阅读笔记8

第8章

Spring web Flow

这是一个web框架 是适用于元素按规定流程运行的程序
Spring web Flow是Spirng MVC的扩展
（支持开发流程的应用程序）
Spring web Flow不是Spring的框架 而是Spirng的子框架

```
<?xml version="1.0" encoding="UTF-8"?><beansxmlns="http://www.springframework.org/schema/beans"xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"xmlns:flow="http://www.springframework.org/schema/webflow-config"xmlns:p="http://www.springframework.org/schema/p"xmlns:context="http://www.springframework.org/schema/context"xsi:schemaLocation="http://www.springframework.org/schema/webflow-config
   http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.3.xsd
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">
```

这就是这个命名空间声明

我们首先要准备织入流程执行器

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

流程执行器 驱动流程的执行 当用户进入一个流程时 流程执行器会为用户创建并启动一个流程执行实例

但是这个本身不负责加载流程定义 这个是落在流程注册表身上的

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

这里的声明中 流程注册表会在WEB/INF/flows 目录下查找流程定义 这是通过bash-path属性定义的 更具有flow-location-pattern这个值 任何以flow-xml文件都视为流程定义
流程都是通过ID来进行引用的 这里流程的ID相对于bash-path的路径--或者双星号所代表的路径

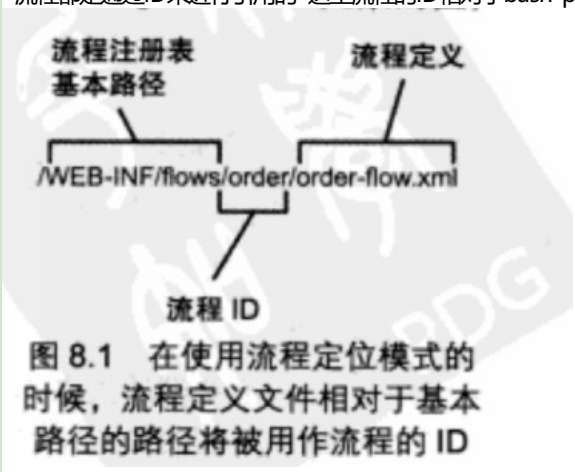


图 8.1 在使用流程定位模式的时候，流程定义文件相对于基本路径的路径将被用作流程的 ID

我们还可以直接去出bash-path这个属性 直接显示定义这个流程定义的位置

```
<flow:flow-registry id="flowRegistry"  >
       <flow:flow-location  path="WEB-INF/flows/springpizza.xml"/>
</flow:flow-registry>
```

这样的流程的ID就是从流程定义的文件的文件名中获得 在这里就是springpizza
当然 我们可以更明显的定义这个流程的ID 由我们在这里自己定义

```
<flow:flow-registry id="flowRegistry"  >
       <flow:flow-location id="spizza"
path="WEB-INF/flows/springpizza.xml"/>
</flow:flow-registry>
```

这样这个ID就是spizza了

同时 我们还需要一个控制器将DispatcherServlet将流程请求发送给Spring web Flow
这个就是FlowHandlerMapping

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

这个FlowHandlerMapping装配了flowRegistry的引用 这样就能知道将请求的URL匹配到流程上

知道如何将请求的 URL 匹配到流程上。例如，如果我们有一个 ID 为 pizza 的流程，FlowHandlerMapping 就会知道如果请求的 URL 模式（相对于应用程序的上下文路径）是“/pizza”的话，就要将其匹配到这个流程上。

然而，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，它会响应发送的流程请求并对其进行处理。FlowHandlerAdapter 可以像下面这样装配成一个 Spring Bean，如下所示：

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

我们还需要一个响应的控制器 FlowHandlerAdpter

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

这个处理适配器是DispatcherServlet与Spring web Flow连接的桥梁 这里装配了流程执行器的引用

另外附一个这个源码写好的配置方式 这个多加了几个bena 但是没有找到相关的解释 这个应该就是能够**成功运行的**

```
<?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:flow="http://www.springframework.org/schema/webflow-config"
xmlns:p="http://www.springframework.org/schema/p"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/webflow-config
   http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <context:component-scan base-package="com.springinaction.pizza.flow" />



  <flow:flow-executor id="flowExecutor"
flow-registry="flowRegistry" />


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



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

```
  <bean id="mvcViewFactoryCreator" class=
      "org.springframework.webflow.mvc.builder.MvcViewFactoryCreator">
     <property name="defaultViewSuffix" value=".jspx" />
  </bean>



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

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



</beans>
```

另外这是MVC的xml

```
<?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-2.0.xsd">

<!-- View resolver for the pizza flow, as shown on page 594 -->
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jspx" />
  </bean>
</beans>
```

另外说明一点这个我们需要的web flow的配置文件是需要在web.xml中进行声明加载的

```
<listener>
 <listener-class>org.springframework.web.context.ContextLoaderListener
</listener-class>
</listener>
```

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value> /WEB-INF/flow.xml
</param-value>
</context-param>
```

当然还有其他的东西的话 也在这里进行加载 这里我将其他的东西进行去掉了 为了比较容易看

流程的组件
在Spring web Flow中 流程时由三个主要元素定义的 状态 转移 流程数据

表 8.1　　Spring Web Flow 可供选择的状态

状态类型	它是用来做什么的
行为（Action）	行为状态是流程逻辑发生的地方
决策（Decision）	决策状态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

视图状态
这里的ID有俩重含义 他在流程里标示这个状态 初次以为 因为在这里没用其他地方指定视图 它就指定了流程到达这个状态时所需要展现的逻辑视图名称为welcome

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

如果你愿意显式指定另外一个视图名称，那么就可以使用 view 属性做到这一点：

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

如果流程为用户展现了一个表单，你可能希望指明表单所绑定的对象。为了做到这一点，可以设置 model 属性：

```
<view-state id="takePayment" model="flowScope.paymentDetails"/>
```

这里我们指定了 takePayment 视图将绑定流程范围内的 paymentDetails 对象（稍后，我们将会更详细地介绍流程范围和数据）。

行为状态
行为状态就是应用程序自身在执行任务 行为状态一般会出发Spring所管理的Bean的一些方法 并根据方法调用的执行方法结果转移到另一个状态去

```
<action-state id="saveOrder">
      <evaluate expression="pizzFlowActions.saveOrder(order)"/>
      <transition to="ThakYou"/>
</action-state>
```

尽管不是严格需要的，但是 <action-state> 元素一般都会有一个 <evaluate> 元素作为子元素。<evaluate> 元素给出了行为状态要做的事情。expression 属性指定了进入这个状态时要评估的表达式。在本示例中，给出的 expression 是 SpEL[2] 表达式，它表明将会找到 ID 为 pizzaFlowActions 的 Bean 并调用其 saveOrder() 方法。

决策状态

有可能流程会完全按照线性执行，从一个状态进入另一个状态，没有其他的替代路线。但是更常见的情况是流程在某一个点根据流程的当前情况进入不同的分支。

决策状态能够使得在流程执行时产生两个分支。决策将评估一个 Boolean 类型

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

可以看到，<decision-state> 并不是独立完成工作的。<if> 元素是决策状态的核心。这是表达式进行评估的地方，如果表达式结果为 true，流程将转移到 then 属性指定的状态中，如果结果为 false，流程将会转移到 else 属性指定的状态中。

子流程状态
将逻辑分散到多个类 方法以及其他结构 将流程分为独立的部分是个不错的主意

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

在这里，<input>元素作为子流程的输入被用于传递订单对象。如果子流程结束的<end-state>状态ID为orderCreated，那么本流程将会转移到名为payment的状态。

结束状态
流程结束的状态

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

当到达<end-state>状态，流程会结束。接下来会发生什么取决于几个因素。

- 如果结束的流程是一个子流程，那调用它的流程将会从<subflow-state>处继续执行。<end-state>的ID将会用作事件触发从<subflow-state>开始的转移。
- 如果<end-state>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的路径也可以是流程模板，加externalRedirect：前缀将重定向到流程外部的页面而添加flowRedirect：将重定向到另一个流程中。
- 如果结束的流程不是子流程也没有指定view属性，那这个流程只是会结束。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。

流程可能是不止一个结束状态 可以多种状态来进行结束子流程 从而触发不同的事件

转移

转移连接了流程中的状态 流程中除结束状态的每个状态 至少需要一个转移 状态可以有多个转移 分别对应当前状态结束后执行不同的路径
转移

```
<transition to="customerReady"/>
```

来进行定义

```
<transition on="orderCreated" to="payment"/>
```

性来指定触发转移的事件：

```
<transition on="phoneEntered" to="lookupCustomer"/>
```

在本例中，如果触发了phoneEntered事件流程，将会进入lookupCustomer状态。

在抛出异常 流程也可以进入另外一个状态

```
<transition on-exception="com.sun.accessibility.internal" to="checkDeilverArea"/>
```

还有这种on-exception属性 只不过这个是指定了要发生转移的异常而不是一个事件

全局转移

顾名思义 就是一种通用的使用转移的方式 这样就能省代码了

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

</global-transitions>
```

定义完这个cancel之后 所有状态都是默认拥有这个转移

流程数据

当流程从一个地方到另外一个地方的时候 会带走一些数据 这就是流程数据

1 声明变量
流程数据保存这变量中 而变量可以这流程的任意地方进行引用

```
<var name="customer" class="com.jamesmurty.utils.XMLBuilder"/>
```

作为**行为状态的一部分** 或者作为视图状态的入口 你有可能会使用<evaluate>元素来**创建变量**

```
<evaluate  result="viewScope.toppingList" expression="T(org.springframework.cache.ehcache.pac
kage-org.springframework.cache.ehcache.package-info)"/>
```

还可以用set来进行设置变量的值

```
<set name="folowScope.pizza" value="new ognl.enhance.ContextClassLoader"></set>
```

<set> 元素与 <evaluate> 元素很类似，都是将变量设置为表达式计算的结果这里，我们设置了一个流程范围的 pizza 变量，它的值是 Pizza 对象的新实例。

定义流程数据的作用域

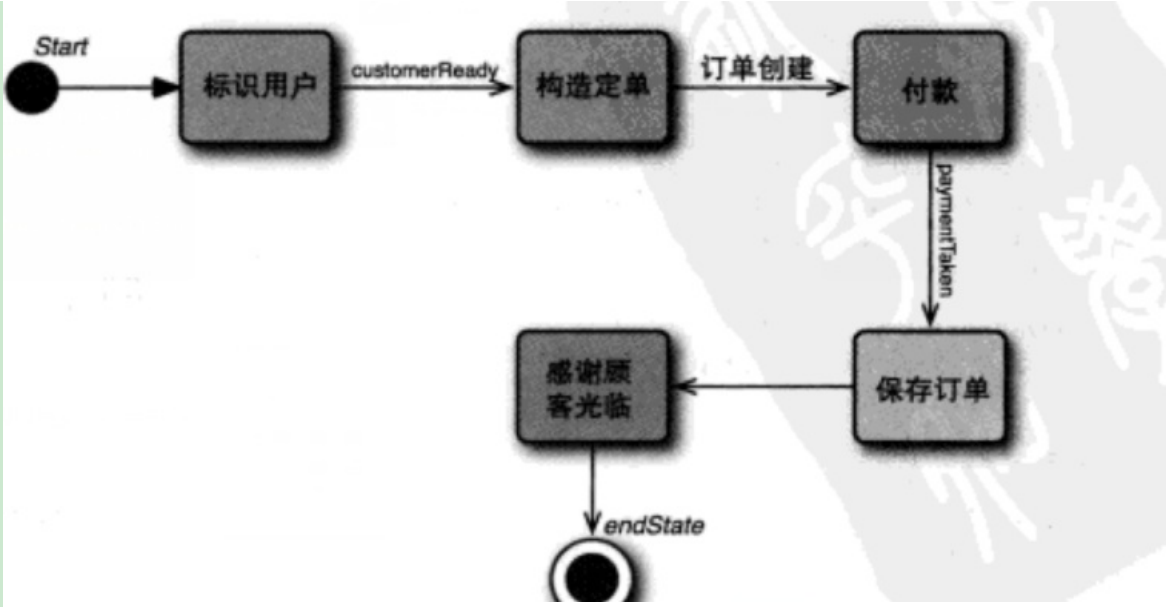
表 8.2 在流程中，数据可以存活的作用域

范围	生命作用域和可见性
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所有的子流程所共享
Flow	当流程开始时创建，在流程结束时销毁。只有在创建它的流程中是可见的
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，在流程结束时销毁。在视图状态渲染后，它也会被清除
View	当进入视图状态时创建，当这个状态退出时销毁。只在视图状态内是可见的

当使用 <var> 元素声明变量时，变量始终是流程作用域的，也就是在流程作用域内定义变量。当使用 <set> 或 <evaluate> 时，作用域通过 name 或 result 属性的前缀指定。例如，将一个值赋给流程作用域的 theAnswer 变量：

```
<set name="flowScope.theAnswer" value="42"/>
```

首先我们需要定义一个基本的订购披萨的流程



代码如下

```
<?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">

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

  <subflow-state id="identifyCustomer" subflow="pizza/customer"> <!--<co id="co_customer_subf
low"/>-->
<output name="customer" value="order.customer"/>
    <transition on="customerReady" to="buildOrder" />
  </subflow-state>

  <subflow-state id="buildOrder" subflow="pizza/order"> <!--<co id="co_order_subflow"/>-->
<input name="order" value="order"/>
    <transition on="orderCreated" to="takePayment" />
  </subflow-state>

    <subflow-state id="takePayment" subflow="pizza/payment"> <!--<co id="co_payment_subflow"/
>-->
<input name="order" value="order"/>
    <transition on="paymentTaken" to="saveOrder"/>
  </subflow-state>

  <action-state id="saveOrder"> <!--<co id="co_saveOrder_action"/>-->
<evaluate expression="pizzaFlowActions.saveOrder(order)" />
    <transition to="thankCustomer" />
  </action-state>

  <view-state id="thankCustomer"> <!--<co id="co_thankCustomer_view"/>-->
<transition to="endState" />
  </view-state>


  <end-state id="endState" />

  <global-transitions>
    <transition on="cancel" to="endState" /><!--<co id="cancel_global_transition"/>-->
</global-transitions>
</flow>
```

我们在上面涉及到了一个关键的变量 叫order 具体的声明如下 每次流程开始的时候 都会创建一个Order实例 Order类会包含关于订单的所有信息 包含顾客信息 订购的披萨 以及具体的支付详情

在这里的具体流程中 我们使用了**<output>来装配一个order的customer属性** bulidOrder和takepayment采用了不同的方式 使用<input>将order流程变量作为输入 这些子流程就能在其内部填充order对象

当得到顾客细节 披萨以及支付情况后 就可以对其进行保存了 转移到了saveOrder是处理这个任务的行为状态 使用**<evaluate>来调用ID为pizzaFlowActions的bena 的saveOrder（ ）方法** 并将保存的对象传递进来 然后就转移到了最后的thakCustomer 这就是简单的视图状态 然后就直接在这个视图页面中直接进行点击结束的按钮发生了直接转发到结束状态就没了

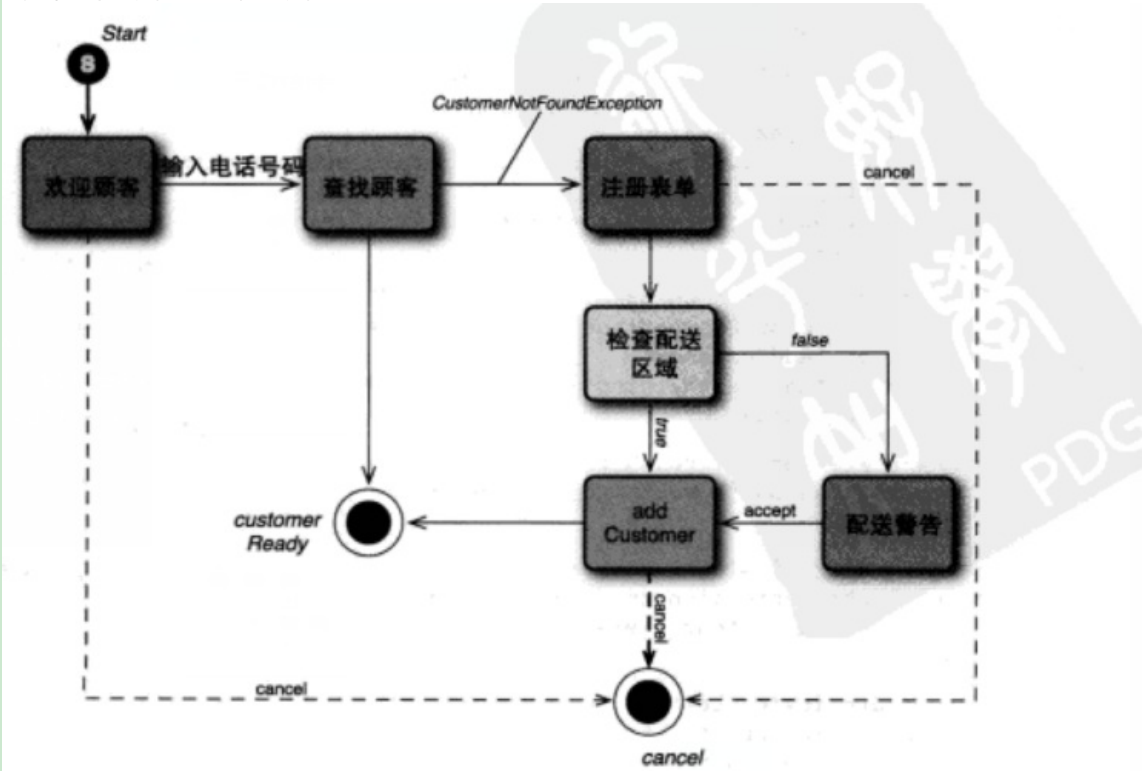
```
public class Order implements Serializable {
private static final long serialVersionUID = 1L;

   private Customer customer;
   private List<Pizza> pizzas;
   private Payment payment;

   public Order() {
pizzas = new ArrayList<Pizza>();
customer = new Customer();
}

public Customer getCustomer() {
return customer;
}

public void setCustomer(Customer customer) {
this.customer = customer;
}

public List<Pizza> getPizzas() {
return pizzas;
}

public void setPizzas(List<Pizza> pizzas) {
this.pizzas = pizzas;
}

public void addPizza(Pizza pizza) {
pizzas.add(pizza);
}

public float getTotal() {
return 0.0f;
}

public Payment getPayment() {
return payment;
}

public void setPayment(Payment payment) {
this.payment = payment;
}
}
```

这 就是视图tankCustomer的视图状态达到的页面 注意这里的**flowExecutionUrl变量** 这里包含着流程的URL 结束链接将一个**_eventId参数**关联到URL上 以便返回的时候出发finsihed事件 达到结束状态

```
<html xmlns:jsp="http://java.sun.com/JSP/Page">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />

  <head><title>Spizza</title></head>

  <body>
   <h2>Thank you for your order!</h2>

<![CDATA[
    <a href='${flowExecutionUrl}&_eventId=finished'>Finish</a> <!--<co id="event_finished"/>-
->
   ]]>
```

```
</body>
</html>
```

收集顾客信息
这是第一个子流程我们需要的流程




```
<?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">

  <var name="customer" class="com.springinaction.pizza.domain.Customer"/>

  <view-state id="welcome"> <!--<co id="welcome_view_state"/>-->
<transition on="phoneEntered" to="lookupCustomer"/>
  </view-state>

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

  <view-state id="registrationForm" model="customer"><!--<co id="registration_view_state"/>--
>
<on-entry>
      <evaluate expression=
          "customer.phoneNumber = requestParameters.phoneNumber" />
    </on-entry>
    <transition on="submit" to="checkDeliveryArea" />
  </view-state>

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

```
  <view-state id="deliveryWarning"><!--<co id="delivery_warning_view_state"/>-->
<transition on="accept" to="addCustomer" />
  </view-state>

  <action-state id="addCustomer"><!--<co id="add_customer_action_state"/>-->
<evaluate expression="pizzaFlowActions.addCustomer(customer)" />
    <transition to="customerReady" />
  </action-state>

  <end-state id="cancel" />
  <end-state id="customerReady">
    <output name="customer" />
  </end-state>

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

这里的就是我们在第一个identifyCustomer
中定义的子流程对应的流程

首先就是一个简单的welcome的视图状态 然后在对应的视图中也有相对的代码如下

```
<view-state id="welcome"> <!--<co id="welcome_view_state"/>-->
<transition on="phoneEntered" to="lookupCustomer"/>
</view-state>
```

```
<html xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:form="http://www.springframework.org/tags/form">
  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />

  <head><title>Spizza</title></head>

  <body>
    <h2>Welcome to Spizza!!!</h2>

    <form:form>
      <input type="hidden" name="_flowExecutionKey"
value="${flowExecutionKey}"/> <!--<co id="flowExecutionKey"/>-->

<input type="text" name="phoneNumber"/><br/>

      <input type="submit" name="_eventId_phoneEntered"
value="Lookup Customer" />  <!--<co id="phoneEnteredEvent"/>-->
</form:form>
  </body>
</html>
```

这里要注意的_**flowExcutionKey输入域** 当进入视图状态时 流程暂停并等待用户采取一些行为 赋予视图的流程执行键（flow execution key）就是一种返回流程的**回程票（claim ticket）** 当用户提交表单的时候 流程执行键将会在**flowExecutionKey输入域**中返回并在流程暂停的位置进行恢复
还有注意这个按钮的名字 **按钮名字的_eventId_部分是Spring web FLOW的一个线索** 它表明接下来**要触发事件** 点击后就触发phoneEntered事件进而转移到lookupCustomer

查找顾客

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

当欢迎表单提交后 顾客的电话号码将讲包含在请求参数中并准备用于查询顾客 lookupCustomer状态的<evaluate>元素是查找发生的位置 它将电话号码从请求参数中抽取出来并传递到pizzaFlowAcions Bean的lookupCustomer（ ）方法中
这里定义了了 俩个转移状态 一个是出错 一个是到下一个

注册新顾客

这个registrtionForm状态时要求用户填写配送地址的 在xml中 我们配置了一个model属性 这个就是跟下面表单的使用的customer是同一个 这里不是通过请求参数一个个处理输入域的 而是以更好的方式将表单绑定到Customer对象上 --让框架来做累的工作 另外我们还使用了一个<on-entry>配置了一个<evaluate>
这个evaluate元素是计算一个表达式（SPEL）

```
<view-state id="registrationForm" model="customer"><!--<co id="registration_view_state"/>-->
<on-entry>
    <evaluate expression=
        "customer.phoneNumber = requestParameters.phoneNumber" />
  </on-entry>
  <transition on="submit" to="checkDeliveryArea" />
</view-state>
```

```
<html xmlns:c="http://java.sun.com/jsp/jstl/core"
xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:spring="http://www.springframework.org/tags"
xmlns:form="http://www.springframework.org/tags/form">

  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />

  <head><title>Spizza</title></head>

  <body>
    <h2>Customer Registration</h2>

    <form:form commandName="customer">
      <input type="hidden" name="_flowExecutionKey"
value="${flowExecutionKey}"/>
      <b>Phone number: </b><form:input path="phoneNumber"/><br/>
      <b>Name: </b><form:input path="name"/><br/>
      <b>Address: </b><form:input path="address"/><br/>
      <b>City: </b><form:input path="city"/><br/>
      <b>State: </b><form:input path="state"/><br/>
      <b>Zip Code: </b><form:input path="zipCode"/><br/>
      <input type="submit" name="_eventId_submit"
value="Submit" />
      <input type="submit" name="_eventId_cancel"
value="Cancel" />
    </form:form>
   </body>
</html>
```

检查配送地址的

这里使用了一个决策状态

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

这里面就样子就是跟我们感觉的一模一样 跟这个本来翻译的意思也很接近
这个方法本身会返回一个Boolean值 true就是第一个 false就是第二个
如果是第一个就到addCustomer状态
否则就是deliverwarning视图状态

```
<view-state id="deliveryWarning"><!--<co id="delivery_warning_view_state"/>-->
```

```
<view-state id="deliveryWarning"><!--<co id="delivery_warning_view_state" />-->
<transition on="accept" to="addCustomer" />
</view-state>

<action-state id="addCustomer"><!--<co id="add_customer_action_state"/>-->
<evaluate expression="pizzaFlowActions.addCustomer(customer)" />
  <transition to="customerReady" />
</action-state>
```

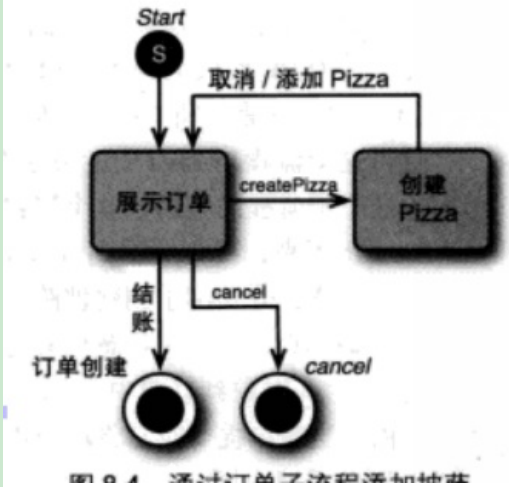
到了addCustomer状态 用户在这个时候已经输过了地址 这个地址我们会存起来 这个存起来的话 我们就又用到了这个<evaluate>表达式 这个表达式匹配了这么一个Bean与方法 最后会进行默认的转移 进入最后的结束状态

结束流程

```
<end-state id="cancel" />
<end-state id="customerReady">
  <output name="customer" />
</end-state>

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

当子流程完成的时候 它会触发一个与结束ID相同的流程事件 如果流程只有一个结束状态的话 那么它会始终触发相同的事件 当然如果有不同的结束状态 就会影响调用状态的方向 当这个customer流程完成所有的路径后 他就最终会到达ID为custmerReady的结束状态 **当调用他的结束状态恢复时 它会接受到一个customerReady事件 这个事件使得进程转移到buildOrder状态**
注意的是这个customerReady结束状态包含了一个**<output>元素 这个元素等同与java中的return**
它从子流程中传递一些数据到调用流程 这里传递的是customer流程变量 给下一个子流程
如果**在这里识别顾客的任意地方触发了cancel事件 将会导致ID为cancel的结束状态退出流程** 这也会在披萨流程中触发cancel事件并导致转移 到披萨流程的结束状态

下面识别完顾客之后 主流程的下一件事件就是确定他们想要的是什么类型的披萨



可用看到showOrder是在流程的中心位置 这是用户到这个流程时看到的第一个状态 也是之后添加披萨到订单后要转移的状态
下面是关于整个Order流程的代码

```
<?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">

  <input name="order" required="true" />

  <view-state id="showOrder">
    <transition on="createPizza" to="createPizza" />
    <transition on="checkout" to="orderCreated" />
    <transition on="cancel" to="cancel" />
  </view-state>

  <view-state id="createPizza" model="flowScope.pizza">
    <on-entry>
```

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

      <evaluate result="viewScope.toppingsList"  expression=
                 "T(com.springinaction.pizza.domain.Topping).asList()" />
    </on-entry>
    <transition on="addPizza" to="showOrder">
      <evaluate expression="order.addPizza(flowScope.pizza)" />
    </transition>
    <transition on="cancel" to="showOrder" />
  </view-state>

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

要添加披萨到订单时 流程会转移到createPizza 状态 这是另外一个视图状态
客户可以选择添加或取消披萨 俩个事件都会使流程转移回showorder状态

这个子流程事件上回操作主流程创建的Order对象 因此 我们需要以某种方法将Order从主流程传递到子流程 注意这里的<input>元素来将Order传递进流程 在这里 使用它来接受Order对象

这个showOrder状态 这是一个基本的视图状态 但是这个状态有三个不同的转移 分别用于创建披萨 提交订单 取消订单
这是createPizza状态更有意思一些 注意这里的<on-entry>元素 定义了一个新的Pizza对象到流程作用域内 当表达提交时将它填充进订单 需要注意的是 **这个视图状态引用的model是流程作用域的同一个Pizza对象**
Pizza对象绑定到创建披萨的表单
当这下面这个页面中 我们点击continue按钮提交订单后 具体的尺寸和配料选择将会绑定到Pizza对象中并且触发addPizza转移 **与这个转移关联的<evaluate>元素表面这转移到showOrder状态之前 流程作用域内的Pizza对象将会传递给addPizza()方法中**
用户可以点击showOrder视图中的Cancel按钮后缀checkout按钮 然后进行不同的结束状态的转移
然后我们就进入了下一个子流程 这就是支付的流程

```
<div xmlns:form="http://www.springframework.org/tags/form"
xmlns:jsp="http://java.sun.com/JSP/Page">

  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />

   <h2>Create Pizza</h2>
   <form:form commandName="pizza">
     <input type="hidden" name="_flowExecutionKey"
value="${flowExecutionKey}"/>

     <b>Size: </b><br/>

  <form:radiobutton path="size"
label="Small (12-inch)" value="SMALL"/><br/>
  <form:radiobutton path="size"
label="Medium (14-inch)" value="MEDIUM"/><br/>
  <form:radiobutton path="size"
label="Large (16-inch)" value="LARGE"/><br/>
  <form:radiobutton path="size"
label="Ginormous (20-inch)" value="GINORMOUS"/>
      <br/>
     <br/>

     <b>Toppings: </b><br/>
     <form:checkboxes path="toppings" items="${toppingsList}"
delimiter="&lt;br/&gt;"/><br/><br/>


     <input type="submit" class="button"
name="_eventId_addPizza" value="Continue"/>
     <input type="submit" class="button"
name="_eventId_cancel" value="Cancel"/>
   </form:form>
</div>
```

下面就由主流程的页面转到了子流程下的支付子流程

和其他流程一样 用一个<input>接受一个Order元素作为输入 开始的话就直接进入takepayment这个视图状态

```
<?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">

  <input name="order" required="true"/>

  <view-state id="takePayment" model="flowScope.paymentDetails">
    <on-entry>
      <set name="flowScope.paymentDetails"
value="new com.springinaction.pizza.domain.PaymentDetails()" />

      <evaluate result="viewScope.paymentTypeList" expression=
               "T(com.springinaction.pizza.domain.PaymentType).asList()" />
    </on-entry>
    <transition on="paymentSubmitted" to="verifyPayment" />
    <transition on="cancel" to="cancel" />
  </view-state>

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

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


</flow>
```

这个视图状态里有选择使用信用卡支付还是 支票还是现金 提交后就进入verifyPayment状态 这是一个行为状态 表示这个支付信息是否可以被接受

在流程进入 takePayment 视图时，<on-entry> 元素将构建一个支付表单并使用 SpEL 表达式在流程范围创建 PaymentDetails 实例。这实际上是表单背后的对象。它也会创建视图作用域的 paymentTypeList 变量，这个变量是一个包含了 PaymentTypeenum（如程序清单 8.11 所示）的值的列表。在这里，SpEL 的 T() 操作用于获得 PaymentType 类，这样就可以调用静态的 asList() 方法。

这下面是takepayment视图状态对应的页面代码

```
<div xmlns:form="http://www.springframework.org/tags/form"
xmlns:jsp="http://java.sun.com/JSP/Page">

  <script>
function showCreditCardField() {
var ccNumberStyle = document.paymentForm.creditCardNumber.style;
ccNumberStyle.visibility = 'visible';
}

function hideCreditCardField() {
var ccNumberStyle = document.paymentForm.creditCardNumber.style;
ccNumberStyle.visibility = 'hidden';
}
</script>

  <jsp:output omit-xml-declaration="yes"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8" />

   <h2>Take Payment</h2>
   <form:form commandName="paymentDetails" name="paymentForm">
     <input type="hidden" name="_flowExecutionKey"
```

```
value="${flowExecutionKey}"/>

    <form:radiobutton path="paymentType"
value="CASH" label="Cash (taken at delivery)"
onclick="hideCreditCardField()"/><br/>
    <form:radiobutton path="paymentType"
value="CHECK" label="Check (taken at delivery)"
onclick="hideCreditCardField()"/><br/>
    <form:radiobutton path="paymentType"
value="CREDIT_CARD" label="Credit Card:"
onclick="showCreditCardField()"/>


     <form:input path="creditCardNumber"
cssStyle="visibility:hidden;"/>

     <br/><br/>
     <input type="submit" class="button"
name="_eventId_paymentSubmitted" value="Submit"/>
     <input type="submit" class="button"
name="_eventId_cancel" value="Cancel"/>
   </form:form>
</div>
```