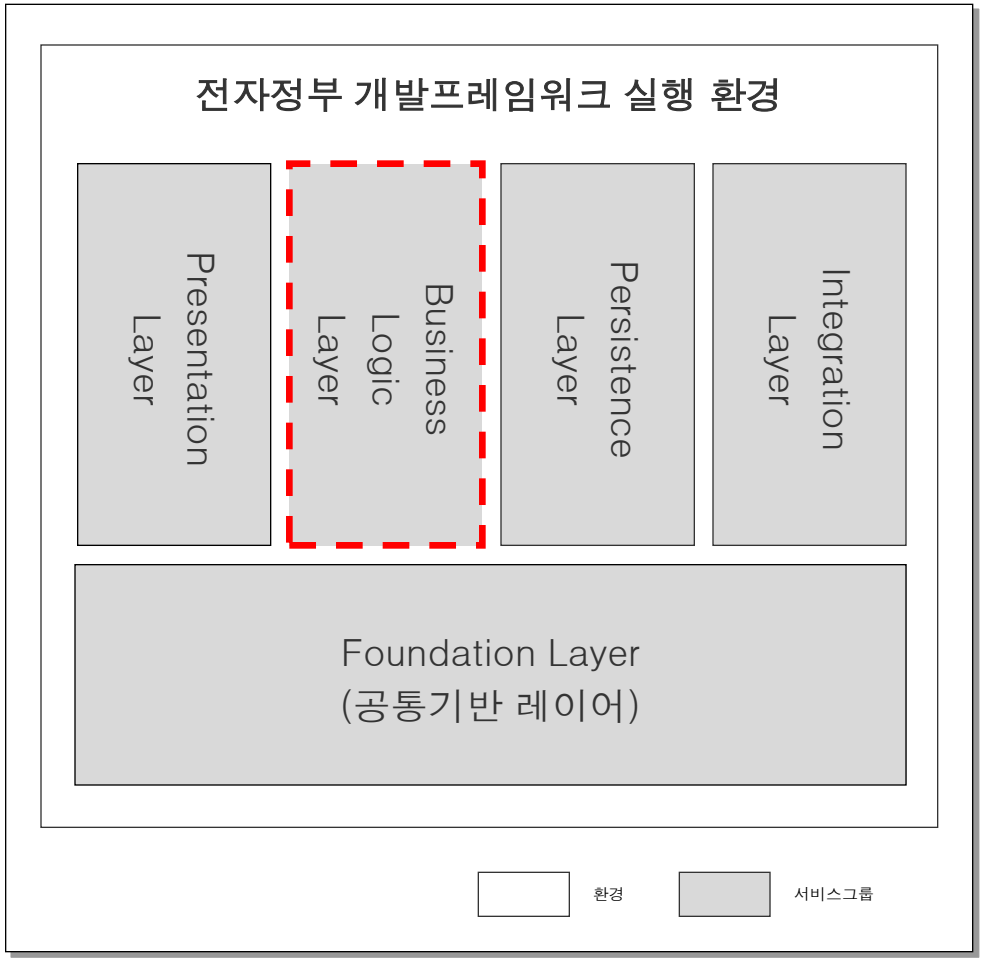
전자정부 개발 프레임워크

실행 환경

1. 개요
2. **Process Control**
3. **Exception Handling**

1. 개요 - 실행환경 업무처리 레이어(1/3)

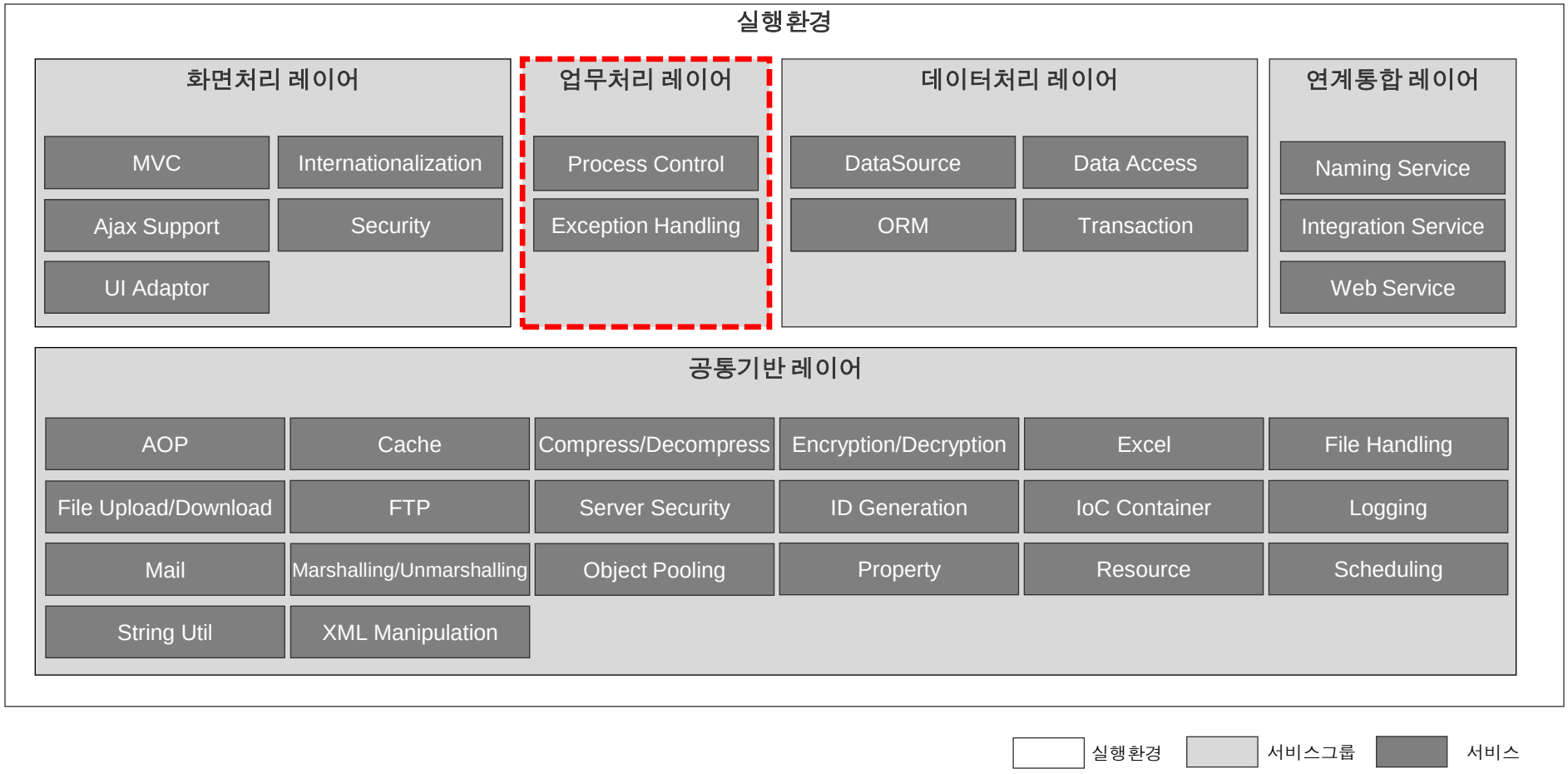
❑ 업무처리 레이어는 업무 프로그램의 업무 로직을 담당하는 **Layer**로서, 업무 흐름 제어, 에러 처리 등의 기능을 제공함



서비스 그룹	설명
Presentation Layer	• 업무 프로그램과 사용자 간의 Interface를 담당하는 Layer로서, 사용자 화면 구성, 사용자 입력 정보 검증 등의 기능을 제공함
Business Logic Layer	• 업무 프로그램의 업무 로직을 담당하는 Layer로서, 업무 흐름 제어, 에러 처리 등의 기능을 제공함
Persistence Layer	• 데이터베이스에 대한 연결 및 영속성 처리, 선언적인 트랜잭션 관리를 제공하는 Layer임
Integration Layer	• 타 시스템과의 연동 기능을 제공하는 Layer임
Foundation Layer (공통기반 레이어)	• 실행 환경의 각 Layer에서 공통적으로 사용하는 공통 기능을 제공함

1. 개요 - 실행환경 업무처리 레이어(2/3)

❑ 업무처리 레이어는 **Process Control, Exception Handling** 등 총 **2**개의 서비스를 제공함



❑ 업무처리 레이어는 **Spring Web Flow, Spring** 등 총 **2**개의 오픈소스 **SW**를 사용하고 있음

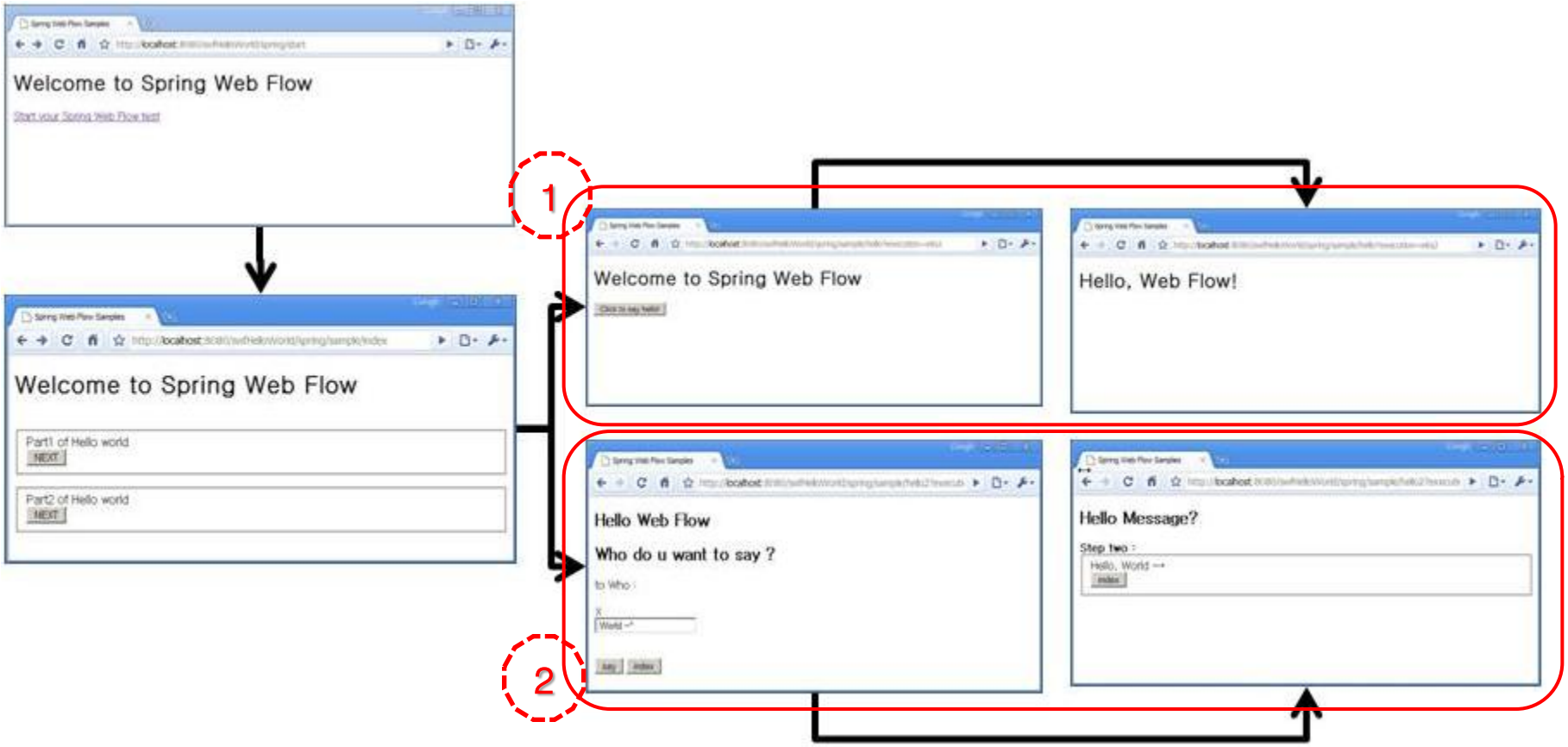
서비스	오픈소스 SW	버전
Process Control	Web Flow	2.0
Exception Handling	Spring	3.0.5

❑ 서비스 개요

- Spring Web Flow(SWF)는 웹 애플리케이션내 페이지 흐름(flow)의 정의와 수행에 집중하는 Spring프레임워크 웹 스택의 컴포넌트이다.
- Spring Web Flow는 추상화의 좀더 높은 레벨에 존재하고 Struts, Spring MVC, Portlet MVC, 그리고 JSF와 같은 기본 프레임워크내에서 자족적인 페이지 흐름(flow) 엔진(page flow engine)처럼 통합된다. SWF는 선언적이고 높은 이식성을 가지며 뛰어난 관리능력을 가지는 형태로 명시적으로 애플리케이션의 페이지 흐름(flow)을 획득하는 기능을 제공한다.

❑ Getting Started - Hello 프로젝트

– 화면 시나리오



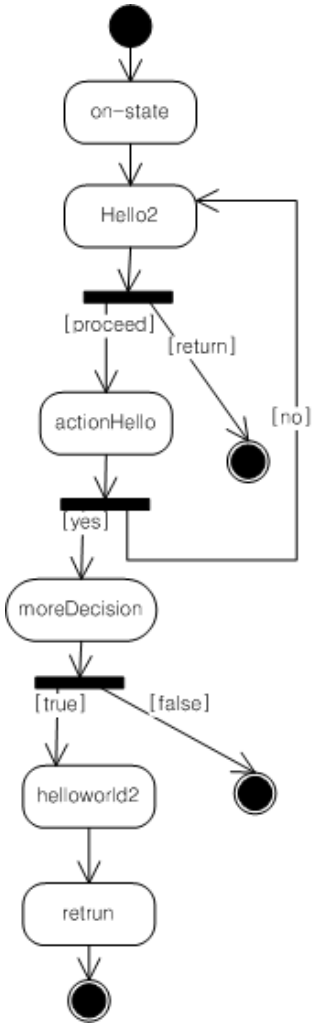
Demo

Lab303-swf

❑ 1 번 Hello 시나리오 Flow 정의


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

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

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

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

</flow>
```

❑ 2 번 Hello 시나리오 Flow 정의




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


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

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

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

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

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


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

❑ Flow Definition

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

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

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

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

</flow>
```

- view-state: Flow 중 화면을 보여주는 State를 정의하는 구성요소
  - 편의상 Flow 정의 파일이 있는 디렉터리 내에서 view-state id와 일치하는 화면 템플릿을 맞추어 보여주게 된다.
- transition: State 내에서 발생한 이벤트를 제어하는 구성 요소. 화면 이동을 일으킨다.
- end-state: Flow의 결과를 정의

– decision state

• action-state를 대신해서 편리하게 if/else 문법을 사용해서 이동하고자 하는 의사결정을 해주는 decision-state를 사용한다.

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

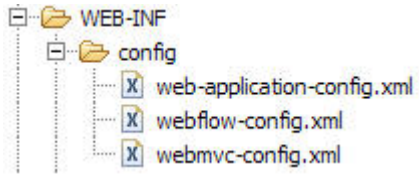
– action-state

• 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 매핑되야 한다.

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

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

❑ 환경설정 with Spring MVC




```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"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-2.5.xsd">

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


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

</beans>
```

```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-2.5.xsd">

       <!--
         flowRegistry 에 등록된 flow 와 요청되는 path 와 매핑해주는 역할을 수행한다.
             예제에선 요청되는 .../swfHelloWorld/spring/sample/hello URL 정보를 이용하여 flow 내에서 sample/hello ID로 찾음.
       -->
       <bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
              <property name="order" value="0" />
              <property name="flowRegistry" ref="flowRegistry" />
       </bean>

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

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

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


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

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

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

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

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

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

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

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


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

</beans>
```

– webmvc-config.xml

```
<!--
  flowRegistry 에 등록된 flow 와 요청되는 path 와 매핑해주는 역할을 수행한다.
        예제에선 요청되는 .../swfHelloWorld/spring/sample/hello  URL 정보를 이용하여 flow 내에서 sample/hello ID로 찾음.
  -->
<bean class="org.springframework.webflow.mvc.servlet.FlowHandlerMapping">
        <property name="order" value="0" />
        <property name="flowRegistry" ref="flowRegistry" />
</bean>

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

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

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


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

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

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

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

– webflow-config.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" xmlns:webflow="http://www.springframework.org/schema/webflow-config"
       xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
          http://www.springframework.org/schema/webflow-config
          http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.0.xsd">

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

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

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


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

</beans>
```

❑ 화면


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

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

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

- transition 은 화면에서 발생한 이벤트와 매핑을 할까? hello.jsp 소스를 잠시 보겠다.
  코드에서 보는 봐와 같이 form으로 둘러싸인 곳에 해답은 있다. <input type="submit" name="_eventId_say" …. /> 에서 name 을 보면 _eventId_say 로 답을 찾을 수 있다. _eventId 가 답이다. say는 transition 의 on 과 같음을 확인 할 수 있다.
  eventId 에 정의된 특정위치의 문자열을 가지고 transition 를 분석한다.
  transition 에 대한 내용은 flow 정의에서 자세히 살펴보길 바란다. eventId 가 "say"를 가지고 form 이 전달되면 flow 정의에 따라 transition 을 찾고 그에 맞는 state 로 넘어가게 된다.
  결과는 별도의 값을 가지고 보여주는 화면이 아닌 단지 아래와 같은 화면을 보여주도록 되어 있다.

❑ The Spring Framework - Reference Documentation

- http://static.springsource.org/spring/docs/2.5.x/reference/
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/
- http://static.springsource.org/spring-webflow/docs/2.0.x/reference/html/

❑ 서비스 개요

- 전자정부 표준프레임워크 기반의 시스템 개발시 Exception 처리, 정확히는 Exception 별 특정 로직을 흐를 수 있도록 하여 Exception 에 따른 적절한 대응이 가능도록 하고자 하는데 목적이 있다.
- AOP 의 After throwing advice 로 정의된다.
- 전자정부 표준프레임워크 기반의 처리되는 Exception 은 현재 **EgovBizException , DataAccessException , FdlException , 그외 나머지 로** 구분되어 제공한다.
- 샘플 코드 : processException (메시지키) 를 던지는 형태이다.

```
        public CategoryVO selectCategory(CategoryVO vo) throws Exception {
                CategoryVO resultVO = categoryDAO.selectCategory(vo);
                try {
    ....

    // 넘어온 resultVO 가 null 인경우 EgovBizException 발생 (result.nodata.msg 는 메세지 키에 해당됨)
                if (resultVO == null)
                        throw processException("result.nodata.msg");
                        // 또는 throw processException("result.nodata.msg", 발생한 Exception );
                return resultVO;
        }
```

- Exception 을 던지지 않고 Exception 후처리 로직처럼 수행후 계속 비즈니스로 돌아오는 방법도 필요 ➔ LeaveaTrace

❑ Exception

- **EgovBizException** 은 Biz flow 상에 Biz Exception.(사용자에 의해 발생시키는 Exception)
- **DataAccessException** 의 경우는 Spring 에서 persistence layer 에서 발생하는 Exception.
- **FdlException** 은 전자정부 표준프레임워크 에서 확장/추가된 영역에서 던져주는 Exception.
- **그외 나머지** 는 앞에서 나열한 Exception 을 제외한 Exception

❑ Exception 후처리 흐름

- Exception 후처리 방식은

  AOP(pointCut ⇒ after-throw) ⇒ ExceptionTransfer.transfer() ⇒ ExceptionHandlerService ⇒ Handler

  순으로 실행된다.

❑ ServiceImpl 클래스 개발

– 업무 로직이 담기는 ServiceImpl 클래스는 아래와 같이 AbstractServiceImpl 을 상속 받는다.

```
@Service("categoryService")
public class CategoryServiceImpl extends AbstractServiceImpl implements CategoryService {
…
}
```

– AbstractServiceImpl 클래스는 추상클래스로 processException 메소드를 하위 클래스에서 사용할 수 있도록 해준다.

```
public abstract class AbstractServiceImpl {
    @Resource(name = "messageSource")
    private MessageSource messageSource;

    @Resource(name = "leaveaTrace")
    private LeaveaTrace traceObj;

    protected EgovBizException processException(final String msgKey) {
        return processException(msgKey, new String[] {});
    }

    protected EgovBizException processException(final String msgKey, final String[] msgArgs) {
        return processException(msgKey, msgArgs, null);
    }
    ……
```

❑ eGov Exception Handling 설정 및 설명

- ServiceImpl 업무 클래스에서 발생한 Exception 은 아래 설정에 따라 ExceptionTransfer.transfer(..) 로 위임된다.

– Aop Config : Bean 설정 (after-throwing Advice)

```
<aop:config>
        <aop:pointcut id="serviceMethod"
            expression="execution(* egov.sample.service.*Impl.*(..))" />

        <aop:aspect ref="exceptionTransfer">
            <aop:after-throwing throwing="exception"
                pointcut-ref="serviceMethod" method="transfer" />
        </aop:aspect>
    </aop:config>

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

- AspectJ req expression (pointcut) 해당되는 클래스상에 Exception 발생시 ExceptionTransfer 클래스로 위임.
- 실제 로직은 transfer 메소드에 구현을 하고 있음.

– ExceptionTransfer : Exception 종류에 따라 구분하고 있다.

```
public class ExceptionTransfer {
…
  public void transfer(JoinPoint thisJoinPoint, Exception exception) throws Exception {
   …
        // BizException 인 경우는 이미 메시지 처리 되었음. 로그만 기록
        if (exception instanceof EgovBizException) {
            …
            EgovBizException be = (EgovBizException) exception;
            getLog(clazz).error(be.getMessage(), be.getCause());
            // Exception Handler 에 발생된 Package 와 Exception 설정.
            processHandling(clazz, exception, pm, exceptionHandlerServices, false);
            throw be;

        } else if (exception instanceof RuntimeException) {
            …
            // Exception Handler 에 발생된 Package 와 Exception 설정.
            processHandling(clazz, exception, pm, exceptionHandlerServices, true);

            if (be instanceof DataAccessException) {
                log.debug("RuntimeException case :: DataAccessException ");
                DataAccessException sqlEx = (DataAccessException) be;
                throw sqlEx;
            }
            throw be;

        } else if (exception instanceof FdlException) {
            …
            FdlException fe = (FdlException) exception;
            getLog(clazz).error(fe.getMessage(), fe.getCause());

            throw fe;

        } else {
            …
            getLog(clazz).error(exception.getMessage(), exception.getCause());
            throw processException(clazz, "fail.common.msg", new String[] {}, exception, locale);
        }
    }
  }
}
```

3. Exception Handling - 설명(5/11)

– ExceptionHandleManager : Exception 발생 클래스 와 등록된 Handlers 사이의 매핑을 해주는 역할수행

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

<bean id="defaultExceptionHandleManager"
        class="egovframework.rte.fdl.cmmn.exception.manager.DefaultExceptionHandleManager">
        <property name="patterns">
            <list>
                <value>**service.*Impl.methodName</value>
            </list>
        </property>
        <property name="handlers">
            <list>
                <ref bean="egovHandler" />
            </list>
        </property>
    </bean>
```

- 여기서는 발생한 Exception 이 **service.*Impl .methodName조건에 맞는 경우 egovHandler 를 실행시킨다.
- Patterns 나 handlers 는 여러 개를 등록할 수 있다.

3. Exception Handling - 설명(6/11)

- **ExceptionHandleManager 구현체는 AbsExceptionHandleManager 를 상속해야 하며 ExceptionHandlerService 를 인터페이스로 취해야 한다. 실제 구현 메소드는 run (..) 이다.**

```
public class DefaultExceptionHandleManager extends AbsExceptionHandleManager implements ExceptionHandlerService {

    @Override
    public boolean run(Exception exception) throws Exception {

        log.debug(" DefaultExceptionHandleManager.run() ");

        // 매칭조건이 false 인 경우
        if (!enableMatcher())
            return false;

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

        return true;
    }

}
```

– **EgovServiceExceptionHandler** (샘플)

- Handler 를 간단하게 구현한 예이다. 발생한 Exception 을 메일로 발송하고 있다.

```
public class EgovServiceExceptionHandler implements ExceptionHandler {
    @Resource(name = "otherSSLMailSender")
    private SimpleSSLMail mailSender;

    public void occur(Exception ex, String packageName) {
        log.debug(" EgovServiceExceptionHandler run..............");
        try {
            mailSender.send(ex, packageName);
            log.debug(" sending a alert mail  is completed ");
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

- ExceptionHander 인터페이스를 구현하여 Bean 설정에 넣어주기만 하면 된다.

```
<bean id="defaultExceptionHandleManager"
        class="egovframework.rte.fdl.cmmn.exception.manager.DefaultExceptionHandleManager">
        <property name="patterns">
            <list>
                <value>**service.*Impl</value>
            </list>
        </property>
        <property name="handlers">
            <list>
                <ref bean="egovHandler" />
            </list>
        </property>
</bean>
<bean
id="egovHandler" class="egovframework.rte.fdl.cmmn.exception.handler.EgovServiceExceptionHandler" />
```

❑ LeavaTrace 설정 및 설명

- Exception 이거나 Exception 이 아닌 경우에도 Trace 후처리 로직을 실행 시키고자 할 때 사용한다.
- 실제 Exception 이 발생하지는 않고 후처리로직만 흐른다.
- 구현/적용방식은 Exception 후처리로직과 유사하다.

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

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

– TraceHandler 구현체 만들기

```
public class DefaultTraceHandler implements TraceHandler {

    public void todo(Class clazz, String message) {

        System.out.println(" log ==> DefaultTraceHandler run..............");
    }

}
```

– leaveaTrace 코드상 발생 Sample

```
public CategoryVO selectCategory(CategoryVO vo) throws Exception {
    CategoryVO resultVO = categoryDAO.selectCategory(vo);
    try {
      //강제로 발생한 ArithmeticException
        int i = 1 / 0;
    } catch (ArithmeticException athex) {
      //Exception 을 발생하지 않고 후처리 로직 실행.
        leaveaTrace("message.trace.msg");
    }

    return resultVO;
}
```

❑ Exception 화면 Handling

- 화면단의 Exception 처리는 Controller 로 전달된 Excpetion에 따른 지정된 에러 화면 이동으로 이루어져 있다. Controller 인터페이스의 handleRequest() 메소드는 Excetpion 을 발생하도록 정의되어 있다.
- **DispathcherServlet 은 HandlerExceptionResolver가 등록 되어 있는 경우에는 Exception이 발생하는 경우 HandlerExceptionResolver에게 위임한다.**

  HandlerExceptionResolver는 아래와 같은 인터페이스이다.

```
public interface HandlerExceptionResolver {

      ModelAndView resolveException(
                  HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex);

}
```

- Bean Configuration

```
<bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">
   <property name="defaultErrorView" value="exception/CommonCaseException"/>
   <property name="exceptionMappings">
       <props>
           <prop key="org.egovframe.exception.ACaseException">exception/ACaseException</prop>
           <prop key="org.egovframe.exception.BCaseException">exception/BCaseException</prop>
       </props>
   </property>
 </bean>
```

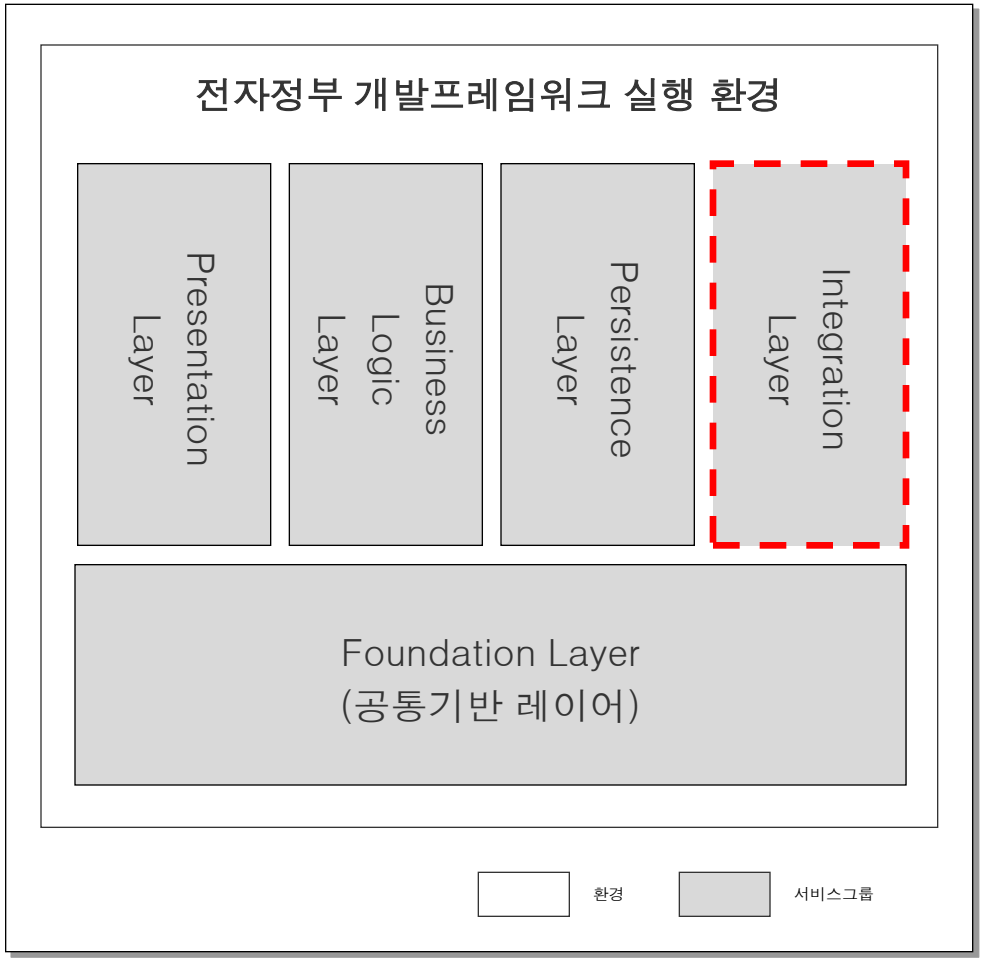
- 발생한 Exception 의 종류를 ACaseException, BCaseException , 기타 Exception 이 발생하는 경우 ACaseException 은 exceptionMapping 규칙에서 보이는 것 처럼 해당 Exception 과 매핑된 excpetion/ACaseExcetpion 을 결과값을 가져오게 된다. BCaseException 의 경우도 마찬가지다. 가져온 값은 ViewResolver에 의해 해당 화면으로 이동을 하게 된다. 그리고 excpetionMappings 에서 정의되지 않은 Exception 인 경우 defaultErrorView 에 매핑된 exception/CommonCaseExceptio 을 리턴하게 된다.
  결과적으로 보면 발생한 Exception 에 대한 별도의 정의를 하지 않는 다면 defaultErrorView 에 지정된 값의 화면으로 이동하게 된다는 것이다.

❑ The Spring Framework - Reference Documentation

- http://static.springsource.org/spring/docs/2.5.x/reference/
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/

1. 개요 - 실행환경 연계통합 레이어(1/3)

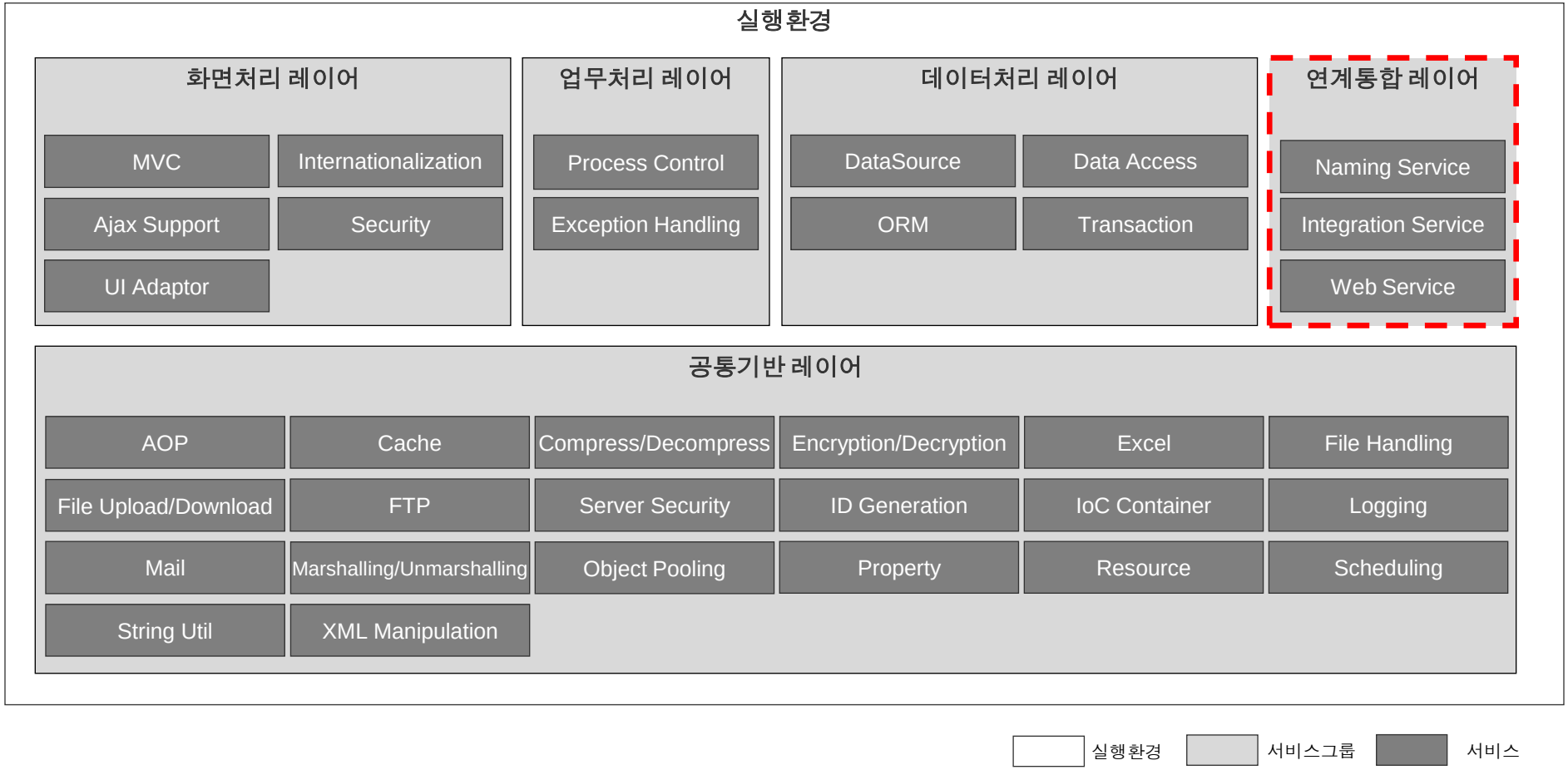
❑ 연계통합 레이어는 타 시스템과의 연동 기능을 제공하는 **Layer**임



서비스 그룹	설명
Presentation Layer	• 업무 프로그램과 사용자 간의 Interface를 담당하는 Layer로서, 사용자 화면 구성, 사용자 입력 정보 검증 등의 기능을 제공함
Business Logic Layer	• 업무 프로그램의 업무 로직을 담당하는 Layer로서, 업무 흐름 제어, 에러 처리 등의 기능을 제공함
Persistence Layer	• 데이터베이스에 대한 연결 및 영속성 처리, 선언적인 트랜잭션 관리를 제공하는 Layer임
Integration Layer	• 타 시스템과의 연동 기능을 제공하는 Layer임
Foundation Layer (공통기반 레이어)	• 실행 환경의 각 Layer에서 공통적으로 사용하는 공통 기능을 제공함

1. 개요 - 실행환경 연계통합 레이어(2/3)

❑ 연계통합 레이어는 **Naming Service, Integration Service, Web Service** 등 총 **3**개의 서비스를 제공함



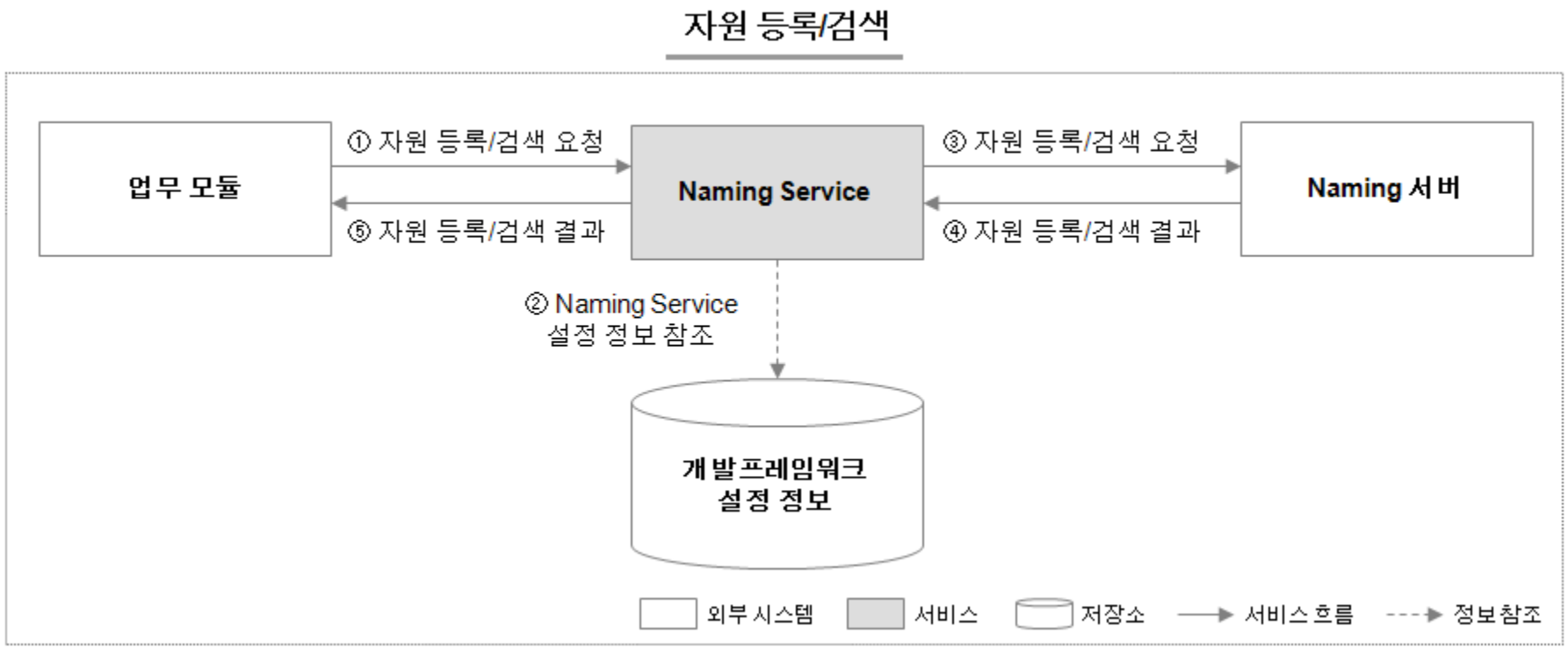
❑ 연계통합 레이어는 **Naming Service, Web Service** 등 총 **2**종의 오픈소스 **SW**를 사용하고 있음

서비스	오픈소스 SW	버전
Naming Service	Spring	3.0.5
Web Service	Apache CXF	2.3.3

❑ 서비스 개요

– 원격에 있는 모듈 및 자원 등을 찾아주는 서비스

- Naming 서비스를 지원하는 Naming 서버에 자원을 등록하여 다른 어플리케이션에서 사용할 수 있도록 공개하고, Naming 서버에 등록되어 있는 자원을 찾아와서 이용할 수 있게 함



❑ 주요 기능

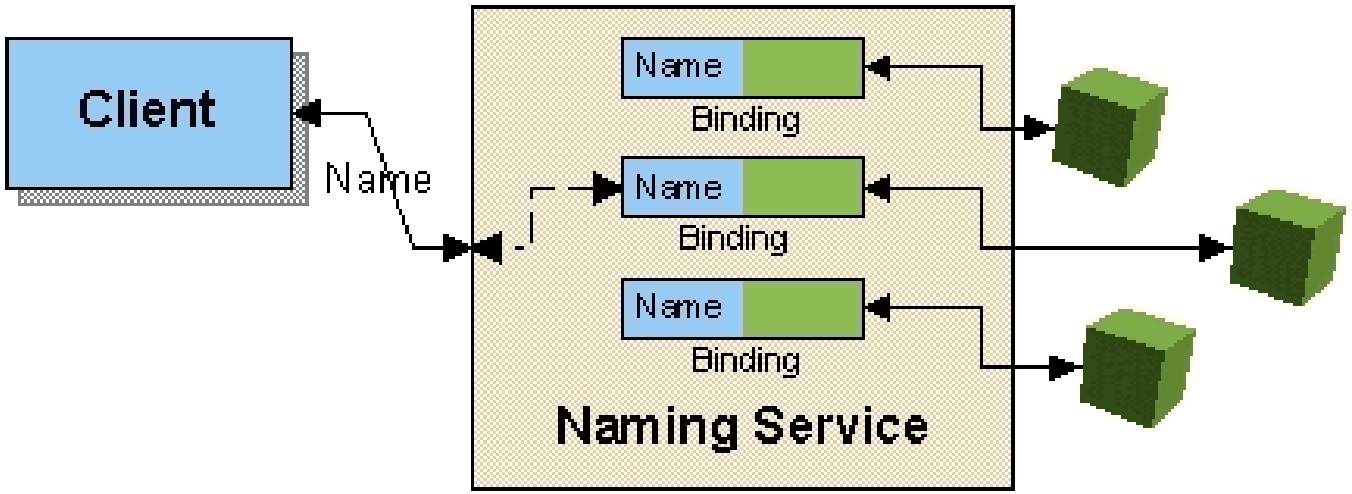
- 자원 등록 (bind, rebind)
  - Naming 서버에 로컬에 있는 자원을 등록함
- 자원 검색 (lookup)
  - Naming 서버에 등록되어 있는 자원을 이름을 이용하여 조회함
- 자원 등록 해제 (unbind)
  - Naming 서버에 등록되어 있는 로컬 자원에 대한 등록을 해제함

❑ Open Source

- Spring Framework : Naming Service는 Spring Framework에서 제공하는 기능을 수정없이 사용함
  - “jee:jndi-lookup” tag : Spring XML Configuration 설정파일에 JNDI 객체를 bean으로 등록하는 방식으로, JNDI 객체를 Lookup 만 할 수 있다. 일반적으로 가장 많이 사용된다.
  - JndiTemplet 클래스 : Spring Framework에서 JNDI API를 쉽게 사용할 수 있도록 제공하는 JndiTemplate class를 직접 사용하는 방식으로, JNDI API 기능을 모두 사용해야 할 경우 사용하는 방식이다.

❑ Java Naming and Directory Interface(JNDI)란

– Java Naming and Directory Interface(JNDI)는 Java 소프트웨어 클라이언트가 이름(name)을 이용하여 데이터 및 객체를 찾을 수 있도록 도와주는 네이밍/디렉토리 서비스에 연결하기 위한 Java API이다.



❑ 개요

– Spring Framework는 XML Configuration 파일에 JNDI 객체를 설정할 수 있다. 단, 설정 파일을 통해서는 JNDI 객체를 lookup하는 것만 가능하므로, bind, rebind, unbind 기능을 사용하려면 Using JndiTemplate 방식을 사용해야 한다.

❑ 설정

– jee tag를 사용하기 위해서는 Spring XML Configuration 파일의 머릿말에 namespace와 schemaLocation를 추가해야 한다.

```
<?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:jee="http://www.springframework.org/schema/jee"
       xsi:schemaLocation="
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-3.0.xsd">

    <!-- <bean/> definitions here -->

</beans>
```

❑ 'jndi-lookup' tag

– 'jndi-lookup' tag는 JNDI 객체를 찾아서 bean으로 등록해주는 tag이다.

```
<jee:jndi-lookup id="bean id" jndi-name="jndi name" cache="true or false"
    resource-ref="true or false" lookup-on-startup="true or false"
    expected-type="java class" proxy-interface="java class">
    <jee:environment>
        name=value
        ping=pong
        ...
    </jee:environment>
</jee:jndi-lookup>
```

JNDI Environment 변수값을 등록할 때 사용한다.
'environment' element는 'foo=bar' 와 같이 <변수명>=<변수값> 형태의 List를 값으로 가진다.

Attribute	설명	Optional	Data Type	Default값
id	Spring XML Configuration의 bean id이다.	N	String	
jndi-name	찾고자 하는 JNDI 객체의 이름이다.	N	String	
cache	한번 찾은 JNDI 객체에 대한 cache여부를 나타낸다.	Y	boolean	true
resource-ref	J2EE Container 내에서 찾을지 여부를 나타낸다.	Y	boolean	false
lookup-on-startup	시작시에 lookup을 수행할지 여부를 타나낸다.	Y	boolean	true
expected-type	찾는 JNDI 객체를 assign할 타입을 나타낸다.	Y	Class	
proxy-interface	JNDI 객체를 사용하기 위한 Proxy Interface이다.	Y	Class	

❑ ‘jndi-lookup’ tag Sample(2/2)

– With multiple JNDI environment settings

아래는 복수 JNDI 환경 설정을 사용하여 JNDI 객체를 찾아오는 예제이다.

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/MyDataSource">
    <!-- newline-separated, key-value pairs for the environment (standard Properties
format) -->
    <jee:environment>
        foo=bar
        ping=pong
    </jee:environment>
</jee:jndi-lookup>
```

– Complex

아래는 이름 외 다양한 설정을 통해 JNDI 객체를 찾아오는 예제이다.

```
<jee:jndi-lookup id="dataSource"
                 jndi-name="jdbc/MyDataSource"
                 cache="true"
                 resource-ref="true"
                 lookup-on-startup="false"
                 expected-type="com.myapp.DefaultFoo"
                 proxy-interface="com.myapp.Foo" />
```

❑ 'jndi-lookup' tag Sample(1/2)

– Simple

가장 단순한 설정으로 이름만을 사용하여 JNDI 객체를 찾아준다. 아래 이름 "jdbc/MyDataSource"로 등록되어 있는 JNDI 객체를 찾아 "userDao" Bean의 "dataSource" property로 Dependency Injection하는 예제이다.

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/MyDataSource" />

<bean id="userDao" class="com.foo.JdbcUserDao">
    <!-- Spring will do the cast automatically (as usual) -->
    <property name="dataSource" ref="dataSource" />
</bean>
```

– With single JNDI environment settings

아래는 단일 JNDI 환경 설정을 사용하여 JNDI 객체를 찾아오는 예제이다.

```
<jee:jndi-lookup id="dataSource" jndi-name="jdbc/MyDataSource">
    <jee:environment>foo=bar</jee:environment>
</jee:jndi-lookup>
```

2. Naming Service – JndiTemplate 클래스 사용(1/3)

❑ 개요

– JndiTemplate class는 JNDI API를 쉽게 사용할 수 있도록 제공하는 wrapper class이다.

❑ Bind

– 아래 Jndi.TemplateSample class의 bind 메소드는 JndiTemplate을 이용하여 argument 'resource'를 argument 'name'으로 JNDI 객체로 bind한다.

```
import javax.naming.NamingException;
import org.springframework.jndi.JndiTemplate;

public class JndiTemplateSample {
    private JndiTemplate jndiTemplate = new JndiTemplate();

    public boolean bind(final String name, Object resource) {
        try {
            jndiTemplate.bind(name, resource);
            return true;
        }
        catch (NamingException e) {
            e.printStackTrace();
            return false;
        }
    }

    ...
}
```

❑ Lookup

– JndiTemplate을 이용하여 argument 'name'으로 등록되어 있는 자원(resource)를 찾을 수 있다.

```
public Object lookupResource(final String name) {
    try {
        return jndiTemplate.lookup(name);
    }
    catch (NamingException e) {
        e.printStackTrace();
        return null;
    }
}
```

❑ Lookup with requiredType

– JndiTemplate의 lookup 메소드는 찾고자 하는 자원의 이름 뿐 아니라 원하는 타입(Type)을 지정할 수 있다.

```
public Foo lookupFoo(final String fooName) {
    try {
        return jndiTemplate.lookup(fooName, Foo.class);
    }
    catch (NamingException e) {
        e.printStackTrace();
        return null;
    }
}
```

❑ Rebind

– JndiTemplate의 rebind 메소드를 사용하여 자원을 재등록할 수 있다.

```
public boolean rebind(final String name, Object resource) {
    try {
        jndiTemplate.rebind(name, resource);
        return true;
    }
    catch (NamingException e) {
        e.printStackTrace();
        return false;
    }
}
```

❑ Unbind

– JndiTemplate의 unbind 메소드를 사용하여 등록된 자원을 등록해제할 수 있다.

```
public boolean unbind(final String name) {
    try {
        jndiTemplate.unbind(name);
        return true;
    }
    catch (NamingException e) {
        e.printStackTrace();
        return false;
    }
}
```

❑ Spring Framework JndiTemplate class API

- http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/jndi/JndiTemplate.html
- http://static.springsource.org/spring/docs/3.0.x/javadoc-api/org/springframework/jndi/package-summary.html

❑ The Spring Framework - Reference Documentation A.2.3. The jee schema

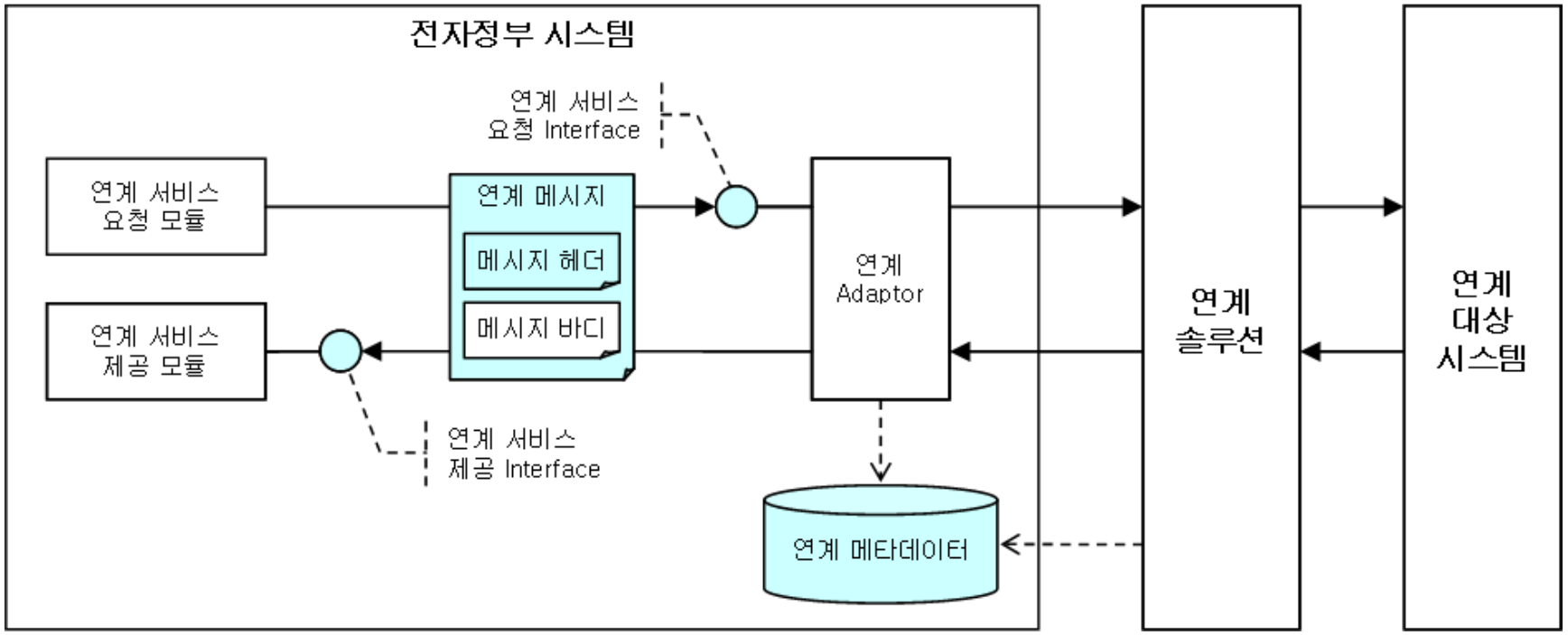
- http://static.springsource.org/spring/docs/2.5.x/reference/xsd-config.html#xsd-config-body-schemas-jee
- http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/xsd-config.html#xsd-config-body-schemas-jee-jndi-lookup

❑ Java SE Guide to JNDI

- http://java.sun.com/j2se/1.5.0/docs/guide/jndi/index.html

❑ 서비스 개요

– Integration 서비스는 전자정부 표준프레임워크 기반의 시스템이 타 시스템과의 연계를 위해 사용하는 Interface의 표준을 정의한 것이다.



❑ 목적

– 기존의 전자정부 시스템은 타 시스템과의 연계를 위해 연계 솔루션을 사용하거나 자체 개발한 연계 모듈을 사용해왔다. 기존에 사용된 연계 솔루션 및 자체 연계 모듈은 각각 고유한 설정 및 사용 방식을 가지고 있어, 동일한 연계 서비스라 할지라도 사용하는 연계 모듈에 따라 각기 다른 방식으로 코딩되어 왔다. 본 Integration 서비스는 이러한 중복 개발을 없애고, 표준화된 설정 및 사용 방식을 정의하여 개발 효율성을 제고한다.

❑ 구성

– Metadata

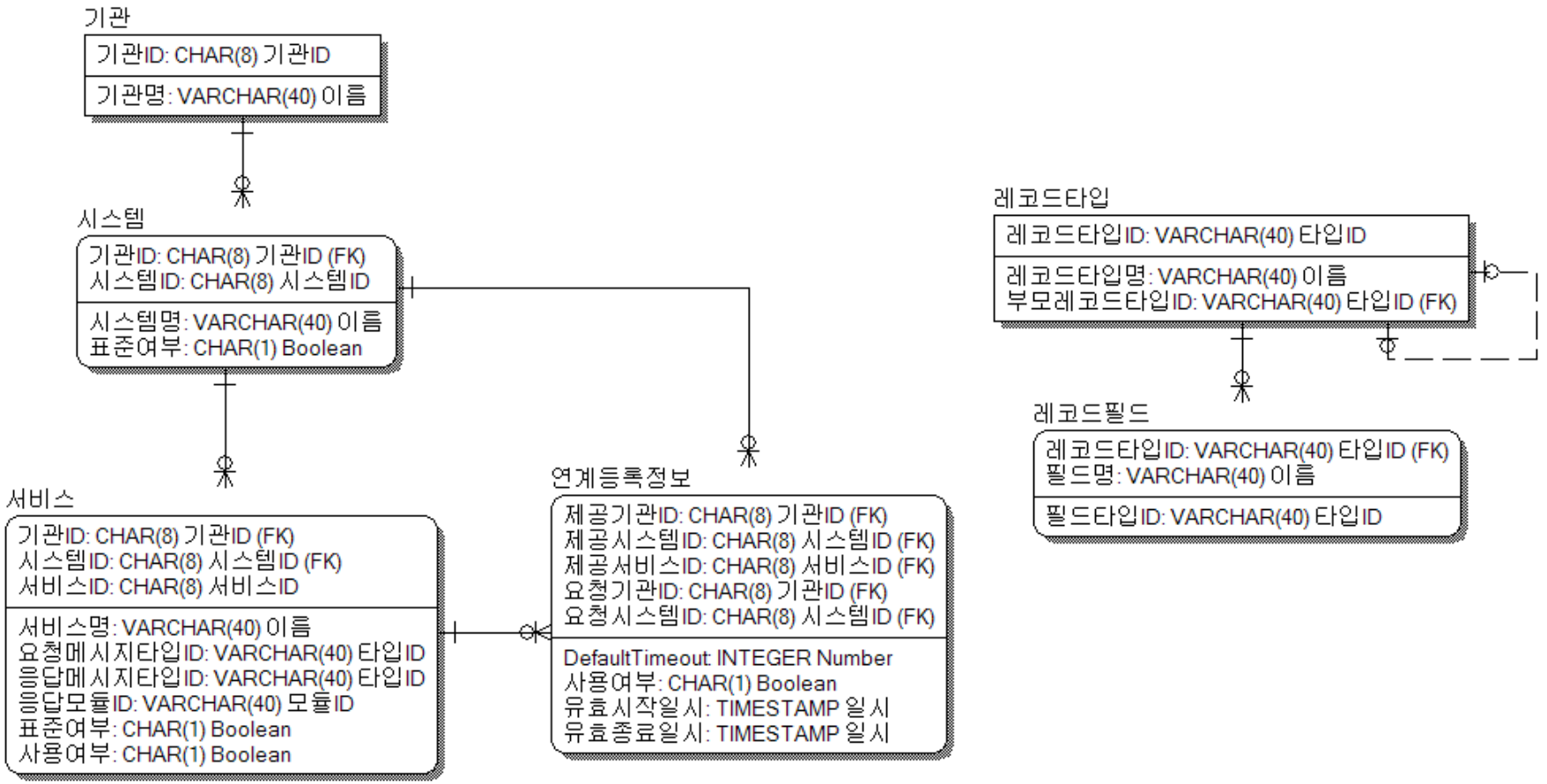
- 연계 Interface를 사용하기 위해 필요한 최소한의 정보(연계 기관 정보, 연계 시스템 정보, 연계 서비스 정보, 메시지 형식 등)을 정의하고 있다.

– 연계 서비스 API

- 연계 서비스 요청 Interface, 연계 서비스 제공 Interface, 연계 메시지 및 메시지 헤더 등을 정의하고 있다.

❑ 논리모델(1/2)

– 논리ERD



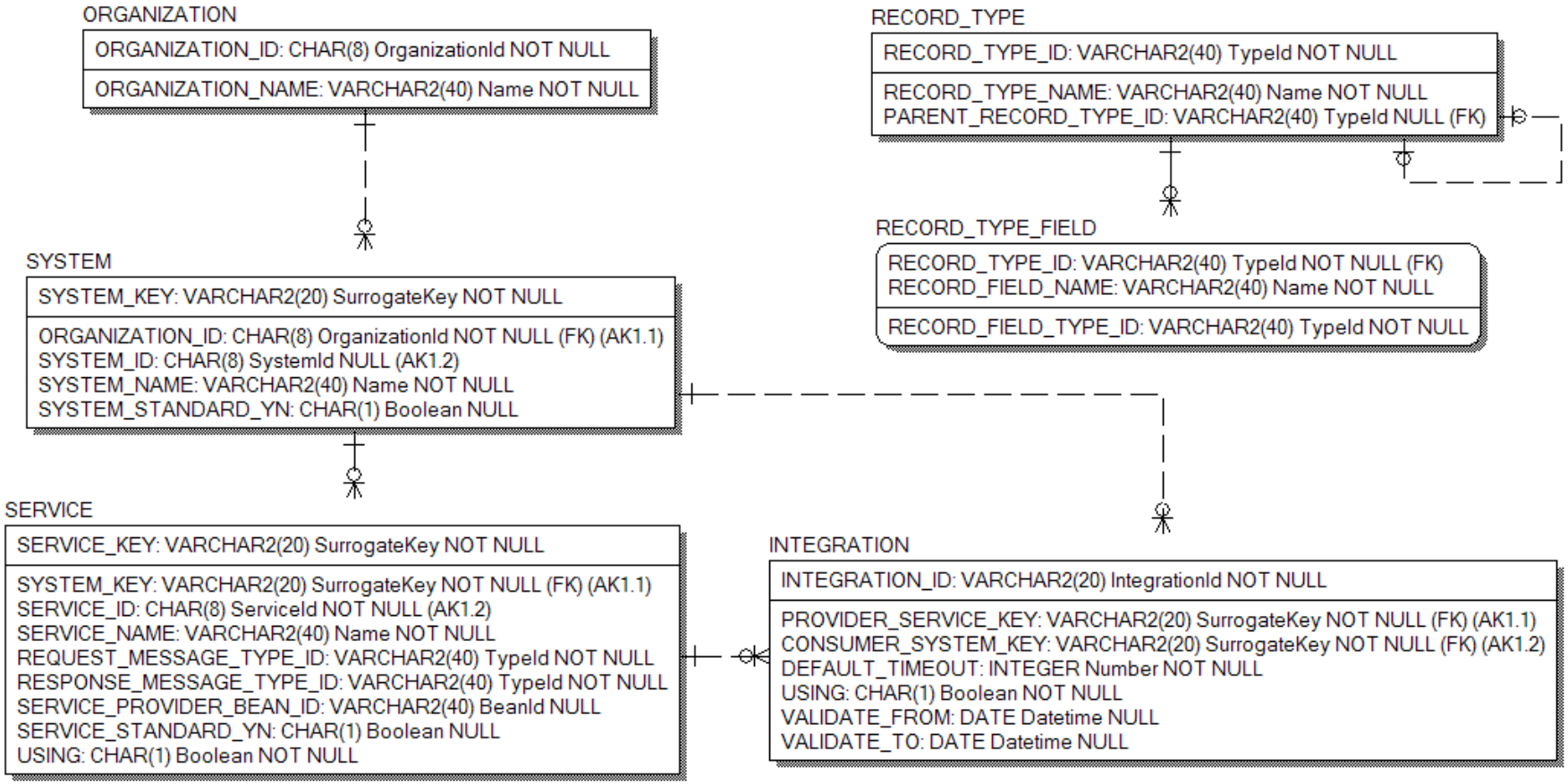
❑ 논리모델(2/2)

– Entity 설명

Entity	설명
기관	• 연계 서비스를 제공 또는 사용하는 기관을 나타낸다. • 하나의 기관은 다수의 시스템을 가지고 있다.
시스템	• 연계 서비스를 제공 또는 사용하는 시스템을 나타낸다. • 하나의 시스템은 반드시 하나의 기관에 속하며, 다수의 서비스를 가지고 있다.
서비스	• 연계 서비스를 제공하는 단위를 나타낸다. • 하나의 서비스는 반드시 하나의 시스템에 속한다.
연계등록정보	• 연계 서비스를 사용하기 위한 단위를 나타낸다. • 연계 요청 시스템이 연계 제공 서비스를 사용하기 위해서 등록해야 하는 정보를 담고 있다.
레코드타입	• 연계에 사용되는 메시지의 형태를 나타낸다. • <Key, Value> 쌍의 정보를 담고 있는 레코드 형태의 타입을 정의하고 있다. • 하나의 레코드타입은 다수의 레코드필드를 가지고 있다.
레코드필드	• 레코드타입에 속하는 내부 필드의 정의를 나타낸다. • 필드의 이름과 타입을 정의한다. • 하나의 레코드필드는 반드시 하나의 레코드타입에 속한다.

❑ 물리모델(1/2)

– 물리ERD



❑ 물리모델(2/2)

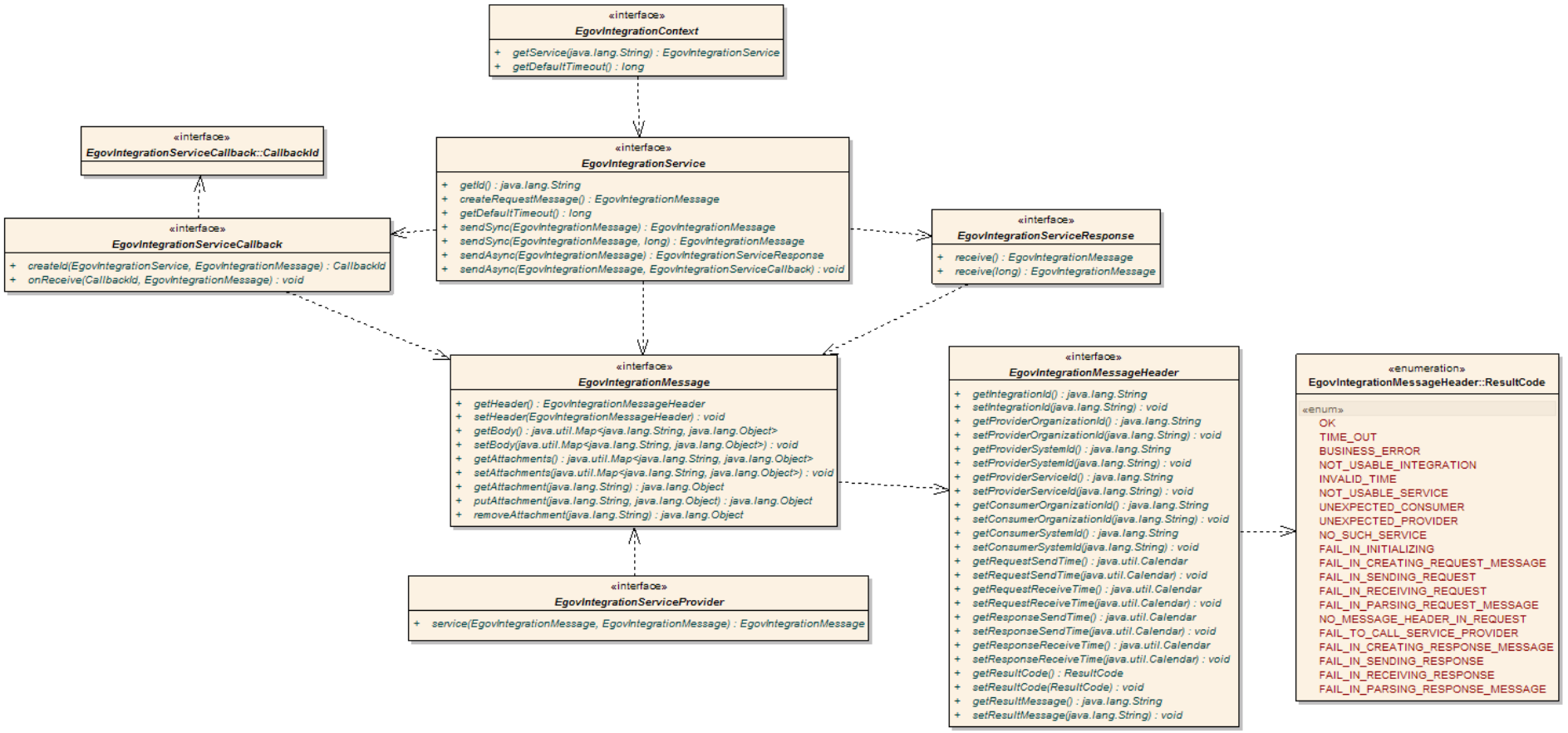
– 물리ERD 설명

- Integration 서비스 Metadata의 물리모델은 논리모델을 실제 물리적인 DB로 구현하기 위한 모델로서, 물리ERD는 Oracle DB를 가정하여 작성된 것이다.
- 물리모델은 Hibernate 등과 같은 Object Relational Mapping(ORM)을 사용하여 Access하는 것을 고려하여, 복수의 Attribute를 Identifier로 갖는 Entity를 Table로 변환할 때 Surrogate Key를 도입하고, 기존 Identifier는 Unique Constraints로 적용하여 정의되었다.
- Entity ↔ Table 매핑

Entity	Table
기관	ORGANIZATION
시스템	SYSTEM
서비스	SERVICE
연계등록정보	INTEGRATION
레코드타입	RECORD_TYPE
레코드필드	RECORD_TYPE_FIELD

❑ 구성(1/2)

– Class Diagram



❑ 구성(2/2)

– 구성요소 설명

구성요소	설명
EgovIntegrationContext	연계 서비스에 대한 설정 및 EgovIntegrationService 객체를 관리한다.
EgovIntegrationMessage	연계 서비스를 통해 주고 받는 표준 메시지를 정의한다.
EgovIntegrationMessageHeader	연계 서비스를 통해 주고 받는 표준 메시지 헤더를 정의한다.
EgovIntegrationMessageHeader:ResultCode	연계 서비스 결과 코드를 담고 있는 enumeration이다.
EgovIntegrationService	연계 서비스를 호출하기 위해 사용한다.
EgovIntegrationServiceResponse	연계 서비스를 비동기 방식으로 호출한 경우, 응답 메시지를 받기 위해 사용한다.
EgovIntegrationServiceCallback	연계 서비스를 비동기 방식으로 호출한 경우, 응답 메시지를 받기 위한 Callback interface이다.
EgovIntegrationServiceCallback:CallbackId	연계 서비스를 Callback을 이용한 비동기 방식으로 호출한 경우, 요청 메시지와 응답 메시지를 연결하기 위한 ID를 나타내는 interface이다.
EgovIntegrationServiceProvider	연계 서비스를 제공하기 위해 사용한다.

❑ EgovIntegrationContext

– EgovIntegrationContext는 연계 서비스에 대한 설정 및 EgovIntegrationService 객체를 관리한다. 연계 서비스를 사용하기 위해서는 EgovIntegrationContext의 getService 메소드를 사용하여 EgovIntegrationService 객체를 얻어와야 한다. 아래는 이름과 주민번호를 이용하여 실명확인 연계 서비스를 요청하는 예제이다.

```
public boolean verifyName(final String name, final String residentRegistrationNumber)
{
    // 연계ID가 "INT_VERIFY_NAME"인 연계 서비스 객체를 얻어온다.
    EgovIntegrationService service =
         egovIntegrationContext.getService("INT_VERIFY_NAME");

    // 요청 메시지 생성
     EgovIntegrationMessage requestMessage = service.createRequestMessage();

    // 요청 메시지 작성
    requestMessage.getBody().put("name", name);
    requestMessage.getBody().put("residentRegistrationNumber",
                                 residentRegistrationNumber);

    // 서비스 요청
     EgovIntegrationMessage responseMessage = service.sendSync(requestMessage);

    // 결과 return
    return responseMessage.getBody().get("result");
}
```

❑ EgovIntegrationMessage(1/2)

– EgovIntegrationMessage는 헤더부, 바디부, 첨부파일로 구성된다.

– 헤더부

• 헤더부를 access하기 위한 메소드는 아래와 같다.

Method Summary	
EgovIntegrationMessageHeader	getHeader()
void	setHeader(EgovIntegrationMessageHeader header)

– 바디부

• 바디부를 access하기 위한 메소드는 아래와 같다.

Method Summary	
Map<String, Object>	getBody()
void	setBody(Map<String, Object> body)

• 바디부는 다음 값들로만 구성될 수 있다.

- Java Primitive Type의 Wrapper 객체(Boolean, Byte, Short, Integer, Long, Float, Double)
- BigInteger, BigDecimal
- String
- Calendar
- List<Object>, Map<String, Object> (* List와 Map의 element 역시 위 값들로 구성되어야 한다.)

❑ EgovIntegrationMessage(2/2)

– 첨부파일

- 첨부파일을 access하기 위한 메소드는 아래와 같다.

Method Summary	
Map<String, Object>	getAttachments()
void	setAttachments(Map<String, Object> attachments)
Object	getAttachment(String name)
void	putAttachment(String name, Object attachment)
Object	removeAttachment(String name)

❑ EgovIntegrationMessageHeader

– EgovIntegrationMessageHeader는 다음과 같은 정보를 담고 있다.

Attribute Name	Data Type	설명
IntegrationId	String	연계ID
ProviderOrganizationId	String	연계 제공 기관ID
ProviderSystemId	String	연계 제공 시스템ID
ProviderServiceId	String	연계 제공 서비스ID
ConsumerOrganizationId	String	연계 요청 기관ID
ConsumerSystemId	String	연계 요청 시스템ID
RequestSendTime	Calendar	요청 송신 시각
RequestReceiveTime	Calendar	요청 수신 시각
ResponseSendTime	Calendar	응답 송신 시각
ResponseReceiveTime	Calendar	응답 수신 시각
ResultCode	ResultCode	결과 코드
ResultMessage	String	결과 메시지

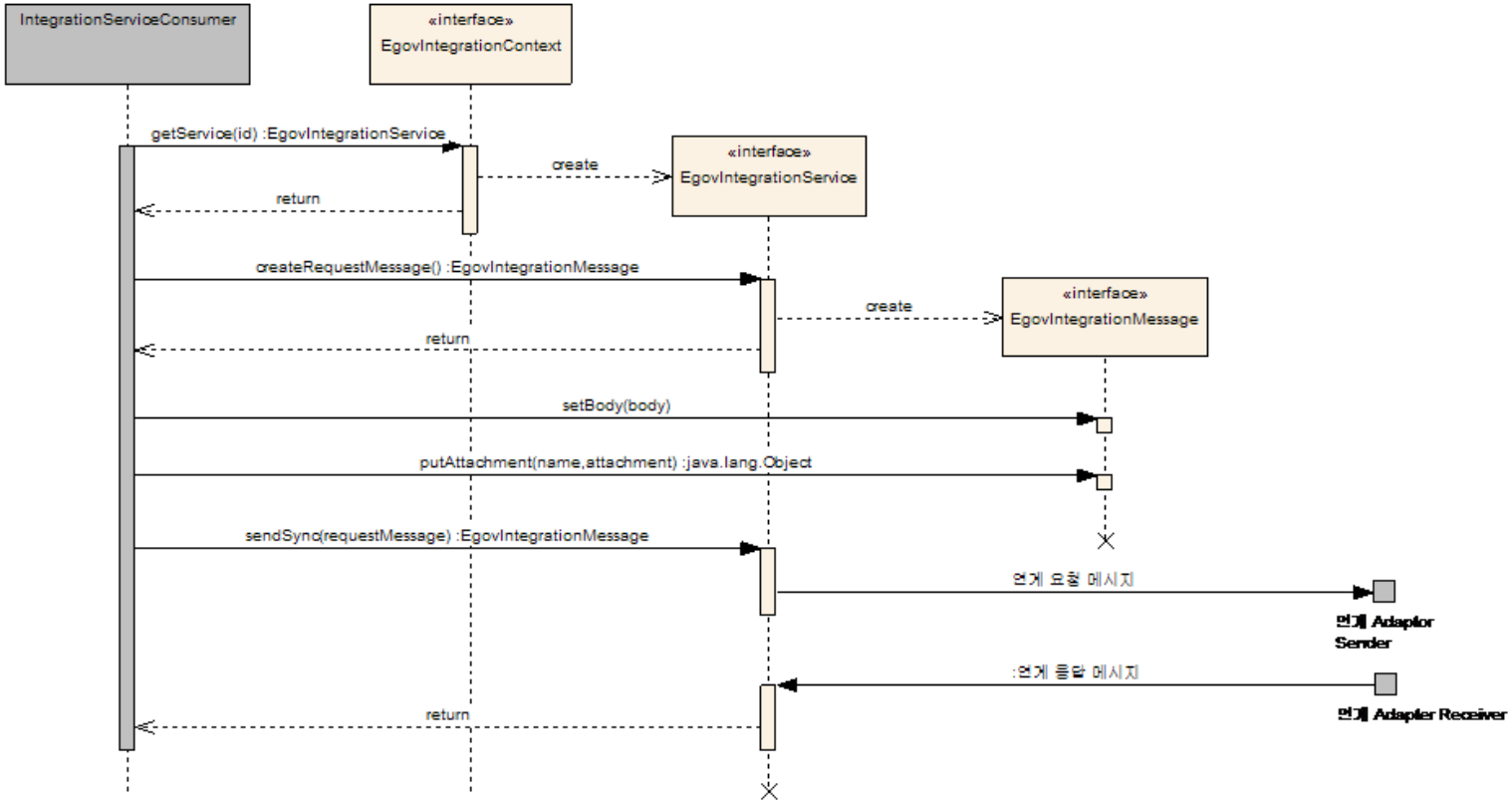
– EgovIntegrationMessageHeader는 위 attribute들에 대한 get/set 메소드를 정의하고 있다.

❑ EgovIntegrationService(1/8)

– EgovIntegrationService는 동기화 방식의 호출과 비동기화 방식의 호출을 지원한다.

– 동기화 방식(1/2)

• Sequence Diagram



❑ EgovIntegrationService(2/8)

– 동기화 방식(2/2)

- Timeout 을 지정한 예제

```
public boolean verifyName(final String name, final String residentRegistrationNumber)
{
    // EgovIntegrationContext에서 EgovIntegrationService 객체를 얻어온 후, 요청 메시지를
    // 생성 및 작성한다.
    ...

    // 동기방식으로 연계 서비스 호출 (timeout = 5000 millisecond)
    EgovIntegrationMessage responseMessage = service.sendSync(requestMessage, 5000);

    // 응답 결과 처리
    ...
 }
```

- Timeout을 설정하지 않은 예제 (이 경우 default timeout 값을 사용한다)

```
    EgovIntegrationMessage responseMessage = servicd.sendSync(requestMessage);
```

❑ EgovIntegrationService(3/8)

– 비동기화 방식 개요

- EgovIntegrationService의 sendAsync 메소드는 비동기화 방식으로 연계 서비스를 호출한다. sendAsync 메소드는 Response 방식과 Callback 방식으로 두 가지 방식이 존재한다.
- Response를 사용한 비동기 방식

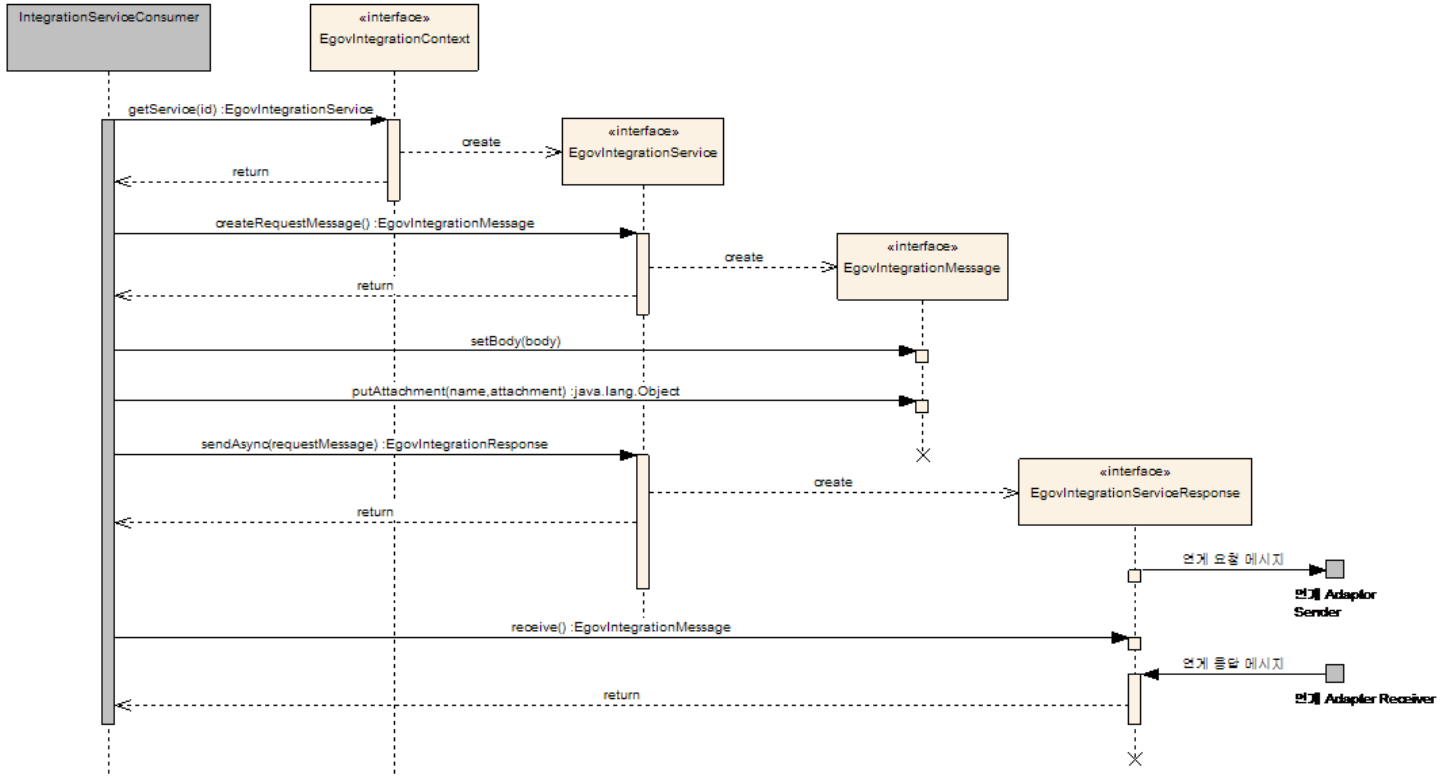
  : EgovIntegrationServiceResponse를 이용한 비동기 호출 방식이다. Response 방식의 비동기 호출은 연계 서비스를 요청하는 업무 모듈의 응답에 대한 ownership를 가지고 있으며, 응답 결과를 스스로 처리해야 하는 경우 사용한다.
- Callback을 사용한 비동기 방식

  : EgovIntegrationServiceCallback을 이용한 비동기 호출 방식이다. Callback 방식의 비동기 호출에서 연계 서비스를 요청하는 업무 모듈은 단지 요청만을 수행하고, 응답에 대한 처리는 Callback 객체에게 위임해도 상관없는 경우 사용한다.

❑ EgovIntegrationService(4/8)

– Response를 사용한 비동기 방식(1/2)

- EgovIntegrationServiceResponse를 이용한 비동기 호출 방식이다. Response 방식의 비동기 호출은 연계 서비스를 요청하는 업무 모듈의 응답에 대한 ownership를 가지고 있으며, 응답 결과를 스스로 처리해야 하는 경우 사용한다.
- Sequence Diagram



❑ EgovIntegrationService(5/8)

– Response를 사용한 비동기 방식(2/2)

- Timeout 을 지정한 예제

```
public boolean verifyName(final String name, final String residentRegistrationNumber)
{
    // EgovIntegrationContext에서 EgovIntegrationService 객체를 얻어온 후, 요청 메시지를
    // 생성 및 작성한다.
    ...

    // 비동기방식으로 연계 서비스 호출
    EgovIntegrationServiceResponse response = service.sendAsync(requestMessage);

    // response 객체를 이용하여 응답 메시지를 받기 전에 필요한 업무를 수행
     ...

    // response 객체를 이용하여 응답 메시지 수신(timeout = 5000 millisecond)
    EgovIntegrationMessage responseMessage = response.receive(5000);

    // 응답 메시지 처리
    ...
}
```

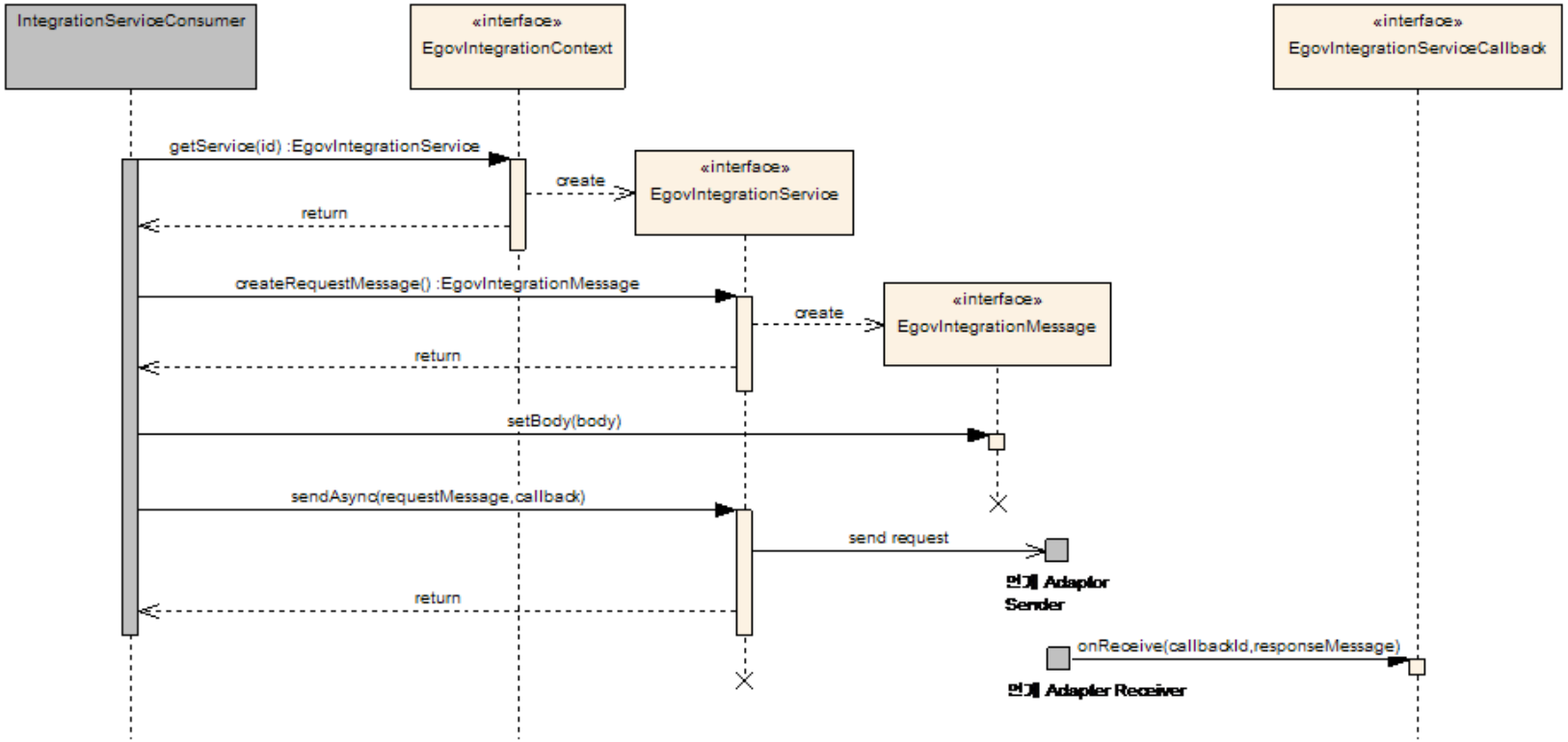
- Timeout 을 지정하지 않은 예제 (이 경우 default timeout 값을 사용한다)

```
    EgovIntegrationMessage responseMessage = response.receive();
```

❑ EgovIntegrationService(6/8)

– Callback을 사용한 비동기 방식(1/3)

- EgovIntegrationServiceCallback을 이용한 비동기 호출 방식이다. Callback 방식의 비동기 호출은 연계 서비스를 요청하는 업무 모듈은 단지 요청만을 수행하고, 응답에 대한 처리는 Callback 객체에게 위임해도 상관없는 경우 사용한다.
- Sequence Diagram



❑ EgovIntegrationService(7/8)

– Callback을 사용한 비동기 방식(2/3)

- 예제(1/2)

  아래는 Callback 방식으로 연계 서비스를 호출하는 예제이다.

```
@Resource(name = "verifyNameServiceCallback")
private EgovIntegrationServiceCallback callback;

public boolean verifyName(final String name, final String residentRegistrationNumber)
{
    // EgovIntegrationContext에서 EgovIntegrationService 객체를 얻어온 후, 요청 메시지를
    // 생성 및 작성한다.
    ...

    // 비동기방식으로 연계 서비스 호출
     service.sendSync(requestMessage, callback);
}
```

❑ EgovIntegrationService(8/8)

– Callback을 사용한 비동기 방식(3/3)

- 예제(2/2)

  아래는 Callback 모듈 예제이다.

```
public class VefiryNameServiceCallback {
    public CallbackId createId(EgovIntegrationService service,
                               EgovIntegrationMessage requestMessage) {
        // 본 메소드는 EgovIntegrationService를 구현한 연계 Adaptor 또는 솔루션에서 불리워진다.
        // 서비스와 요청 메시지를 이용하여 CallbackId를 생성하여 return한다.
        // 생성한 CallbackId는 응답 메시지 수신 시, 해당하는 서비스 및 요청 메시지를 식별하기 위해
        // 사용한다.

        // CallbackId 생성
        CallbackId callbadkId = ...

        return callbackId;
    }

    public vod onReceive(CallbackId callbackId, EgovIntegrationMessage responseMessage) {
        // 본 메소드는 처리해야 하는 응답 메시지가 도착했을 때, 연계 Adaptor 또는 솔루션에 의해
        // 불리워진다.

        // 응답 메시지 처리
         ...
    }
}
```

❑ EgovIntegrationServiceProvider

– EgovIntegrationServiceProvider interface는 연계 서비스를 제공하기 위한 interface로 연계 서비스를 제공하는 모듈은 본 interface를 implements 해야 한다.

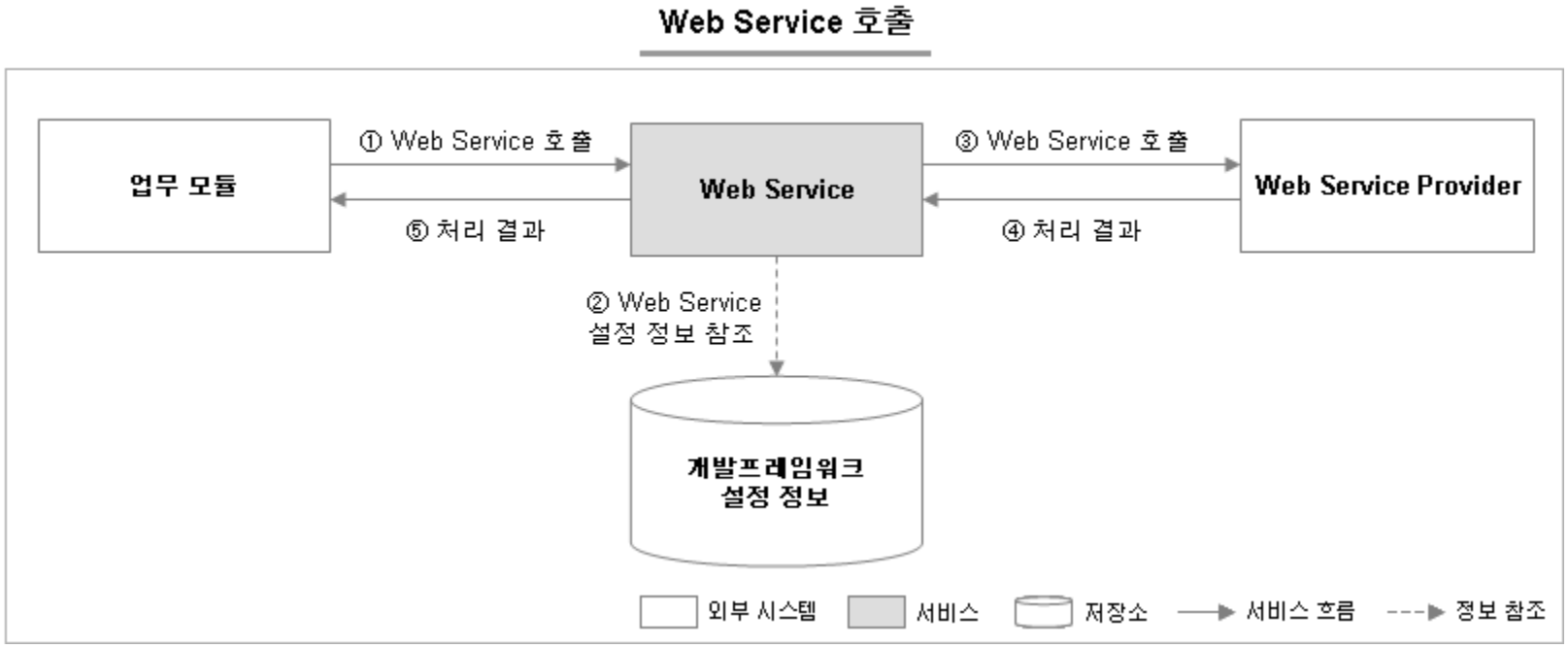
– 예제

```
package itl.sample;

import egovframework.rte.itl.integration.EgovIntegrationMessage;
import egovframework.rte.itl.integration.EgovIntegrationServiceProvider;

public class ServiceVerifyName implements EgovIntegrationServiceProvider {
    public void service(EgovIntegrationMessage requestMessage,
                        EgovIntegrationMessage responseMessage) {
        String name = requestMessage.getBody().get("name");
        String residentRegistrationNumber =
            requestMessage.getBody().get("residentRegistrationNumber");

        // 실명 확인
        boolean result = varifyName(name, residentRegistrationNumber);

        responseMessage.getBody().put("result", result);
    }
}
```

```
    <bean id="serviceVerifyName" class="itl.sample.ServiceVerifyName"/>
```

❑ 서비스 개요

– Integration Service 표준에 따라 작성된 Service로, Web Service를 호출하고 제공할 수 있도록 지원한다.



❑ 주요 기능

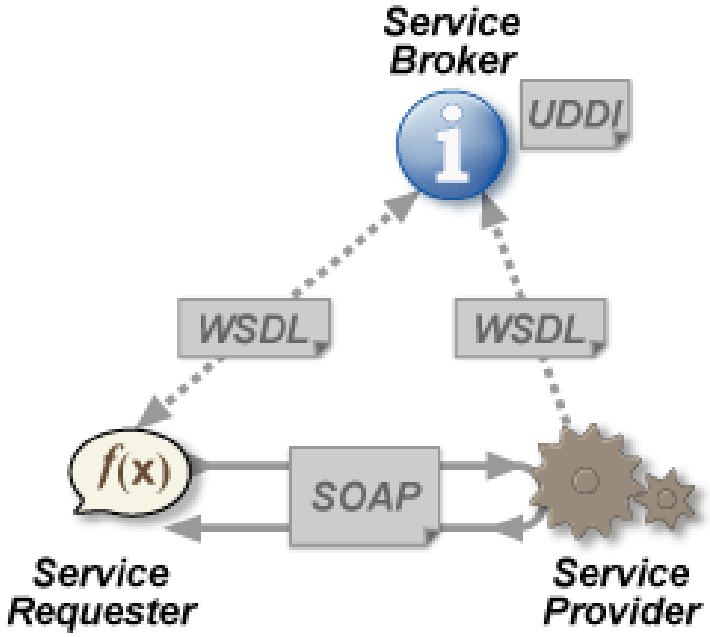
- Web Service 호출
  - 공개되어 있는 Web Service를 호출하고, 처리 결과를 돌려준다.
- Web Service 공개
  - 개발한 업무 모듈을 Web Service로 공개할 수 있도록 Proxy 등을 제공하고, 완성된 Web Service로 공개한다.

❑ Open Source

- WebService는 Web Service를 호출 및 제공하기 위해서 Apache CXF를 사용한다.

❑ Web Service란?

– W3C는 Web Service를 “네트워크 상에서 발생하는 컴퓨터 간의 상호작용을 지원하기 위한 소프트웨어 시스템”으로 정의하고 있다. 일반적으로 Web Service는 인터넷과 같은 네트워크 상에서 접근되고, 요청된 서비스를 제공하는 원격 시스템에서 수행되는 Web APIs이다.



❑ 문서 구성

– WebService는 Integration Service 표준에 따라 구현된 서비스로서, Integration Service에서 정의하고 있는 연계 서비스 API를 제공하고 있다. 따라서 본 문서는 WebService를 프로그램적으로 호출하는 방식이 아닌, WebService를 사용하기 위한 설정 방법을 중점적으로 설명한다(호출방식은 Integration Service의 연계 서비스 API를 참조한다).

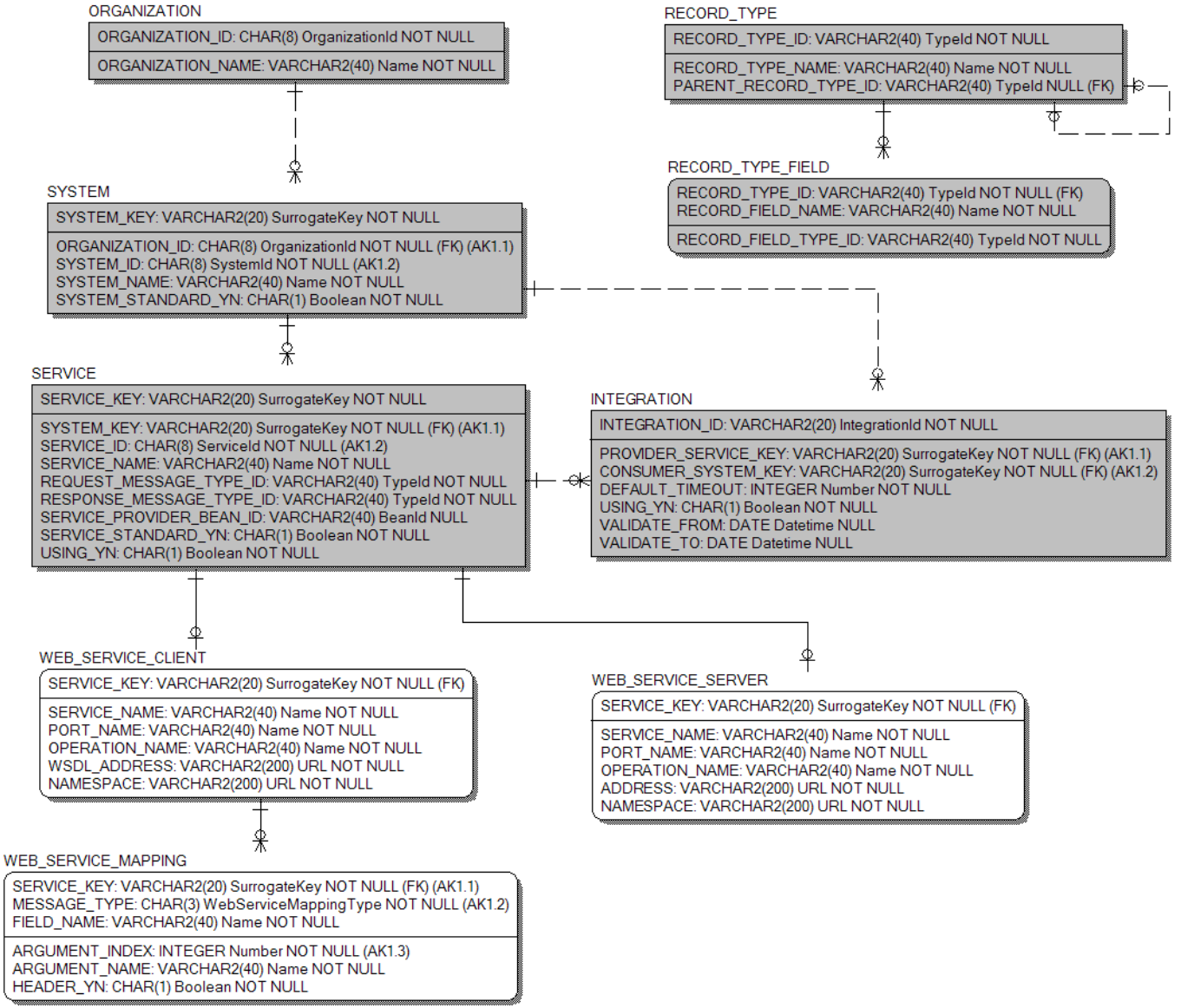
– 본 문서의 구성은 아래와 같다.

- Metadata
  1. WebService를 사용하기 위해 필요한 설정정보 구조를 설명한다.
- 설정 방법
  1. WebService를 사용하기 위해 필요한 설정 방법을 설명한다.
- Client 모듈 개발
  1. WebService Client 모듈을 개발하는 방법을 설명한다.
- Server 모듈 개발
  ➢ WebService Server 모듈을 개발하는 방법을 설명한다.

❑ Metadata(1/2)

– 물리ERD

- WebService는 Integration Service 표준을 따르므로 Integration Service의 Metadata와 연동하여 동작한다.
  (회색 부분이 Integration Service의 Metadata이다)



❑ Metadata(2/2)

– Table 설명

Table	설명
WEB_SERVICE_SERVER	연계 서비스를 Web Service 형태로 공개(publish)하기 위해 필요한 정보를 담고 있다.
WEB_SERVICE_CLIENT	Web Service 형태로 공개(publish)되어 있는 연계 서비스를 호출하기 위해 필요한 정보를 담고 있다.
WEB_SERVICE_MAPPING	전자정부 Integration 서비스 표준에 따라 개발된 서비스가 아닌 기존의 Legacy 시스템의 Web Service를 호출하기 위해, 표준 메시지와 Web Service 메시지 간의 mapping 정보를 담고 있다.

4. WebService – 설정 방법(1/3)

❑ 개요

– WebService를 사용하기 위해서는 다음의 설정이 필요하다.

- “pom.xml” 파일에 dependency 설정 추가
- Spring XML Configuration 설정

❑ “pom.xml” 파일에 dependency 설정 추가

– WebService를 사용하기 위해서 pom.xml의 dependencies tag에 다음 dependency를 추가한다.

- <version/> element의 값인 ${egovframework.versioin}에는 사용할 egovframework의 version을 기재한다.

```
<dependency>
    <groupId>egovframework.rte</groupId>
    <artifactId>egovframework.rte.itl.webservice</artifactId>
    <version>${egovframework.version}</version>
</dependency>
```

❑ Spring XML Configuration 설정(1/2)

– WebService를 위한 기본적인 설정이 포함된 “context-webservice.xml” 파일을 Spring XML Configuration 파일에 import한다.

```
<import resource="classpath:/egovframework/rte/itl/webservice/context/context-webservice.xml"/>
```

– 그리고 Context와 DataSource를 등록해야 한다.(DataSource의 경우, 프로젝트에서 사용하는 것이 있을 경우 설정하지 않아도 된다. 단, 반드시 id가 “dataSource”이여야 한다.)

```
<!-- EgovWebServiceContext 이다.
     organizationId 와 systemId 는 현재 시스템의 기관ID 및 시스템ID를 넣어야 한다. -->
<bean id="egovWebServiceContext"
      class="egovframework.rte.itl.webservice.EgovWebServiceContext"
      init-method="init">
    <property name="organizationId" value="ORG_EGOV"/>
    <property name="systemId" value="SYS00001"/>
    <property name="defaultTimeout" value="5000"/>
    <property name="integrationDefinitionDao" ref="integrationDefinitionDao"/>
    <property name="webServiceServerDefinitionDao" ref="webServiceServerDefinitionDao"/>
    <property name="webServiceClientDefinitionDao" ref="webServiceClientDefinitionDao"/>
    <property name="typeLoader" ref="typeLoader"/>
    <property name="classLoader" ref="classLoader"/>
</bean>
```

이 부분(기관ID, 시스템ID, defaultTimeout)만 수정하면 된다.

나머지 DAO 및 typeLoader, classLoader는 “context-webservice.xml” 파일에 정의되어 있다.

❑ Spring XML Configuration 설정(2/2)

```
<!-- DataSource 설정이다. 시스템에 맞게 재작성 해야 한다. 아래는 HSQL Sample이다. -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="net.sf.log4jdbc.DriverSpy"/>
    <property name="url" value="jdbc:log4jdbc:hsqldb:hsql://localhost/test"/>
    <property name="username" value="sa"/>
    <property name="password" value=""/>
    <property name="defaultAutoCommit" value="false"/>
    <property name="poolPreparedStatements" value="true"/>
</bean>
```

❑ 개요

– Client 모듈을 설정하기 위해서는 다음 과정이 필요하다.

- Metadata WEB_SERVICE_CLIENT에 설정 추가
- (Optional) Metadata WEB_SERVICE_MAPPING에 설정 추가

❑ Metadata WEB_SERVICE_CLIENT에 설정 추가

– Client 모듈을 설정하기 위해서는 Metadata의 WEB_SERVICE_CLIENT Table에 설정을 추가해야 한다. 다음과 같이 Integration 서비스의 Metadata인 INTEGRATION Table에 연계등록정보가 설정되어 있다고 가정한다. (기관, 시스템, 서비스, 메시지타입 등의 정보는 설정되어 있으며, 개발하는 시스템은 'SYSTEM_CONSUMER'라고 가정함)

INTEGRATION						
ID	PROVIDER_SERVICE_KEY	CONSUMER_SYSTEM_KEY	DEFAULT_TIMEOUT	USING_YN	VALIDATE_FROM	VALIDATE_TO
'INT_VERIFY_NAME'	'SERVICE_VERIFY_NAME'	'SYSTEM_CONSUMER'	5000	'Y'	NULL	NULL

– Web Service 'SERVICE_VERIFY_NAME'를 호출하기 위해서 WEB_SERVICE_CLIENT에 'SERVICE_VERIFY_NAME'을 SERVICE_KEY로 갖는 설정을 추가해야 한다.

WEB_SERVICE_CLIENT					
SERVICE_KEY	WSDL_ADDRESS	NAMESPACE	SERVICE_NAME	PORT_NAME	OPERATION_NAME
'SERVICE_VERIFY_NAME'	'http://localhost:8080/Sample/services/VerifyName?wsdl'	'http://itl/sample/'	'VerifyNameService'	'VerifyNamePort'	'service'

❑ (Optional) Metadata WEB_SERVICE_MAPPING에 설정 추가

- 만약 호출하는 Web Service가 전자정부 Integration 서비스 표준에 따라 개발된 서비스가 아닌 경우, 메시지 헤더부가 다를 수 있어 별도의 Mapping 정보가 필요하다.
- 전자정부 Integration 서비스 표준은 Web Service Header부에 들어갈 Attribute들이 EgovIntegrationMessageHeader에 정의되어 있고, 바디부는 EgovIntegrationMessage의 body에 정의되어 있으므로 별도의 mapping 정보 없이 header와 body 부의 구분이 가능하지만, 표준을 따르지 않은 Web Service의 경우 EgovIntegrationMessage의 body부에 정의되어 있는 일부 값들을 헤더에 포함시켜야 한다.

4. WebService – Server 모듈 개발(1/2)

❑ 개요

– Web Service Server 모듈을 개발하는 과정은 다음과 같다.

- “web.xml” 파일에 EgovWebServiceServlet 추가
- Metadata WEB_SERVICE_SERVER 설정 추가

❑ “web.xml” 파일에 EgovWebServiceServlet 추가

– web.xml에 EgovWebServiceServlet 설정을 추가한다.

```
<servlet>
    <description></description>
    <display-name>EgovWebServiceServlet</display-name>
    <servlet-name>EgovWebServiceServlet</servlet-name>
    <servlet-class>egovframework.rte.itl.webservice.EgovWebServiceServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>EgovWebServiceServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

개발한 Server 모듈을 공개할 URL path를 설정한다. WEB_SERVICE_SERVER의 ADDRESS는 이 값에 대한 상대적인 위치를 나타낸다.

❑ Metadata WEB_SERVICE_SERVER 설정 추가

– 다음과 같이 Integration 서비스의 Metadata인 INTEGRATION Table에 연계등록정보가 설정되어 있다고 가정한다. (기관, 시스템, 서비스, 메시지타입 등의 정보는 설정되어 있으며, 공개할 서비스는 'SERVICE_VERIFY_NAME'이라고 가정함)

INTEGRATION						
ID	PROVIDER_SERVICE_KEY	CONSUMER_SYSTEM_KEY	DEFAULT_TIMEOUT	USING_YN	VALIDATE_FROM	VALIDATE_TO
'INT_VERIFY_NAME'	'SERVICE_VERIFY_NAME'	'SYSTEM_CONSUMER'	5000	'Y'	NULL	NULL

– Web Service 'SERVICE_VERIFY_NAME'를 공개하기 위해서 WEB_SERVICE_SERVER에 'SERVICE_VERIFY_NAME'을 SERVICE_KEY로 갖는 설정을 추가해야 한다.

WEB_SERVICE_SERVER					
SERVICE_KEY	ADDRESS	NAMESPACE	SERVICE_NAME	PORT_NAME	OPERATION_NAME
'SERVICE_VERIFY_NAME'	'/VerifyName'	'http://itl/sample/'	'VerifyNameService'	'VerifyNamePort'	'service'

- <servlet-mapping> tag의 <url-pattern> tag의 값은 서비스를 제공하기 위한 주소로, WEB_SERVICE_SERVER Table의 ADDRESS Column 값은 <url-pattern> tag값에 대한 상대 위치를 나타낸다.
  예를 들어, Web Application의 IP가 192.168.0.1, Port가 8080, Context Root가 "Sample", url-patterns이 "/services/*"인 경우, 위 'SERVICE_VERIFY_NAME'의 WSDL Address는 http://192.168.0.1:8080/Sample/services/VerifyName?wsdl이다.