# Spring Web Flow

Spring Web Flow是Spring MVC的扩展，它支持开发基于流程的应用程序，它将流程的定义与实现流程行为的类和视图分离开来。

# 1 在Spring中配置WebFlow

Spring Web Flow构建与Spring MVC之上

## 1.1 装配流程执行器

流程执行器驱动流程的执行。当用户进入一个流程时，流程执行器回味用户创建并启动一个流程执行器实例，当流程暂停的时候，流程执行器会在用户执行操作后回复流程

创建一个流程执行器

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

尽管流程执行器分则创建和执行流程，但它并不负责加载流程定义。这个职责落在了流程注册表上

# 1.2 配置流程注册表

流程注册表的工作室加载流程定义并让那个流程执行器能够古使用他们。

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

流程注册表会在"/WEB-INF/flows"目录下朝朝流程定义。flow-location-pattern定义了任意名以”-flow.xml“结尾的XML文件都将视为流程定义。

所有的流程都是通过其ID来进行引用的。这里使用flow-location-pattern元素，流程的ID就是相对于base-path的路径--或者双星号所代表的路径

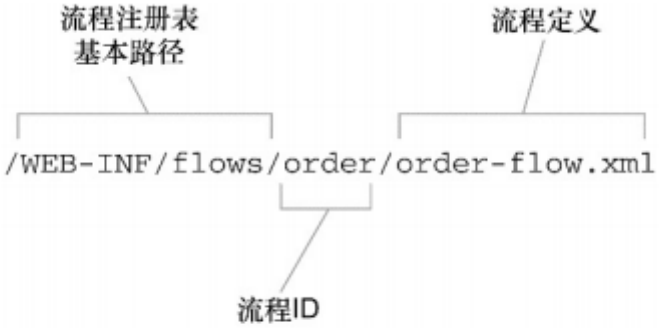


图8.1 在使用流程定位模式的时候，
流程定义文件相对于基本路径的路径将被用作流程的ID

另外一种定义方式

```
<flow:flow-registry id="flowRegistry">
    <flow:flow-location-pattern value="/WEB-INF/flows/apringpizza.xml"/>
</flow:flow-registry>
```

flow-location-pattern属性指明了"/WEB-INF/flows/apringpizza.xml"作为流程定义，流程的ID是从流程定义文件的文件名中获取的，这里是springpizza

可以通过flow:flow-location元素的id属性来设置

## 1.3 流程处理请求

对于流程，需要一个FlowHandlerMapping来帮助DispatcherServlet将流程请求发送给Spring Web Flow

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

FlowHandlerMapping 装配了流程注册表的引用，这样他就能知道如何将请求的URL匹配到流程上。例如，如果有一个id为pizza的流程，FlowHandlerMapping就会知道如果请求的URL模式是”pizza“的的化，就要将其匹配到这个流程上。

FlowHandlerMapping的工作仅仅是将流程请求定向到Spring Web Flow上，响应请求的是FlowHandlerAdapter。FlowHandlerAdapter等同于Spring MVC的控制器，他会响应发送的流程请求并将其惊醒处理

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

这个处理适配器是DispatcherServlet和Spring Web Flow之前的桥梁。他会处理请求并管理基于这些请求的流程。

# 2 流程的额组件

流程由三个主要元素定义：

- **状态** 流程中事件发生的地点。流程中的状态时业务逻辑执行、做出决策或将页面展现给用户的地方。
- **转移**
- **流程数据**

## 2.1 状态

Spring Web Flow可供选择的状态

| 类型 | todo |
| --- | --- |
| 行为（Action） | 行为状态是流程逻辑发生的地方 |
| 决策（Decision） | 决策将流程分成两个方向，它会基于流程数据的评估结果确定流程方向 |
| 结束（End） | 结束状态是流程的最后一站，一旦进入ENd状态，流程就会终止 |
| 子流程（Subflow） | 子流程状态会在当前正在运行的流程上下文中共启动一个新的流程 |
| 视图（View） | 视图状态会暂停流程并邀请用户参与流程 |

### 2.1.1 视图状态

视图状态用于为用户展现信息并使用户在流程中发挥作用。实际的视图实现可以是Spring支持的任意视图类型。通常是JSp来实现。

```
<view-state id="welcome"/>
```

id属性由两个含义。它在流程中标示这个状态。此外，因为没有指定视图。所以他爷制定了流程到达这个状态时要展现的逻辑视图名为welcome。同时也可以显显式指定另外一个视图名

```
<view-state id="welcome" view="greeting"/>
```

如果流程未用户展现了一个表单，并指明表单绑定的对象

```
<view-state id="registrationForm" model="order">
```

这里指定registrationForm视图中的表单将绑定流程作用域内的order对象

### 2.1.2 行为状态

视图状态会涉及到流程应用程序的用户，而行为状态则是应用程序自生在执行任务。行为状态一般会触发Spring所管理bean的一些方法并根据方法调用的执行结果转移到另一个状态。

```
<action-state id="lookupCustomer">
    <evaluate result="order.customer" expression=
            "pizzaFlowActions.lookupCustomer(requestParameters.phoneNumber)"/>
    <transition to="registrationForm" on-exception=
            "com.springinaction.pizza.service.CustomerNotFoundException"/>
    <transition to="customerReady"/>
</action-state>
```

尽管不是严格要求的，但是元素一般都会有一个作为子元素，给出行为状态要做的事情，expression制定了进入这个状态时要评估的表达式。(找到 id未pizzaFlowActions的bean，并执行lookupCustomer方法)。此外还可以使用SpEL。

### 2.1.3 决策状态

有可能流程会按照线性执行，从一个状态进入另一个状态，没有其他的替代路线。但更常用的情况是流程在某一点更具流程的当前情况进入不同的分支。

决策状态能偶在流程执行时产生两个分支。决策状态将评估一个boolean类型的表达式，然后再两个状态转移中选择一个。

```
<decision-state id="checkDeliveryArea">
    <if test="pizzaFlowActions.checkDeliveryArea(order.customer.zipCode)"
        then="addCustomer"
        else="deliveryWarning"/>
</decision-state>
```

### 2.1.4 子流程状态

```
<subflow-state id="order" subflow="pizza/order">
    <input name="order" value="order"/>
    <transition on="orderCreated" to="payment"/>
</subflow-state>
```

元素用于产地订单对象作为子流程的输入。如果子流程结束的状态Id未orderCreated，那么流程将会转移到名为payment的状态。

### 2.1.5 结束状态

最后所有的流程都要结束。这就是当流程转移到结束状态时要做的。元素制定了流程的结束

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

当到达状态，流程结束。接下来发生什么取决于：

- **如果结束的是一个子流程，那调用它的流程将会从处继续执行。的UD将会用作时间触发从开始的转移**
- **如果设置了view属性，指定的视图将会被渲染。视图可以是相对于流程路径的试图模板，如果添加了"externalRedirect：前缀的化，将会重定向到流程外部的页面，如果添加"flowRedirect：“将会从定向到另一个流程**
- **如果结束的流程不是子流程，也没有指定view属性，那么这流程只是会结束而已。浏览器最后将会加载流程的基本URL地址，当前已没有活动的流程，所以会开始一个新的流程实例。**

需要注意到流程可能会有不止一个结束状态。子流程的结束状态ID确定了激活的事件，所以可能会希望通过多种结束状态来结束子流程，从能能够再调用流程中触发不同的事件。即使不是在子流程中，也有可能再流程结束后，根据流程的执行情况有多个显示页面供选择。

## 2.2 转移

**转移能够连接流程中的状态。流程中除了结束状态之外的每个状态，至少都需要一个转移，这样就能够知到一旦这个状态完成时流程要去向哪里。状态可以有多个转移。对应于当前状态结束时可以执行的不同路径。

```
<subflow-state id="order" subflow="pizza/order">
    <input name="order" value="order"/>
    <transition on="orderCreated" to="payment"/>
</subflow-state>
```

会作为各种状态元素的子元素。属性to用于指定六册灰姑娘的下一个状态。只用了to属性，这个转移就会是当前状态的默认转移选项，如果没有其他可用转移，就会使用它。

更常见的是基于世家出发来进行的。

- 视图状态，时间通常是用户采取的动作。
- 行为状态，事件是评估表达式得到的结果。

- 子流程中，事件取决于子流程结束状态的ID

  再任意的事件中，可以使用on属性来指定出发转移的事件。

再抛出异常时，流程也可以进入另一个状态。

```
<transition on-
exception="com.springinaction.pizza.service.CustomerNotFoundException"
            to="registrationForm"/>
```

### 全局转移

创建流程之后，发现有下状态使用了一个通用的转移

```
<transition on="cancel" to="cancel"/>
```

与其再多个状态中都重复通用的转移，我们可以将元素作为的子元素，把他们定义为全局转移。

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

## 2.3 流程数据

### 2.3.1 声明变量

流程数据保存在变量中，而变量可以在流程的各个地方进行引用

```
<var name="order" class="com.springinaction.pizza.domain.Order"/>
```

- 作为行为状态的一部分或者作为视图状态的入口，可以使用元素来创建变量

```
<action-state id="verifyPayment">
    <evaluate result="order.payment" expression=
            "pizzaFlowActions.verifyPayment(flowScope.paymentDetails)"/>
    <transition to="paymentTaken"/>
</action-state>
```

evaluate元素计算了一个表达式的结果并将结果放到payment的变量中

- 元素也可以设置变量的值

  ```
  <set name="flowScope.pizza" value="new
  com.springinaction.pizza.domain.Pizza()"/>
  ```

### 2.3.2 定义流程数据的作用域

流程中携带的数据会拥有不同的生命作用域和可见性，这取决于保存数据的变量本身的作用域。

| 范围 | 声明作用域和可见性 |
| --- | --- |
| Conversation | 最高层级的流程开始时创建，在最高层级的流程结束时销毁。被最高层级的流程和其他所有的子流程共享 |
| Flow | 流程开始时创建，流程结束时销毁。只有在创建它的流程中时可见的 |
| Request | 当一个请求进入流程时创建，在流程返回时销毁 |
| Flash | 当流程开始时创建，流程结束时销毁。在视图状态选然后，他会被清除 |
| View | 当进入视图状态时创建，当这个状态退出时销毁。只在视图状态内时可见的 |

*元素声明变量时，变量是中共是流程作用域的，也就是在定义变量的流程中有效。当使用或，作用域通过name或result属性的前缀指定*

https://github.com/giantwugui/JavaStudy/tree/master/SprintStudy/WebFlow