니다.'); });
window.SKIAF.program	• 프로그램 관리에서 등록한 도움말 첨부파일이 있을 경우, 도움말 첨부파일 ID 값이 들어있다. • 도움말이 등록되어 있을 경우, 해당 view 페이지에 메뉴얼 아이콘이 생성된다.	SKIAF.program
window.SKIAF.programPopup	• 프로그램 관리에서 등록한 도움말 첨부파일이 있고, 해당 view 페이지가 popup 일 경우에, 도움말 첨부파일 ID 값이 들어있다. • 도움말이 등록되어 있을 경우, 해당 view 페이지에 메뉴얼 아이콘이 생성된다.	SKIAF.programPopup
window.SKIAF.ui	• ui 관련 함수가 정의 되어있다. • common-ui.js 에 정의 되어있다.	SKIAF.ui.lnb();
window.SKIAF.util	• 업무관련 공통 유틸리티 함수가 정의되어있다. • 함수의 상세한 설명은 "V. SKIAF 개발자매뉴얼 - 업무공통기능 > V.15 업무공통 Utility" 을 참고	SKIAF.util.removeChar("abcdefg","e")
window.SKIAF.variable	• SKIAF 공통 상수 오브젝트의 변할 수 있는 prefix 명을 설정한다.	SKIAF.variable.preFix
window.SKIAF.webeditor	• Alopex 의 WebEditor plugin 사용시 필요한 함수가 정의 되어있다.	SKIAF.webeditor.setWebeditor('#WebeditorArticleCreatePopup');

IV.2.2 Thymeleaf

[Thymeleaf 란]

- Thymeleaf 는 XML/XHTML/HTML5 용 Java 템플릿 엔진이다.
- HTML 태그 및 속성 기반이기 때문에 HTML의 형식을 해치지 않으며, 웹과 비 웹(오프라인) 환경에서 모두 작동 가능하다.

- Spring Boot 에서 사용을 권장하고있는 템플릿 엔진이다. (참고로 jsp 는 spring boot에서 몇가지 제한 사항이 있어 지양하고 있다.)
- Spring Boot starter 로 제공되기 때문에 의존성 설정하기에 간편하다.

> 참고
> - Thymeleaf 개요 - https://www.thymeleaf.org/index.html
> - Spring Boot MVC template engines - https://docs.spring.io/spring-boot/docs/1.5.15.RELEASE/reference/html/boot-features-developing-web-applications.html#boot-features-spring-mvc-template-engines
> - Spring Boot jsp 제한 사항 - https://docs.spring.io/spring-boot/docs/1.5.15.RELEASE/reference/html/boot-features-developing-web-applications.html#boot-features-jsp-limitations

[Thymeleaf 설정]

maven dependency 추가

Spring Boot 에서 Thymeleaf 설정하는 방법은 간단하다. Thymeleaf 는 Spring Boot starter 로 제공되기 때문에, maven 의 경우 다음과 같이 pom.xml 에 spring-boot-starter-thymeleaf 만 추가해주면 된다.

의존성 설정, 라이브러리 버전 관리 등은 spring boot 에서 대신해주게된다.

pom.xml

```
<!-- Thymeleaf -->
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

ViewResolver 설정

Spring 이 Thymeleaf 를 사용해 View를 처리할 수 있도록 ViewResolver를 등록해준다.

SKIAF 에서는 Web MVC 관련 설정을 담당하는 WebMvcConfig.java 에서 Thymeleaf 설정을 하고 있으며, 다음과 같다.

- SpringResourceTemplateResolver 에서 템플릿으로 사용할 파일들의 prefix 경로와 확장자를 설정한다.
- 템플릿의 캐시 설정은, application.properties 에서 Thymeleaf 캐시 사용여부에 대한 값(bcm.thymeleaf.cache)을 관리 할 수 있도록 설정했다.
  (bcm.thymeleaf.cache 값이 false 일 경우, 서버 재시작 없이 새로고침만으로 반영이되기 때문에, 개발을 할 때는 false 로 설정하는 것이 편리하다)
- SpringTemplateEngine 에 addDialect(new LayoutDialect()); 하는 부분은 Thymeleaf의 layout dialect 기능을 사용하기 위한 설정이다.

WebMvcConfig.java (Thymeleaf 관련 설정 부분만)

```
package com.skiaf.core.config;

import java.nio.charset.StandardCharsets;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;
import org.thymeleaf.spring4.SpringTemplateEngine;
import org.thymeleaf.spring4.templateresolver.SpringResourceTemplateResolver;
import org.thymeleaf.spring4.view.ThymeleafViewResolver;

import nz.net.ultraq.thymeleaf.LayoutDialect;

/**
 * Web MVC 설정
 */
@Configuration
@EnableWebMvc
public class WebMvcConfig extends WebMvcConfigurerAdapter {

    /** Thymeleaf 캐시 설정 */
    @Value("${bcm.thymeleaf.cache}")
    private boolean isThymeleafCache;

    /**
     * View Resolver 설정 : Thymeleaf
     */
    @Bean
    public SpringResourceTemplateResolver templateResolver() {
        SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();
        templateResolver.setPrefix("classpath:/templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setCacheable(isThymeleafCache);
        return templateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine() {
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.setTemplateResolver(templateResolver());
        templateEngine.addDialect(new LayoutDialect());
        return templateEngine;
    }

    @Bean
    public ThymeleafViewResolver viewResolver() {
        ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
        viewResolver.setCharacterEncoding(StandardCharsets.UTF_8.name());
        viewResolver.setTemplateEngine(templateEngine());
        return viewResolver;
    }

}
```

[Thymeleaf MVC 예제]

Thymeleaf 를 사용해 MVC 를 구현한 간한단 예제이다. Controller 메소드에서 리턴된 값이 템플릿의 경로가 되며, 실제 템플릿에서는 model 에 넣은 값을 사용할 수 있다.

NoticeController.java

```
@Controller
public class NoticeController {

 @GetMapping(value = "/view/notice/list" )
 public String noticeList(Model model) {

  model.addAttribute("list", noticeService.getNoticeList());

  return "views/notice/list";
 }
}
```

templates/views/notice/list.html

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org">
<div>
 <table id="datatable">
  <thead>
   <tr>
    <th>번호</th>
    <th>제목</th>
    <th>작성자</th>
    <th>게시기간</th>
    <th>작성일</th>
   </tr>
  </thead>
  <tbody>
   <tr th:each="item : ${list}">
    <td class="tdSeq" th:text="${item.seq}"></td>
    <td th:text="${item.title}"></td>
    <td th:text="${item.author}"></td>
    <td>
     <span th:text="${item.postStrDt}"></span> ~ <span th:text="${item.postEndDt}"></span>
    </td>
    <td th:text="${item.regDt}"></td>
   </tr>
  </tbody>
 </table>
</div>
</html>
```

list.html 소스를 보면, html 태그에 Thymeleaf 를 선언한다. 그리고 선언된 th를 태그의 속성 처럼 사용해 필요한 데이터를 바인딩한다.

HTML 형식을 해치지 않기 때문에 WAS가 없는 환경에서도 HTML 파일을 열어서 확인할 수 있다.

[Thymeleaf 표현식]

SKIAF 에서 주로 사용한 Thymeleaf 표현식은 다음과 같다.

> 참고
>
> 더 자세한 가이드는 Thymeleaf 사용 가이드 https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html 를 참고

간단한 표현식 (https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#standard-expression-syntax)

변수 표현식 : ${...}

OGNL(Object-Graph Navigation Language) 표현식으로 해당 Context에 포함된 변수들을 사용할 수 있다.

```
<p>Today is: <span th:text="${today}">13 February 2011</span></p>
```

선택된 변수 표현식 : *{...}

전체 컨텍스트 변수 맵이 아닌 선택한 객체 안에서 변수를 찾는다. 선택한 객체(th:object)가 없다면 변수 표현 방식과 동일하다.

```
<div th:object="${session.user}">
 <p>Name: <span th:text="*{firstName}">Sebastian</span>.</p>
   <p>Surname: <span th:text="*{lastName}">Pepper</span>.</p>
   <p>Nationality: <span th:text="*{nationality}">Saturn</span>.</p>
</div>
```

메세지 표현식 : #{...}

Thymeleaf engine에 등록된 메세지를 나타내는 표현식이다.

SKIAF 는 message properties 파일과 DB에서 다국어 메세지를 관리하고 있으며, 이를 Thymeleaf engine에 등록해서 사용하고 있다. html 에서 다국어 메세지를 표현하는 방법으로 Thymeleaf 의 메세지 표현식을 사용한다.

아래 코드에서 사용중인 utext 와 text는 주의해서 사용할 필요가 있는데,

- th:utext 는 HTML 태그를 바인딩하고 싶을 때 사용한다.
- th:text 는 HTML 태그를 바인딩 하더라도 문자열로 자동 치환이 된다. 때문에 일반적인 텍스트 바인딩에서 사용한다.

```
<p th:utext="#{home.welcome(${session.user.name})}">
  Welcome to our grocery store, Sebastian Pepper!
</p>


<title><th:block th:text="#{bcm.menu.title}" /></title>
```

Link URL 표현식 : @{...}

URL을 표현할 때 사용한다. 서버의 context name을 자동으로 추가해준다.

```
<!-- Will produce 'http://localhost:8080/gtvg/order/details?orderId=3' (plus rewriting) -->
<a href="details.html"
  th:href="@{http://localhost:8080/gtvg/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/details?orderId=3' (plus rewriting) -->
<a href="details.html" th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/gtvg/order/3/details' (plus rewriting) -->
<a href="details.html" th:href="@{/order/{orderId}/details(orderId=${o.id})}">view</a>


<link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/css/common/skiaf-common.css}" />
```

조건문 (https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#conditional-evaluation)

단순 조건문 : "if"와 "unless"

조건에 따라서 해당 영역 랜더링을 설정 할 수 있다. th:class 의 경우 조건에 맞는 값이 리턴된다. th:if 와 th:unless 를 사용해서 조건에 맞는 부분만 랜더링 할 수 있다.

```
<!-- 조건에 따라 class 사용 -->
<tr th:class="${row.even}? 'even' : 'odd'">
  ...
</tr>


<!-- th:if -->
<table>
  <tr>
    <th>NAME</th>
    <th>PRICE</th>
    <th>IN STOCK</th>
    <th>COMMENTS</th>
  </tr>
  <tr th:each="prod : ${prods}" th:class="${prodStat.odd}? 'odd'">
    <td th:text="${prod.name}">Onions</td>
    <td th:text="${prod.price}">2.41</td>
    <td th:text="${prod.inStock}? #{true} : #{false}">yes</td>
    <td>
      <span th:text="${#lists.size(prod.comments)}">2</span> comment/s
      <a href="comments.html"
         th:href="@{/product/comments(prodId=${prod.id})}"
         th:if="${not #lists.isEmpty(prod.comments)}">view</a>

    </td>
  </tr>
</table>


<!-- th:unless -->
<a href="comments.html"
  th:href="@{/comments(prodId=${prod.id})}"
  th:unless="${#lists.isEmpty(prod.comments)}">view</a>
```

그외 조건문 : "switch"

조건에 따라서 해당 영역 랜더링을 설정 할 수 있다.

```
<div th:switch="${user.role}">
  <p th:case="'admin'">User is an administrator</p>
  <p th:case="#{roles.manager}">User is a manager</p>
</div>
```

[Thymeleaf Script Inlining]

Thymeleaf 는 javascript 와 Dart 에 대해 스크립트 inline 기능을 제공하고 있다.(https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#script-inlining-javascript-and-dart)

사용 방법은 script 태그에 th:inline="javascript" 속성을 정의하고, 해당 스크립트에서 [[..]] 표현식을 사용해 서버데이터를 스크립트 영역에 표현한다.

하지만 [[..]] 표현식은 정적인 페이지로 html을 열었을 때 문법 오류로 인식되기 때문에, 자바 스크립트 주석과 함께 다음과 같이 사용한다.

SKIAF 에서 서버데이터를 javascript 전역객체로 바인딩하는 data.html 의 예제이다.

```
<script th:inline="javascript">
 (function() {
  // 공통 데이터
  /*<![CDATA[*/
          window.SKIAF.i18n = /*[[${skiaf.i18n}]]*/ [];
          window.SKIAF.elementRole = /*[[${skiaf.elementRole}]]*/ [];
          window.SKIAF.program = /*[[${skiaf.program}]]*/ {};
          window.SKIAF.bcmHomeUrl = /*[[${skiaf.bcmHomeUrl}]]*/ {};
          window.SKIAF.bcmUIHomeUrl = /*[[${skiaf.bcmUIHomeUrl}]]*/ {};
          window.SKIAF.loginUserInfo = /*[[${skiaf.loginUserInfo}]]*/ {};
   window.SKIAF.PATH = /*[[${skiaf.path}]]*/ [];
   window.SKIAF.CONSTATNT = /*[[${skiaf.constant}]]*/ [];
       /*]]>*/
    })();
</script>
```

- 자바 스크립트 주석(/*...*/)을 사용하면 페이지를 브라우저에 정적으로 표시 할 때 주석 안에 포함 되어 있는 표현식은 무시된다.
- 따라서 페이지를 정적으로 표시하면 window.SKIAF.i18n = []; 가 실행된다.
- Thymeleaf 는 서버 랜더링을 통해 [[..]] 안에 값을 판단해서 출력한다.

[Thymeleaf Layout Dialect]

SKIAF 에서 전체적인 공통 레이아웃을 설정하는데 사용하고 있는 기능이다.

기존의 Thymeleaf 의 코드 재사용성을 개선하기 위해 재사용이 가능한 레이아웃과 템플릿을 작성할 수 있도록 도와주는 라이브러리이며,
spring-boot-starter-thymeleaf 에 포함되어 있기 때문에, 따로 dependency를 추가할 필요는 없다.

기존 ViewResolver 설정에서 layout dialect를 사용하기 위해 추가한 부분은 SpringTemplateEngine 에 아래 코드 부분이다.

```
templateEngine.addDialect(new LayoutDialect());
```

Layout Dialect 속성

- layout:include

Thymeleaf의 th:include와 유사하지만 선언한 태그 하위 템플릿을 전달할 수 있다.

재사용이 가능하며, 컨텍스트 변수만으로 내용을 결정하거나 구성하기에는 너무 복잡한 내용이있는 HTML이 있는 경우 유용하다.

- layout:replace

layout:include 와 비슷하다.

layout:include 는 선언된 태그 하위에 template 이 임포트가 되지만, layout:replace 는 선언된 태그를 포함해서 templete 이 변경된다.

- layout:fragment

동일한 이름을 공유하는 내용 템플릿의 섹션으로 대체 될 수있는 섹션을 표시한다.

- layout:decorator

어떤 레이아웃 템플릿을 사용할지를 지정한다.

<html> 태그에 선언이 되며, 적용 할 decorator template의 위치를 지정한다.

decorator를 사용하는 메커니즘은 기존 Thymeleaf가 사용하는 것과 동일하다.

Layout 예제

- 공통으로 사용할 상위 레이아웃 html

templates/skiaf/layout/default-layout.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

<!-- 레이아웃 -->
<head layout:replace="skiaf/fragments/config :: configFragment"></head>
<body layout:replace="skiaf/fragments/body :: bodyFragment"></body>
</html>
```

SKIAF에서 실제 컨텐츠 페이지에서 decorator로 사용하는 기본 상위 레이아웃이다.

복잡한 구조를 단순하게 하기 위해 head 와 body 태그 부분을 나누었다. layout:replace 기능을 사용해서 각 해당 경로에 있는 파일의 fragment 로 선언된 부분으로 대체한다.

- default-layout.html 에서 layout:replace 로 대체되는 공통 head 레이아웃

templates/skiaf/fragments/config.html

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head layout:fragment="configFragment">

    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />

    <meta id="_csrf" name="_csrf" th:content="${_csrf.token}"/>
    <meta id="_csrf_header" name="_csrf_header" th:content="${_csrf.headerName}"/>

    <meta charset="UTF-8"/>
    <base th:href="@{/}" />

    <!-- 페이지 head -->
    <th:block layout:fragment="html_head"></th:block>

    <!-- css -->
    <!-- alopex css -->
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/alopex/ui/css/alopex-ui-default.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/alopex/grid/alopex-grid.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/alopex/ui/css/src/alopex-ext.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/alopex/grid-plugin-search/alopex-grid-plugin-search.css}"
/>
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/alopex/ui/script/src/webeditor/alopex-webeditor.css}" />

    <!-- common css -->
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/css/common/skiaf-common.css}" />
    <link rel="stylesheet" type="text/css" th:href="@{/static/skiaf/css/common/skiaf-layout.css}" />

    <!-- 페이지별 lib css -->
    <th:block layout:fragment="add_lib_css"></th:block>

    <!-- 페이지별 css -->
    <th:block layout:fragment="custom_css"></th:block>

    <!-- data include -->
    <th:block layout:include="skiaf/fragments/data"></th:block>

    <!-- js -->
    <!-- jquery js -->
    <script type="text/javascript" th:src="@{/static/skiaf/script/jquery/jquery.min.js}"></script>
    <script type="text/javascript" th:src="@{/static/skiaf/script/jquery/js.cookie.js}"></script>

    <!-- alopex js -->
    <script type="text/javascript" th:src="@{/static/skiaf/alopex/ui/script/alopex-ui.min.js}"></script>
    <script type="text/javascript" th:src="@{/static/skiaf/alopex/grid/alopex-grid.min.js}"></script>
    <script type="text/javascript" th:src="@{/static/skiaf/alopex/ui/script/src/alopex-ext.min.js}"></script>
    <script type="text/javascript" th:src="@{/static/skiaf/alopex/ui/script/src/alopex-ext-setup.js}"></script>
    <script type="text/javascript" th:src="@{/static/skiaf/alopex/ui/script/src/webeditor/alopex-webeditor.min.js}"></script>
    <script type="text/javascript" async="async" th:src="@{/static/skiaf/alopex/grid/alopex-grid-excel.min.js}"></script>
    <script type="text/javascript" async="async"
th:src="@{/static/skiaf/alopex/grid-plugin-search/alopex-grid-plugin-search.min.js}"></script>
    <script type="text/javascript" async="async"
th:src="@{/static/skiaf/alopex/ui/script/src/webeditor/alopex-webeditor-setup.js}"></script>

</head>
</html>
```

configFragment 네이밍과 일치하는 default-layout.html 의 head 부분으로 대체된다.

- default-layout.html 에서 layout:replace 로 대체되는 공통 body 레이아웃

templates/skiaf/fragments/body.html

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">

<body layout:fragment="bodyFragment">

    <div class="skiaf-wrap"> <!-- skiaf-collapse :close -->

        <th:block layout:include="skiaf/fragments/header"></th:block>

        <section class="skiaf-layout-content">

            <!-- lnb-->
            <th:block layout:include="skiaf/fragments/lnb"></th:block>
            <!-- // lnb -->

            <!-- Content Area -->
            <th:block layout:fragment="content_body"></th:block>
            <!-- // Content Area -->

        </section>

        <th:block layout:include="skiaf/fragments/footer"></th:block>

    </div>

    <th:block layout:include="skiaf/fragments/common-js"></th:block>
    <th:block layout:fragment="custom_js"></th:block>

</body>

</html>
```

bodyFragment 네이밍과 일치하는 default-layout.html 의 body 부분으로 대체된다.

- 위의 공통 레이아웃을 조립해서 사용하는 실제 컨텐츠 페이지 html

templates/skiaf/view/code/code-list.html

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorator="skiaf/layout/default-layout">

    <th:block layout:fragment="html_head">
        <title th:text="#{bcm.code.title}"></title>
    </th:block>

    <th:block layout:fragment="add_lib_css"></th:block>

    <th:block layout:fragment="custom_css"></th:block>

    <th:block layout:fragment="content_body">
        <!-- Content Area -->
        <div class="skiaf-wrap-content">
            (컨텐츠 html...)
        </div>
        <!-- // Content Area -->
    </th:block>

    <th:block layout:fragment="custom_js">
        <script type="text/javascript" th:src="@{/static/skiaf/script/code/code-common.js}"></script>
        <script type="text/javascript" th:src="@{/static/skiaf/script/code/code-list.js}"></script>
    </th:block>

</html>
```

layout:decorator 에 레이아웃 템플릿으로 사용할 default-layout 을 선언한다.
config.html 과 body.html 의 동일한 이름의 layout:fragment 부분에 대체된다.

[Thymeleaf 와 Alopex 역할 구분]

SKIAF 는 Thymeleaf 와 Alopex의 역할을 구분해서 사용하고 있다.

그 이유는 Thymeleaf 와 Alopex 에서 데이터를 바인딩하는 부분의 코드가 중첩이 되기 때문인데,

아래 Alopex의 데이터 바인딩 예제를 보면 알 수 있듯이, Alopex 또한 Thymeleaf 처럼 html 태그의 속성을 사용해 템플릿을 랜더링 한다.

때문에 Alopex의 data-bind 를 사용한 부분에 Thymeleaf의 th:text 등을 사용하게되면 혼란을 줄 수 있고, 오류가 발생할 수 있다.

Alopex ajax 데이터 바인딩 예제

NoticeRestController.java

```
@RestController
public class NoticeRestController {

 @GetMapping(value = "/api/notice/list")
 public List<Notice> getNoticeList() {
  return noticeService.getNoticeList();
 }
}
```

views/notice/list.html

```
<div>
 <table id="datatable">
  <thead>
   <tr>
    <th>번호</th>
    <th>제목</th>
    <th>작성자</th>
    <th>게시기간</th>
    <th>작성일</th>
   </tr>
  </thead>
  <tbody data-bind="foreach: list">
   <tr>
    <td class="tdSeq" data-bind="text : seq"></td>
    <td data-bind="text : title"></td>
    <td data-bind="text : author"></td>
    <td>
     <span data-bind="text : postStrDt""></span> ~ <span data-bind="text : postEndDt""></span>
    </td>
    <td data-bind="text : regDt"></td>
   </tr>
  </tbody>
 </table>
</div>
```

notice.js

```
$a.page(function() {

 // 초기화 함수
 this.init = function(id, param) {
  this.search();
 }

 this.search = function() {
  $.ajax({
   url : '/api/notice/list',
   method : 'GET',
   success : function(data, textStatus, jqXHR) {
    $('#datatable').setData({ list : data });
   }
  });
 }
});
```

이를 방지하기위해 SKIAF에서는 이 둘의 역할을 다음과 같이 나누어서 사용한다.

그리고 Contoller에서 url path에서 구분할 수 있도록 prefix를 지정했다.

Thymeleaf 의 역할

- ViewResolver 설정
- MVC 패턴에서의 ModelAndView 에 addObject 된 값 사용(라벨의 다국어 처리)
- HTML 레이아웃 설정
- 구분을 위한 Controller path prefix - "view" (ex. /view/bcm/codes)

Alopex 의 역할

- Alopex 에서 제공해주는 UI 컴포넌트들을 사용해서 마크업
- Ajax 통신 데이터바인딩
- 구분을 위한 Controller path prefix - "api" (ex. /api/codes)

Thymeleaf 와 Alopex 역할 구분 예제

```
<tr>
   <!-- Thymeleaf 예제 -->
   <th><th:block th:text="#{bcm.user.login-id}"/></th>
   <!-- Alopex 예제 -->
   <td data-bind="text : loginId"></td>
</tr>
```

IV.2.3 SKIAF 스크립트 작성원칙

[SKIAF 스크립트 특징]

- Alopex 프레임워크 기반이며, Alopex UI, Alopex Grid에서 제공하는 기능을 최대한 활용하여 개발하였다.
- Alopex 권고사항을 바탕으로 각 페이지 별로 모듈화 되어있다.
- javascript 코딩 컨벤션을 준수하고있다.
- 여러 페이지에서 공통으로 사용하는 함수 또는 오브젝트는 window.SKIAF 객체로 등록되어있다.

[Alopex UI]

SKIAF 는 front-end 개발에 Alopex UI를 사용했으며, Alopex UI 관련 표준과 가이드는 Alopex UI 홈페이지를 참고한다.

- Alopex UI/UX 표준
  http://ui.alopex.io/demonstration/alopexUIUX 참고
- Alopex UI 개발가이드
  http://ui.alopex.io/document/development 참고
- Alopex UI 퍼블리싱가이드
  http://ui.alopex.io/document/publish 참고

[Alopex Grid]

SKIAF 는 게시판 개발에 Alopex Grid를 사용했으며, Alopex Grid 가이드는 Alopex Grid 홈페이지를 참고한다.

- Alopex Grid 개발가이드
  API Document : http://grid.alopex.io/html/docs.html 참고
  DEMO : http://grid.alopex.io/html/demo.html#!gridoption/personalization 참고
- Alopex Grid 퍼블리싱가이드
  Publishing Document : http://grid.alopex.io/html/docs_guide_publishing.html

[SKIAF 에서 사용하는 javascript 라이브러리와 버전 정보]

- Alopex UI v2.11.4
- Alopex Grid v3.11.20
- Alopex Grid Search Plugin v1.0.1
- jQuery JavaScript Library v3.3.1
- JavaScript Cookie v2.2.0

[SKIAF 에서 HTML 내에 스크립트 import]

Thymeleaf 로 HTML를 랜더링 하기 때문에 예제 처럼 th 속성을 사용해서 스크립트를 import 한다.

```
<!DOCTYPE html>
<html lang="ko" xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
    layout:decorator="skiaf/layout/default-layout">


<!-- script import. Thymeleaf 속성을 사용한다. -->
<script type="text/javascript" th:src="@{/static/skiaf/alopex/ui/script/alopex-ui.min.js}"></script>


<!-- layout decorator 로 스크립트가 import 되는 위치를 fragments 처리해서 아래와 같이 사용할 수도 있다. -->
<th:block layout:fragment="custom_js">
 <script th:inline="javascript" th:src="@{/static/skiaf/script/menu/menu-list.js}"></script>
</th:block>


</html>
```

[SKIAF 스크립트 작성 원칙]

SKIAF 는 Alopex 에서 권고하는 사항을 바탕으로 스크립트를 각 페이지 별로 모듈화를 하였으며, Alopex 사용시 반드시 지켜야 할 부분을 충족하는 템플릿 형태로 모든 스크립트를 일관성있게 작성하였다.

다음의 template.js 파일은 SKIAF의 스크립트 작성 예제이며, resources/static/skiaf/script/template.js 에 위치한다. 개발자들은 이 템플릿을 예제로 개발하면된다.

스크립트 상단에 주석으로 파일 생성날짜와 작성자 정보, 그리고 간단하게 해당 스크립트에 대한 설명을 적는다.

- ECMA Script 5 기능인 "use strict"; 를 선언해서 자바스크립트 문법을 보다 엄격하게 검사한다.
- $(document).ready 대신 $a.page 를 사용한다.
- Alopex UI, Alopex Grid 의 setup 은 $a.page 상단에 정의하고 주석(Alopex setup)으로 구분한다. 해당 페이지에서 설정하는 부분이 없을 경우 생략한다.
- 전역변수는 Alopex setup 다음으로 위치하고 주석(Variables)으로 구분한다.
- Alopex는 $a.page에 초기화 함수(init)을 제공한다. 주석(init)으로 구분하고 전역변수 다음으로 위치한다.
- 이벤트 바인딩하는 함수들은 구분하기 위해 따로 addEvent 라고 정의한 함수에 추가하고, 주석(ADD Event)로 구분한다.
  그리고 init 함수에서 addEvent 함수를 호출해서 이벤트들을 일괄적으로 등록한다.
- 그외 함수들은 주석(Functions)로 구분하고 하단에 정의한다.

template.js

```
/**
 * create by 2018.09.04 - in01869
 * description : 페이지 스크립트 샘플 js
 */
"use strict";
var Template = $a.page(function() {

    /*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    | Alopex setup
    |-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

    // alopex ui setup
    $a.setup({
        locale : SKIAF.i18n.langCurrentCode
    });

    /*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    | Variables
    |-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

    var duplChkYn = false;

    /*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    | Init
    |-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

    this.init = function(id, param) {

        Template.addEvent();

        Template.setMenuData();
    };

    /*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    | ADD Event
    |-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

    this.addEvent = function() {

        // 버튼 클릭 이벤트
        $('#btnSave').on('click', function(e) {
            Template.save();
        });

    };

    /*-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    | Functions
    |-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=*/

    /**
     * 저장
     */
    this.save = function() {

        if(Template.duplchk()) {
            /*...*/
        }
    };

    /**
     * 중복 체크
     */
    this.duplchk = function() {
        /*...*/
    };

});
```

[SKIAF 에서 사용한 Alopex 주요 기능]

- $a.page (http://ui.alopex.io/development/javascript/page)

  웹 페이지 로딩에 대한 초기화 시점을 보장하여 Alopex UI 컴포넌트와 애플리케이션 자바스크립트 함수를 규격화된 형태로 작성할 수 있도록 하는 페이지 컨트롤러 함수이다.

  스크립트페이지 초기화 (init) 과 함께 사용한다.

  $a.page 예제

```
var PageScriptSample = $a.page(function() {

 this.init = function(id, param) {

       PageScriptSample.addEvent();
       PageScriptSample.setData();
   };

});
```

- $a.setup (http://ui.alopex.io/development/javascript/setup)

  Alopex UI 컴포넌트의 옵션 중 공통적으로 적용할 옵션들에 대해 셋업하는 기능이다.

  SKIAF 에서는 common.js 에서 다국어 설정과 파일업로드 공통 셋업을 사용하고있다.

common.js $a.setup 공통 처리

```
/**
    * alopex ui 기본 설정
    * - 다국어설정 : alopex에서 기본으로 지원하는 다국어로는 'ko'(한국어), 'en'(영어), 'zh'(중국어), 'ja'(일본어) 가 있다.
    */
   $a.setup({
      locale : SKIAF.i18n.langCurrentCode
   });


...


   /**
    * Alopex UI FileUpload API 사용 시 필요한 공통 처리
    */
   $a.setup('fileupload', {
      locale : SKIAF.i18n.langCurrentCode,
      fileName : 'file',
      showCancel : false,
      showAbort : false,
      showDone : false,
      showCheckedAll : false,
      showUnCheckedAll  : false,
      showDeleteChecked : false,
      allowedTypes  : acceptType,
      acceptFiles : acceptTypeWithDot,
      maxFileSize : maxFileSize,
      beforeUpload : function (xhr) {

         xhr.setRequestHeader($('meta[name="_csrf_header"]').attr('content'), $('meta[name="_csrf"]').attr('content'));

      },
      onError : function(files, status, errMsg, pd, xhr) {

         CommonModule.fileUploadErrorHandle(files, status, errMsg, pd, xhr);
      }
   });
```

- $a.request (http://ui.alopex.io/development/javascript/request)

  SKIAF 는 모든 비동기 HTTP(Ajax) 통신을 Alopex Request API를 사용하고 있으며, 모든 request에 대한 공통 처리를 하기위해 다음과 같이 common.js 에 Alopex Request API setup 을 추가했다.

common.js $a.request 공통 처리

```
/**
    * $a.request() API 사용 시 필요한 공통 처리
    */
   $a.request.setup({
      url : function(id, param) {
         request_id = id;
         request_param = param;

         if (request_id == null) {
            return null;
         } else {
            return SKIAF.contextPath + id;
         }
      },
      timeout: 30000,
      before: function(id, option) {
```

```
            ++requestCount;

            this.requestHeaders["Content-Type"] ="application/json; charset=UTF-8";

            //spring security - csrf_token
            if(option.method != "GET"){
                this.requestHeaders[$("meta[name='_csrf_header']").attr("content")] =
$("meta[name='_csrf']").attr("content");
            }

            //loading 시작
            body_progress = $('body').progress();

        },
        after: function(res) {
            // 통신은 성공이지만, 서버업무오류 조건일 경우,
            // after콜백에서 this.isSuccess = false로 변경해야 success콜백이 아닌, fail콜백이 호출됨
            // if(fail condition ) {
            //      this.isSuccess = false;
            // }
        },
        success: function(res) {
            // 통신이 성공적으로 이루어 진 경우 호출되는 콜백함수

        },
        fail: function(res){
            // 통신은 성공적으로 이루어 졌으나, 서버오류가 발생한 경우 호출되는 콜백함수
            // after콜백에서 this.isSuccess = false로 변경해야 success콜백이 아닌, fail콜백이 호출됨

            SKIAF.console.error(res);

            /*
             * request fail 일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
             * $a.request 요청시 파라미터에 failConfirmCallback 함수를 추가해준다.
             * ex)
             *
             * $a.request('/api/articles/'+id, {
                    method : 'GET',
                    success : function(res) {
                        $('#frm').setData(res.data);
                    },
                    //request fail일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
                    failConfirmCallback : function() {
                        console.log('test fail alert confirm');
                    },
                    //request error일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
                    errorConfirmCallback : function() {
                        console.log('test error alert confirm');
                    }
                });
             */
            if(request_param.hasOwnProperty('failConfirmCallback')){
                if(typeof request_param.failConfirmCallback === "function"){
                    // 에러 알림 팝업 호출(callback function 포함)
                    SKIAF.popup.exception(JSON.parse(res.responseText), request_param.failConfirmCallback);
                }
            } else {
                // 에러 알림 팝업 호출
                SKIAF.popup.exception(JSON.parse(res.responseText));
            }
        },
        error: function(errObject) {
            // 통신이 실패한 경우 호출되는 콜백함수

            SKIAF.console.error(errObject);

            // 401,403 에러시에 해당 에러 페이지로 이동
            if(errObject.status == 401){
                location.href = SKIAF.contextPath + SKIAF.PATH.ERROR_401;
                return;
            }else if(errObject.status == 403){
                // 페이지 이동이 아닌 alert으로 처리
                //location.href = SKIAF.contextPath + SKIAF.PATH.ERROR_403;
```

```
            var result = JSON.parse(errObject.response);
            result.meta.userMessage = SKIAF.i18n.messages["bcm.common.exception.forbidden-detail"];
            SKIAF.popup.exception(result);
            return;
        }


        var errorRes = JSON.parse(errObject.response)
        var code = errorRes.meta.code;

        if(code == null){

            /*
             * request error일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
             * $a.request 요청시 파라미터에 errorConfirmCallback 함수를 추가해준다.
             * ex)
             *
             * $a.request('/api/articles/'+id, {
                    method : 'GET',
                    success : function(res) {
                        $('#frm').setData(res.data);
                    },
                    //request fail일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
                    failConfirmCallback : function() {
                        console.log('test fail alert confirm');
                    },
                    //request error일 경우 에러알림팝업이 뜨고 '확인'버튼을 누르면 실행되는 callback 함수
                    errorConfirmCallback : function() {
                        console.log('test error alert confirm');
                    }
                });
             */

            if(request_param.hasOwnProperty('errorConfirmCallback')){
                if(typeof request_param.errorConfirmCallback === "function"){
                    // 에러 알림 팝업 호출(callback function 포함)
                    SKIAF.popup.exception(JSON.parse(errObject.responseText), request_param.errorConfirmCallback);
                }
            } else {
                // 에러 알림 팝업 호출
                SKIAF.popup.exception(JSON.parse(errObject.responseText));
            }
        }

    },
    last : function(res, status, httpstatus) {
        // 통신성공,실패여부와 관계없이 맨 마지막에 호출되는 콜백함수

        --requestCount;
        //loading 종료
        if(requestCount == 0){
            setTimeout(function(){
                body_progress.remove();
            }, 300);
```

```
            }
        }
    });
```

setup 의 공통처리를 바탕으로 $a.request는 다음과 같이 사용하고 있다.

```
$a.request(SKIAF.PATH.MENUS, {
            method : 'POST',
            data : saveData,
            success : function(res) {
                $a.close({
                    type : 'success',
                    menuId : saveData.menuId
                });
            }
        });
```

- $a.popup (http://ui.alopex.io/development/javascript/popup)

  윈도우 팝업, 레이어 팝업 등을 호출할 때 사용한다.

$a.popup 예제

```
$a.popup({
        url: SKIAF.contextPath + SKIAF.PATH.VIEW_MENUS_CHANGE,
        title: SKIAF.i18n.messages['bcm.menu.menu-order-change'],
        iframe: false,
        movable:true,
        width: 400,
        height: 600,
        center: true,
        callback : function(data) {

            if (data !== null) {
                if(data.type == 'confirm') {
                    MenuModule.viewMenuTree();
                }
            }
        }
    });
```

$a.popup으로 팝업 호출시 오류가 발생 할 경우 공통으로 오류에 대한 처리를 할 수 있도록 common.js에 $a.popup.setup 을 추가했다.

common.js $a.popup.setup 공통처리

```
/**
 * $a.popup() API 사용 시 필요한 공통 처리
 */
$a.popup.setup({
    errorCallback : function(res) {
        // 401,403 에러시에 해당 에러 페이지로 이동
        if(res == 401){
            location.href = SKIAF.contextPath + SKIAF.PATH.ERROR_401;
            return;
        }else if(res == 403){
            // 페이지 이동이 아닌 alert으로 처리
            //location.href = SKIAF.contextPath + SKIAF.PATH.ERROR_403;
            var result = JSON.parse(res.response);
            result.meta.userMessage = SKIAF.i18n.messages["bcm.common.exception.forbidden-detail"];
            SKIAF.popup.exception(result);
            return;
        }
    }
});
```

또한, SKIAF 에서는 알림, 에러, 확인 팝업을 공통으로 사용하고 있으며, common-util.js 에 SKIAF.popup 공통 함수를 정의해서 사용하고 있다.

common-util.js SKIAF.popup

```
window.SKIAF.popup = {
    /*
     * 알림 팝업 호출
     *
     *  1. 메세지
     * ex) SKIAF.popup.alert('ID를 입력해주세요.');
     *
     * 2. 메세지와 콜백함수
     * ex) SKIAF.popup.alert('ID를 입력해주세요.', function(){ alert('확인 버튼을 누르셨습니다.'); });
     *
     * 3. 메세지와 콜백함수(매개변수 있는경우)
     * ex) SKIAF.popup.alert('ID를 입력해주세요.', function(callbackParam){ alert(callbackParam.message); }, callbackParam);
     *
     */
    alert : function(message, callbackFunction) {

        // 콜백함수의 매개변수
        var callbackParam = Array.prototype.splice.call(arguments, 2);

        var pop = $a.popup({
            title : SKIAF.i18n.messages["bcm.common.alert"],
            url : '/static/skiaf/popup/alert.html',
            data : {
                message : message
            },
            iframe: false,
            movable:true,
            width: 400, //width 필요
            height: 253, //height 필요
            callback : function(data) {
                if(data.type == 'confirm') {
                    if(typeof callbackFunction === "function"){
                        callbackFunction.apply(this, callbackParam);
                    }
                }
            }
        });
        $(pop).addClass('alert');
    },
```

```
    /*
     * 확인 팝업 호출
     *
     * 1. 메세지
     * ex) SKIAF.popup.confirm('저장하시겠습니까?');
     *
     * 2. 메세지와 콜백함수
     * ex) SKIAF.popup.confirm('저장하시겠습니까?', function callback(){alert('callback!!')});
     *
     * 3. 메세지와 콜백함수(매개변수 있는경우)
     * ex) SKIAF.popup.confirm('저장하시겠습니까?', function callback(param){alert(param.message);}, callbackParam);
     */
    confirm : function(message, callbackFunction) {

        // 콜백함수의 매개변수
        var callbackParam = Array.prototype.splice.call(arguments, 2);

        var pop = $a.popup({
            title : SKIAF.i18n.messages["bcm.common.confirm"],
            url : '/static/skiaf/popup/confirm.html',
            data : {
                message : message
            },
            iframe: false,
            movable:true,
            width: 400, //width 필요
            height: 253, //height 필요
            callback : function(data) {
                if(data.type == 'confirm') {
                    if(typeof callbackFunction === "function"){
                        callbackFunction.apply(this, callbackParam);
                    }
                }
            }
        });
        $(pop).addClass('alert');
    },
    /*
     * 에러 팝업 호출
     *
     * 1. 에러 메타(RestResponse) 정보
     * ex) SKIAF.popup.exception({meta : {systemMessage : null, userMessage : '찾지못했습니다.', displayType :
'LAYER_POPUP'}});
     *
     * 2. 에러 메타(RestResponse) 정보와 콜백함수
     * ex) SKIAF.popup.exception({meta : {systemMessage : 'not found', userMessage : '찾지못했습니다.', displayType :
null}}, function callback(){alert('callback!!')});
     *
     * 3. 에러 메타(RestResponse) 정보와 콜백함수(매개변수 있는경우)
     * ex) SKIAF.popup.exception({ meta : {systemMessage : 'not found', userMessage : '찾지못했습니다.', displayType : null}
}, function callback(callbackParam){alert(callbackParam.message);}, callbackParam);
     */
    exception : function(response, callbackFunction) {

        // 콜백함수의 매개변수
        var callbackParam = Array.prototype.splice.call(arguments, 2);

        var displayType = 'LAYER_POPUP';
        if(response.meta.displayType) {
            displayType = response.meta.displayType;
        }

        if(displayType == 'LAYER_POPUP') {

            var pop = $a.popup({
                title : SKIAF.i18n.messages["bcm.common.error"],
                url : '/static/skiaf/popup/exception.html',
                data : {
                    meta : response.meta
                },
                iframe: false,
                movable:true,
                width: 420, //width 필요
                height: 320, //height 필요
```

```
    callback : function(data) {
        if(data.type == 'confirm') {
            callbackFunction.apply(this, callbackParam);
        }
    }
});
$(pop).addClass('alert');
```

```
        }
    }
};
```

오픈된 팝업을 닫을 때는 $a.close 을 사용한다. $a.close(data) 로 넘긴 data는 $a.popup({callback : function(data) {... }}) 에서 callback 에서 받을 수 있다.

$a.close 예제

```
$a.close({
        type : 'success',
        menuId : saveData.menuId
    });


    var pop = $a.popup({
        url: SKIAF.PATH.VIEW_MENUS_UPDATE,
        title: SKIAF.i18n.messages['bcm.menu.menu-modify'],
        data : {
            'menu' : node.data
        },
        iframe: false,
        movable:true,
        width: 1000,
        height: 600,
        center: true,
        callback : function(data) {
            if(data !== null) {
                if(data.type == 'success') {
                    var modifyMenuId = data.menuId;
                    MenuModule.viewMenuTree(modifyMenuId);
                }
            }
        }
    });
```

- $a.maskedinput (http://ui.alopex.io/development/javascript/maskedinput)
input 태그 text에 대해 특정 format의 형태로 쉽게 입력 가능하도록 도와주는 기능이다.

$a.maskedinput script 예제

```
//날짜 시간 입력 mask 처리
    $a.maskedinput($("input[name=startDate]")[0], "0000-00-00");
    $a.maskedinput($("input[name=endDate]")[0], "0000-00-00");
    $a.maskedinput($("input[name=startTime]")[0], "00:00");
    $a.maskedinput($("input[name=endTime]")[0], "00:00");
```

$a.maskedinput html 예제

```
<div class="Daterange" data-default-date="false" data-type="daterange">
                        <div class="Startdate Dateinput" data-type="dateinput">
                            <input class="Textinput" name="startDate" data-maskedinput-rule="0000-00-00"
maxlength="16" placeholder="yyyy-MM-dd" type="text" />
                            <div class="Calendar"></div>
                        </div>
                        <input class="Textinput" name="startTime" style="width: 50px;" placeholder="00:00"
data-maskedinput-rule="00:00" data-type="textinput" /> ~
                        <div class="Enddate Dateinput" data-type="dateinput">
                            <input class="Textinput" name="endDate" data-maskedinput-rule="0000-00-00"
data-type="textinput" data-classinit="true" maxlength="16" placeholder="yyyy-MM-dd"
                                type="text" />
                            <div class="Calendar"></div>
                        </div>
                        <input class="Textinput" name="endTime" style="width: 50px;" placeholder="23:59"
data-maskedinput-rule="00:00" />
                    </div>
```

IV.2.4 Front-End 보안코딩

[XSS]

Front-End에서는 별도의 XSS를 처리 하지 않고, 순수 HTML로써 출력하는 형태를 최대한 피함으로써 XSS를 회피한다.

- thymeleaf 의 utext, alopex 의 html 형식 data-bind, javascript 의 innerHTML 이 해당된다.

불가피 하게 순수 HTML을 출력해야 하는 경우, Back-End에서 문제 소지가 있는 HTML 키워드를 걸러내고 이를 출력한다.

XSS에 대한 구체적인 설정 내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.3 보안 > XSS" 부분을 참조하면 된다.

[CSRF]

실질적인 CSRF에 대한 판단은 Back-End에서 이뤄진다.

Front-End에서는 Back-End에서 내려준 csrf token을 Apopex UI의 reqeust / fileupload 시에 header값에 설정해서 보내주는 역할을 한다.

사용 예제

config.js

```
...
<head layout:fragment="configFragment">
    ...
    <meta id="_csrf" name="_csrf" th:content="${_csrf.token}"/>
    <meta id="_csrf_header" name="_csrf_header" th:content="${_csrf.headerName}"/>
    ...
</head>
...
```

common.js

```
// request에 대한 csrf 설정
$a.request.setup({
    ...
    before: function(id, option) {
        ...
        //spring security - csrf_token
        if(option.method != "GET"){
            this.requestHeaders[$("meta[name='_csrf_header']").attr("content")] = $("meta[name='_csrf']").attr("content");
        }
        ...
    },
    ...
}


// fileupload에 대한 csrf 설정
$a.setup('fileupload', {
    ...
    beforeUpload : function (xhr) {
        xhr.setRequestHeader($('meta[name="_csrf_header"]').attr('content'), $('meta[name="_csrf"]').attr('content'));
    },
    ...
}
```

CSRF에 대한 구체적인 설정 내용은 "VI. SKIAF 개발자매뉴얼 - 시스템공통기능 > VI.3 보안 > CSRF" 부분을 참조하면 된다.

Appendix#1. Javascript Coding Convention

SKIAF 는 Alopex의 Javascript Coding Convention을 표준으로 참고하여 스크립트를 작성하였다,

주된 설명은 Alopex UI 개발가이드 문서의 목차 "6.3 Javascript 코딩컨벤션"에 작성되어 있다.

- Alopex UI 개발가이드
  http://ui.alopex.io/document/development 참고

Appendix#2. CSS Coding Convention

[기본규칙]

※ 퍼블리싱 가이드

Aloepx UI 에서 제공하는 퍼블리싱 가이드(http://ui.alopex.io/document/publish)를 따른다.

모든 속성은 숫자, 대문자, 특수문자로 시작할 수 없으며, 영문 소문자로 작성한다.
단어의 구분을 위하여 하이픈 표기법을 사용한다.

소문자, 숫자 만을 이용해서 작명한다.

영역은 큰 순으로 (skiaf)-wrap 〉(skiaf)-layout 〉(skiaf)-ui 〉(skiaf)-box 또는 (skiaf)-area 이름이 붙는다.

css 이름은 block1, block2 형태로 사용할 수 있고 block2에 여백이나 위치를 지정하고 block1은 독립적으로 유지 할 수 있다.

[선언순서]

선언의 순서는 아래와 같은 순서로 그룹핑 한다.

1. 위치 관련
2. 박스모델 관련
3. 타이포그래피
4. 시각효과
5. 기타

[@import 사용금지]

CSS 내에 @import는 성능을 저하시키기 때문에 여러 개의 CSS를 link한다.

[줄바꿈과 들여쓰기]

CSS의 선언은 여러 선언을 하나의 라인으로 구성하면 디버깅하기 어렵기 때문에 하나의 선언 당 하나의 라인으로 구성한다.

미디어 쿼리 및 vendor prefix 대한 속성 사용으로 속성을 한 줄에 작성을 하게 되면 코드의 가독성이 떨어져 유지 보수와 수정 작업에 불편함이 따르기 때문이다.

[주석]

시작 주석만 작성하며 끝 주석은 작성하지 않는다.

[세미콜론]

마지막 속성 값의 끝에도 세미콜론을 사용한다.