

Anyframe Webflow Plugin



Version 1.0.1

저작권 © 2007-2011 삼성SDS

본 문서의 저작권은 삼성SDS에 있으며 Anyframe 오픈소스 커뮤니티 활동의 목적하에서 자유로운 이용이 가능합니다. 본 문서를 복제, 배포할 경우에는 저작권자를 명시하여 주시기 바라며 본 문서를 변경하실 경우에는 원문과 변경된 내용을 표시하여 주시기 바랍니다. 원문과 변경된 문서에 대한 상업적 용도의 활용은 허용되지 않습니다. 본 문서에 오류가 있다고 판단될 경우 이슈로 등록해 주시면 적절한 조치를 취하도록 하겠습니다.

I. Introduction	1
II. Spring Webflow	2
1. Configuration	3
1.1. 기본 설정	3
1.1.1. FlowRegistry 정의	3
1.1.2. FlowExecutor 정의	4
1.2. Spring MVC와 연계하기 위한 설정	4
1.2.1. FlowHandlerAdaptor 정의	4
1.2.2. FlowHandlerMapping 정의	4
1.2.3. Spring MVC의 ViewResolver 지정	5
2. Flow	7
2.1. 필수 요소	7
2.1.1. view-state	7
2.1.2. transition	7
2.1.3. end-state	8
2.2. 메소드 호출	8
2.2.1. evaluate	9
2.3. Transition Decision	9
2.3.1. action-state	9
2.3.2. decision-state	10
2.4. Expression Language	10
2.4.1. Special EL variables	10
2.5. Subflow	11
2.5.1. subflow-state	11
2.5.2. input	11
2.5.3. output	12
2.6. 플로우 상속	12
2.6.1. flow 레벨 상속	12
2.6.2. state 레벨 상속	12
3. View	14
3.1. model 바인딩	14
3.2. view backtracking	14
3.2.1. discard	14
3.2.2. invalidate	15
4. Validator	16
4.1. model 객체 내에 validate 메소드 구현	16
4.2. validator 클래스 및 메소드 구현	16

I.Introduction

webflow plugin은 XML 기반으로 작성된 Flow Definition 파일을 기반으로 페이지 흐름 제어를 지원하는 Spring Web Flow [<http://www.springsource.org/webflow>]를 활용하는 방법을 가이드하기 위한 샘플 코드와 이 오픈소스를 활용하는데 필요한 참조 라이브러리들로 구성되어 있다.

Installation

Command 창에서 다음과 같이 명령어를 입력하여 webflow plugin을 설치한다.

```
mvn anyframe:install -Dname=webflow
```

installed(mvn anyframe:installed) 혹은 jetty:run(mvn clean jetty:run) command를 이용하여 설치 결과를 확인해볼 수 있다.

Dependent Plugins

Plugin Name	Version Range
query [http://dev.anyframejava.org/docs/anyframe/plugin/optional/query/1.1.2/reference/htmlsingle/query.html]	2.0.0 > *

II.Spring Webflow

일반적인 웹 어플리케이션을 개발 할 때 페이지 처리 흐름을 제어하기 위해서는 복잡하고 반복적인 코드가 들어가게 된다. 이 때, Spring Web Flow를 사용하면 선언적인 Flow Definition 파일을 작성함으로써 보다 쉽게 페이지 흐름을 제어할 수 있다. 이번 장에서는 Spring Web Flow에 대한 소개 및 Flow Definition에 대한 소개 및 Flow Definition 파일 작성 방법과 Spring Web Flow를 사용한 웹 어플리케이션 개발 방법에 대해 알아본다.

1. Configuration

Spring Web Flow를 사용하게 위해서는 다음과 같은 기본 설정과 Spring MVC와의 연계를 위한 추가 설정이 필요하다.

1.1. 기본 설정

1.1.1. FlowRegistry 정의

Spring Web Flow를 실행 시키기 위해 Flow Definition 파일이 있는 위치를 FlowRegistry에 등록하고 각각의 Flow Definition 파일에 대해 Flow ID를 부여한다. 필요에 따라 base-path를 지정해 줄 수도 있다.

```
<webflow:flow-registry id="flowRegistry" base-path="/WEB-INF/jsp/webflow">
  <webflow:flow-location path="/sales/product/addProduct-flow.xml"
    id="webflowAddProduct" />
</webflow:flow-registry>
```

위와 같이 정의할 경우 base-path를 포함하여 /WEB-INF/jsp/webflow/sales/product/addProduct-flow.xml 파일을 Flow Registry에 등록하게 된다.

1.1.1.1. Flow Registry 정의 방법

Flow Registry를 정의 하는 방법은 아래와 같다.

- path를 이용한 Flow Definition 파일 위치 직접 지정 : Flow Definition 파일의 위치를 직접 정의하여 Flow Registry에 등록 할 수 있다.

```
<webflow:flow-registry id="flowRegistry">
  <webflow:flow-location path="/WEB-INF/jsp/webflow/sales/product/addProduct-flow.xml"
    id="webflowAddProduct" />
</webflow:flow-registry>
```

- pattern을 이용한 위치 지정 : pattern을 이용하여 여러 개의 Flow Definition 파일을 한꺼번에 등록 할 수 있다.

```
<webflow:flow-registry id="flowRegistry"
  base-path="/WEB-INF/jsp/webflow/sales">
  <webflow:flow-location-pattern value="/**/*-flow.xml"/>
</webflow:flow-registry>
```

위와 같이 정의할 경우 /WEB-INF/jsp/webflow/sales 하위 모든 폴더의 -flow.xml로 끝나는 모든 파일을 Flow Registry에 등록 하게 된다.

1.1.1.2. Flow ID 생성

Flow Registry 설정 시 path를 이용하여 id 요소에 id를 지정해 줄 경우 해당 id로 Flow ID를 생성하지만 별도의 id를 정의하지 않았을 때나 pattern을 이용하여 Flow Registry를 정의했을 때에는 base-path 설정 유무에 따라 아래와 같이 Flow ID가 생성된다. Flow ID는 실제 Flow를 실행시키게 되는 요청명이 되므로 어떠한 Flow ID로 생성되는지 개발자는 인지하고 있어야 할 것이다.

- base-path를 정의할 경우 : base-path를 정의한 경우에는 실제 파일의 경로에서 base-path와 파일명을 제외한 문자열이 Flow의 ID가 된다.

```
<webflow:flow-registry id="flowRegistry" base-path="/WEB-INF/jsp/webflow">
  <webflow:flow-location path="/sales/product/addProduct-flow.xml"
```

```
id="webflowAddProduct" />
</webflow:flow-registry>
```

위와 같이 정의할 경우 Flow의 ID는 sales/product가 된다. pattern을 사용하였을 경우에도 마찬가지다. 그러나 이 때, 폴더 명이 ID가 되므로 하나의 폴더에는 하나의 Flow Definition 파일만 존재해야 한다.

- base-path를 정의하지 않을 경우 : base-path를 정의하지 않을 경우에는 Flow Definition 파일의 이름에서 확장자를 제외한 파일 이름이 Flow ID가 된다.

```
<webflow:flow-registry id="flowRegistry">
  <webflow:flow-location-pattern value="/WEB-INF/jsp/webflow/sales/**/*-flow.xml"/>
</webflow:flow-registry>
```

위에서 정의한 pattern과 일치하는 파일이 category-flow.xml, product-flow.xml 이라고 할 때 각각의 ID는 category-flow, product-flow가 된다.



생성된 flow id를 확인할 수 있는 방법은?

개발자가 flow의 id를 따로 지정안해 줄 경우 생성된 flow id는 log4j.xml 파일에서 org.springframework.webflow logger를 DEBUG 레벨로 정의 하면 확인할 수 있다. 다음은 이렇게 정의하였을 때 출력되는 로그의 일부이다.

```
[/WEB-INF/jsp/webflow/sales/category/category-flow.xml] under id 'category'
[/WEB-INF/jsp/webflow/sales/category/category-flow.xml] under id 'category-flow'
```

1.1.2.FlowExecutor 정의

Flow를 실행 시키기 위해서 FlowExecutor 배포하여야 한다.

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

1.2.Spring MVC와 연계하기 위한 설정

Spring Web Flow는 Spring MVC, JSP, Faces, Portlet 등의 환경에서 사용이 가능하다. 이 절에서는 Spring MVC와의 연계 방법에 대해 알아보게 될 것이며 기본적으로 이를 위해서는 DispatcherServlet이 정의되어 있고 Spring Web Flow로 구현될 모든 요청이 DispatcherServlet을 servlet으로 사용해야 한다.

1.2.1.FlowHandlerAdaptor 정의

Spring MVC 환경에서 Flow가 핸들링 될 수 있도록 해주기 위해 FlowHandlerAdaptor를 정의한다.

```
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerAdapter">
  <property name="flowExecutor" ref="flowExecutor" />
</bean>
```

속성 중 flowExcutor는 위에서 정의한 flowExcutor의 id를 참조하게 된다.

1.2.2.FlowHandlerMapping 정의

요청에 대한 Flow를 매핑시켜 주기위해 FlowHandlingMapping을 정의 한다.

```
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
  <property name="order" value="0" />
  <property name="flowRegistry" ref="flowRegistry" />
</bean>
```

```
</bean>
```

위에서 정의한 FlowHandlerMapping은 Spring MVC의 BeanNameHandlerMapping, SimpleHandlerMapping과 같이 interceptor, order등의 속성을 줄 수 있다. 속성 중 flowRegistry는 위에서 정의한 flowRegistry의 id를 참조 하며 실제 요청을 처리하게 될 flow 파일들을 관리하고 있는 저장소이다.



특정 URL에만 Interceptor를 적용해야 할 경우에는?

특정 URL에만 Interceptor를 적용해야 할 경우 Spring Web Flow에서 제공하는 FlowController를 이용하여 정의할 수 있다.

```
<bean class="org.springframework.
    web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="order" value="0"/>
    <property name="mappings">
        <value>
            /webflowProduct.do=flowController
        </value>
    </property>
    <property name="interceptors" ref="loginInterceptor" />
</bean>
<bean id="flowController"
    class="org.springframework.webflow.mvc.servlet.FlowController">
    <!-- 필수 -->
    <property name="flowExecutor" ref="flowExecutor"/>
</bean>
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
    <!-- order는 SimpleUrlHandlerMapping 보다 낮게 준다. -->
    <property name="order" value="1" />
    <property name="flowRegistry" ref="flowRegistry" />
</bean>
<bean id="loginInterceptor" class="common.LoginInterceptor" />
```

FlowController는 flowExecutor를 속성으로 가질 수 있으며 이를 통해 Flow가 요청을 처리할 수 있게 된다. 위와 같이 정의할 경우 "/webflowProduct.do"라는 요청에 대해서만 loginInterceptor를 적용하게 된다.

1.2.3.Spring MVC의 ViewResolver 지정

Spring MVC를 사용할 때 정의한 ViewResolver를 사용하기 위해 flow-builder-services를 등록하고 정의한 viewResolver를 참조하도록 한다. 정의하는 방법은 아래와 같은 Step을 따른다.

1. flowRegistry에 flow-builder-services 속성 추가

```
<webflow:flow-registry id="flowRegistry"
    flow-builder-services="flowBuilderServices">
    <webflow:flow-location path="/sample/product/addProduct-flow.xml"
        id="addProduct" />
</webflow:flow-registry>
```

2. flowBuilderServices 정의

```
<webflow:flow-builder-services id="flowBuilderServices"
    view-factory-creator="mvcViewFactoryCreator" development="true" />
```

3. mvcViewFactoryCreator 정의

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

```
</bean>
```

2.Flow

flow가 시작되어 종료될 때까지의 재사용 가능한 범위가 Flow 정의 파일의 단위가 된다. 이러한 flow를 정의하기 위해서 작성해야 할 요소에 대해 알아보도록 한다.

2.1.필수 요소

view-state, transition, end-state 세 요소를 통해 기본적인 view navigation을 구성할 수 있다.

2.1.1.view-state

view-state는 flow의 step을 정의하며 해당 view를 출력해주는 역할을 한다.

```
<view-state id="addProductView" model="product"
    view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
</view-state>
```

위와 같이 정의할 수 있으며 view-state의 id는 플로우내에서 유일해야 한다. model attribute로 model 객체를 정의해 줄 수 있으며 해당 변수에 대한 선언은 <var>를 이용해서 할 수 있다.

```
<var name="product" class="domain.Product" />
```

또한, view를 사용하여 출력해 줄 view 이름을 정의해 줄 수 있으며 정의해 주지 않을 시에는 view-state의 id가 view 이름이 된다. 위의 코드에서 view 이름을 따로 정의해 주지 않았다면 출력해 줄 view 이름은 addProductView가 된다. 플로우 정의 파일에 여러 개의 view-state가 정의되어 있다면 플로우가 시작될 때 실행되는 view-state는 맨 처음에 정의되어 있는 view-state가 된다.

2.1.2.transition

화면에서 일어난 event에 의해 다음으로 진행해야 할 view-state에 대해 정의해준다.

```
<view-state id="confirmAddProduct"
    view="/WEB-INF/jsp/webflow/sales/product/reviewProduct.jsp">
    <transition on="revise" to="addProductView" />
    <transition on="confirm" to="backtolist" history="invalidate">
        <evaluate expression="coreProductService.add(product)" />
    </transition>
    <transition on="cancel" to="backtolist" />
</view-state>
```

event id에 따라 위와 같이 각각의 view-state로 분기할 수 있다. 이 때, on에는 event id를 to에는 분기시킬 view-state id를 정의해 준다.



view에서 event id는 어떻게 지정해 줄까?

Spring MVC와 연동하기 위해서는 JSP 페이지 내에 특정 화면에서 발생하는 각 event에 대해 event id를 부여해 주어야 한다. 일반적으로 폼을 포함한 JSP 페이지에서는 "_eventId"라는 키의 값이 event id가 되며 아래와 같이 정의할 수 있다.

```
<input type="submit" value="save"/>
<input type="hidden" name="_eventId" value="success" />
```

submit 타입의 input 태그인 경우에는 name 속성을 "_eventId" + "_" + "transition의 on에 속하는 속성값"과 같이 정의할 수 있다.

```
<input type="submit" name="_eventId_success" value="save" />
```

단순한 HTML 링크 형식의 event일 경우에는 아래와 같이 `<a href ... >` 태그를 사용하여 event id를 정해줄 수 있다.

```
<a href="{flowExecutionUrl}&_eventId=success">success</a>
```

위에서 사용한 Expression Language인 `flowExecutionUrl`을 사용하여 현재의 `flowExecutionUrl`을 가지고 올 수 있다.

2.1.3.end-state

플로우를 종료시키기 위해 end-state를 정의해준다.

```
<end-state id="cancel" />
```



execution과 snapshot 갯수의 제한

보통 플로우를 작성하게 되면 end-state로 플로우를 종료시키게 되고 이 때, 해당 플로우에 대한 정보와 데이터가 메모리에서 지워지게 된다. 하지만 사용자의 메뉴이동이나 기타 다른 이유에 의해 플로우가 end-state로 정상 종료 되지 않았을 때는 관련 데이터가 삭제되지 않으며 메모리 오버헤드가 발생하거나 OutOfMemory exception이 발생할 수도 있다. 이에 Spring Web Flow에서는 사용자 당 execution과 snapshot의 갯수를 제한할 수 있다.

```
<webflow:flow-execution-repository max-executions="5" max0execution-
snapshots="30" />
```

위와 같이 정의 할 경우 사용자 당 execution 갯수를 5개로, execution 당 snapshot의 갯수를 30개로 제한하게 된다.

2.2.메소드 호출

플로우에서는 대부분 화면 이동에 대한 로직이 들어가게 되며 아래와 같은 시점에서 Business Service를 호출하거나 다른 클래스의 메소드를 호출할 수 있다.

- 플로우가 시작될 때

```
<on-start>
  <evaluate expression="..." />
</on-start>
```

- state에 진입할 때

```
<view-state id="addProductView" model="product"
  view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
  <on-entry>
    <evaluate expression="..." />
  </on-entry>
  ....
</view-state>
```

- view가 출력될 때

```
<view-state id="addProductView" model="product"
  view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
  <on-render>
    <evaluate expression="..." />
  </on-render>
  ....
```

```
</view-state>
```

- transition이 실행될 때

```
<view-state id="addProductView" model="product"
  view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
  <transition on="add" to="confirmAddProduct">
    <evaluate expression="..." />
  </transition>
  ....
</view-state>
```

- state가 끝날 때

```
<view-state id="addProductView" model="product"
  view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
  ....
  <on-exit>
    <evaluate expression="..." />
  </on-exit>
</view-state>
```

- flow가 끝날 때

```
<on-end>
  <evaluate expression="..." />
</on-end>
```

보통 Spring Bean으로 등록된 클래스의 메소드를 실행 시킬 수 있으며 이 경우 해당 bean을 Autowired 방식으로 찾게 된다. 또한, 일반 클래스의 메소드도 variable로 정의한 후 호출해서 쓸 수 있다.

2.2.1.evaluate

action을 실행시키기 위해서는 위에서 언급한 시점에서 <evaluate>를 사용하면 된다.

```
<evaluate expression="coreProductService.save(product)" />
```

```
<var name="list" class="java.util.ArrayList" />
<evaluate expression="list.add(product)" />
```

위에서 처럼 Spring Bean으로 정의되지 않은 클래스를 variable로 정의하여 호출할 때에는 variable로 정의된 클래스가 플로우 요청 간 인스턴스의 상태를 유지하기 위해서 java.io.Serializable를 구현하여야 한다. 메소드 실행 후 결과값을 받아야 한다면 result를 사용한다.

```
<evaluate expression="coreProductService.get(prodNo)" result="flowScope.product" />
```

2.3.Transition Decision

view-state에서 transition을 결정 할 때 단순히 사용자의 입력 값을 받아 처리할 수도 있지만 어떠한 action을 실행시킨 후 그 결과에 따라 transition을 결정할 수도 있다. 이 때, 사용할 수 있는것이 action-state와 decision-state이다.

2.3.1.action-state

action-state를 사용하면 action을 실행시킨 후 리턴 값에 의해 다음으로 진행될 transition을 정할 수 있다.

```
<action-state id="checkUserLogin">
  <evaluate expression="userService.isUserLogin(userId)"/>
  <transition on="yes" to="getCategory"/>
  <transition on="no" to="backtolist"/>
</action-state>
```

위와 같이 정의할 경우 userService의 isUserLogin() 메소드를 실행 시킨 후 리턴 값이 true일 경우 event id는 yes가 되어 getCategory로 이동하게 되고 false일 event id가 no가 되어 backtolist로 이동하게 될 것이다. action 수행 후 리턴 값에 매핑 되는 event id는 아래와 같다.

메소드 리턴 타입	event id
java.lang.String	String 값
java.lang.Boolean	yes(true일 경우), no(false일 경우)
java.lang.Enum	Enum 이름
그 밖의 타입	success

2.3.2.decision-state

decision-state는 action-state와 같은 역할을 하지만 if/else 문을 사용할 수 있다. 위에서 정의한 코드를 decision-state로 정의하면 아래와 같은 코드가 된다.

```
<decision-state id="checkUserLogin">
  <if test="UserService.isUserLogin(userId)" then="getCategory" else="backtolist" />
</decision-state>
```

2.4.Expression Language

Spring Web Flow에서는 Expression Language(EL)을 통해 모델 객체에 접근하거나 action을 실행시킬 수 있다. 이 때, Unified EL을 사용하며 현재 Spring Web Flow에서는 구현체로 jboss-el을 사용하고 있다. 개발자는 이러한 EL문을 사용하여 client가 입력한 input 데이터나 request parameter에 담겨온 데이터들에 접근할 수 있고 flowScope 같이 플로우 내부에서 사용하는 데이터에 접근할 수 있다. 또한, Spring Bean으로 정의된 class를 호출할 수도 있다.

2.4.1.Special EL variables

다음은 flow 파일 내에서 유용하게 사용할 수 있는 변수들이다.

- flowScope : flow 내에서 사용할 수 있는 변수에 할당할 수 있다. flowScope은 플로우가 요청을 처리하는 동안 존재 하게 되며 플로우가 호출될 때 생성되며 플로우가 응답을 반환하고 나면 파괴된다.

```
<evaluate expression="coreProductService.get(prodNo)" result="flowScope.product"/>
```

- viewScope : view에서 사용할 수 있는 변수에 할당할 수 있다. view 상태가 시작되서 존재하는 동안 지속된다. 또한 외부에서는 참조할 수 없다.

```
<evaluate expression="coreProductService.get(prodNo)" result="viewScope.product"/>
```



view에서는 viewScope에 있는 변수를 어떻게 접근 할까?

jsp에서 viewScope에 선언되어 있는 변수를 꺼내어 쓰고 싶을 때 JSP 내부에서 Expression Language를 사용하면 쉽게 사용할 수 있다.

```
${product}
```

- requestScope : request 내에서 사용할 수 있는 변수에 할당할 수 있다. 플로우가 요청을 처리하는 동안에 존재하며 플로우가 호출될 때 생성되고 플로우가 응답을 반환하면 파괴된다.

```
<evaluate expression="coreProductService.get(prodNo)"
  result="requestScope.product"/>
```

- flashScope : 플로우가 시작될 때 생성되며 값이 할당된 후에는 다음 뷰가 출력될 때 값이 지워진다.

```
<evaluate expression="coreProductService.get(prodNo)"
  result="flashScope.product"/>
```

- conversationScope : 최상위 플로우가 시작될 때 할당되며 최상위 플로우가 종료될 때 파괴된다. 그러므로 최상위 플로우가 가지고 있는 subflow에서도 conversationScope으로 설정된 변수에 접근할 수 있다.

```
<evaluate expression="coreProductService.get(prodNo)"
  result="conversationScope.product"/>
```

- requestParameters : client단에서 request parameter로 넘어온 값들에 대해 접근할 수 있다.

```
<set name="flowScope.userId" value="requestParameters.userId" />
```

위와 같이 정의할 시 Java에서 HttpServletRequest의 `getParameter("userId")`한 것과 같다. Spring Web Flow에서 제공하는 Expression Language에 대한 더 많은 정보는 Spring Web Flow Reference [<http://static.springsource.org/spring-webflow/docs/2.0.x/reference/htmlsingle/spring-webflow-reference.html>]를 참고 한다.

2.5.Subflow

플로우내에서 하위 플로우를 호출하려고 할 때 subflow를 이용한다. 이 때, 하위의 flow가 결과를 리턴할 때 까지 상위 플로우는 대기하게 된다.

2.5.1.subflow-state

먼저 subflow를 호출하게될 state에서 subflow-state를 정의하고 실행시킬 subflow의 id를 지정해 준다. 아래의 코드는 상위 플로우 정의 부분의 일부이다.

```
<subflow-state id="viewCategory" subflow="viewCategorySubFlow">
  <!-- transition 정의 -->
  <transition ... />
</subflow-state>
```

위에서 정의한것 처럼 viewCategory state에 오게 되면 viewCategorySubFlow라는 ID를 가진 플로우를 subflow로 실행시키게 된다.

2.5.2.input

subflow에 전달해줄 객체가 있으면 input을 사용한다.

```
<subflow-state id="viewCategory" subflow="viewCategory">
  <!-- subflow에서 필요한 data를 input 값으로 정의 -->
  <input name="categoryNo" value="product.category.categoryNo"/>
  <!-- transition 정의 -->
  <transition ... />
```

```
</subflow-state>
```

상위 플로우에서 전달해준 input값을 전달받기 위해서는 하위 플로우에서도 input을 정의해주어야 한다. 아래의 코드는 하위 플로우 정의 부분의 일부이다.

```
<!-- 상위 플로우에서 받게 될 input값 정의 -->
<input name="categoryNo"/>
<view-state id="viewCategory"
  view="/WEB-INF/jsp/webflow/sales/category/viewCategoryForFlow.jsp">
  <on-render>
    <evaluate
      expression="coreCategoryService.get(categoryNo)"
      result="category"/>
    <set name="viewScope.category" value="category"/>
  </on-render>
  <transition ... />
</view-state>
```

2.5.3.output

하위 플로우에서 플로우를 진행시킨 후에 상위 플로우에 전달해 줄 객체가 있을 경우는 output으로 정의한다.

```
<end-state id="submit">
  <!-- 상위플로우에 전달해 줄 data를 output으로 셋팅 -->
  <output name="category" value="category" />
</end-state>
```

input과 마찬가지로 하위 플로우에서 전달받은 output 객체를 상위 플로우에서도 output으로 정의해줘야 한다.

```
<output name="category" />
```

2.6.플로우 상속

플로우 파일을 작성할 때 한 플로우에서 다른 플로우를 상속받아 구현할 수 있으며 flow 레벨과 state 레벨로 가능하다. 보통 global transition이나 global exception을 parent로 정의하여 사용한다. child 플로우에서 parent 플로우를 상속받을 경우 해당 element는 override되는 것이 아니라 merge되게 된다. 단, bean-import, evaluate, exception-handler, persistence-context, render 속성에 대해서는 merge가 불가능하다.

2.6.1.flow 레벨 상속

flow 레벨 상속은 여러개의 플로우 상속이 가능하다. 이 때 콤마(,)로 구분하여 정의한다.

```
<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"
  parent="webflowParent1, webflowParent2">
```

2.6.2.state 레벨 상속

state 레벨 상속은 한 개의 state에서 한 개의 state만 상속받을 수 있다. 또한, child state와 parent state는 같은 타입이어야 한다. 예를 들어 child state가 view-state 이면 parent state도 view-state여야 한다. flow id와 state id간의 구분자는 "#"이다.

```
<view-state id="getProduct" model="product"  
  view="/WEB-INF/jsp/webflow/sales/product  
  /viewProduct.jsp" parent="webflowParent#stateParent">  
  ...  
</view-state>
```

3.View

위에서 살펴본 view-state에서 view 속성을 정의해 주면 사용자가 원하는 view를 출력해 줄 수 있다. 이 때, view 속성을 지정하지 않을 경우에 view-state의 id가 view 이름이 된다.

```
<view-state id="confirmAddProduct" view="/WEB-INF/jsp/webflow/sales/product/
reviewProduct.jsp">
```

view는 아래와 같은 방법으로 정의할 수 있다.

- 플로우로부터의 상대 경로로 지정 : 플로우의 working directory로 부터 상대적인 경로로 view를 지정할 수 있다.

```
<view-state id="confirmAddProduct"
view="reviewProduct.jsp">
```

- 절대 경로로 지정 : webapp의 루트 디렉토리부터의 경로로 view를 절대 경로로 지정할 수 있다.

```
<view-state id="confirmAddProduct"
view="/WEB-INF/jsp/webflow/sales/product/reviewProduct.jsp">
```

- 논리적인 이름으로 지정 : Spring MVC의 viewResolver를 사용하는 것과 같이 논리적 이름을 지정하여 viewResolver에 의해 view를 찾게 할 수 있다.

```
<view-state id="confirmAddProduct" view="confirmAddProductView">
```

3.1.model 바인딩

사용자가 입력한 데이터를 model 객체로 바인딩하기 위해 Spring Web Flow에서는 model 속성을 정의하여 쓸 수 있다.

```
<var name="product" class="domain.Product" />
<view-state id="getProduct" model="product"
view="/WEB-INF/jsp/webflow/sales/product/viewProduct.jsp">
```

위와 같이 정의할 경우 사용자가 입력한 데이터의 parameter 이름과 model 객체의 attribute의 이름과 일치하면 자동으로 바인딩이 된다. model 객체는 하나만 정의할 수 있으며 이렇게 정의된 model 객체에 대해 validation 체크를 할 수 있다. 보통 model 속성만 정의할 경우 해당 model 객체에 대한 모든 public attribute들이 바인딩되게 된다. <binder>를 사용하면 특정 attribute에 대해서만 바인딩 시킬 수 있다.

```
<binder>
<binding property="userId" />
<binding property="userName" />
</binder>
```

3.2.view backtracking

사용자는 어플리케이션을 이용할 때 브라우저의 뒤로가기 버튼을 이용하여 이미 끝난 view-state나 transition으로 되돌아 갈 수 있다. history 정책에 대해 Spring Web Flow에서는 history 속성 설정만으로 제어할 수 있으며 history에 대한 설정이 없을 시에는 기본적으로 backtracking이 허용된다.

3.2.1.discard

해당 뷰에 대해 backtracking을 방지 한다.

```
<transition on="confirm" to="backtolist" history="discard">  
  <evaluate expression="coreProductService.add(product)" />  
</transition>
```

3.2.2.invalidate

이전에 출력되었던 모든 뷰에 대한 backtracking을 방지 한다.

```
<transition on="confirm" to="backtolist" history="invalidate">  
  <evaluate expression="coreProductService.add(product)" />  
</transition>
```

4.Validator

Spring Web Flow에서는 Validator를 구현하여 지정된 model 객체에 대한 프로그램적인 Validation을 수행할 수 있다. Validator를 구현하는 방법에는 model 객체에 validate 메소드를 구현하는 방법과 validator 클래스 및 메소드를 구현하는 방법이 있다.

4.1.model 객체 내에 validate 메소드 구현

model 객체 내에 validate 메소드를 구현하여 해당 model 객체에 대한 validation을 수행할 수 있다. 이 때, 메소드의 이름은 일정한 룰에 따라 정의해줘야 하는데 그 이름은 "validate\${state}"가 되며 ValidationContext를 입력 argument로 받게 된다. 예를 들어, "domain.Category"라는 타입의 model 객체를 갖는 "addCategoryView" state에 대한 validate 메소드는 아래와 같이 정의할 수 있다.

```
public class Category implements java.io.Serializable {
//....
    public void validateAddCategoryView(ValidationContext context){
        MessageContext messages = context.getMessageContext();
        if(categoryName.length() <=3)
            messages.addMessage(new MessageBuilder().error().source("categoryName").
                defaultText("카테고리명은 4자 이상이어야 합니다. ").build());
    }
}
```

4.2.validator 클래스 및 메소드 구현

위와 같이 model 객체 내에 validate를 정의할 경우 각각의 model 객체가 validate 메소드를 포함하고 있는 모습이 된다. Validator 클래스를 따로 작성하여 Validation 체크 부분에 대한 코드를 따로 관리할 수 있다. 이 때, 클래스명은 "\${model}Validator"가 되며 state에 따라 수행되게될 메소드의 이름은 "validate\${state}"가 된다. 또한 입력 argument는 model 객체와 ValidationContext가 된다.

```
@Component
public class CategoryValidator {
    public void validateGetCategory(Category category, ValidationContext context) {
        MessageContext messages = context.getMessageContext();
        if (category.getCategoryName().length() <= 3)
            messages.addMessage(new MessageBuilder().error().source(
                "categoryName").code("category.length.error").build());
    }
}
```

위에 코드에서 볼 수 있듯이 Validator 클래스를 @Component로 정의하여 Bean으로 등록하여 scan할 수 있도록 한다.