

개요

전자정부 표준프레임워크 기반의 시스템 개발시 Exception 처리, 정확히는 Exception 별 특정 로직(후처리 로직이라고 부르기도 함)을 흐를 수 있도록 하여 Exception 에 따른 적절한 대응이 가능도록 하고자 하는데 목적이 있다.
AOP 의 도움을 받아 비즈니스 POJO와 분리되어 After throwing advice 로 정의하였다.
AOP 관련한 내용은 [AOP 모듈](#)을 참조하길 바란다.

- Exception Handling 서비스
 - 개요
 - 설명
 - Aop Config, ExceptionTransfer 설정 및 설명
 - leaveTrace 설정 및 설명
 - 참고자료

Exception 에 대해 이야기 하겠다.
Exception 이 발생시 Exception 발생 클래스 정보와 Exception 종류가 중요하다.
Exception 발은생 클래스 정보와 Exception 종류는 모두 후처리 로직의 대상일지 아닐지를 결정하는데 사용된다.

```
public CategoryVO selectCategory(CategoryVO vo) throws Exception {
    CategoryVO resultVO = categoryDAO.selectCategory(vo);
    try {
        ....
        // 넘어온 resultVO 가 null 인경우 EgovBizException 발생 (result.nodata.msg 는 메세지 키에 해당됨)
        if (resultVO == null)
            throw processException("result.nodata.msg");
        // 또는 throw processException("result.nodata.msg", 발생한 Exception );
        return resultVO;
    }
}
```

설명

우리는 앞에서 언급했던 Exception 후처리 방식과 Exception 아니지만 후처리 로직(leaveTrace)을 실행하는 방식에 대해 설명하도록 하겠다.
간략하게 보면
Exception 후처리 방식은 **AOP(pointCut ⇒ after-throw) ⇒ ExceptionTransfer.transfer() ⇒ ExceptionHandlerService ⇒ Handler** 순으로 실행된다.

LeavaTrace 는 AOP를 이용하는 구조는 아니고 Exception 을 발생하지도 않는다. 단지 후처리 로직을 실행하도록 하고자 함에 목적이 있다.
실행 순서는 **LeavaTrace ⇒ TraceHandlerService ⇒ Handler** 순으로 실행한다.

먼저 Exception Handling 에 대해 알아보도록 하자.

Aop Config, ExceptionTransfer 설정 및 설명

Bean 설정

Exception 후처리와 leaveaTrace 설정을 위해서 샘플에서는 두개의 xml 파일을 이용한다. (context-aspect.xml, context-common.xml)

먼저 Exception 후처리를 위한 부분을 보겠다.
Exception Handling 을 위한 AOP 설정은 아래와 같다.
비즈니스 개발시 패키지 구조는 바뀌기 때문에 Pointcut은 egov.sample.service.*Impl.*(..) 을 수정하여 적용할 수 있다.
ExceptionTransfer 의 property 로 존재하는 exceptionHandlerService 는 다수의 HandleManager 를 등록 가능하도록 되어 있다.
여기서는 defaultExceptionHandlerManager을 등록한 것을 볼 수 있다.

context-aspect.xml

```
...
<aop:config>
    <aop:pointcut id="serviceMethod"
        expression="execution(* egov.sample.service.*Impl.*(..))" />
    <aop:aspect ref="exceptionTransfer">
        <aop:after-throwing throwing="exception"
            pointcut-ref="serviceMethod" method="transfer" />
    </aop:aspect>
</aop:config>

<bean id="exceptionTransfer" class="egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer">
    <property name="exceptionHandlerService">
        <list>
            <ref bean="defaultExceptionHandlerManager" />
        </list>
    </property>
</bean>

<bean id="defaultExceptionHandlerManager"
    class="egovframework.rte.fdl.cmmn.exception.manager.DefaultExceptionHandlerManager">
```

```

        <property name="patterns">
            <list>
                <value>**service.*Impl</value>
            </list>
        </property>
        <property name="handlers">
            <list>
                <ref bean="egovHandler" />
            </list>
        </property>
    </bean>

    <bean id="egovHandler"
        class="egovframework.rte.fdl.cmmn.exception.handler.EgovServiceExceptionHandler" />
    ...

```

defaultExceptionHandler는 setPatters() , setHandlers() 메소드를 가지고 있어 상단과 같이 등록된 pattern 정보를 이용하여 Exception 발생 클래스와의 비교하여 true 인 경우 handlers 에 등록된 handler를 실행한다.

패턴 검사시 사용되는 pathMatcher 는 AntPathMatcher 를 이용하고 있다.

특정 pattern 그룹군을 만든후 patterns 에 등록하고 그에 해당하는 후처리 로직을 정의하여 등록할 수 있는 구조이다.

Handler 구현체

먼저 클래스에 대한 이해가 필요하다. 앞단에서 간단하게 설명을 했지만 다시 정리 하자면 Exception 발생시 AOP pointcut "After-throwing" 걸려 ExceptionTransfer 클래스의 transfer 가 실행된다. transfer 메소드는 ExceptionHandlerManager 의 run 메소드를 실행한다. 아래는 구현예로 DefaultExceptionHandlerManager 코드이다.

(구현시 필수사항) 상위클래스는 AbsExceptionHandlerManager 이고 인터페이스는 ExceptionHandlerService 이다.
구현되는 메소드는 run(Exception exception) 인 것을 확인할 수 있다.

DefaultExceptionHandlerManager.java

```

public class DefaultExceptionHandlerManager extends AbsExceptionHandlerManager implements ExceptionHandlerService {
    @Override
    public boolean run(Exception exception) throws Exception {
        log.debug(" DefaultExceptionHandlerManager.run() ");
        // 매칭조건이 false 인 경우
        if (!enableMatcher())
            return false;

        for (String pattern : patterns) {
            log.debug("pattern = " + pattern + ", thisPackageName = " + thisPackageName);
            log.debug("pm.match(pattern, thisPackageName) = " + pm.match(pattern, thisPackageName));
            if (pm.match(pattern, thisPackageName)) {
                for (ExceptionHandler eh : handlers) {
                    eh.occure(exception, getPackageName());
                }
                break;
            }
        }
        return true;
    }
}

```

Customizable Handler 등록

시나리오 : CustomizableHandler 클래스를 만들어 보고 sample 패키지에 있는 Helloworld 클래스 Exception 시에 CustomizableHandler 를 실행한다.

먼저 CustomHandler 클래스를 아래와 같이 만든다.
ExceptionHandlerManager 에서는 occur 메소드를 실행한다.

Handler 구현체는 반드시 (필수사항) ExceptionHandler Interface를 갖는다.

CustomizableHandler.java

```

public class CustomizableHandler implements ExceptionHandler {
    protected Log log = LoggerFactory.getLog(this.getClass());
    public void occur(Exception ex, String packageName) {
        log.debug(" CustomHandler run.....");
        try {
            log.debug(" CustomHandler 실행 ... ");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

주의해야 하는 부분은 patterns 에 sample 패키지에 있는 Helloworld 클래스를 지정해야하며 여러개의 Handler를 등록 할 수 있다.

```

<bean id="exceptionTransfer" class="egovframework.rte.fdl.cmmn.aspect.ExceptionTransfer">
    <property name="exceptionHandlerService">
        <list>
            <ref bean="customizableExceptionHandler" />
        </list>
    </property>
</bean>

<bean id="customizableExceptionHandler"
class="egovframework.rte.fdl.cmmn.exception.manager.DefaultExceptionHandler">
    <property name="patterns">
        <list>
            <value>**sample.Helloworld</value>
        </list>
    </property>
    <property name="handlers">
        <list>
            <ref bean="customizableHandler1" />
            <ref bean="customizableHandler2" />
            <ref bean="customizableHandler3" />
        </list>
    </property>
</bean>

<bean id="customizableHandler1" class="sample.CustomizableHandler" />
<bean id="customizableHandler2" class="sample.CustomizableHandler" />
<bean id="customizableHandler3" class="sample.CustomizableHandler" />

```

leaveaTrace 설정 및 설명

Exception 이거나 Exception 이 아닌 경우에 Trace 후처리 로직을 실행 시키고자 할 때 사용한다.
 설정하는 기본적인 구조는 Exception 후처리 하는 방식과 같다. 설정파일이 context-common.xml 이다.
 DefaultTraceHandleManager 에 TraceHandler를 등록하는 형태로 설정된다.

Bean 설정

```

...
<bean id="leaveaTrace" class="egovframework.rte.fdl.cmmn.trace.LeaveaTrace">
    <property name="traceHandlerServices">
        <list>
            <ref bean="traceHandlerService" />
        </list>
    </property>
</bean>

<bean id="traceHandlerService"
class="egovframework.rte.fdl.cmmn.trace.manager.DefaultTraceHandleManager">
    <property name="patterns">
        <list>
            <value>*</value>
        </list>
    </property>
    <property name="handlers">
        <list>
            <ref bean="defaultTraceHandler" />
        </list>
    </property>
</bean>

<bean id="antPathMater" class="org.springframework.util.AntPathMatcher" />

<bean id="defaultTraceHandler"
class="egovframework.rte.fdl.cmmn.trace.handler.DefaultTraceHandler" />
...

```

TraceHandler 확장 개발 Sample

Interface TraceHandler를 아래와 같이 implements 한다.

```

package egovframework.rte.fdl.cmmn.trace.handler;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

public class DefaultTraceHandler implements TraceHandler {

    public void todo(Class clazz, String message) {
        //수행하고자 하는 처리로직을 넣는 부분...
        System.out.println(" log ==> DefaultTraceHandler run.....");
    }

}

```

leaveaTrace 코드상 발생 Sample

사용방법을 다시 상기 해보면 아래와 같다.
 메세지키(message.trace.msg) 를 이용하여 메세지 정보를 넘겨 Handler 를 실행한다.

```

public CategoryVO selectCategory(CategoryVO vo) throws Exception {
    CategoryVO resultVO = categoryDAO.selectCategory(vo);
    try {
        //강제로 발생한 ArithmeticException
    }
}

```

```
        int i = 1 / 0;
    } catch (ArithmeticException athex) {
        //Exception 을 발생하지 않고 후처리 로직 실행.
        leaveaTrace ("message.trace.msg");
    }

    return resultVO;
}
```

참고자료

-  exception-handling-framework-for-j2ee
- Effective Java (Joshua Bloch) : Chapter 8 예외처리

개요

- Spring Web Flow
 - 개요
 - 설명
 - 참고자료

Spring Web Flow(SWF)는 웹 애플리케이션내 페이지 흐름(flow)의 정의와 수행에 집중하는 Spring프레임워크 웹 스택의 컴포넌트다.

시스템은 다른 위치에서 재사용될수 있는 자족적 모듈처럼 웹 애플리케이션의 로직적인 흐름(flow)을 획득하는 것을 허용한다. 이러한 흐름(flow)은 비즈니스 프로세스의 구현을 통해 단일 사용자를 안내하고 단일 사용자 대화를 표현한다. 흐름(flow)은 종종 HTTP요청에 대해 수행되고 상태를 가지며, 트랜잭션적인 특징을 보이고 동적이며 오랜시간 구동된다.

Spring Web Flow는 추상화의 좀더 높은 레벨에 존재하고 Struts, Spring MVC, Portlet MVC, 그리고 JSF와 같은 기본 프레임워크 내에서 자족적인 페이지 흐름(flow) 엔진(page flow engine)처럼 통합된다. SWF는 선언적이고 높은 이식성을 가지며 뛰어난 관리능력을 가지는 형태로 명시적으로 애플리케이션의 페이지 흐름(flow)을 획득하는 기능을 제공한다.

설명

Spring Web Flow는 여타의 API에 대한 몇가지 요구 의존성을 가진 자족적인 page flow engine처럼 구조화되었다. 모든 의존성은 주의깊게 관리된다.

대부분의 사용자들은 좀더 큰 웹 애플리케이션 개발 프레임워크내 컴포넌트로 SWF를 사용할것이다.

SWF는 요청 매핑과 응답 표현을 다루기 위한 호출 시스템을 기대하는 컨트롤러 기술에 집중한다.

이 경우, 이러한 사용자는 환경을 위한 가는(thin) 통합 조각에 의존할것이다.

예를 들어, Servlet내 수행 흐름(flow)은 SWF에 대한 요청에 대한 할당(dispatch)과 SWF view선택을 책임지는 표현을 다루는 Spring MVC통합을 사용한다.

Spring처럼, Spring Web Flow는 그들이 필요한 부분을 사용하는 것을 허용하는 방법으로 패키징되는 계층화된(layered) 프레임워크라는 것을 아는게 중요하다.

SWF의 중요한 이득은 어떤 환경에서도 수행될수 있는 자족적인 컨트롤러의 모듈 재사용하여 정의할수 있도록 하는 것이다.

우리는 구체적으로 알아보기전에 Hello World를 찍어 보자.

- [Hello world](#)

Spring Web Flow 의 기본 샘플로 Hotel Booking 을 Spring Source 에서는 제공하고 있다.

우리는 Spring Web Flow 레퍼런스문서를 기준으로 하고 샘플인 Hotel Booking 을 참고하는 형태로 설명하도록 하겠다.

Hotel Booking 샘플 데모 : <http://richweb.springframework.org/swf-booking-faces/spring/intro>

SWF Configuration

- [SWF 시스템 설정](#)
- [Spring Web Flow 와 MVC 연동](#)
- [Securing Flows](#)
- [Flow Managed Persistence](#)

SWF

- [Getting Started -Hello world](#)
- [Flow Definition](#)
- [Expression Language](#)
- [Rendering Views](#)
- [Executing actions](#)
- [Flow Inheritance](#)

참고자료

- [Spring Web Flow reference 2.0.x](#)
- [openframework swf 소개](#)

개요	▪Spring Web Flow 환경 구성하기 <ul style="list-style-type: none">▪개요▪설정<ul style="list-style-type: none">▪기본적인 설정▪flow-registry 옵션▪커스텀 FlowBuilder 서비스 설정▪flow-executor 옵션▪참고자료
설정	

Spring Web Flow 의 Flow 정의를 위한 XML 문서는 아래와 같은 Schema를 갖는다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:webflow="http://www.springframework.org/schema/webflow-config"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/webflow-config
http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd">

    <!-- Setup Web Flow here -->
</beans>
```

기본적인 설정

Spring Web Flow 를 사용하기 해서는 FlowRegistry 와 FlowExecutor 를 설정해야 한다.

FlowRegistry는 등록될 시나리오에 따라 작성된 flow xml 을 가져오는 역할[1]을 수행한다.

FlowExecutor는 등록된 flow 설정 xml을 실행[2]한다. 차후 Spring MVC 와 결합하여 Web Flow 시스템을 실행되는 부분에 대해 다루겠다.

```
<!-- [1] Flow 설정 파일 등록-->
<webflow:flow-registry id="flowRegistry">
    <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<!-- [2] Flow 실행의 중추 역할을 하는 서비스 제공-->
<webflow:flow-executor id="flowExecutor" />
```

flow-registry 옵션

flow-registry는 아래 보는 것과 같이 설정을 할 수 있다.

```
<!-- [1] 기본은 이름-flow.xml이지만, 직접 지정할 수도 있다. -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />

<!-- [2]id로 식별이 가능하도록 할 수도 있다 -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" id="bookHotel" />

<!-- [3] 메타 정보도 등록할 수 있다. -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml">
    <flow-definition-attributes>
        <attribute name="caption" value="Books a hotel" />
    </flow-definition-attributes>
</webflow:flow-location>

<!-- [4]ANT 패턴을 지정할 수도 있다. -->
<webflow:flow-location-pattern value="/WEB-INF/flows/**/*-flow.xml" />

<!--
    [5] 기본 앞 첨자 경로를 지정해서 위치를 조합해서 사용할 수도 있다.
    Flow 정의 파일은 모듈화를 높이기 위해서 관련 있는 폴더에 각각 위치해 있는게 가장 좋다.
-->
<webflow:flow-registry id="flowRegistry" base-path="/WEB-INF">
    <webflow:flow-location path="/hotels/booking/booking.xml" />
</webflow:flow-registry>

<!-- [6]Flow 상속 구조 구성 가능 -->
<!-- my-system-config.xml -->
<webflow:flow-registry id="flowRegistry" parent="sharedFlowRegistry">
    <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<!-- shared-config.xml -->
<webflow:flow-registry id="sharedFlowRegistry">
    <!-- Global flows shared by several applications -->
</webflow:flow-registry>
```

커스텀 FlowBuilder 서비스 설정

flow-registry에서 flow-builder-services 는 flow를 구축하는데 사용되는 서비스나 설정 등을 커스터마이징 할 수 있다.

지정하지 않는 경우에는 기본 서비스가 사용 된다.

```
<webflow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
  <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<webflow:flow-builder-services id="flowBuilderServices" />
```

구성 가능한 서비스

conversion-service

SWF 시스템에서 사용하는 ConversionService를 커스터마이징. Flow 실행 동안에 필요한 경우 특정 타입을 다른 타입으로 변환해 줌(propertyEditor 성격)

expression-parser

ExpressionParser 커스터마이징하는데 사용. 기본은 Unified EL이 사용되며, 사용하는 영역은 classpath 이다. 다른 ExpressionParser로는 OGNL이 사용 됨.

view-factory-creator

ViewFactoryCreator 를 커스터마이징. 디폴트 ViewFactoryCreator는 JSP, Velocity, Freemaker 등을 화면에 보여주게 해주는 Spring MVC ViewFactories로 되어 있음.

development

Flow 개발 모드 설정. true인 경우, Flow 정의가 변경되면 hot-reloading 적용(message bundles 과 같은 리소스 포함).

별도로 커스터마이징 된경우

```
<webflow:flow-builder-services id="flowBuilderServices"
  conversion-service="conversionService"
  expression-parser="expressionParser"
  view-factory-creator="viewFactoryCreator" />

<bean id="conversionService" class="..." />
<bean id="expressionParser" class="..." />
<bean id="viewFactoryCreator" class="..." />
```

flow-executor 옵션

Flow 실행 리스너 붙이기

Flow 실행의 Lifecycle 에 관련된 리스너(listener) 는 flow-execution-listeners 를 이용하여 등록한다.

```
<flow-execution-listeners>
  <listener ref="securityListener" />
  <listener ref="persistenceListener" />
</flow-execution-listeners>
```

또한 특정 흐름에 대해서만 적용 가능하다.

```
<listener ref="securityListener" criteria="securedFlow1,securedFlow2"/>
```

FlowExecution persistence 조정

```
<flow-execution-repository max-executions="5" max-execution-snapshots="30" />
```

max-executions

사용자 세션 당 생성될 수 있는 Flow 실행 개수 지정

max-execution-snapshots

Flow 실행 당 받을 수 있는 이력 snapshot 개수 지정. snapshot을 사용하지 못하게 하려면, 0으로 지정. 제한이 없게 하려면 -1로 설정.

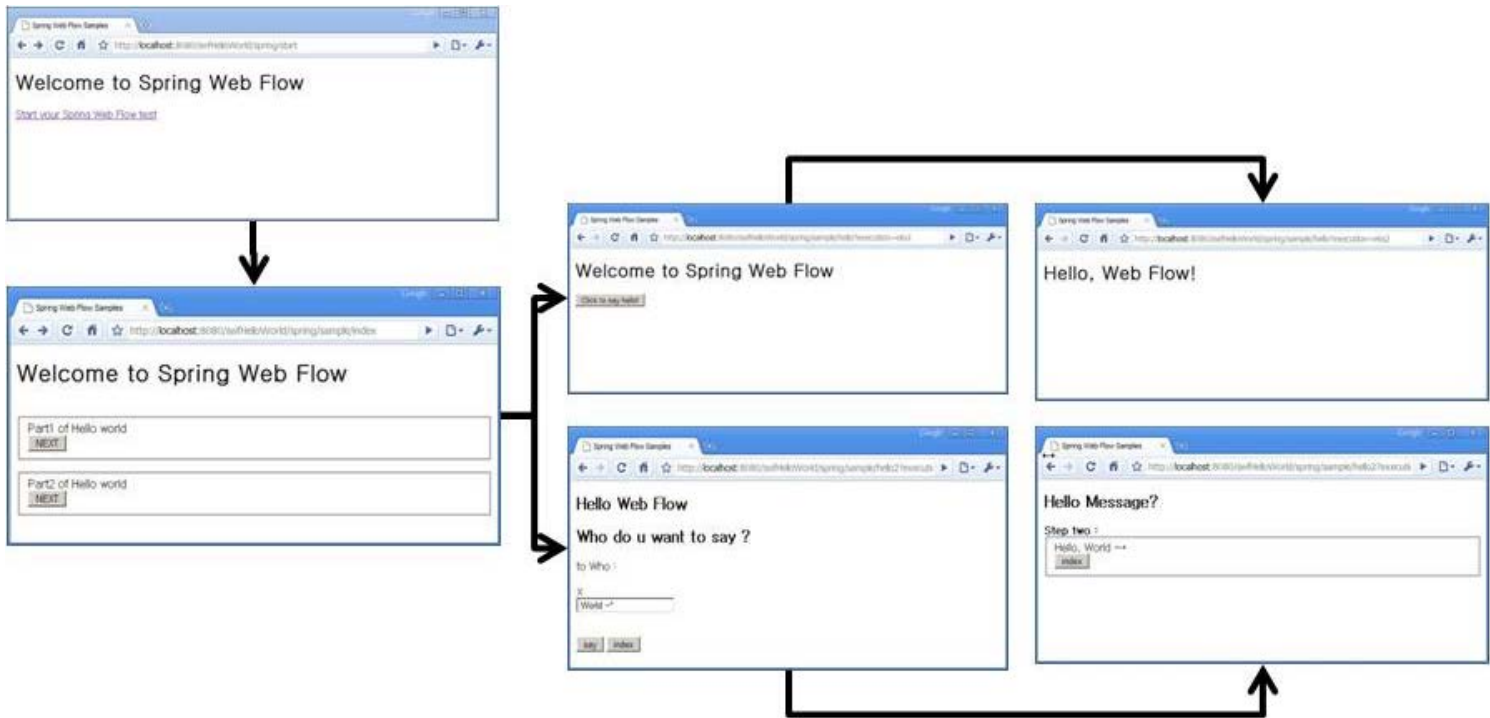
참고자료

- Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

개요

처음으로 접하므로 여기서는 Hello World 를 찍어 보면서 실행하는 것을 살펴 보도록 하겠다.
Hello World 는 두가지 버전으로 입력되는 값이 없이 단지 Hello, Web Flow 화면을 호출 하는 것과 입력값을 가지고 분기처리등 서비스 메소드를 실행후 결과를 화면으로 보여주는 버전으로 나누어 설명하도록 하겠다. 실행하여 보고자 하는 화면결과는 아래와 같다.

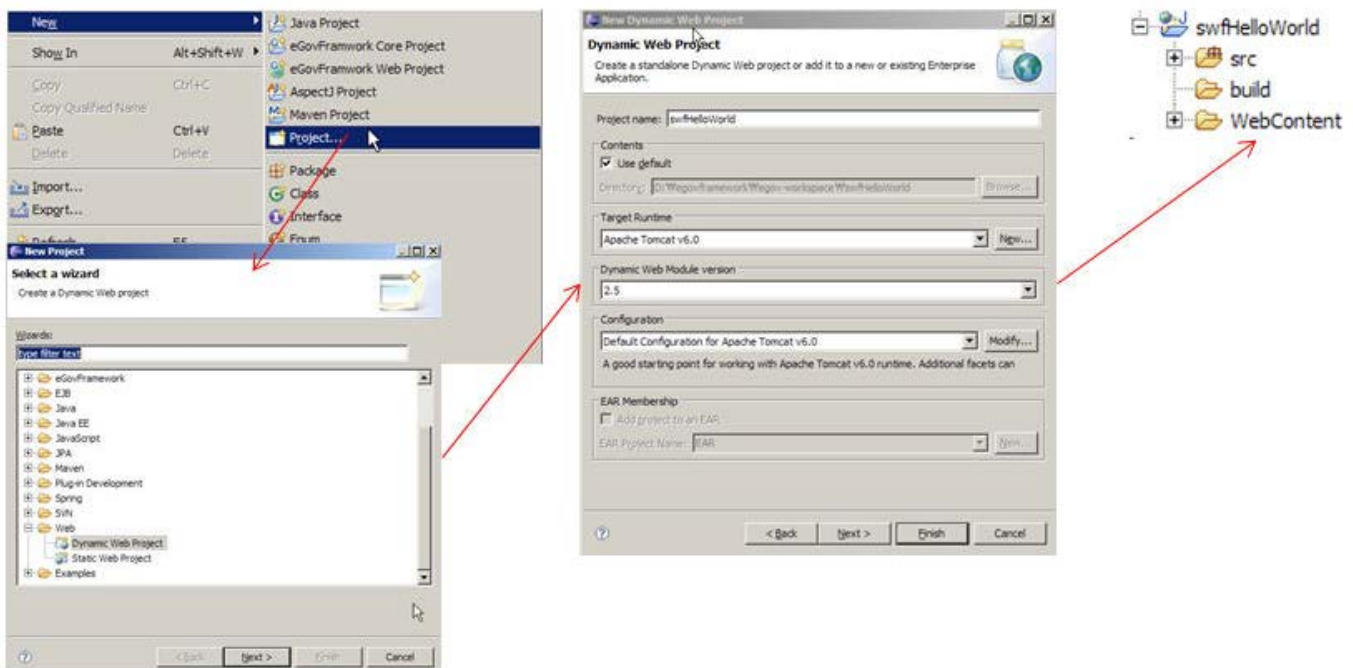
- Hello, World
 - 개요
 - 설명
 - swfHelloWorld 프로젝트 환경 생성
 - web.xml
 - web-application-config.xml
 - Hello, Web Flow
 - Hello, Web Flow with input value



설명

Spring Web Flow 는 사용자와 Service를 제공하는 서버간의 대화하듯한 화면의 이동을 정의 하는 것이다.
SWF(Spring Web Flow)는 사용자와 화면간의 대화형태로 웹 대화형 시나리오를 중심으로 접근한다.

swfHelloWorld 프로젝트 환경 생성



web.xml

webContent/WEB-INF 아래 web.xml을 아래와 같이 작성한다.
contextConfigLocation 의 값으로 /WEB-INF/config/web-application-config.xml 을 설정한다.

servlet 으로 org.springframework.web.servlet.DispatcherServlet 를 등록하고 /spring/* URL 정보를 매핑해준다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
version="2.4">

    <!-- Spring Web 어플리케이션을 위한 메인 설정파일을 등록한다. -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            </param-value>
        </param-value>
    </context-param>

    <!-- Spring Web 어플리케이션 컨테스트를 로딩한다. -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!-- Spring Web 어플리케이션의 맨 앞단 Controller(DispatcherServlet) 를 등록한다. -->
    <servlet>
        <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value></param-value>
        </init-param>
        <load-on-startup>0</load-on-startup>
    </servlet>

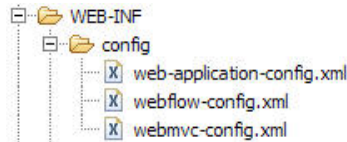
    <!-- 모든 spring 요청에 대한되는 request를 DispatcherServlet 와 매핑하여 처리 할 수 있도록 한다. -->
    <servlet-mapping>
        <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
        <url-pattern>/spring/*</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>

</web-app>
```

web-application-config.xml

Spring MVC 와 Spring Web Flow 를 위한 설정파일은 아래와 같다.



먼저 web-application-config.xml 를 살펴보겠다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-2.5.xsd">

    <!-- 어플리케이션 소스를 스캔하여 로딩 하도록 한다. -->
    <context:component-scan base-package="org.egovframe.swf.sample.service" />

    <!-- 편의를 위하여 Spring MVC 설정과 Spring Web Flow 를 위한 설정을 별도로 분리하여 가져오도록 한다. -->
    <import resource="webmvc-config.xml" />
    <import resource="webflow-config.xml" />

</beans>
```

webmvc-config.xml

Spring MVC 를 위한 설정파일.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <!--
    flowRegistry 에 등록된 flow 와 요청되는 path 와 매핑해주는 역할을 수행한다.
    예제에선 요청되는 ../swfHelloWorld/spring/sample/hello URL 정보를 이용하여 flow 내에서 sample/hello ID로 찾음.
    -->
    <bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
        <property name="order" value="1" />
        <property name="flowRegistry" ref="flowRegistry" />
    </bean>

    <bean
        class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping">
        <property name="order" value="1" />
        <property name="defaultHandler">
            <!-- UrlFilenameViewController 는 spring/start 으로 접근하는 path 정보를 이용하여
            View 이름을 추출하여 View 를 반환하게 된다. 여기서는 tiles 의 view 반환하게 된다. -->
            <bean class="org.springframework.web.servlet.mvc.UrlFilenameViewController" />
        </property>
    </bean>

    <!--
    Controller 에 의해 반환된 View 명을 tiles 로 보내어 tiles 에 미리 정의된 화면을 보여주도록 한다.
    -->
    <bean id="tilesViewResolver" class="org.springframework.js.ajax.AjaxUrlBasedViewResolver">
        <property name="viewClass"
            value="org.springframework.webflow.mvc.view.FlowAjaxTilesView" />
    </bean>

    <!-- tiles 설정 정보를 정의한다. -->
    <bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
        <property name="definitions">
            <list>
                <value>/WEB-INF/layouts/layouts.xml</value>
                <value>/WEB-INF/views.xml</value>
                <value>/WEB-INF/sample/views.xml</value>
            </list>
        </property>
    </bean>
```

```

        </list> <value>/WEB-INF/sample/hello/views.xml</value>
    </property>
</bean>

<!-- Dispatches requests mapped to POJO @Controllers implementations-->
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

<!--
    Dispatches requests mapped to
    org.springframework.web.servlet.mvc.Controller implementations
-->
<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />

<!--
    requests 에 맞는 등록된 FlowHandler 구현부를 연결해준다.
-->
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
    <property name="flowExecutor" ref="flowExecutor" />
</bean>

<!-- Custom FlowHandler for the hello flow-->
<bean name="sample/hello" class="org.egovframe.web.HelloFlowHandler" />

</beans>

```

webflow-config.xml

Web Flow 관련된 설정 파일.

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:webflow="http://www.springframework.org/schema/webflow-config"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/webflow-config
        http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd">

    <webflow:flow-executor id="flowExecutor" />

    <!-- flow 를 정의한 파일을 가져와 flow registry 구성한다. -->
    <webflow:flow-registry id="flowRegistry"
        flow-builder-services="flowBuilderServices" base-path="/WEB-INF">
        <webflow:flow-location-pattern value="/**/*-flow.xml" />
    </webflow:flow-registry>

    <!-- Web Flow views 에 커스터마이징 할 수 있도록 확장하여 사용한다. -->
    <webflow:flow-builder-services id="flowBuilderServices"
        view-factory-creator="mvcViewFactoryCreator" conversion-service="conversionService"
        development="true" />

    <!-- Web Flow 에서 tiles 를 사용할 수 있도록 설정한다. -->
    <bean id="mvcViewFactoryCreator"
        class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
        <property name="viewResolvers" ref="tilesViewResolver" />
    </bean>

</beans>

```

상세 : [Web Flow views 에 커스터마이징 할 수 있도록 확장하여 사용한다.](#)

tiles

URL : <http://localhost:8080/swfHelloWorld>으로 처음 접근할 때 index.html 파일이 열리게 된다.

```

<html>
<head>
    <meta http-equiv="Refresh" content="0; URL=spring/start">
</head>
</html>

```

위에서 보는 것처럼 "spring/start" URL 을 호출한다.
spring/start 에 해당하는 화면은 먼저 설정된 tiles 설정정보에서 찾게 된다.

```

<tiles-definitions>
    <definition name="start" extends="standardLayout">
        <put-attribute name="body" value="/WEB-INF/main.jsp" />
    </definition>
</tiles-definitions>

```

tiles 관련된 것은 <http://tiles.apache.org/> 를 참조하시길 바랍니다.
등록된 tiles 설정파일은 앞 설정에서 나왔다. 다시 보면 ..

```

...
<!-- tiles 설정 정보를 정의한다. -->
<bean id="tilesConfigurer" class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
    <property name="definitions">
        <list>
            <value>/WEB-INF/layouts/layouts.xml</value>
            <value>/WEB-INF/views.xml</value>
            <value>/WEB-INF/sample/views.xml</value>
            <value>/WEB-INF/sample/hello/views.xml</value>
        </list>
    </property>
</bean>
...

```

Hello, Web Flow

다시 돌아와서 Hello , Web Flow 를 화면에 찍어 보도록 하겠다.
Web Flow 로 해당 화면의 흐름을 작성한 예를 보자.

hello-flow.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/webflow
        http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

```

```

<view-state id="hello">
  <transition on="say" to="helloworld" />
</view-state>

<view-state id="helloworld">
  <transition on="return" to="return" />
</view-state>

<end-state id="return"          view="externalRedirect:servletRelative:/start" />

</flow>

```

자세한 설명은 [flow 정의](#) 에서 다루고 있다.

간단하게 보면 .

view-state , end-state 로 나뉘져 있는 것을 볼 수 있다. 처음으로 존재하는 view-state 는 시작점이라고 생각해도 무방하다. 또한 문자 그대로 end-state 는 마지막점이다. hello 라는 화면이 맨처음 나오고 거기서 helloworld 화면이 보이고 다음은 return 이라는 마지막실행을 하는 것이다. view-state 안쪽의 transition 는 화면에서 클릭하여 이동하게 하는 버튼의 실행이라고 할 수 있다. 여기선 say 를 눌러서 실행하면 helloworld 라는 view-state 로 이동하는 것이다. 마찬가지로. return 을 누르면 externalRedirect:servletRelative:/start 으로 이동하는 것이다.

- 참조 : [externalRedirect](#) , [servletRelative](#)

view-state 에서 별도의 view 를 정의하지 않은 경우 id 를 가지고 view 를 가져오게 된다. 여기서는 hello 라는 id 가 곧 view 명이 되게 된다. default는 flow.xml 과 같은 디렉토리에 있는 화면소스(JSP, xhtml, 등)을 찾게 된다. 여기선 tiles 로 정의된 부분을 참조한다. .../hello/views.xml 파일 내용을 살펴보면,

```

...
<definition name="hello" extends="standardLayout">
  <put-attribute name="body" value="/WEB-INF/sample/hello/hello.jsp" />
</definition>
...

```

로 화면에 해당되는 hello.jsp 를 가져오는 것을 확인 할 수 있다.

그렇다면 transition 은 화면에서 발생한 이벤트와 매핑을 할까? hello.jsp 소스를 잠시 보겠다.

```

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Welcome to Spring Web Flow</title>
</head>
<body>
<h1>Welcome to Spring Web Flow</h1>
<form:form id="start">
  <input type="submit" name="_eventId_say" value="Click to say hello!" />
</form:form>
</body>
</html>

```



보는 봐와 같이 form으로 둘러싸인 곳에 해답은 있다. <input type="submit" name="_eventId_say" /> 에서 name 을 보면 **_eventId_say** 로 답을 찾을 수 있다.

_eventId 가 답이다. say는 transition 의 on 과 같음을 확인 할 수 있다. eventId 에 정의된 특정위치의 문자열을 가지고 transition 를 분석하다. transition 에 대한 내용은 [flow 정의](#)에서 자세히 살펴볼길 바란다. eventId 가 "say"를 가지고 form 이 전달되면 flow 정의에 따라 transition 을 찾고 그에 맞는 state 로 넘어가게 된다.

결과는 별도의 값을 가지고 보여주는 화면은 아니고 단지 아래와 같은 화면을 보여주도록 되어 있다.



다음은 입력값이 있는 예를 살펴 보도록 하겠다.

Hello, Web Flow with input value

먼저 flow 정의 파일인 hello2-flow.xml 을 보도록 하자.

실행 시나리오는 on-start ⇒ view-state ⇒ action-state ⇒ decision-stat ⇒ end-state 이다. **hello2-flow.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

  <on-start>
    <evaluate expression="helloService.sayMessage()" result="flowScope.message" />
  </on-start>

  <view-state id="hello2" model="message">
    <binder>
      <binding property="str" required="true" />
    </binder>
    <transition on="proceed" to="actionHello" />
    <transition on="return" to="return" />
  </view-state>

  <action-state id="actionHello">
    <evaluate expression="helloService.addHello(message)" />
    <transition on="yes" to="moreDecision" />
  </action-state>

```

```

        <transition on="no" to="hello" />
    </action-state>

    <decision-state id="moreDecision">
        <if test="helloService.getDecision(message)" then="helloworld2" else="return" />
    </decision-state>

    <view-state id="helloworld2">
        <transition on="return" to="return" />
    </view-state>

    <end-state id="return"          view="externalRedirect:servletRelative:/start" />
</flow>
</xml>

```

보여주고자하는 것은 hello2 화면(view-state) 에서 입력데이터를 객체에 바인딩하고, helloService 서비스 객체를 통해 addHello 메소드 실행, 그후 결과에 따라 분기문(decision-state) 를 통과하여 helloworld2 화면으로 가는 것이다. 간략하게 설명드리면, on-start 는 flow 를 처음 실행할 때 선행하여 실행된다. 여기서는 helloService의 sayMessage 를 실행하여 flowScope 내의 message 객체로 저장한다.

HelloService.java

```

...
@Service("helloService")
public class HelloService implements IService {

    public Message sayMessage() {
        return new Message();
    }
    ...
}

```

flow가 시작할 때 첫번째로 만나는 view-state 는 시작점으로 인식한다. 따라서 view-state "hello2" 는 시작점에 해당한다. hello2.jsp 를 화면에 보여주는데 앞단의 예제와 같다. Spring MVC 의 tiles 를 이용하여 보여주게 된다.

views.xml

```

...
<definition name="hello2" extends="standardLayout">
    <put-attribute name="body" value="inhello2.body" />
</definition>

<definition name="inhello2.body" template="/WEB-INF/sample/hello2/main.jsp">
    <put-attribute name="helloSection" value="/WEB-INF/sample/hello2/hello.jsp" />
</definition>
...

```

hello.jsp

```

<form:form method="post" >
    <p>
        to Who : <input type="text" id="str" name="str" value=" World ~*"/>
        <script type="text/javascript">
            Spring.addDecoration(new Spring.ElementDecoration({
                elementId : "str",
                widgetType : "diigit.form.ValidationTextBox",
                widgetAttrs : { promptMessage : "for who ? ", required : true }}});
        </script>
    </p>

    <input id="proceed" type="submit" class="button"
        name="_eventId_proceed" value="say">
    <script type="text/javascript">
        Spring.addDecoration(new Spring.ValidateAllDecoration({elementId:'proceed', event:'onclick'}));
    </script>
    <input type="submit" class="button"
        name="_eventId_return" value="index">
</form:form>
...

```

상단의 화면은 아래 hello2-flow.xml 내의 view-state 와 매핑된다.

여기서 봐야 할 부분은 화면내의 str 이름의 input 데이터를 message 라는 객체로 바인딩하는 부분이다.

```

...
<view-state id="hello2" model="message">
    <binder>
        <binding property="str" required="true" />
    </binder>

    <transition on="proceed" to="actionHello" />
    <transition on="return" to="return" />
</view-state>
...

```

이벤트에 해당하는 proceed 버튼을 클릭하면 다음 state 로 이동하게된다.

```

...
<transition on="proceed" to="actionHello" />
...

```

actionHello 은 아래와 같다. 하는 기능은 helloService 객체의 addHello 메소드 호출이다.

```

...
<action-state id="actionHello">
    <evaluate expression="helloService.addHello(message)" />
    <transition on="yes" to="moreDecision" />
    <transition on="no" to="hello" />
</action-state>
...

```

addHello 메소드는 아래와 같다. 반환되는 값이 boolean 인 것을 주목할 필요가 있다. 리턴되는 boolean 값은 transition 의 yes, no 와 매핑된다.

```

...
public boolean addHello(Message msg){
    try{
        msg.setStr("Hello,"+msg.getStr());
    }catch (Exception e){
        return false;
    }
    return true;
}
...

```

다음 나오는 decision-state는 아래와 같이 분기문의 기능을 수행한다.

```
...
<decision-state id="moreDecision">
  <if test="helloService.getDecision(message)" then="helloworld2" else="return" />
</decision-state>
...
```

helloworld2 화면으로 이동하게 되면 아래와 같은 jsp 소스를 확인할 수 있다. message 객체의 str 값을 EL 을 이용하요 \${message.str} 으로 보여주고 있다.

```
<%@ taglib prefix="form"
    uri="http://www.springframework.org/tags/form" %>
<h2>Hello Message?</h2>
<form:form>
  <b>Step two :</b>
  <fieldset>
    <div class="field">
      <div class="label">
        <label>${message.str}</label>
      </div>
    </div>
    <div class="buttonGroup">
      <input type="submit" class="button" name="_eventId_return" value="index">
    </div>
  </fieldset>
</form:form>
```

화면을 다시 보면



say 버튼을 누르면,



Hello , 뒤에 넣었던 문장이 붙어서 나오게 된다.

개요

Spring Web Flow 이용하여 WEB 을 개발하려할 때 Spring MVC 와 연동하여 개발을 하려 한다. 따라서 Spring MVC 연동하는 모듈등을 설정하여야 한다.
여기서는 booking-mvc sample([실행데모](#)(faces이지만 시나리오는 같음)) 을 기준으로 설정하도록 하겠다.

- Spring Web Flow 와 MVC 연동
 - 개요
 - 설명
 - web.xml 환경 구성
 - Flow로 전달
 - 커스텀 FlowHandler 구현
 - 뷰 결정
 - 뷰에서 이벤트 보내기
 - 참고자료

설명

Spring MVC 와의 연동을 위해 우리는 web.xml 안에 있는 DispatcherServlet 설정을 보도록 하겠다.

web.xml 환경 구성

Spring MVC를 구성하는 첫 단계는 web.xml에 DispatcherServlet을 구성하는 것이다.
DispatcherServlet은 웹어플리케이션별 하나를 등록한다.

이 예제에서는 /spring/으로 시작하는 모든 요청을 받도록 설정하고 있다. init-param을 사용해서 contextConfigLocation 을 설정하고 있다.

web.xml

```
<servlet>
  <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/web-application-config.xml</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>Spring MVC Dispatcher Servlet</servlet-name>
  <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Flow로 전달

DispatcherServlet은 애플리케이션 자원에 대한 요청과 핸들러를 매핑시켜 준다. Flow도 핸들러의 하나의 유형으로 처리 된다.

FlowHandlerAdapter 등록

먼저 FlowHandlerAdapter Bean 정의하고 나서 property(flowExecutor) 로 flowExecutor 빈을 설정함으로써 Spring MVC 내에서 Flow를 제어할 수 있도록 한다.

```
<!-- Enables FlowHandler URL mapping -->
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
  <property name="flowExecutor" ref="flowExecutor" />
</bean>
```

flow 매핑 정의

다음 단계로 애플리케이션 자원을 특정 Flow와 매핑시키는 것이다. 가장 간단한 방법은 FlowHandlerMapping을 정의 하는 방법이다.

```
<!-- Maps request paths to flows in the flowRegistry; e.g. a path of /hotels/booking looks for a flow with id "hotels/booking" -->
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
  <property name="flowRegistry" ref="flowRegistry" />
</bean>
```

이 설정은 Dispatcher가 애플리케이션 자원 경로를 flow registry에 등록된 Flow로 매핑시킬 수 있도록 해준다.
예를 들어, /hotels/booking 요청은 hotels/booking이란 Flow ID를 갖는 Flow에게 요청이 가게 된다.
flow registry에서 해당 ID 못 찾게 되면, Dispatcher의 순서에 따라 다음 핸들러 매핑에서 찾고, 없다면 "noHandlerFound"를 반환되게 된다.

작업 흐름을 제어하는 Flow

조건이 맞는 flow 가 매핑되면 FlowHandlerAdapter는 새로운 flow 실행을 시작할 것인지 아니면 HTTP 요청에 담겨있는 정보를 기반으로 기존 실행을 계속할 것인지를 판단하게 된다.

개요

Spring Web Flow 를 사용하기 위한 Web 개발환경에 대한 세팅을 설명하겠다.

설정

- Spring Web Flow 환경 구성하기
 - 개요
 - 설정
 - 기본적인 설정
 - flow-registry 옵션
 - 커스텀 FlowBuilder 서비스 설정
 - flow-executor 옵션
 - 참고자료

Spring Web Flow 의 Flow 정의를 위한 XML 문서는 아래와 같은 Schema를 갖는다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:webflow="http://www.springframework.org/schema/webflow-config"
        xsi:schemaLocation="
            http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
            http://www.springframework.org/schema/webflow-config
            http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd" >

    <!-- Setup Web Flow here -->
</beans>
```

기본적인 설정

Spring Web Flow 를 사용하기 해서는 FlowRegistry 와 FlowExecutor 를 설정해야 한다.

FlowRegistry는 등록될 시나리오에 따라 작성된 flow xml 을 가져오는 역할[1]을 수행한다.

FlowExecutor는 등록된 flow 설정 xml을 실행[2]한다. 차후 Spring MVC 와 결합하여 Web Flow 시스템을 실행되는 부분에 대해 다루겠다.

```
<!-- [1] Flow 설정 파일 등록-->
<webflow:flow-registry id="flowRegistry">
    <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<!-- [2] Flow 실행의 중추 역할을 하는 서비스 제공-->
<webflow:flow-executor id="flowExecutor" />
```

flow-registry 옵션

flow-registry는 아래 보는 것과 같이 설정을 할 수 있다.

```
<!-- [1]기본은 이름-flow.xml이지만, 직접 지정할 수도 있다. -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />

<!-- [2]id로 식별이 가능하도록 할 수도 있다 -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" id="bookHotel" />

<!-- [3]메타 정보도 등록할 수 있다. -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml">
    <flow-definition-attributes>
        <attribute name="caption" value="Books a hotel" />
    </flow-definition-attributes>
</webflow:flow-location>

<!-- [4]ANT 패턴을 지정할 수도 있다. -->
<webflow:flow-location-pattern value="/WEB-INF/flows/**/*-flow.xml" />

<!--
    [5]기본 앞 첨자 경로를 지정해서 위치를 조합해서 사용할 수도 있다.
    Flow 정의 파일은 모듈화를 높이기 위해서 관련 있는 폴더에 각각 위치해 있는게 가장 좋다.
-->
<webflow:flow-registry id="flowRegistry" base-path="/WEB-INF">
    <webflow:flow-location path="/hotels/booking/booking.xml" />
</webflow:flow-registry>

<!-- [6]Flow 상속 구조 구성 가능 -->
<!-- my-system-config.xml -->
<webflow:flow-registry id="flowRegistry" parent="sharedFlowRegistry">
    <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<!-- shared-config.xml -->
<webflow:flow-registry id="sharedFlowRegistry">
    <!-- Global flows shared by several applications -->
</webflow:flow-registry>
```

커스텀 FlowBuilder 서비스 설정

flow-registry에서 flow-builder-services 는 flow를 구축하는데 사용되는 서비스나 설정 등을 커스터마이징 할 수 있다. 지정하지 않는 경우에는 기본 서비스가 사용 된다.

```
<webflow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
  <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<webflow:flow-builder-services id="flowBuilderServices" />
```

구성 가능한 서비스

conversion-service

SWF 시스템에서 사용하는 ConversionService를 커스터마이징. Flow 실행 동안에 필요한 경우 특정 타입을 다른 타입으로 변환해 줌(propertyEditor 성격)

expression-parser

ExpressionParser 커스터마이징하는데 사용. 기본은 Unified EL이 사용되며, 사용하는 영역은 classpath 이다. 다른 ExpressionParser로는 OGNL이 사용 됨.

view-factory-creator

ViewFactoryCreator 를 커스터마이징. 디폴트 ViewFactoryCreator 는 JSP, Velocity, Freemake 등을 화면에 보여주게 해주는 Spring MVC ViewFactories로 되어 있음.

development

Flow 개발 모드 설정. true인 경우, Flow 정의가 변경되면 hot-reloading 적용(message bundles 과 같은 리소스 포함).

별도로 커스터마이징 된경우

```
<webflow:flow-builder-services id="flowBuilderServices"
  conversion-service="conversionService"
  expression-parser="expressionParser"
  view-factory-creator="viewFactoryCreator" />

<bean id="conversionService" class="..." />
<bean id="expressionParser" class="..." />
<bean id="viewFactoryCreator" class="..." />
```

flow-executor 옵션

Flow 실행 리스너 붙이기

Flow 실행의 Lifecycle 에 관련된 리스너(listener) 는 flow-execution-listeners 를 이용하여 등록한다.

```
<flow-execution-listeners>
  <listener ref="securityListener" />
  <listener ref="persistenceListener" />
</flow-execution-listeners>
```

또한 특정 흐름에 대해서만 적용 가능하다.

```
<listener ref="securityListener" criteria="securedFlow1,securedFlow2" />
```

FlowExecution persistence 조정

```
<flow-execution-repository max-executions="5" max-execution-snapshots="30" />
```

max-executions

사용자 세션 당 생성될 수 있는 Flow 실행 개수 지정

max-execution-snapshots

Flow 실행 당 받을 수 있는 이력 snapshot 개수 지정. snapshot을 사용하지 못하게 하려면, 0으로 지정. 제한이 없게 하려면 -1로 설정.

참고자료

- Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)


```
<webflow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
  <webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />
</webflow:flow-registry>

<webflow:flow-builder-services id="flowBuilderServices" />
```

구성 가능한 서비스

conversion-service

SWF 시스템에서 사용하는 ConversionService를 커스터마이징. Flow 실행 동안에 필요한 경우 특정 타입을 다른 타입으로 변환해 줌(propertyEditor 성격)

expression-parser

ExpressionParser 커스터마이징하는데 사용. 기본은 Unified EL이 사용되며, 사용하는 영역은 classpath 이다. 다른 ExpressionParser로는 OGNL이 사용 됨.

view-factory-creator

ViewFactoryCreator 를 커스터마이징. 디폴트 ViewFactoryCreator 는 JSP, Velocity, Freemake 등을 화면에 보여주게 해주는 Spring MVC ViewFactories로 되어 있음.

development

Flow 개발 모드 설정. true인 경우, Flow 정의가 변경되면 hot-reloading 적용(message bundles 과 같은 리소스 포함).

별도로 커스터마이징 된경우

```
<webflow:flow-builder-services id="flowBuilderServices"
  conversion-service="conversionService"
  expression-parser="expressionParser"
  view-factory-creator="viewFactoryCreator" />

<bean id="conversionService" class="..." />
<bean id="expressionParser" class="..." />
<bean id="viewFactoryCreator" class="..." />
```

flow-executor 옵션

Flow 실행 리스너 붙이기

Flow 실행의 Lifecycle 에 관련된 리스너(listener) 는 flow-execution-listeners 를 이용하여 등록한다.

```
<flow-execution-listeners>
  <listener ref="securityListener" />
  <listener ref="persistenceListener" />
</flow-execution-listeners>
```

또한 특정 흐름에 대해서만 적용 가능하다.

```
<listener ref="securityListener" criteria="securedFlow1,securedFlow2" />
```

FlowExecution persistence 조정

```
<flow-execution-repository max-executions="5" max-execution-snapshots="30" />
```

max-executions

사용자 세션 당 생성될 수 있는 Flow 실행 개수 지정

max-execution-snapshots

Flow 실행 당 받을 수 있는 이력 snapshot 개수 지정. snapshot을 사용하지 못하게 하려면, 0으로 지정. 제한이 없게 하려면 -1로 설정.

참고자료

- Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

- HTTP 요청 파라미터는 모든 Flow 실행의 입력 맵에서 사용할 수 있다.
- Flow 실행이 마지막 응답 전송없이 끝나는 경우, Default 핸들러가 동일한 요청을 새로운 Flow 실행을 요청하게 된다.
- 발생한 Exception이 NoSuchFlowExecutionException인 경우에는 새 Flow 실행을 시작해 복구 시도를 해보 다른 예외는 제어하지 않는다.

자세한 내용은 SWF JavaDoc 에서 FlowHandlerAdapter 클래스를 참고하길 바란다.

커스텀 FlowHandler 구현

FlowHandler는 HTTP 서블릿 환경에서 Flow가 실행에 대한 커스터마이징할 수 있는 확장 영역이다. FlowHandlerAdapter에 의해 실행/사용되며, 아래와 같은 수행을 한다.

- 실행되는 Flow의 id 반환
- Flow 실행시 입력될 값 생성
- Flow 실행이 종료되면서 반환하는 결과 처리
- Flow 실행에서 발생해서 던져진 예외 처리

이러한 수행을 위한 메소드는 org.springframework.mvc.servlet.FlowHandler 인터페이스 형태로 되어 있다.

```
public interface FlowHandler {
    public String getFlowId();
    public MutableAttributeMap createExecutionInputMap (HttpServletRequest request);
    public String handleExecutionOutcome (FlowExecutionOutcome outcome, HttpServletRequest request,
    HttpServletResponse response);
    public String handleException (FlowException e, HttpServletRequest request, HttpServletResponse
    response);
}
```

FlowHandler를 구현시, AbstractFlowHandler를 상속하면 된다. 모든 연산은 선택적이 되어서, 필요할 때만 구현하면 되며, 구현하지 않으면 기본 구현 내용(AbstractFlowHandler 내에 구현된)이 적용된다. 특히 다음과 같은 경우에는 구현을 생각해 볼 수 있다.

- getFlowId(HttpServletRequest) 재정의: HTTP 요청에서 직접적으로 Flow id를 받을 수 없을 때. 일반적으로는 요청 URI에서 경로 정보를 얻게 됨. 예를 들어, <http://localhost/app/hotels/booking?hotelId=1>는 hotels/booking이란 Flow id로 매핑 됨
- createExecutionInputMap(HttpServletRequest) 재정의: HttpServletRequest에서 Flow 입력 파라미터를 세부적으로 직접 추출해야 하는 경우. 기본적으로는 모든 요청 파라미터가 Flow 입력 파라미터로 넘겨짐.
- handleExecutionOutcome 재정의: 직접 Flow 실행 결과를 제어할 필요가 있을 경우. 기본 행동은 Flow의 새로운 실행을 재시작하려고 마지막 Flow의 URL로 redirect 보냄
- handleException 재정의: 제어되지 못한 Flow 실행을 세심하게 조정할 필요가 있는 경우. 기본적으로는 제어하지 않은 Exception 은 Spring MVC ExceptionResolver 로 다시 보내진다.

FlowHandler 예제

가장 일반적인 스프링 MVC와의 상호 작용은, Flow가 종료 됐을 때 @Controller로 재전송 하는 방법이다. FlowHandler는 특정 controller URL을 Flow 정의와 관련없이 가능하도록 해준다. 예를 보자.

```
public class BookingFlowHandler extends AbstractFlowHandler {
    public String handleExecutionOutcome (FlowExecutionOutcome outcome, HttpServletRequest request,
    HttpServletResponse response) {
        if (outcome.getId().equals("bookingConfirmed")) {
            return "/booking/show?bookingId=" + outcome.getOutput().get("bookingId");
        } else {
            return "/hotels/index";
        }
    }
}
```

재정의된 handleExecutionOutcome 메소드에서는 Flow 결과로 나온 flow id 가 bookingConfirmed 인 경우 특정 URL(/booking/show?bookingId=...) 로 보낸다. flow id 가 bookingConfirmed 와 다른 경우 /hotels/index 에 대한되는 URL 로 가게 된다.

커스텀 FlowHandler 배포

커스텀 FlowHandler를 설치하려면, 그냥 빈으로 등록하기만 하면 된다. 빈 이름은 반드시 적용하고자 하는 Flow의 id와 일치해야 한다.

```
<bean name="/hotels/booking" class="org.springframework.webflow.samples.booking.BookingFlowHandler"/>
```

이 설정을 통해서 /hotels/booking 자원에 대한 접근은 커스텀 핸들러인 BookingFlowHandler를 사용해서 hotels/booking이 실행되게 된다.
booking flow 가 끝나는 시점에서 BookingFlowHandler의 handleExecutionOutcome이 실행되며 String(URL) 결과에 따라 적당한 controller 로 재전송된다.

FlowHandler 재전송

FlowExecutionOutcome이나 FlowException을 제어하는 FlowHandler는 제어를 한 후에 재전송하는 경로를 지정하는 String을 반환하게 된다. 이전 예에서 BookingFlowHandler는 bookingConfirmed 결과에 대해서는 booking/show로 재전송하고, 다른 결과에 대해서는 hotels/index 자원 URI로 반환하게 된다. 기본적으로 반환되는 자원의 위치는 현재 서블릿 매핑에 관계된다. 이는 flow handler가 상대 경로를 사용해서 애플리케이션 내에 있는 다른 컨트롤러로 redirect 할 수 있게 해준다. 여기에 더해서 좀더 제어가 필요한 경우에 사용할 수 있는 명시적인 앞첨자를 제공한다.

- servletRelative: 현재 서블릿에 대한 상대적인 자원으로 재전송
- contextRelative: 현재 웹 애플리케이션 컨텍스트 경로(web application context path)에 대한 상대적인 자원으로 재전송
- servletRelative: 서블릿 루트에 대한 상대적인 자원으로 재전송
- http: 또는 https: : 완전한 자원 URI로 재전송

이 앞첨자는 externalRedirect: 지시어와 함께 Flow 정의 내에서도 사용할 수 있다.
예를 들면, view="externalRedirect: http://springframework.org".

뷰 결정

Web Flow 2는 따로 지정하지 않는다면 Flow 파일이 있는 디렉토리에 있는 파일과 선택된 뷰 식별자를 매핑해주게 된다. 기존 스프링 MVC+Web Flow 애플리케이션에서는 이미 외부 ViewResolver가 매핑 처리를 해주고 있다. 그러므로 기존 resolver를 계속 사용하고, 기존 Flow 뷰가 패키징된 방법이 변경되는 것을 피하기 위해서 다음처럼 설정을 하도록 하자.

```
<webflow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
  <webflow:location path="/WEB-INF/hotels/booking/booking.xml" />
</webflow:flow-registry>

<webflow:flow-builder-services id="flowBuilderServices" view-factorycreator="mvcViewFactoryCreator" />

<bean id="mvcViewFactoryCreator" class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
  <property name="viewResolvers" ref="myExistingViewResolverToUseForFlows" />
</bean>
```

MvcViewFactoryCreator는 당신이 Spring MVC view 시스템(여기선 "myExistingViewResolverToUseForFlows")을 Spring Web Flow 내에서 사용할 수 있도록 해준다.
Booking Hotels 샘플에서는 아래와 같이 설정되어 있다.(tilesViewResolver 를 이용할 수 있도록 되어 있다.)

```
<bean id="mvcViewFactoryCreator" class="org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
  <property name="viewResolvers" ref="tilesViewResolver" />
</bean>
```

또한 useSpringBinding 의 값을 true 하는 경우 Spring MVC 의 고유한 BeanWrapper 를 이용하여 데이터 바인딩을 할 수 있다.

뷰에서 이벤트 보내기

flow가 view-state에 들어가서 잠시 멈추게 되면, 사용자를 해당 실행 URL로 재전송해서, 사용자 이벤트가 다시 시작되기를 기다리게 된다. 이번 절에서는 JSP, Velocity나 Freemarker처럼 템플릿 엔진에 의해서 생성된 HTML 기반 뷰에서 이벤트를 발생시키는 방법을 알아보자.

HTML 버튼을 사용해서 이벤트 보내기

다음은 proceed와 cacle 이벤트를 발생시키는 같은 폼 내의 두 개 버튼을 보여준다.

```
<input type="submit" name="_eventId_proceed" value="Proceed" />
<input type="submit" name="_eventId_cancel" value="Cancel" />
```

버튼이 선택되면, SWF는 _eventId로 시작하는 요청 파라미터를 찾아서, 그 부분을 잘라내고 남은 문자열을 id로 사용하게 된다.

_eventId_proceed는 proceed가 된다. 그러기 때문에 동일한 폼에서 다양한 여러 이벤트를 발생시킬 수 있다.

hidden HTML 폼 파라미터 사용해 이벤트 신호 보내기

form이 submit될 때 proceed 이벤트가 발생하려면 다음처럼 하자.

```
<input type="submit" value="Proceed" />
<input type="hidden" name="_eventId" value="proceed" />
```


여기서는 _eventId 파라미터로 오는 값을 찾아서 event id로 해당 값을 사용하게 된다.
이런 방법은 form으로 전송할 수 있는 이벤트가 하나일 때만 생각해봐야 한다.

HTML link 사용해서 이벤트 보내기

```
<a href="${flowExecutionUrl}&_eventId=cancel">Cancel</a>
```

매개변수 식별 순서는 “eventId” ⇒ “_eventId” ⇒ 없음 이다.

참고자료

-  Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

개요

보안은 어플리케이션 에서 매우 중요한 이슈이다.
Spring Security 는 어플리케이션과 결합하여 여러 수준에서 보안을 책임지는 플랫폼의 기능을 수행한다.
여기서는 Web Flow 에 적용되는 Spring Security 에 대해 알아보도록 하겠다.

설명

- Flow에 보안 적용하기(Securing Flows)
- 개요
- 설명
 - 어떻게 Flow를 안전하게 할 수 있을까?
 - secured 구성요소
 - 속성에 보안 적용
 - 타입 맞춰보기
 - SecurityFlowExecutionListener
 - Spring Security 환경 구성
 - 스프링 환경 구성
 - web.xml 환경 구성
 - 참고자료

어떻게 Flow를 안전하게 할 수 있을까?

Flow 실행에 보안을 적용시키고 싶다면 다음 단계에 따르자.

1. Spring Security에서 인증(authentication)과 권한(authorization) 규칙을 설정한다.
2. secured 구성요소로 Flow 정의에 보안 규칙을 추가한다.
3. 보안 규칙을 처리해주는 SecurityFlowExecutionListener 추가한다.

secured 구성요소

secured 구성요소는 접근 하기 전에 권한 확인을 적용해주며, Flow 실행 단계마다 한 번 이상은 나올 수 없다.
Flow 실행에서 세단계로 보안을 적용할 수 있다. Flow, state, transition 에 보안 적용이 가능하다.
사용되는 문법은 동일하다. secured 구성요소는 보안이 적용되어야 하는 구성요소 내에 위치하면 된다.
예를 들어 view state에 보안을 적용하고자 하면,

```
<view-state id="secured-view">
  <secured attributes="ROLE_USER" />
  ...
</view-state>
```

속성에 보안 적용

attributes 속성은 ','(콤마)로 구분해서 SS의 권한 속성을 리스트로 입력할 수 있다. 이 속성은 대부분 허가된 보안 롤(role)을 명시하게 된다.
스프링 시큐리티 접근 결정 매니저(access decision manager)에 의해 이 속성에 입력한 값과 사용자가 가지고 있는 값을 비교한다.

```
<secured attributes="ROLE_USER" />
```

기본적으로, 롤 기반 접근 결정 관리자를 사용하여 사용자가 접근할 수 있는지 확인한다.
만약 애플리케이션이 권한 롤을 사용하지 않는다면 이 부분을 오버라이딩할 필요가 있다.

타입 맞춰보기

두 가지 유형의 일치 유형 제공: any, all.

```
<secured attributes="ROLE_USER, ROLE_ANONYMOUS" match="any" />
```

이 속성은 필수가 아니다. 정의하지 않으면 기본 값은 any다.

SecurityFlowExecutionListener

Web Flow 설정에 추가한다.
SecurityFlowExecutionListener가 Web Flow 설정에 정의되어 있어야 플로우 실행기(executor)에 적용된다.

```
<webflow:flow-executor id="flowExecutor"
  flow-registry="flowRegistry">
  <webflow:flow-execution-listeners>
    <webflow:listener ref="securityFlowExecutionListener" />
  </webflow:flow-execution-listeners>
</webflow:flow-executor>
<bean id="securityFlowExecutionListener"
  class="org.springframework.webflow.security.SecurityFlowExecutionListener" />
```

보안 설정에 의해서 접근이 거절되면, `AccessDeniedException` 발생한다.
기본으로 룰 기반 의사결정이 이루어 지지만, 커스텀 의사결정 관리자 지정 가능하다.

```
<bean id="securityFlowExecutionListener"
class="org.springframework.webflow.security.SecurityFlowExecutionListener">
  <property name="accessDecisionManager" ref="myCustomAccessDecisionManager" />
</bean>
```

Spring Security 환경 구성

스프링 환경 구성

http와 authentication-provider로 정의하면 된다.

```
<security:http auto-config="true">
  <security:form-login login-page="/spring/login"
    login-processing-url="/spring/loginProcess" default-target-url="/spring/main"
    authentication-failure-url="/spring/login?login_error=1" />
  <security:logout logout-url="/spring/logout" logout-success-url="/spring/logout-success" />
</security:http>

<security:authentication-provider>
  <security:password-encoder hash="md5" />
  <security:user-service>
    <security:user name="keith" password="417c7382b16c395bc25b5da1398cf076"
    authorities="ROLE_USER,ROLE_SUPERVISOR" />
    <security:user name="erwin" password="12430911a8af075c6f41c6976af22b09"
    authorities="ROLE_USER,ROLE_SUPERVISOR" />
    <security:user name="jeremy" password="57c6cbff0d421449be820763f03139eb" authorities="ROLE_USER" />
    <security:user name="scott" password="942f2339bf50796de535a384f0d1af3e"
    authorities="ROLE_USER" />
  </security:user-service>
</security:authentication-provider>
```

web.xml 환경 구성

필터 설정.(SS 기본)

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

참고자료

- 🌐 Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)
- 🌐 Whitespace's Note

개요

대부분의 애플리케이션은 여러 방법으로 데이터에 접근한다. 여러 사용자가 공유하는 데이터를 여럿이 수정한다.

따라서 트랜잭션 데이터 접근 속성이 필요하다. 관계형 데이터 집합을 도메인 객체로 변형하여 애플리케이션 처리를 도와준다.

Web Flow는 "Flow가 관리하는 영속성"(flow managed persistence)을 제공하여 Flow가 객체 영속성 문맥을 만들고, commit하고, 닫을 수 있도록 한다.

Web Flow는 하이버네이트와 JPA 객체 영속화 기술과 연동한다.

Flow-관리 영속성과 별도로 PersistenceContext 관리를 애플리케이션의 서비스 계층에서 완전히 캡슐화하는 패턴이 있다.

이런 경우 Web 계층은 영속화에 관여하지 않는다. 그 대신 서비스 계층으로 넘주거나 반환받은 detached object를 가지고 동작한다.

이번 장은 Flow-관리 영속성에 초점을 맞추고 이 기능을 언제 어떻게 사용하는지 살펴보겠다.

설명

Flow 범위(FlowScoped) PersistenceContext

이 패턴은 Flow 시작 시에 flowScope으로 PersistenceContext를 생성해준다.

이 persistence context는 Flow 실행 과정 동안에 데이터 접근을 하는데 사용하며, Flow가 종료될 때 persistent entity에서 변경된 내용을 반영(commit)한다.

이 패턴은 대부분 다중 사용자에게 의해서 동시에 수정되는 데이터의 정합성을 보호하고자 optimistic locking 전략과 함께 사용된다.

저장이나 재시작 능력이 필요없다면, Flow 상태를 표준 HTTP 세션 기반 저장 방법이 충분하다.

이 때 커밋 전의 세션 만료나 종료(termination)는 잠재적으로 변경 사항을 손실하게 할 수 있다.

FlowScoped PersistenceContext 패턴을 사용하려면 먼저 persistence-context로 Flow를 식별하게 해야 한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
  <persistence-context />
</flow>
```

그 다음으로 이 패턴을 적용할 Flow에 적절한 FlowExecutionListener를 설정하자.

하이버네이트를 사용한다면, HibernateFlowExecutionListener를 등록하고, JPA를 사용한다면, JpaFlowExecutionListener를 등록하자.

```
<webflow:flow-executor id="flowExecutor"
  flow-registry="flowRegistry">
  <webflow:flow-execution-listeners>
    <webflow:listener ref="jpaFlowExecutionListener" />
  </webflow:flow-execution-listeners>
</webflow:flow-executor>

<bean id="jpaFlowExecutionListener" class="org.springframework.webflow.persistence.JpaFlowExecutionListener">
  <constructor-arg ref="entityManagerFactory" />
  <constructor-arg ref="transactionManager" />
</bean>
```

Flow가 종료되는 시점에 커밋이 일어나게 하려면, end-state의 commit 속성을 입력하자.

```
<end-state id="bookingConfirmed" commit="true" />
```

이걸로 끝이다.

이제 Flow가 시작할 때 리스너가 flowScope에 새로운 EntityManager를 할당해서 제어하게 된다.

Flow 내에서 스프링 기반 데이터 접근 객체를 사용해서 발생하는 데이터 접근 시에는 항상 이 EntityManager를 자동으로 사용하게 된다.

이러한 데이터 접근 연산은 중간 수정 내용의 고립성 유지를 위해 항상 트랜잭션 처리 대상이 되지 않고, 읽기 전용 트랜잭션에서만 실행되어야 한다.

참고자료

- Spring Web Flow reference 2.0.x Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)
- Whiteship's Note

개요

Flow

Flow란 상이한 상황(context)에서 실행될 수 있는 재사용이 가능한 여러 단계들의 흐름을 캡슐화한 것을 의미한다.

모든 Flow는 아래와 같은 Root 로 시작한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
      http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

</flow>
```

- Flow 정의
 - 개요
 - Flow
 - Flow 의 구성
- 설명
 - Flow 의 필수적인 언어 구성요소
 - Actions
 - 입력/출력 매핑
 - 변수들
 - Sub Flow 호출
- 참고자료

Flow 의 구성

SWF에서 Flow는 "Sate(state)"로 부르는 일련의 단계들로 구성된다. Flow로 진입하게 되는 Sate는 일반적으로 사용자에게 보여지는 뷰가 된다.

이 뷰에서는 Sate를 제어하게 되는 이벤트가 발생한다. 이들 이벤트는 결과적으로 다른 뷰로 이동하게 되는 Transition(transition)을 일으키게 된다.

모든 state 는 <flow/> 안에 정의하게 된다. 맨처음 정의되는 state가 Flow의 시작점이게 된다.

Flow 의 작성법

Flow 는 웹 애플리케이션 개발자가 XML 기반 Flow 정의 언어를 사용해서 작성된다.

설명

Flow 의 필수적인 언어 구성요소

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
      http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

    <view-state id="enterBookingDetails" />

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <end-state id="bookingCancelled" />

</flow>
```

- view-state: Flow 중 화면을 보여주는 Sate를 정의하는 구성요소
 - 편의상 Flow 정의 파일이 있는 디렉터리 내에서 view-state id와 일치하는 화면 템플릿을 맞춰 보게 됨
- transition: Sate 내에서 발생한 이벤트를 제어하는 구성 요소. 화면 이동을 일으킴.
- end-state: Flow의 결과를 정의

Actions

대부분의 Flow는 화면 이동 로직 뿐만 아니라, 애플리케이션의 비즈니스 서비스나 다른 행동을 호출할 필요가 있을 수 있다.

Flow 내에서 Action을 취할 수 있는 여러 지점이 존재한다.

- Flow가 시작할 때
- Sate에 들어갈 때
- 화면을 보여줄 때
- Transition이 일어날 때
- Sate가 종료될 때
- Flow가 종료될 때

SWF 에서 Action은 기본적으로 Unified EL이라는 간결한 표현 언어를 사용해서 정의하게 된다.

evaluate

대부분 evaluate 구성요소를 사용하게 된다. 이를 통해 Spring Bean 에 있는 메소드나 다른 Flow 변수를 호출할 수 있다.
예를 들자면 아래와 같다.

```
<!-- [1] entityManager Bean 의 persist 메소드에 booking 객체를 넣어 호출한다. -->
<evaluate expression="entityManager.persist(booking)" />

<!-- [2] findHotels 메소드 호출하고 실행결과 Hotels 객체를 flowScope 데이터 모델에 저장한다. -->
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels" />

<!-- [3] findHotels 메소드 호출하고 실행결과 Hotels 객체를 flowScope 데이터 모델에 저장시 dataModel 타입으로 변환하여 저장한다. -->
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels" result-type="dataModel"/>
```

아래 예에서는 Flow가 시작할 때 Flow 범위에 Booking 객체를 생성해 저장한다. hotelId는 Flow의 입력 속성으로 받게 된다.

```
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
      http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

    <input name="hotelId" />

    <on-start>
        <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
        result="flowScope.booking" />
    </on-start>

    <view-state id="enterBookingDetails">
        <transition on="submit" to="reviewBooking" />
    </view-state>

    <view-state id="reviewBooking">
        <transition on="confirm" to="bookingConfirmed" />
        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <end-state id="bookingConfirmed" />
    <end-state id="bookingCancelled" />

</flow>
```

입력/출력 매핑

각각의 Flow는 잘 정의된 입력/출력 계약(input/output contract)를 갖고 있다.
Flow는 시작할 때 입력 속성을 건네 받게되고, 종료될 때 출력 속성을 반환하게 된다. 이처럼 Flow호출은 개념적으로 다음과 같은 메소드 호출과 비슷하다.

```
FlowOutcome flowId(Map<String, Object> inputAttributes);
```

반환되는 FlowOutcome은 다음과 같은 메소드 선언부를 갖게 된다.

```
public interface FlowOutcome {
    public String getName();
    public Map<String, Object> getOutputAttributes();
}
```

입력

```
<!-- [1] 해당 변수의 값은 flow scope 내에 hotelId 이란 이름으로 저장된다. -->
<input name="hotelId" />

<!-- [2] type 속성으로 속성 지정 가능. 타입이 일치하지 않다면 타입 변환 시도 -->
<input name="hotelId" type="long" />

<!-- [3] value 속성으로 입력 값을 할당 -->
<input name="hotelId" value="flowScope.myParameterObject.hotelId" />

<!-- [4] required 속성으로 null이나 비어있지 못하도록 강제 -->
<input name="hotelId" type="long" value="flowScope.hotelId" required="true" />
```

출력

Flow 출력 속성은 output 구성요소를 사용한다. output 속성은 end-state 내에 선언한다. 출력 값은 속성의 이름으로 Flow 범위 내에서 얻어오게 된다.

```
<end-state id="bookingConfirmed">
    <output name="bookingId" />
</end-state>
```

```

</end-state>
<!-- 직접 대상 값 지정 -->
<end-state id="bookingConfirmed">
  <output name="confirmationNumber" value="booking.confirmationNumber" />
</end-state>

```

입력/출력 매핑:샘플

```

<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
        http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
  <input name="hotelId" />
  <on-start>
    <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
    result="flowScope.booking" />
  </on-start>
  <view-state id="enterBookingDetails">
    <transition on="submit" to="reviewBooking" />
  </view-state>
  <view-state id="reviewBooking">
    <transition on="confirm" to="bookingConfirmed" />
    <transition on="revise" to="enterBookingDetails" />
    <transition on="cancel" to="bookingCancelled" />
  </view-state>
  <end-state id="bookingConfirmed">
    <output name="bookingId" value="booking.id" />
  </end-state>
  <end-state id="bookingCancelled" />
</flow>

```

위 Flow는 이제 hotelId를 입력 값으로 받아서, 새로운 예약이 끝나게 되면 bookingId 출력 속성을 결과로 반환하게 된다.

변수들

Flow에는 하나이상의 인스턴스 변수 선언이 가능하다. 이 변수들은 flow가 시작할 때 할당되며, 변수를 유지하게 되는 모든 @Autowired transient 참조는 Flow가 재시작될 때 다시 값이 할당(rewired)되게 된다. var 구성 요소를 사용해서 Flow 변수를 선언하자.

```

<var name="searchCriteria" class="com.mycompany.myapp.hotels.search.SearchCriteria"/>

```

변수로 사용하는 클래스가 Flow 요청 간 인스턴스의 Sate를 유지하기 위해서 java.io.Serializable을 interface로 가지고 있어야 함을 기억하자.

Sub Flow 호출

Flow 내에서 하위 Flow로써 또 다른 Flow 호출이 가능하다. 이 때 하위 Flow가 결과를 반환할 때까지 기존 Flow는 대기하게 된다.

subflow-state

subflow-state 구성요소를 사용해서 하위 Flow 호출을 하게 된다.

```

<subflow-state id="addGuest" subflow="createGuest">
  <transition on="guestCreated" to="reviewBooking">
    <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
  </transition>
  <transition on="creationCancelled" to="reviewBooking" />
</subflow-state>

```

이 예제에서는 createGuest Flow를 호출하게 된다. guestCreated 출력이 반환되게 되면, 새로운 손님이 예약 손님 리스트에 추가되게 된다.

subflow input 전달

input 구성요소를 사용하면 하위 Flow에 입력값을 건낼 수 있다.

```

<subflow-state id="addGuest" subflow="createGuest">
  <input name="booking" />
  <transition to="reviewBooking" />
</subflow-state>

```

subflow output 매핑

출력 값의 이름으로 하위 Flow에서 출력하는 속성을 참조해서 Transition를 하게 된다.

```

<subflow-state ..>
  <transition on="guestCreated" to="reviewBooking">
    <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
  </transition>
  ..

```

이 예에서는 guestCreated 을 반환하게 될 때 gest 이름으로 넘어온 값을 booking 내의 guests (currentEvent.attributes.guest) 의 일부로 추가 해주고 있다.

샘플:Sub Flow 호출하기

아래는 샘플 코드이다.

```

<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
        http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">

  <input name="hotelId" />
  <on-start>
    <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
              result="flowScope.booking" />
  </on-start>

  <view-state id="enterBookingDetails">
    <transition on="submit" to="reviewBooking" />
  </view-state>


  <view-state id="reviewBooking">
    <transition on="addGuest" to="addGuest" />
    <transition on="confirm" to="bookingConfirmed" />
    <transition on="revise" to="enterBookingDetails" />
    <transition on="cancel" to="bookingCancelled" />
  </view-state>

  <subflow-state id="addGuest" subflow="createGuest">
    <transition on="guestCreated" to="reviewBooking">
      <evaluate expression="booking.guests.add(currentEvent.attributes.guest)" />
    </transition>
    <transition on="creationCancelled" to="reviewBooking" />
  </subflow-state>

  <end-state id="bookingConfirmed">
    <output name="bookingId" value="booking.id" />
  </end-state>
  <end-state id="bookingCancelled" />
</flow>

```

참고자료

-  Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

개요

Web Flow 는 데이터 모델 및 action 실행을 위해 EL 을 이용한다. 우리는 EL에 대해 알아보면서 flow 정의 를 해보도록 하겠다.

- Expression Language
 - 개요
 - 설명
 - 지원하는 EL 구현체
 - EL 사용법
 - 참고자료

설명

지원하는 EL 구현체

Unified EL

기본으로는 Unified EL을 사용하도록 되어 있음. jboss-el이 기본 구현체로 되어 있다.

참고 : web 컨테이너에서는 대개 el-api 를 지원해준다. 톰캣 6 의 경우처럼 말이다.\

OGNL

OGNL은 SWF2에서 제공하는 또 다른 el. 클래스스페이스에만 추가하면 자동으로 찾아서 사용한다.

EL 호환성

Unified EL과 OGNL은 비슷한 문법을 가지고 있다. 가능하면 Unified EL만 사용하자.

EL 사용법

Flow에서 EL 사용하는 경우

- 클라이언트에서 제공되는 데이터에 접근하는 경우. 입력 속성이나 요청 파라미터
- flowScope처럼 내부 데이터 구조에 접근하는 경우
- 스프링 빈에 있는 메소드 호출
- 생성자 결정할 때

Flow에 의해서 보여지는 뷰는 EL을 사용해서 Flow 데이터 구조에 접근하게 됨.

표현 타입

표준 eval 표현

가장 일반적인 방법은 eval 표현으로 이 경우 \${}나 #{}을 사용하면 안 됨. 이 예는 searchCriteria에 있는 nextPage() 호출.

```
<evaluate expression="searchCriteria.nextPage()" />
```

표현 템플릿

다음은 "template" 표현식으로 아래와 같은 형태로 \${} 을 사용할 수 있다.

```
<view-state id="error" view="error-${externalContext.locale}.xhtml" />
```

externalContext 에 세팅되어 있는 locale 결과를 대체하여 error-결과.xhtml 로 생성된다.

특별한 EL 변수

Scope

- flowScope: flow 변수에 할당되며, Flow 범위를 가진 객체. 기본적으로 Flow범위에 저장되는 모든 객체는 Serializable이 되어 함.

```
<evaluate expression="searchService.findHotel(hotelId)" result="flowScope.hotel" />
```

- viewScope: view 변수에 할당되며, view-state 내 범위를 갖음. 그러므로 view-state 내에서만 참조 가능. 역시 모든 객체는 Serializable 되어 함.

```
<on-render>  
<evaluate expression="searchService.findHotels(searchCriteria)" result="viewScope.hotels" result-type="dataModel" />  
</on-render>
```

- requestScope: request 변수에 할당. 한 번의 Flow 내에서 공유

```
<set name="requestScope.hotelId" value="requestParameters.id" type="long" />
```

- flashScope: flash 변수에 할당. Flow가 시작될 때 할당되고, 뷰가 보여지고 난 후 clear 됐다가, Flow가 종료되면 정리되는 범위. 객체는 Serializable 해야 함.

```
<set name="flashScope.statusMessage" value="'Booking confirmed'" />
```

- conversationScope

conversation 변수에 할당. 최상위 Flow가 시작할 때 할당되며, 최상위 Flow가 종료될 때 정리. 최상위 Flow의 자식 Flow에서 공유. HTTP session에 저장되며, 세션 복제를 할 경우를 대비해 Serializable을 구현해야 함.

```
<evaluate expression="searchService.findHotel(hotelId)" result="conversationScope.hotel"/>
```

context

- flowRequestcontext: 현재 Flow 요청을 표현. RequestContext API.
- messageContext: 에러나 성공 메시지를 포함해서 Flow 실행 메시지를 받아오고, 만드는데 대한 context에 접근할 수 있음. MessageContext 참조.

```
<evaluate expression="bookingValidator.validate(booking, messageContext)" />
```

- flowExecutionContext: 현재 Flow 상태를 표현. FlowExecutionContext API.
- externalContext: 사용자 세션 속성 등 외부 환경에 접근할 수 있음. ExternalContext API.

```
<evaluate expression="searchService.suggestHotels(externalContext.sessionMap.userProfile)" result="viewScope.hotels" />
```

그 외

- requestParameters: 사용자로부터 넘어온 request 매개변수 접근

```
<set name="requestScope.hotelId" value="requestParameters.id" type="long" />
```

- currentEvent: 현재 Event 객체에 접근

```
<evaluate expression="booking.guests.add(currentEvent.guest)" />
```

- currentUser: 인증된 Principal에 접근

```
<evaluate expression="bookingService.createBooking(hotelId, currentUser.name)" result="flowScope.booking" />
```

- resourceBundle: message 자원 관리

```
<set name="flashScope.successMessage" value="resourceBundle.successMessage" />
```

- flowExecutionUrl: 현재 flow execution view-state에 대한 context-relative URI에 접근

범위 검색 알고리즘

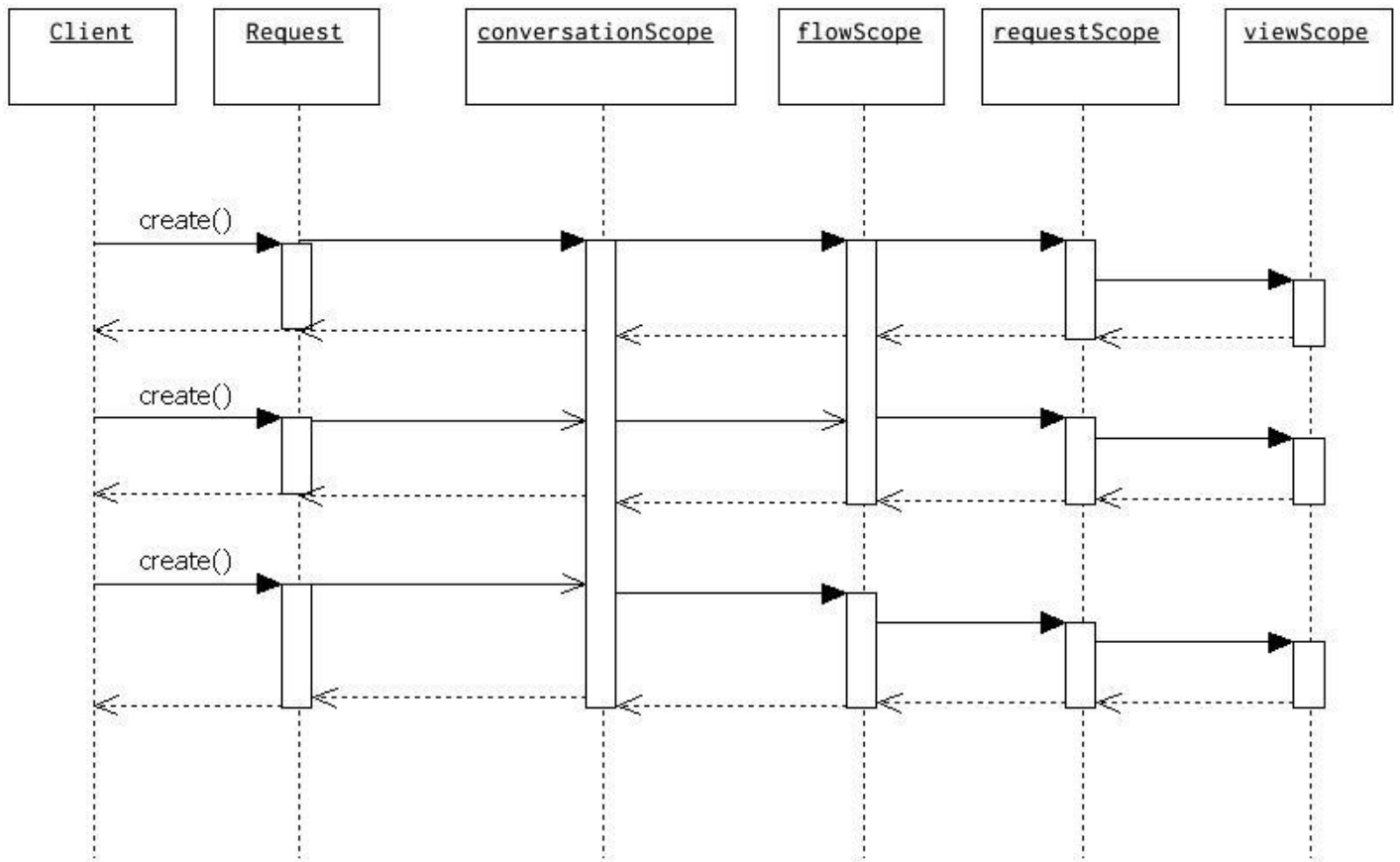
특정 범위에 변수를 할당할 때는 반드시 범위를 명시해야 한다.

```
<set name="requestScope.hotelId" value="requestParameters.id" type="long" />
```

특정 범위에 있는 변수에 접근할 때는 꼭 범위를 명시할 필요는 없다.

```
<evaluate expression="entityManager.persist(booking)" />
```

booking처럼 범위를 명시하지 않은 경우, 범위 검색 알고리즘(scope searching algorithm)이 동작하며, 이 알고리즘은 request→flash→view→flow→conversation 범위의 순서로 찾게 된다. 없을 경우 EvaluationException 발생. 아래그림은 검색되는 Scope 순서를 잘 보여주고 있다.



참고자료

- [Spring Web Flow reference 2.0.x](#)
- [Spring Web-Flow Framework Reference beta with Korean \(by 박찬욱\)](#)
- [Pro Spring 2.5\(Apress\) - Chapter 18 Spring Web Flow](#)

개요

view-state 는 flow 내에서 화면을 생성하는 요소이다.
여기서는 view-state 에 대해서 알아보도록 하자.

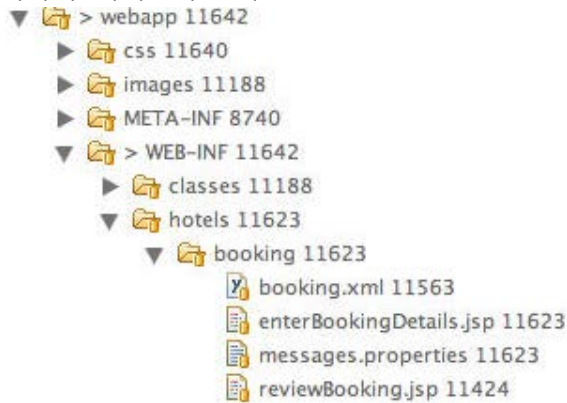
설명

뷰 상태(view state) 정의하기

view-state 는 기본적으로 해당 뷰를 생성하여 보여준 후, 사용자가 화면을 통해 응답하는 것을 기다린다.
아래는 view-state는 enterBookingDetails 라는 ID 를 가지고 있으며 또한 별도의 view 설정이 없기 때문에 ID 가 곧 view 를 뜻한다.

```
<view-state id="enterBookingDetails">
  <transition on="submit" to="reviewBooking" />
</view-state>
```

따라서, 디렉토리 상의



booking.xml(or booking-flow.xml) 이 존재하는 디렉토리에 있는 enterBookingDetails.jsp 이 자동으로 view 로 동작한다.
또는 절대경로를 이용하여 명시적으로 view="/WEB-INF/hotels/booking/enterBookingDetails.jsp" 설정할 수도 있다.
아래에서 다시 설명하겠다.

뷰 식별자 지정하기

뷰 속성을 여러 방법으로 지정할 수 있다.

- 상대 경로 사용

```
<view-state id="enterBookingDetails" view="bookingDetails.xhtml">
```

- 절대 경로 사용

```
<view-state id="enterBookingDetails" view="/WEB-INF/hotels/booking/bookingDetails.jsp">
```

- 논리적인 경로 사용: Spring MVC 등과 통합 시

```
<view-state id="enterBookingDetails" view="bookingDetails">
```

뷰 범위

view-state 내부에서 유지되는 변수. Ajax 요청처럼 동일한 뷰가 여러번 보여줘야 하는 경우 유용.

- var 태그를 사용해서 view 변수 선언.

```
<var name="searchCriteria" class="com.mycompany.myapp.hotels.SearchCriteria" />
```

- 뷰(View) 보여주기
 - 개요
 - 설명
 - 뷰 상태(view state) 정의하기
 - 뷰 식별자 지정하기
 - 뷰 범위
 - 화면을 보여줄 때 액션 실행
 - Model Binding
 - 타입 변환 수행
 - 바인딩 금지하기
 - 명시적으로 바인딩 지정하기
 - Model 유효성 검증
 - 유효성 검증 하지 않기
 - 뷰 transition 실행
 - 메세지 사용하기
 - 팝업 띄우기
 - 뷰 백트래킹(View backtracking)
 - 참고자료

- viewScope 변수에 할당하기

```
<on-render>
  <evaluate expression="bookingService.findHotels(searchCriteria)" result="viewScope.hotels" />
</on-render>
```

뷰 범위내에서 Object 다루기

아래코드는 화면 ID 가 searchResults 인 view-state 화면을 그리되
그리기전 bookingService.findHotels(searchCriteria) 메소드를 호출한 후 그 결과를 viewScope내의 hotels 로 저장한 후
화면을 보여주고 있다.
그리고 화면상에 next 또는 previous 이벤트 발생시 eval(searchCriteria.nextPage()/previousPage()) 이 발생
그 결과를 fragments으로 지정된 영역에 뿌려주고 있다.

```
<view-state id="searchResults">
  <on-render>
    <evaluate expression="bookingService.findHotels(searchCriteria)"
      result="viewScope.hotels" />
  </on-render>
  <transition on="next">
    <evaluate expression="searchCriteria.nextPage()" />
    <render fragments="searchResultsFragment" />
  </transition>
  <transition on="previous">
    <evaluate expression="searchCriteria.previousPage()" />
    <render fragments="searchResultsFragment" />
  </transition>
</view-state>
```

자세한 것은 아래에서 다시 설명하도록 하겠다.

화면을 보여줄 때 액션 실행

뷰를 보여주기 전에 특정 액션을 실행하려면 on-render 사용한다.

```
<on-render>
  <evaluate expression="bookingService.findHotels(searchCriteria)" result="viewScope.hotels" />
</on-render>
```

Model Binding

```
<view-state id="enterBookingDetails" model="booking">
```

뷰 이벤트가 발생했을 때 지정된 모델에 대해서 다음 행동이 일어난다.

1. view-to-model binding.
2. 모델 유효성 검증.

타입 변환 수행

변환기(Converter) 구현

org.springframework.binding.convert.converters.TwoWayConverter을 구현하면 됨. StringToObject를 구현하는게 더 좋다.

```
protected abstract Object toObject(String string, Class targetClass) throws Exception;
protected abstract String toString(Object object) throws Exception;
```

구현 예.

```
public class StringToMonetaryAmount extends StringToObject {
    public StringToMonetaryAmount() {
        super(MonetaryAmount.class);
    }
    @Override
    protected Object toObject(String string, Class targetClass) {
        return MonetaryAmount.valueOf(string);
    }
    @Override
    protected String toString(Object object) {
        MonetaryAmount amount = (MonetaryAmount) object;
        return amount.toString();
    }
}
```

org.springframework.binding.convert.converters에 이미 구현된 변환기가 위치.

변환기 등록하기

org.springframework.binding.convert.service.DefaultConversionService를 상속해서 addDefaultConverters() 메소드를 재정의 하면 된다.

자세한 것은 시스템 설정 에서 ConversionService 확장을 이용하여 설정하는 곳에서 다루고 있다.

바인딩 금지하기

bind 속성으로 특정 뷰 이벤트에서 모델 바인딩과 유효성 검증을 안 하게 할 수도 있다.

```
<view-state id="enterBookingDetails" model="booking">
  <transition on="proceed" to="reviewBooking">
    <transition on="cancel" to="bookingCancelled" bind="false" />
  </view-state>
```

명시적으로 바인딩 지정하기

아래와 같이 binder 속성으로 바인딩 할 프로퍼티를 명시적으로 지정할 수 있다.

```
<view-state id="enterBookingDetails" model="booking">
  <binder>
    <binding property="creditCard" />
    <binding property="creditCardName" />
    <binding property="creditCardExpiryMonth" />
    <binding property="creditCardExpiryYear" />
  </binder>
  <transition on="proceed" to="reviewBooking" />
  <transition on="cancel" to="cancel" bind="false" />
</view-state>
```

binder로 지정하지 않으면 모든 프로퍼티를 바인딩된다. converter를 이용하여 변환기 지정 가능하다.

```
<view-state id="enterBookingDetails" model="booking">
  <binder>
    <binding property="checkinDate" converter="shortDate" />
    <binding property="checkoutDate" converter="shortDate" />
    <binding property="creditCard" />
    <binding property="creditCardName" />
    <binding property="creditCardExpiryMonth" />
    <binding property="creditCardExpiryYear" />
  </binder>
  <transition on="proceed" to="reviewBooking" />
  <transition on="cancel" to="cancel" bind="false" />
</view-state>
```

Model 유효성 검증

Model 유효성 검사에 대한 부분은 Web flow 에서는 프로그래밍적으로 제약사항을 강제화 하는 형태로 지원하고 있다.

프로그램 내에서 유효성 검증

첫 번째 방법으로 유효성 검증 로직을 모델 객체 내에 정의하는 방법이다.

Web Flow 는 view-stat 에서 모델로 넘어간 시점(view-state postback lifecycle)에서 자동적으로 validate 메소드를 자동으로 호출한다.

```
<view-state id="enterBookingDetails" model="booking">
  <transition on="proceed" to="reviewBooking">
  </view-state>
```

Booking class내의 validate(view-state 명) 코드는 아래와 같이 볼 수 있다.(메소드명 : validate + EnterBookingDetails)

```
public class Booking {
    private Date checkinDate;
    private Date checkoutDate;

    ...

    public void validateEnterBookingDetails(ValidationContext context) {
        MessageContext messages = context.getMessage();
        if (checkinDate.before(today())) {
            messages.addMessage(new MessageBuilder().error().source("checkinDate").
                defaultText("Check in date must be a future date").build());
        } else if (!checkinDate.before(checkoutDate)) {
            messages.addMessage(new MessageBuilder().error().
                source("checkoutDate").
                defaultText("Check out date must be later than
check in date")
                .build());
        }
    }
}
```

enterBookingDetails에 대한 이벤트가 발생했을 때 자동으로 validateEnterBookingDetails이 호출 된다.
메소드 이름을 validate\$<state> 로 정의하면 된다.

Validator 구현

Validator로 불리는 별도의 객체로 정의할 수도 있다. 클래스 이름을 \$<model>Validator 로 지정하면 된다. 메소드 이름은 역시 validate\$<state>로 한다. 아래 클래스명은 Booking + Validator 이며 메소드 이름은 validate + EnterBookingDetails 임을 볼 수 있다.

```
@Component
public class BookingValidator {
    public void validateEnterBookingDetails(Booking booking, ValidationContext context) {
        MessageContext messages = context.getMessages();
        if (booking.getCheckinDate().before(today())) {
            messages.addMessage(new MessageBuilder().error()
                .source("checkinDate")
                .defaultText("Check in date must be a future
date")
                .build());
        } else if (!booking.getCheckinDate().before(booking.getCheckoutDate())) {
            messages.addMessage(new MessageBuilder().error()
                .source("checkoutDate")
                .defaultText("Check out date must be later
than check in date")
                .build());
        }
    }
}
```

spring mvc의 Error 객체도 받을 수 있다.

ValidationContext

유효성 검증 동안에 MessageContext에 접근할 수 있게 해주며, 다양한 객체에 접근 가능하게 해준다.

유효성 검증 하지 않기

validate="false" 설정함으로 유효성 검사를 하지 않을 수 있다

```
<view-state id="chooseAmenities" model="booking">
    <transition on="proceed" to="reviewBooking">
    <transition on="back" to="enterBookingDetails" validate="false" />
</view-state>
```

뷰 transition 실행

전이 대상은 (1)다른 뷰, (2)현재 뷰를 다시, (3)action을 실행, (4)Ajax 이벤트를 제어할 때 'fragments'로 불리는 일부 뷰를 보여주라는 요청일 수도 있다.

전이 액션(Transition actions)

```
<transition on="submit" to="bookingConfirmed">
    <evaluate expression="bookingAction.makeBooking(booking, messageContext)" />
</transition>
```

```
public class BookingAction {
    public boolean makeBooking(Booking booking, MessageContext context) {
        try {
            bookingService.make(booking);
            return true;
        } catch (RoomNotAvailableException e) {
            context.addMessage(builder.error().defaultText("No room is available at this
hotel").build());
            return false;
        }
    }
}
```

글로벌 전이(Global transitions)

```
<global-transitions>
    <transition on="login" to="login">
    <transition on="logout" to="logout">
</global-transitions>
```

이벤트 핸들러(Event handlers)

```
<transition on="event">
    <!-- Handle event -->
</transition>
```

프레그먼트 보여주기(fragments)

현재 뷰 중 일부만을 다시 보여줄 수 있는 방법으로, Ajax 기반일 때 주로 사용한다.

```
<transition on="next">
  <evaluate expression="searchCriteria.nextPage()" />
  <render fragments="searchResultsFragment" />
</transition>
```

';'로 구분해서 다수의 fragment를 지정할 수도 있다.

메세지 사용하기

MessageContext는 플로우 실행 동안에 메세지를 저장하는데 사용되는 API다.

일반 메세지나 국제화가 지원된 메세지 모두 사용 가능하다.

메세지 수준도 지정 가능하며, 지원되는 수준은 info, warning, error이 있다. 메세지를 추가할 때는 MessageBuilder를 사용하자.

■ 일반 메세지 추가

```
MessageContext context = ...
MessageBuilder builder = new MessageBuilder();
context.addMessage(builder.error().source("checkinDate").defaultText("Check in date must be a future date").build());
context.addMessage(builder.warn().source("smoking").defaultText("Smoking is bad for your health").build());
context.addMessage(builder.info().defaultText("We have processed your reservation - thank you and enjoy your stay").build());
```

■ 국제화가 지원되는 메세지 추가

```
MessageContext context = ...
MessageBuilder builder = new MessageBuilder();
context.addMessage(builder.error().source("checkinDate").code("checkinDate.notFuture").build());
context.addMessage(builder.warn().source("smoking").code("notHealthy").resolvableArg("smoking").build());
```

메세지 번들 사용하기

스프링의 MessageSource를 사용해서 메세지 번들을 정의가 가능하다. 간단히 프로퍼티 파일로 관리하면 된다.

```
#messages.properties
checkinDate=Check in date must be a future date
notHealthy={0} is bad for your health
reservationConfirmation=We have processed your reservation - thank you and enjoy your stay
```

뷰나 플로우에서는 resourceBundle EL 변수로 접근도 가능하다.

```
<h:outputText value="#{resourceBundle.reservationConfirmation}" />
```

시스템 생성 메세지 이해하기

시스템에서 발생한 예외에 대해 메세지 지정 가능하다. 예를 들어 타입 변환 시 예외가 발생하면 typeMismatch를 통해서 메세지 지정 가능하다.

```
booking.checkinDate.typeMismatch=The check in date must be in the format yyyy-mm-dd.
```

팝업 띄우기

모달 팝업 다이얼로그를 뷰로 렌더링하고 싶다면, view-state 내에 popup="true" 설정하면 된다.

```
<view-state id="changeSearchCriteria" view="enterSearchCriteria.xhtml" popup="true">
```

특히 스프링 자바 스크립트와 함께 사용하면, 팝업을 보여주는데 클라이언트 코드가 전혀 필요 없다. SWF가 클라이언트 요청을 팝업으로 재전송(redirect)해준다.

뷰 백트래킹(View backtracking)

기본적으로 브라우저의 백 버튼으로 이전 view-state로 돌아갈 수 있다. history를 사용해서 이에 대한 설정이 가능하다.


■ 'discard'로 설정하면 백트래킹(backtracking) 예방 가능하다.

```
<transition on="cancel" to="bookingCancelled" history="discard">
```

■ 'invalidate'로 설정하면 이전에 보여줬던 모든 뷰 뿐만 아니라 현재 뷰까지도 백트래킹 예방된다.

```
<transition on="confirm" to="bookingConfirmed" history="invalidate">
```

참고자료

-  Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

개요

action-state 은 flow 내에서 action 실행을 제어하기 위한 요소이다.
decision-state 를 이용하여 if-else 와 같은 흐름제어를 할 수 있다. 좀 더 자세히 알아보도록 하자.

설명

- Action 실행
 - 개요
 - 설명
 - 액션 상태 정의하기
 - 의사결정 상태(decision states) 정의
 - 액션 출력 이벤트 매핑
 - 액션 구현
 - 액션 예외
 - 다른 Action 실행 예제
- 참고자료

액션 상태 정의하기

특정 액션을 호출한 다음에, 그 결과에 따라서 다른 상태로 전이하고 싶은 경우에는 action-state 구성요소를 사용하자. 직관적으로 봤을 때 아래 코드는 interview.moreAnswersNeeded() 의 결과값에 의해 transition 이 실행될 것을 예상할 수 있다.

```
<action-state id="moreAnswersNeeded">
  <evaluate expression="interview.moreAnswersNeeded()" />
  <transition on="yes" to="answerQuestions" />
  <transition on="no" to="finish" />
</action-state>
```

좀더 완전한 예를 살펴보자.

```
<flow xmlns="http://www.springframework.org/schema/webflow"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/webflow
    http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
  <on-start>
    <evaluate expression="interviewFactory.createInterview()"
      result="flowScope.interview" />
  </on-start>

  <view-state id="answerQuestions" model="questionSet">
    <on-entry>
      <evaluate expression="interview.getNextQuestionSet()"
        result="viewScope.questionSet" />
    </on-entry>
    <transition on="submitAnswers" to="moreAnswersNeeded">
      <evaluate expression="interview.recordAnswers(questionSet)" />
    </transition>
  </view-state>

  <action-state id="moreAnswersNeeded">
    <evaluate expression="interview.moreAnswersNeeded()" />
    <transition on="yes" to="answerQuestions" />
    <transition on="no" to="finish" />
  </action-state>

  <end-state id="finish" />
</flow>
```

의사결정 상태(decision states) 정의

action-state를 대신해서 편리하게 if/else 문법을 사용해서 이동하고자 하는 의사결정을 해주는 decision-state를 사용한다. 이전 예제를 의사결정 상태로 구현한 예를 보자.

```
<decision-state id="moreAnswersNeeded">
  <if test="interview.moreAnswersNeeded()" then="answerQuestions" else="finish" />
</decision-state>
```

액션 출력 이벤트 매핑

액션은 대부분 POJO의 메소드를 호출한다. action-state와 decision-state을 호출했을 때, 이들 메소드가 반환하는 값은 상태를 전이하게 해주는데 사용할 수 있다. 전이가 이벤트에 의해서 발생되기 때문에, 우선 메소드가 반환하는 값은 반드시 Event 객체에 매핑되어야 한다. 다음 테이블은 공통적으로 반환하는 값 타입에 따라 Event 객체가 어떻게 매핑되는지를 설명해준다. 메소드 반환 타입 매핑된 Event 식별자 표현

결과로 리턴되는 타입	매핑되는 이벤트 값
java.lang.String	String 값
java.lang.Boolean	yes(true에 해당), no(false에 해당)
java.lang.Enum Enum	Enum 이름

나머지 다른 타입

success

예제.moreAnswersNeeded() 메소드의 리턴 타입은 boolean 인 것을 예상할 수 있으면 그에 따라 yes, no 에 매핑됨을 알 수 있다.

```
<action-state id="moreAnswersNeeded">
  <evaluate expression="interview.moreAnswersNeeded()" />
  <transition on="yes" to="answerQuestions" />
  <transition on="no" to="finish" />
</action-state>
```

액션 구현

POJO 로직처럼 action 코드를 작성하는 것이 가장 일반적이다.

때로는 flow context에 접근할 필요가 있는 액션 코드를 작성할 필요가 있다.

이럴 때는 POJO를 호출하면서, EL 변수로 flowRequestContext를 건낼 수 있다.

그 대신 Action 인터페이스를 구현하거나, MultiAction 기본 클래스를 상속할 수도 있다.

POJO 메소드 호출

```
<evaluate expression="pojoAction.method(flowRequestContext)" />
```

```
public class PojoAction {
    public String method(RequestContext context) {
        ...
    }
}
```

custom Action 구현 호출

```
<evaluate expression="customAction" />
```

```
public class CustomAction implements Action {
    public Event execute(RequestContext context) {
        ...
    }
}
```

MultiAction 구현 호출

```
<evaluate expression="multiAction.actionMethod1" />
```

```
public class CustomMultiAction extends MultiAction {
    public Event actionMethod1(RequestContext context) {
        ...
    }
    public Event actionMethod2(RequestContext context) {
        ...
    }
}
```

액션 예외

action은 복잡한 비즈니스 로직을 캡슐화하고 있는 서비스를 호출할 수도 있다.

이 서비스들은 비즈니스 예외를 던질 수도 있으니 이를 처리해야 할 수도 있다.

POJO 액션 사용 시 비즈니스 예외 제어하기

```
<evaluate expression="bookingAction.makeBooking(booking, flowRequestContext)" />
```

```
public class BookingAction {
    public String makeBooking(Booking booking, RequestContext context) {
        try {
            BookingConfirmation confirmation = bookingService.make(booking);
            context.getFlowScope().put("confirmation", confirmation);
            return "success";
        } catch (RoomNotAvailableException e) {
            context.addMessage(new MessageBuilder().error().defaultText("No room is available at
this hotel").build());
            return "error";
        }
    }
}
```

MultiAction 사용 시 비즈니스 예외 제어하기

아래 예제는 이전 예제와 기능적으로는 동일하지만, POJO 액션 대신 MultiAction으로 구현했다. Event \${methodName}(RequestContext) 규약에 따라 메소드를 구성하면 되고, POJO의 자유스러움에 비해, 보다 더 강력한 타입 안정성을 제공한다.

```
<evaluate expression="bookingAction.makeBooking" />

public class BookingAction extends MultiAction {
    public Event makeBooking(RequestContext context) {
        try {
            Booking booking = (Booking) context.getFlowScope().get("booking");
            BookingConfirmation confirmation = bookingService.make(booking);
            context.getFlowScope().put("confirmation", confirmation);
            return success();
        } catch (RoomNotAvailableException e) {
            context.getMessageContext().addMessage(new MessageBuilder().error().defaultText("No room is available at
this hotel").build());
            return error();
        }
    }
}
```

다른 Action 실행 예제

on-start

```
<flow xmlns="http://www.springframework.org/schema/webflow"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
    <input name="hotelId" />
    <on-start>
        <evaluate expression="bookingService.createBooking(hotelId, currentUser.name)"
result="flowScope.booking" />
    </on-start>
</flow>
```

on-entry

```
<view-state id="changeSearchCriteria" view="enterSearchCriteria.xhtml"
popup="true">
    <on-entry>
        <render fragments="hotelSearchForm" />
    </on-entry>
</view-state>
```

on-exit

```
<view-state id="editOrder">
    <on-entry>
        <evaluate expression="orderService.selectForUpdate(orderId, currentUser)"
result="viewScope.order" />
    </on-entry>
    <transition on="save" to="finish">
        <evaluate expression="orderService.update(order, currentUser)" />
    </transition>
    <on-exit>
        <evaluate expression="orderService.releaseLock(order, currentUser)" />
    </on-exit>
</view-state>
```

on-end

```
<flow xmlns="http://www.springframework.org/schema/webflow"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/webflow
http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
    <input name="orderId" />
    <on-start>
        <evaluate expression="orderService.selectForUpdate(orderId, currentUser)"
result="flowScope.order" />
    </on-start>
    <view-state id="editOrder">
        <transition on="save" to="finish">
            <evaluate expression="orderService.update(order, currentUser)" />
        </transition>
    </view-state>
    <on-end>
        <evaluate expression="orderService.releaseLock(order, currentUser)" />
    </on-end>
</flow>
```

on-render

```
<view-state id="reviewHotels">
    <on-render>
```



```

        <evaluate expression="bookingService.findHotels(searchCriteria)"
            result="viewScope.hotels" result-type="dataModel" />
    </on-render>
    <transition on="select" to="reviewHotel">
        <set name="flowScope.hotel" value="hotels.selectedRow" />
    </transition>
</view-state>

```

on-transition

```

<subflow-state id="addGuest" subflow="createGuest">
    <transition on="guestCreated" to="reviewBooking">
        <evaluate expression="booking.guestList.add(currentEvent.attributes.newGuest)" />
    </transition>
</subflow-state>

```

Named actions

```

<action-state id="doTwoThings">
    <evaluate expression="service.thingOne()">
        <attribute name="name" value="thingOne" />
    </evaluate>
    <evaluate expression="service.thingTwo()">
        <attribute name="name" value="thingTwo" />
    </evaluate>
    <transition on="thingTwo.success" to="showResults" />
</action-state>

```

Streaming actions

아래 예는 flow 에서 printBoardingPassAction 를 호출하는 것으로 PDF 로 프린트 하고자할 때 구현하는 예를 보여주고 있다.

AbstractAction 을 상속한 PrintBoardingPassAction 의 doExecute() 메소드안에 실제 pdf 관련 소스를 구현하고 success() 를 리턴한다.

```

<view-state id="reviewItinerary">
    <transition on="print">
        <evaluate expression="printBoardingPassAction" />
    </transition>
</view-state>


```

```

public class PrintBoardingPassAction extends AbstractAction {
    public Event doExecute(RequestContext context) {
        // stream PDF content here...
        // - Access HttpServletResponse by calling context.getExternalContext().getNativeResponse();
        // - Mark response complete by calling context.getExternalContext().recordResponseComplete();
        return success();
    }
}

```

참고자료

-  Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)

개요

Flow 상속은 한 Flow가 다른 Flow 설정을 상속할 수 있게 되어 있다. 상속은 Flow와 State 레벨에서 모두 발생할 수 있다.
가장 흔한 유즈케이스는 상위 Flow로 global transition과 예외 핸들러를 정의하고 하위 Flow로 그 설정을 상속받는 것이다.
상위 Flow를 찾으려면 다른 Flow들처럼 flow-registry에 추가해야 된다.

- Flow 상속
 - 개요
 - 설명
 - Flow 상속은 자바 상속과 비슷한가?
 - Flow 상속 타입
 - 추상 Flow
 - 상속 알고리즘
 - 참고자료

설명

Flow 상속은 자바 상속과 비슷한가?

상위에 정의한 요소를 하위에서 접근할 수 있다는 측면에서는 자바 상속과 Flow 상속이 비슷하다.
하지만 몇가지 차이점을 가지고 있다.

하위 Flow는 상위 Flow의 요소를 재정의 할 수 없다. 상위와 하위 Flow에 있는 같은 요소는 병합된다.
상위 Flow에만 있는 요소는 하위 Flow에 추가된다.

하위 Flow는 여러 상위 Flow를 상속받을 수 있다. 그러나, 자바 상속은 단일 클래스로 제한된다.

Flow 상속 타입

Flow 수준 상속

Flow 수준 상속은 flow 내에 parent 속성을 이용하여 정의한다. 이 속성은 콤마로 구분하여 상속받을 Flow 를 표현한다.

하위 flow는 목록에 명시된 순서대로 각각의 상위 Flow를 상속받는다.
첫 번째 상속으로 상위 Flow에 있는 요소와 내용을 추가하고 나면 그것을 다시 하위 Flow로 간주하고 그 다음 상위 flow를 상속 받는다.
아래 예를 보면 common-transitions를 먼저 상속 받고 다음에 common-states 를 상속 받는다.

```
<flow parent="common-transitions, common-states">
```

State 수준 상속

State 수준 상속은 Flow 수준 상속과 비슷하다. 유일한 차이점은 Flow 전체가 아니라 오직 해당 **State 하나만 상위로 부터 상속 받는다.**
Flow 상속과 달리 오직 하나의 상위만 허용한다. 또한 상속받을 Flow State의 식별자가 반드시 정의되어 있어야 한다.
Flow와 State 식별자는 #로 구분한다.
상위와 하위 State는 반드시 같은 타입이어야 한다. 예를 들어 view-state는 ent-state를 상속받을 수 없다. 오직 view-state만 상속받을 수 있다.

```
<view-state id="child-state" parent="parent-flow#parent-view-state">
```

추상 Flow

종종 상위 Flow는 직접 호출하지 않도록 설계한다. 그런 Flow를 실행하지 못하도록 abstract로 설정할 수 있다.
만약 추상 Flow를 실행하려고 하면 FlowBuilderException가 발생한다.

```
<flow abstract="true">
```

상속 알고리즘

하위 Flow가 상위 Flow를 상속할 때 발생하는 기본적인 일은 상위와 하위 Flow를 병합하여 새로운 Flow를 만드는 것이다.
웹 Flow 정의 언어에는 각각의 엘리먼트에 대해 어떻게 병합할 것인가에 대한 규칙이 있다.
엘리먼트에는 두 종류가 있다. 병합 가능한 것(mergeable)과 병합이 가능하지 않은 것(non-mergeable)이 있다.
병합가능(mergeable)한 엘리먼트는 만약 엘리먼트가 같다면 병합을 시도한다. 병합이 가능하지 않은(non-mergeable) 엘리먼트는 항상 최종 Flow에 직접 포함된다.
병합 과정 중에 수정하지 않는다.

주의

- *상위 Flow에있는 외부 리소스 경로는 절대 경로여야 한다.** 상대 경로는 두 Flow를 병합할 때 상위 Flow와 하위 Flow가 위치한 디렉토리가

다르면 깨질 수 있다. 일반 병합하면, 상위 Flow에 있던 모든 상대 경로는 하위 Flow 기준으로 바뀐다.

병합 가능한(mergeable) 엘리먼트

만약 같은 타입의 엘리먼트고 입력한 속성이 같다면 상위 엘리먼트의 내용을 하위 엘리먼트로 병합한다.

병합 알고리즘은 계속해서 병합하는 상위와 하위의 서브 엘리먼트를 각각 병합한다.

그렇지 않으면 상위 Flow의 엘리먼트를 하위 Flow에 새로운 엘리먼트로 추가한다.

대부분의 경우 상위 flow의 엘리먼트가 하위 Flow 엘리먼트에 추가된다.

이 규칙에 예외로는 시작할 때 추가 될 action 엘리먼트(evaluate, render, set)가 있다. 상위 action의 결과를 하위 action 결과로 사용하게 한다.

병합이 가능한 엘리먼트는 다음과 같다.



- action-state: id
- attribute: name
- decision-state: id
- end-state: id
- flow: 항상 병합
- if: test
- on-end: 항상 병합
- on-entry: 항상 병합
- on-exit: 항상 병합
- on-render: 항상 병합
- on-start: 항상 병합
- input: name
- output: name
- secured: attributes
- subflow-state: id
- transition: on and on-exception
- view-state: id

병합 할 수 없는 엘리먼트

병합할 수 없는 엘리먼트는 다음과 같다

- bean-import
- evaluate
- exception-handler
- persistence-context
- render
- set
- var

참고자료

-  Spring Web Flow reference 2.0.x
- Spring Web-Flow Framework Reference beta with Korean (by 박찬욱)
-  Whiteship's Note