

Spring

Web-Flow

Framework

Reference **beta**

with Korean

제 작: 박찬욱

메 일: chanwook.god@gmail.com

블로그: <http://chanwook.tistory.com>

한국 스프링 사용자 모임: <http://ksug.org>



최초 배포 : 2009.01.05

스프링 웹-플로우(SWF) 프레임워크 레퍼런스 번역 문서 베타 버전입니다.

2.0.5 버전을 대상으로 한 번역 문서입니다. 특히 본 문서는 공식 레퍼런스 번역 문서가 아닌 학습을 위해 번역한 문서로, 일부 내용을 임의로 수정하여 작성했음을 말씀 드립니다.

수정, 배포는 자유입니다. 출처 정도만 남겨주세요.

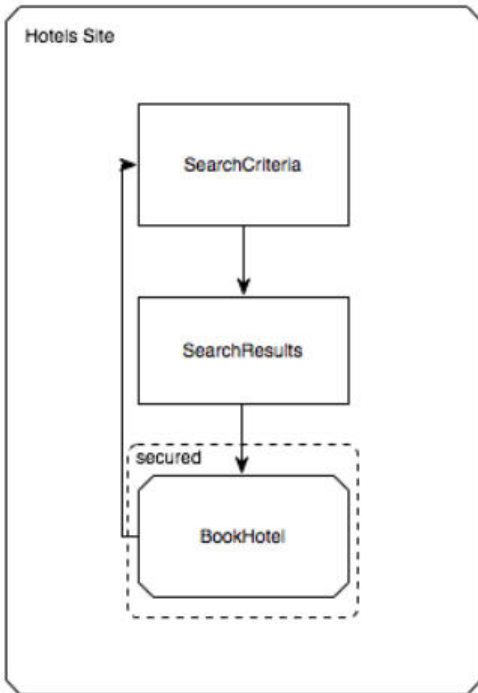
감사합니다.



2. 플로우(Flow) 정의

2.2. 플로우란 무엇인가?

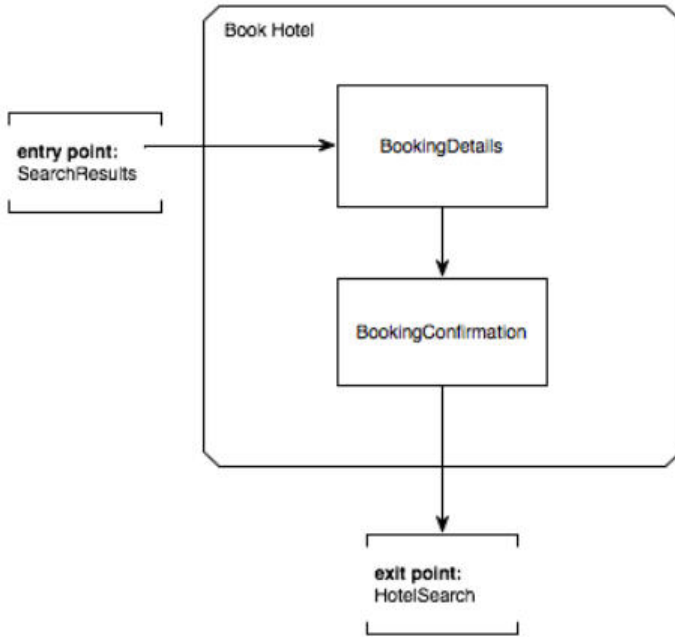
플로우는 상이한 상황(context)에서 실행될 수 있는 재사용이 가능한 여러 단계들의 흐름을 캡슐화한 것을 의미한다. 아래는 호텔 예약 프로세스의 각 단계들을 캡슐화한 flow를 설명하고 있는 [Garrett Information Architecture](#) 다이어그램이다.



2.3. 일반적인 플로는 무엇으로 구성되는가?

SWF에서 플로는 "상태(state)"로 부르는 일련의 단계들로 구성된다. 플로우로 진입하게 되는 상태는 일반적으로 사용자에게 보여지는 뷰가 된다. 이 뷰에서는 상태를 제어하게 되는 이벤트가 발생한다. 이들 이벤트는 결과적으로 다른 뷰로 이동하게 되는 전이(transition)을 일으키게 된다.

아래 그림은 이전 다이어그램에서 설명한 호텔 예약 플로우의 구조를 보여준다.



2.4. 플로우를 작성하는 방법은?

플로우는 웹 애플리케이션 개발자가 XML 기반 플로우 정의 언어를 사용해서 작성하게 된다.

2.5. 필수적인 언어 구성요소

```

//flow
<?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: 플로우 중 화면을 보여주는 상태를 정의하는 구성요소
 - 편의상 플로우 정의 파일이 있는 디렉터리 내에서 view-state id와 일치하는 화면 템플릿을 마취 보게 됨
- transition: 상태 내에서 발생한 이벤트를 제어하는 구성 요소. 화면 이동을 일으킴.
- end-state: 플로우의 결과를 정의

2.5.5. 확인지점: 필수적인 언어 구성요소

위 세 가지 구성요소로 화면 이동 로직을 손쉽게 표현할 수 있다. 보통 팀에서는 최종 사용자에게 맞는 애플리케이션의 사용자 인터페이스를 개발하는데 초점을 맞추고, 차후에 플로우에 행동을 추가하게 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">

  <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>

```

2.6. 액션(Action)

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

플로우 내에서 액션을 취할 수 있는 여러 지점이 존재한다.

- 플로우가 시작할 때
- 상태에 들어갈 때
- 화면을 보여줄 때
- 전이가 일어날 때
- 상태가 종료될 때
- 플로우가 종료될 때

SWF에서 액션은 기본적으로 Unified EL이라는 간결한 표현 언어를 사용해서 정의하게 된다.

2.6.1. 평가하기

대부분 evaluate 구성요소를 사용하게 된다. 이를 통해 스프링 빈에 있는 메소드나 다른 플로우 변수를 호출할 수 있다.

```

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

<!-- ## ## -->
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels" />

<!-- ## ## ## -->
<evaluate expression="bookingService.findHotels(searchCriteria)" result="flowScope.hotels"
    result-type="dataModel"/>

```

2.6.2. 확인지점: 플로우 액션

```

<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" />
    </view-state>

```

```

        <transition on="revise" to="enterBookingDetails" />
        <transition on="cancel" to="bookingCancelled" />
    </view-state>

    <end-state id="bookingConfirmed" />

    <end-state id="bookingCancelled" />

</flow>

```

이 플로우 예에서는 플로우가 시작할 때 플로우 범위에 Booking 객체를 생성해 저장한다. hotelId는 플로우의 입력 속성으로 받게 된다.

2.7. 입력/출력 매핑

각 플로우는 잘 정의된 입력/출력 계약(input/output contract)를 갖고 있다. 플로우는 시작할 때 입력 속성을 건네 받게 되고, 종료될 때 출력 속성을 반환하게 된다. 이처럼 플로우 호출은 개념적으로 다음과 같은 메소드 호출과 비슷하다.

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

그리고 FlowOutcome은 다음과 같은 메소드 선언부를 갖게 된다.

```

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

```

2.7.1. 입력

입력 속성은 플로우 범위에 저장되게 된다.

```

<input name="hotelId" />

<!-- [1] -->
<input name="hotelId" type="long" />

<!-- [2] -->
<input name="hotelId" value="flowScope.myParameterObject.hotelId" />

<!-- [3] -->
<input name="hotelId" type="long" value="flowScope.hotelId" required="true" />

```

1 type 속성으로 속성 지정 가능. 타입이 일치하지 않다면 타입 변환 시도

2 value 속성으로 입력 값을 할당

3 required 속성으로 null이나 비어있지 못하도록 강제

2.7.2. 출력

플로우 출력 속성은 output 구성요소를 사용한다. output 속성은 end-state 내에 선언한다.

```

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

<!-- [4] -->
<output name="confirmationNumber" value="booking.confirmationNumber" />

```

4 직접 대상 값 지정

출력 값은 속성의 이름으로 플로우 범위 내에서 얻어오게 된다.

2.7.3. 확인지점: 입력/출력 매핑하기

```
<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 플로우 는 이제 hotelId를 입력 값으로 받아서, 새로운 예약이 끝나게 되면 bookingId 출력 속성을 결과로 반환하게 된다.

2.8. 변수들

플로우에는 다수의 인스턴스 변수 선언이 가능하다. 이 변수들은 flow가 시작할 때 할당되며, 변수를 유지하게 되는 모든 @Autowired transient 참조는 플로우가 재시작될 때 다시 값이 할당(rewired)되게 된다.

var 구성 요소를 사용해서 플로우 변수를 선언하자.

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

변수로 사용하는 클래스가 플로우 요청 간 인스턴스의 상태를 유지하기 위해서 java.io.Serializable을 구현하는지 확인하자.

2.9. 하위 플로우 호출하기

플로우 내에서 하위 플로우로써 또 다른 플로우 호출이 가능하다. 이 때 하위 플로우가 결과를 반환할 때까지 기존 플로우 는 대기하게 된다.

subflow-state 구성요소를 사용해서 하위 플로우 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 플로우를 호출하게 된다. guestCreated 출력이 반환되게 되면, 새로운 손님이 예약 손님 리스트에 추가되게 된다.

input 구성요소를 사용하면 하위 플로우에 입력값을 건낼 수 있다.

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

출력 값의 이름으로 하위 플로우에서 출력하는 속성을 참조해서 전이를 하게 된다.

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

2.9.2. 확인지점: 하위 플로우 호출하기

```
<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>
```

3. Expression Language (EL)

3.2. 지원하는 EL 구현체

3.2.1. Unified EL

기본으로는 Unified EL을 사용하도록 되어 있음. jboss-el이 기본 구현체로 되어 있다. 웹 컨테이너에서 기본적으로 제공하는 EL도 있다. 예를 들면, 톰캣 6처럼.

3.2.2. OGNL

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

3.3. EL 호환성

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

3.4. EL 사용법

플로우에서 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

플로우에 의해서 보여지는 뷰는 EL을 사용해서 플로우 데이터 구조에 접근하게 됨.

3.4.1. 표현 타입

3.4.1.1. 표준 eval 표현

가장 일반적인 방법은 eval 표현. 이 경우 \${}나 #{}을 사용하면 안 됨.

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

이 예는 searchCriteria에 있는 nextPage() 호출.

3.4.1.2. 표현 템플릿

\${}을 사용.

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

이 템플릿의 결과가 합쳐져서 error-결과.xhtml 이렇게 구성 됨.

3.5. 특별한 EL 변수

*scope

- flowScope: flow 변수에 할당되며, 플로우 범위를 가진 객체. 기본적으로 플로우범위에 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 플로우 내에서 공유

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

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

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

- conversationScope: conversation 변수에 할당. 최상위 플로우가 시작할 때 할당되며, 최상위 플로우가 종료될 때 정리. 최상위 플로우의 자식 플로우에서 공유. HTTP session에 저장되며, 세션 복제를 할 경우를 대비해 Serializable을 구현해야 함

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

- context

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

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

- flowExecutionContext: 현재 플로우 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

3.6. 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.

4. 뷰(View) 보여주기

4.2. 뷰 상태(view state) 정의하기

view-state 구성요소로 정의. view-state는 해당 뷰를 보여준 다음, 사용자의 응답을 기다림.

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

4.3. 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.xhtml">
```

- 논리적인 경로 사용: Spring MVC 등과 통합 시

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

4.4. 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" />
```

- viewScope 변수에 할당하기

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

```
</on-render>
```

4.5. 화면을 보여줄 때 액션 실행하기

뷰를 보여주기 전에 특정 액션을 실행하려면 on-render 사용.

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

4.6. 모델 바인딩 하기

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

뷰 이벤트가 발생했을 때 지정된 모델에 대해서 다음 행동이 일어난다.

1. view-to-model 바인딩.
2. 모델 유효성 검증.

4.7. 타입 변환 수행하기

4.7.1. 변환기(Converter) 구현하기

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.

4.7.2. 변환기 등록하기

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

4.8. 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>
```

4.9. 명시적으로 바인딩 지정하기

binder 속성으로 바인딩 할 프로퍼티를 명시적으로 지정할 수 있음.

```
<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 바인딩.

변환기 지정 가능.

```
<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>
```

4.10. 모델 유효성 검증하기

4.10.1. 프로그램 내에서 유효성 검증 하기

첫 번째 방법으로 유효성 검증 로직을 모델 객체 내에 정의하는 방법이다. view-state 생명주기에서 자동으로 호출 된다.

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

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

  public void validateEnterBookingDetails(ValidationContext context) {
    MessageContext messages = context.getMessages();
    if (checkinDate.before(today())) {
      messages.addMessage(new MessageBuilder().error().source("checkinDate").
```

```

        defaultMessage("Check in date must be a future date").build());
    } else if (!checkinDate.before(checkoutDate)) {
        messages.addMessage(new MessageBuilder().error().source("checkoutDate").
            defaultMessage("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.

4.10.1.2. Validator 구현하기

Validator로 불리는 별도의 객체로 정의할 수도 있다. 클래스 이름을 \$<model>Validator 로 하자. 메소드 이름은 역시 validate\$<state>로 하자.

```

@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").
                defaultMessage("Check in date must be a future date").build());
        } else if (!booking.getCheckinDate().before(booking.getCheckoutDate())) {
            messages.addMessage(new MessageBuilder().error().source("checkoutDate").
                defaultMessage("Check out date must be later than check in date").build());
        }
    }
}

```

spring mvc의 Error 객체도 받을 수 있다.

4.10.2. ValidationContext

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

4.11. 유효성 검증 하지 않기

```

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

```

4.12. 뷰 전이 실행하기

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.

4.12.1. 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;
}
}
}

```

4.12.2. 글로벌 전이(Global transitions)

```

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

```

4.12.3. 이벤트 핸들러(Event handlers)

```

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

```

4.12.4. 프래그먼트 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.

4.13. 메시지 사용하기

MessageContext는 플로우 실행 동안에 메시지를 저장하는데 사용되는 API다. 일반 메시지나 국제화가 지원된 메시지 모두 사용 가능하다.

메시지 수준도 지정 가능하며, 지원되는 수준은 info, warning, error이 있다. 메시지를 추가할 때는 MessageBuilder를 사용하자.

1. 일반 메시지 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());

```

2. 국제화가 지원되는 메시지 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());

```

4.13.3. 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}" />
```

4.13.4. 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.
```

4.14. 팝업 띄우기

모달 팝업 다이얼로그를 뷰로 렌더링하고 싶다면,

```
<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.

4.15. 뷰 백트래킹(View backtracking)

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

1. 'discard'로 설정하면 백트래킹(backtracking) 예방 가능

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

#'invalidate'로 설정하면 이전에 보여줬던 모든 뷰 뿐만 아니라 현재 뷰까지도 백트래킹 예방(더 강력한 예방이 된다는 뜻..)

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

5. 액션(action) 실행하기

5.2. 액션 상태 정의하기

특정 액션을 호출한 다음에, 그 결과에 따라서 다른 상태로 전이하고 싶은 경우에는 action-state 구성요소를 사용하자.

```
<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>
```

5.3. 의사결정 상태(decision states) 정의하기

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

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

5.4. 액션 출력 이벤트 매핑하기

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
java.lang.String	String 값
java.lang.Boolean	yes(true에 해당), no(false에 해당)
java.lang.Enum	Enum 이름
나머지 다른 타입	성공!

예제.

```
<action-state id="moreAnswersNeeded">
  <evaluate expression="interview.moreAnswersNeeded()" />

  <transition on="yes" to="answerQuestions" />
  <transition on="no" to="finish" />
</action-state>
```

5.5. 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.

5.5.1. Invoking a POJO action

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

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

5.5.3. MultiAction 구현 호출하기

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

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

    public Event actionMethod2(RequestContext context) {
        ...
    }

    ...
}
```

5.6. 액션 예외

action은 복잡한 비즈니스 로직을 캡슐화하고 있는 서비스를 호출할 수도 있다. 이 서비스들은 비즈니스 예외를 던질 수도 있으니 이를 처리해야 할 수도 있다.

5.6.1. 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";
        }
    }
}
```

5.6.2. 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();
        }
    }
}

```

5.7. 다른 액션 실행 예제

5.7.1. 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>

```

5.7.2. on-entry

```

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

```

5.7.3. 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>

```

5.7.4. 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>
```

5.7.5. 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>
```

5.7.6. on-transition

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

5.7.7. 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>
```

6. 플로우가 관리하는 영속성(Flow Managed Persistence)

6.2. 플로우 범위(FlowScoped) PersistenceContext

이 패턴은 플로우 시작 시에 flowScope으로 PersistenceContext를 생성해준다. 이 persistence context는 플로우 실행 과정 동안에 데이터 접근을 하는데 사용하며, 플로우가 종료될 때 persistent entity에서 변경된 내용을 반영(commit)한다. 이 패턴은 대부분 다중 사용자에 의해서 동시에 수정되는 데이터의 정합성을 보호하고자 optimistic locking 전략과 함께 사용된다. 저장이나 재시작 능력이 필요없다면, 플로우 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 플로우를 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 플로우에 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>
```

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

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

이걸로 끝이다. 이제 플로우가 시작할 때 리스너가 flowScope에 새로운 EntityManager를 할당해서 제어하게 된다. 플로우 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.

7. 플로우에 보안 적용하기(Securing Flows)

7.2. 어떻게 플로우를 안전하게 할 수 있을까?

플로우 실행에 보안을 적용시키고 싶다면 다음 단계에 따르자.

1. Spring Security에서 인증(authentication)과 권한(authorization) 규칙을 구성
2. secured 구성요소로 플로우 정의에 보안 규칙을 등록
3. 보안 규칙을 처리해주는 SecurityFlowExecutionListener 추가

7.3. secured 구성요소

secured 구성요소는 접근 하기 전에 권한 확인을 적용해주며, 플로우 실행 단계마다 한 번 이상은 나올 수 없다.

플로우 실행에서 세 개의 측면, 플로우, 상태, 전이에 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>
```

7.3.1. 속성에 보안 적용하기

'/'로 구분해서 SS의 권한 속성을 리스트로 입력할 수 있다. 이 속성은 대부분 허가된 보안 룰이 된다.

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

7.3.2. 타입 맞춰보기

두 가지 유형의 일치 유형 제공: any, all.

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

이 설정은 선택이며, 기본은 any.

7.4. SecurityFlowExecutionListener

웹 플로우 설정에 추가해야 함.

```
<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>
```

7.5. Spring Security 환경 구성하기

7.5.1. 스프링 환경 구성하기

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: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>

```

7.5.2. web.xml 환경 구성하기

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>

```

9. 시스템 설정

9.2. webflow-config.xsd

```

<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>

```

9.3. 기본 시스템 환경 구성하기

```

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

<!-- [2] -->
<webflow:flow-executor id="flowExecutor" />

```

- 1 플로우 설정 파일 등록
- 2 플로우 실행의 중추 역할을 하는 서비스 제공

9.4. flow-registry 옵션

```
<!-- [1] -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" />

<!-- [2] -->
<webflow:flow-location path="/WEB-INF/flows/booking/booking.xml" id="bookHotel" />

<!-- [3] -->
<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] -->
<webflow:flow-location-pattern value="/WEB-INF/flows/**/*-flow.xml" />

<!-- [5] -->
<webflow:flow-registry id="flowRegistry" base-path="/WEB-INF">
  <webflow:flow-location path="/hotels/booking/booking.xml" />
</webflow:flow-registry>

<!-- [6] -->
<!-- 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>
```

1 기본은 이름 -flow.xml이지만, 직접 지정할 수도 있다.

2 id로 식별이 가능하도록 할 수도 있다

3 메타 정보도 등록할 수 있다.

4 패턴을 지정할 수도 있다.

5 기본 앞 접자 경로를 지정해서 위치를 조합해서 사용할 수도 있다. 플로우 정의 파일은 모듈화를 높이기 위해서 관련 있는 폴더에 각각 위치해 있는게 가장 좋다.

6 플로우 상속 구조 구성 가능

9.4.7. 커스텀 FlowBuilder 서비스 구성하기

flow-registry에서 flow를 구축하는데 사용되는 서비스나 설정 등을 커스터마이징 해주는 속성이다. 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" />
```

1. 구성 가능한 서비스

- conversion-service: SWF 시스템에서 사용하는 ConversionService를 커스터마이징. 플로우 실행 동안에 필요한 경우 특정 타입을 다른 타입으로 변환해 줌(propertyEditor 성격)
- expression-parser: ExpressionParser 커스터마이징. 기본은 Unified EL이 사용되며, 다른 경우에는 OGNL이 사용 됨.
- view-factory-creator: ViewFactoryCreator 커스터마이징. JSP, Velocity, Freemaker 등을 화면에 보여주게 해 주는 Spring MVC ViewFactories를 만들어 줌
- development: 플로우 개발 모드 설정. true인 경우, 플로우 정의가 변경되면 hot-reloading 적용.

```

<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="..." />

```

9.5. flow-executor 옵션

플로우 실행 리스너 붙이기

리스너 등록

```

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

```

특정 흐름에 대해서만 적용 가능하다.

```

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

```

9.5.2. FlowExecution persistence 조정하기

- max-executions: 사용자 세션 당 생성될 수 있는 플로우 실행 개수 지정
- max-execution-snapshots: 플로우 실행 당 받을 수 있는 이력 snapshot 개수 지정. snapshot을 사용하지 못하게 하려면, 0으로 지정. 제한이 없게 하려면 -1로 설정.

```

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

```

10. 스프링 MVC 통합

10.2. web.xml 환경 구성하기

스프링 MVC를 구성하는 첫 단계는 web.xml에 DispatcherServlet을 구성하는 것이다. DispatcherServlet은 웹 애플리케이션 당 하나를 등록한다.

이 예제에서는 /spring/으로 시작하는 모든 요청을 받도록 설정하고 있다. init-param을 사용해서 contextConfigLocation을 설정하고 있다.(참고: 또 다른 방법은 <context-param>을 사용하는 것)

```

<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>

```

10.3. 플로우로 전달하기(dispatch)

DispatcherServlet은 애플리케이션 자원에 대한 요청과 핸들러를 매핑시켜 준다. 플로우도 핸들러의 하나의 유형일 뿐이다.

10.3.1. FlowHandlerAdapter 등록하기

FlowHandlerAdapter를 설치해서 스프링 MVC 내에서 플로우를 제어할 수 있게 하는 것이다.

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

10.3.2. flow 매핑 정의하기

다음 단계로 애플리케이션 자원을 특정 플로우와 매핑시키는 것이다. 가장 간단한 방법은 FlowHandlerMapping을 정의하는 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"/>
  <property name="order" value="0"/>
</bean>
```

이 설정은 Dispatcher가 애플리케이션 자원 경로를 flow registry에 등록된 플로우로 매핑시킬 수 있도록 해준다. 예를 들어, /hotels/booking 요청은 hotels/booking이란 플로우id를 갖는 플로우에게 요청이 가게 된다. flow registry에서 못 찾게 되면, Dispatcher의 순서에 따라 다음 핸들러 매핑에서 찾고, 없다면 "noHandlerFound" 응답이 반환되게 된다.

10.3.3. 작업 흐름을 제어하는 플로우

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

- HTTP 요청 파라미터는 모든 플로우 실행의 입력 맵에서 사용할 수 있다.
- 최종 응답없이 플로우 실행이 끝나게 되면, 기본 핸들러는 동일한 요청을 기반으로 새로운 실행 시작을 시도하게 된다.
- NoSuchFlowExecutionException인 경우에는 새로운 플로우 실행을 시작해 복구 시도를 해보며, 다른 예외는 제어하지 않는다.

10.4. 커스텀 FlowHandler 구현하기

FlowHandler는 HTTP 서블릿 환경에서 플로우가 실행되는 방법을 커스터마이징 하는데 사용하는 확장 지점이 된다. FlowHandlerAdapter 내에서 사용하며 다음과 같은 책임을 갖고 있다.

- 실행되는 플로우의 id 반환
- 플로우 실행을 시작하면서 건널 입력 값 생성
- 플로우 실행이 종료되면서 반환하는 결과 관리
- 플로우 실행에서 발생해서 던져진 예외 관리

이러한 책임은 org.springframework.mvc.servlet.FlowHandler로 표현되어 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 구현하려면, 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 고려해보자.

- getFlowId(HttpServletRequest) 재정의: HTTP 요청에서 직접적으로 플로우 id를 받을 수 없을 때. 일반적으로는 요청의 URI에서 경로 정보를 얻게 됨. 예를 들어, <http://localhost/app/hotels/booking?hotelId=1>는 hotels/booking 이란 플로우 id로 매핑 됨
- createExecutionInputMap(HttpServletRequest) 재정의: HttpServletRequest에서 플로우 입력 파라미터를 세부적으로 직접 추출해야 하는 경우. 기본적으로는 모든 요청 파라미터가 플로우 입력 파라미터로 넘겨짐.
- handleExecutionOutcome 재정의: 직접 플로우 실행 결과를 제어할 필요가 있을 경우. 기본 행동은 플로우의 새로운 실행을 재시작하려고 마지막 플로우의 URL로 redirect 보냄
- handleException 재정의: 제어되지 못한 플로우 실행을 세심하게 조정할 필요가 있는 경우. 기본 행동은 끝났거나 만료된 플로우 실행에 접근했을 때는 플로우 실행을 다시 시작하려 하고, 그렇지 않다면 ExceptionResolver로 다시 던져버린다.

10.4.1. FlowHandler 예제

가장 일반적인 스프링 MVC와의 상호 작용은, 플로우가 종료 됐을 때 @Controller로 재전송 하는 방법이다. FlowHandler는 이를 특정 controller URL을 플로우 정의와 상호작용 없이 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";
        }
    }
}

```

10.4.2. 커스텀 FlowHandler 배포하기

커스텀 FlowHandler를 설치하려면, 빈으로 등록하기만 하면 된다. 빈 id는 반드시 적용하고자 하는 플로우의 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.

10.4.3. FlowHandler 재전송(redirect)

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 redirect할 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 플로우 정의 내에서도 사용할 수 있다. 예를 들면, view="externalRedirect:<http://springframework.org>".

10.5. 뷰 결정

Web Flow 2는 따로 지정하지 않는다면 플로우 파일이 있는 디렉터리에 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 플로우 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-factory-creator="mvcViewFactoryCreator"/>

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

10.6. 뷰에서 이벤트 신호 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.

10.6.1. HTML 버튼을 사용해서 이벤트 신호 보내기

다음은 proceed와 cancel 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.

10.6.2. hidden HTML 폼 파라미터 사용해 이벤트 신호 보내기

폼이 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 폼으로 전송할 수 있는 이벤트가 하나일 때만 생각해봐야 한다.

10.6.3. HTML link 사용해서 이벤트 신호 보내기

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

매개변수 식별 순서는 "eventId" => "_eventId" => 없음 이다.

11. 스프링 자바스크립트 킷 레퍼런스

11.1 소개

Spring JavaScript(spring-js)는 Dojo와 같은 공통 자바 스크립트 킷을 경량화, 추상화했다. spring-js의 목적은 리치 위젯 행위나 Ajax 리모팅을 사용하는 웹 페이지를 혁신적으로 강화시켜주는 공통의 클라이언트 단 프로그래밍 모델을 제공하는 것이다.

11.2. 자바스크립트 자원 제공하기

spring-js는 웹 루트 디렉토리 뿐만 아니라 jar 파일에 있는 JavaScript와 CSS 파일과 같은 웹 자원을 가져올 수 있는 일반적인 ResourceServlet을 제공한다. 이 서블릿은 Spring.js를 더 편리하게 사용할 수 있게 해준다.

```
<!-- Serves static resource content from .jar files such as spring-js.jar -->
<servlet>
  <servlet-name>Resource Servlet</servlet-name>
  <servlet-class>org.springframework.js.resource.ResourceServlet</servlet-class>
</servlet>

<!-- Map all /resources requests to the Resource Servlet for handling -->
<servlet-mapping>
  <servlet-name>Resource Servlet</servlet-name>
  <url-pattern>/resources/*</url-pattern>
</servlet-mapping>
```

11.3 페이지에서 스프링 자바 스크립트 포함하기

spring-js 최초 버전은 Dojo를 기반으로 구축됐다.

페이지에서는 기본 인터페이스 파일인 Spring.js와 킷에 기반을 둔 Spring-(library implementation).js를 포함시키면 된다.(이로 보아 충분히 확장이 가능할 듯.)

```
<script type="text/javascript" src="<c:url value="/resources/dojo/dojo.js" />"> </script>
<script type="text/javascript" src="<c:url value="/resources/spring/Spring.js" />"> </script>
<script type="text/javascript" src="<c:url value="/resources/spring/Spring-Dojo.js" />"> </script>
```

11.4. 스프링 자바 스크립트 데코레이션

spring-js의 핵심 개념은 기존 DOM 노드에 대한 데코레이션(decoration)을 적용한다는 점이다. addDecoration 메소드가 데코레이션을 적용하는데 사용 된다.

```
<form:input id="searchString" path="searchString"/>
<script type="text/javascript">
  Spring.addDecoration(new Spring.ElementDecoration({
    elementId: "searchString",
    widgetType: "dijit.form.ValidationTextBox",
    widgetAttrs: { promptMessage : "Search hotels by name, address, city, or zip." } }));
</script>
```

ElementDecoration은 기존 DOM 노드에 풍부한 위젯 행위(rich widget behavior)를 적용해주는데 사용된다. 데코레이션의 목적은 기반이 되는 킷을 숨기고자 하는게 목적이 아니고, 킷 고유의 위젯 타입과 속성을 바로 사용하게 해주는 데 목적이 있다. 이러한 접근 방법은 특정 킷 기반에 있는 모든 위젯을 통합해서 공통의 데코레이션 모델을 사용할 수 있도록 해준다. booking-mvc 예제를 보면 클라이언트 단 유효성 검증에 대한 더 많은 예제가 있다.

유효성이 모두 통과될 때까지는 폼을 서버로 submit하지 말라는 공통 요구사항이 있는 예는 다음처럼 ValidateAllDecoration을 하면 된다.

```
<input type="submit" id="proceed" name="_eventId_proceed" value="Proceed" />
```

```

<script type="text/javascript">
    Spring.addDecoration(new Spring.ValidateAllDecoration({ elementId: 'proceed',
    event: 'onclick' }));
</script>

```

AjaxEventDecoration를 사용해서 클라이언트 단 이벤트 리스터 적용하자.

```

<a id="prevLink" href="search?searchString=${criteria.searchString}&page=${criteria.page -
1}">Previous</a>
<script type="text/javascript">
    Spring.addDecoration(new Spring.AjaxEventDecoration({
        elementId: "prevLink",
        event: "onclick",
        params: { fragments: "body" }
    }));
</script>

```

이 예는 prevLink의 onclick 이벤트 발생 시, 요청을 Ajax 호출로 해서 지정된 fragments만 response로 받아서 다시 보여 주라는 내용이다.

다수의 데코레이션도 적용 가능하다.

```

<input type="submit" id="proceed" name="_eventId_proceed" value="Proceed" />
<script type="text/javascript">
    Spring.addDecoration(new Spring.ValidateAllDecoration({elementId: 'proceed', event: 'onclick'}));
    Spring.addDecoration(new Spring.AjaxEventDecoration({elementId: 'proceed',
    event: 'onclick',formId: 'booking', params: {fragments: 'messages'}}));
</script>

```

기반이 되는 툴킷 API를 사용해서 여러 구성요소에 동시에 적용 가능하다.

```

<div id="amenities">
<form:checkbox path="amenities" value="OCEAN_VIEW" label="Ocean View" /></li>
<form:checkbox path="amenities" value="LATE_CHECKOUT" label="Late Checkout" /></li>
<form:checkbox path="amenities" value="MINIBAR" label="Minibar" /></li>
<script type="text/javascript">
    dojo.query("#amenities input[type='checkbox']").forEach(function(element) {
        Spring.addDecoration(new Spring.ElementDecoration({
            elementId: element.id,
            widgetType : "dijit.form.CheckBox",
            widgetAttrs : { checked : element.checked }
        }));
    });
</script>
</div>

```

11.5. Ajax 요청 제어하기

spring-js의 클라이언트 단 Ajax 응답 처리는 서버에서 보내주는 '프래그먼트(fragments)'를 기반으로 한다. 프래그먼트는 표준 HTML로, 기존 페이지에서 교체되는 부분을 의미한다. 서버 단에서 핵심은 전체 응답에서 보여줘야 할 필요한 부분을 결정하는 방법이다.

이런 방법이 가능하려면 전체 응답은 반드시 templating 기술을 사용해서 구축되어야 한다. spring-js는 이를 위해서 Tiles를 사용한 스프링 MVC 확장을 제공한다.

11.5.1. 스프링 MVC 컨트롤러로 Ajax 요청 제어하기

스프링 MVC 컨트롤러에서 Ajax 요청을 제어하려면 응답의 일부를 보여줄 수 있게 해주는 스프링 MVC 확장을 설정하면 된다.

```
<bean id="tilesViewResolver" class="org.springframework.js.ajax.AjaxUrlBasedViewResolver">
  <property name="viewClass" value="org.springframework.webflow.mvc.view.FlowAjaxTilesView" />
</bean>
```

AjaxUrlBasedViewResolver은 Ajax 요청을 적절한 프래그먼트로 보여줄 수 있도록 제어해주는 FlowAjaxTilesView을 만들어 변환해준다. FlowAjaxTilesView는 스프링 MVC와 SWF에서 모두 보여줄 수 있다.

```
<definition name="hotels/index" extends="standardLayout">
  <put-attribute name="body" value="index.body" />
</definition>

<definition name="index.body" template="/WEB-INF/hotels/index.jsp">
  <put-attribute name="hotelSearchForm" value="/WEB-INF/hotels/hotelSearchForm.jsp" />
  <put-attribute name="bookingsTable" value="/WEB-INF/hotels/bookingsTable.jsp" />
</definition>
```

Ajax 요청에서 렌더링하고자 하는 'body', 'hotelSearchForm', 'bookingsTable' 등을 지정해주면 된다.

11.5.2. 스프링 MVC + 스프링 웹플로우에서 Ajax 요청 제어하기

swf에서는 플로우 정의 언어를 사용해서 직접 프래그먼트를 정의할 수도 있다. 이러한 접근 방법의 장점은 프래그먼트의 선택이 클라이언트 사이드와 완전히 독립적이라는 점이다.

```
<view-state id="changeSearchCriteria" view="enterSearchCriteria.xhtml" popup="true">
  <on-entry>
    <render fragments="hotelSearchForm" />
  </on-entry>
  <transition on="search" to="reviewHotels">
    <evaluate expression="searchCriteria.resetPage()" />
  </transition>
</view-state>
```

14. Flow 테스트하기

14.2. AbstractXmlFlowExecutionTests 상속받기

XML 기반 플로우 정의에 대한 실행 테스트를 하려면, AbstractXmlFlowExecutionTests를 상속하자.

```
public class BookingFlowExecutionTests extends AbstractXmlFlowExecutionTests {
}
```

14.3. 테스트 대상 플로우 경로 지정하기

최소한 테스트하려면 플로우 에 대한 경로를 반환하는 getResource(FlowDefinitionResourceFactory)를 재정의해야만 한다.

```
@Override
protected FlowDefinitionResource getResource(FlowDefinitionResourceFactory resourceFactory) {
  return resourceFactory.createFileResource("src/main/webapp/WEB-INF/hotels/booking/booking.xml");
}
```

14.4. 플로우 의존성 등록하기

플로우 가 외부 서비스에 의존한다면, 이에 대한 stub이나 mock을 등록하기 위해 `configureFlowBuilderContext(MockFlowBuilderContext)`를 재정의 하자.

```
@Override
protected void configureFlowBuilderContext(MockFlowBuilderContext builderContext) {
    builderContext.registerBean("bookingService", new StubBookingService());
}
```

또 다른 플로우를 상속하거나, 다른 상태를 상속하는 상태를 갖는 경우, 다른 플로우를 반환하는 `getModelResources(FlowDefinitionResourceFactory)`를 재정의하자.

```
@Override
protected FlowDefinitionResource[] getModelResources(FlowDefinitionResourceFactory resourceFactory)
{
    return new FlowDefinitionResource[] {
        resourceFactory.createFileResource("src/main/webapp/WEB-INF/common/common.xml")
    };
}
```

14.5. 플로우 시작 테스트하기

```
public void testStartBookingFlow() {

    Booking booking = createTestBooking();

    MutableAttributeMap input = new LocalAttributeMap();
    input.put("hotelId", "1");
    MockExternalContext context = new MockExternalContext();
    context.setCurrentUser("keith");
    startFlow(input, context);

    assertCurrentStateEquals("enterBookingDetails");
    assertTrue(getRequiredFlowAttribute("booking") instanceof Booking);
}
```

14.6. 플로우 이벤트 제어 테스트하기

flow 이벤트 제어 행동을 시험해보는 테스트를 정의해보자. 특정 플로우 상태로 이동하기 위해 `setCurrentState(String)`을 편리하게 사용할 수 있다.

```
public void testEnterBookingDetails_Proceed() {

    setCurrentState("enterBookingDetails");

    getFlowScope().put("booking", createTestBooking());

    MockExternalContext context = new MockExternalContext();
    context.setEventId("proceed");
    resumeFlow(context);

    assertCurrentStateEquals("reviewBooking");
}
```

14.7. 하위 플로우 테스트하기

```
public void testBookHotel() {
```

```

setCurrentState("reviewHotel");

Hotel hotel = new Hotel();
hotel.setId(1L);
hotel.setName("Jameson Inn");
getFlowScope().put("hotel", hotel);

getFlowDefinitionRegistry().registerFlowDefinition(createMockBookingSubflow());

MockExternalContext context = new MockExternalContext();
context.setEventId("book");
resumeFlow(context);

// verify flow ends on 'bookingConfirmed'
assertFlowExecutionEnded();
assertFlowExecutionOutcomeEquals("finish");
}

public Flow createMockBookingSubflow() {
    Flow mockBookingFlow = new Flow("booking");
    mockBookingFlow.setInputMapper(new Mapper() {
        public MappingResults map(Object source, Object target) {
            // assert that 1L was passed in as input
            assertEquals(1L, ((AttributeMap) source).get("hotelId"));
            return null;
        }
    });
    // immediately return the bookingConfirmed outcome so the caller can respond
    new EndState(mockBookingFlow, "bookingConfirmed");
    return mockBookingFlow;
}

```