

Tasarım Örüntüleri ile Spring Eğitimi

9

Spring ve SOAP/WSDL Web Servisleri

Spring WS Nedir?

SOAP/WSDL tabanlı WS geliştirmeyi sağlayan **contract first yaklaşımını** benimsemiş **Spring üzerine kurulu** bir framework'tür

Contract First ve Contract Last Yaklaşımları

- **Contract first**

- Önce WS kontratının yani WSDL'in oluşturulup, ardından bu kontrata uygun servisin implement edilmesini savunur
- WSDL » Java code

- **Contract last**

- Önce servisin implement edilip, ardından bu servis implemantasyonundan kontratın yani WSDL'in üretilmesini savunur
- Java code » WSDL

Contract Last Yaklaşımın Problemleri

- Java kodu değiştiği anda WSDL'de değişebilir
- Ancak WSDL iki farklı sistem arasındaki **kontrattır**, kodun değişmesi WSDL'in değişmesine **neden olmamalıdır**
- Java kodu üzerinden akacak **verinin boyutunun ve kapsamının** kontrol edilmesi zor olabilir
- Sunucuda çalışan kodun içerisindeki **tiplerin bire bir karşılıkları** istemci tarafında olmayabilir

Spring WS ve Contract First Yaklaşım

- Spring WS **contract-first** yaklaşımı benimser
- En temel özelliği, **XML doküman/mesaj bakış açısı** ile düşünmeyi ön plana çıkarmasıdır
- Spring WS'e **mesaj odaklı** veya **doküman odaklı** bir WS framework de denilebilir
- **Gelen-giden veri** en önemli şeydir
- **Java kodu** daha geri plandadır ve bir **implementasyon detayıdır**

Spring WS ve Contract First Yaklaşım

- Spring WS'in “contract first” yaklaşımı kendi içinde **iki temel kısımdan** oluşur
 - Data contract (XSD)
 - Service contract (WSDL)
- XSD ve WSDL oluşturmak Spring WS ile çalışmanın **başlangıç noktasıdır**

Data Contract

- **Data Contract** ile gelen-giden **XML mesajların yapısı** belirlenir
- Çalışma zamanında data contract tarafından tanımlanmış bu **XML mesajları** Spring WS tarafından alınır veya dönülür
- Bu mesajlara “**request**” ve “**response**” adı verilir
- Bu XML mesajlarının yapısı **XSD** kullanılarak tanımlanır

Service Contract

- Service contract ile de WS **operasyonları** tanımlanır
- Bu tanımlama **WSDL** ile gerçekleştirilir
- Ancak Spring WS ile çalışırken sıfırdan **WSDL yazmaya gerek yoktur**
- “XSD + bazı convention”lar ile WSDL **otomatik olarak üretilebilir**
- Bu işleme **otomatik wsdl publish** adı verilir

Extensible Markup Language (XML)

- XML, **verinin anlamının/yapısının** verinin kendisi ile birlikte taşınmasını sağlar
- Veri **transferi ve saklama** da kullanılır
- XML sadece **veri hakkında bir bilgiyi** ifade eder
- Bunun dışında o veri ile ilgili **başka herhangi bir şey** yapmaz
- Veriye özel XML **elemanları** ve **attribute**'lar kullanılarak XML mesajları oluşturulur

XML vs HTML

- Verinin oluşturulması, sistemler arası transferi veya veritabanına kaydedilmesi, verinin gösterimi **farklı farklı sistemler** tarafından ele alınan işlemlerdir
- Verinin XML formatında olması bunlarla ilgili **herhangi bir kabiliyetin** olması/sergilenmesi anlamına gelmez
- XML sadece **verinin ne olduğuna** odaklanır
- **HTML** ise **verinin gösterimine** odaklanır

XML Örneği

```
<vets>
```

```
  <vet id="101">  
    <firstName>Ali</firstName>  
    <lastName>Zor</lastName>  
    <graduationYear>2011</graduationYear>  
  </vet>
```

```
  <vet id="102">  
    <firstName>Veli</firstName>  
    <lastName>Uysal</lastName>  
    <graduationYear>2005</graduationYear>  
  </vet>
```

```
</vets>
```

XML Namespace Nedir?

```
<table>
  <tr>
    <td>Elma</td>
    <td>Armut</td>
  </tr>
</table>
```

HTML sayfa içerisindeki
table elemanıdır

```
<table>
  <name>Yemek Masasi</name>
  <width>80</width>
  <length>120</length>
</table>
```

Bir masa bilgisini ifade eden
table elemanıdır

```
<data>
  <table>
    <tr>
      <td>Elma</td>
      <td>Armut</td>
    </tr>
  </table>
</data>
```

```
<data>
  <table>
    <name>Yemek Masasi</name>
    <width>80</width>
    <length>120</length>
  </table>
</data>
```

Problem:

İki farklı table yapısının
aynı mesajda ele alınması
karışıklıklara yol açacaktır
Hangisi HTML table,
hangisi masayı ifade eden
table net değildir

Çözüm:

XML namespace tanımlarını
kullanmaktır

XML Namespace Nedir?

- XML mesajındaki eleman ve attribute'ları **benzersiz biçimde ayrıştıran** bir URI'dır
- Genellikle kök elemanda tanımlanır

```
<data
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns:furniture="http://www.furnitureworld.com">
  <html:table>
    <html:tr>
      <html:td>Elma</html:td>
      <html:td>Armut</html:td>
    </html:tr>
  </html:table>
  <furniture:table>
    <furniture:name>Yemek Masasi</furniture:name>
    <furniture:width>80</furniture:width>
    <furniture:length>120</furniture:length>
  </furniture:table>
</data>
```

`xmlns:prefix="URI"`
ile tanımlanır. Prefix sayesinde
namespace'i uzun biçimde
elemanların başına yazmaya
gerek kalmaz

Default XML Namespace

- XML içindeki namespace'lerden bir tanesi **default** olarak tanımlanabilir

```
<data
xmlns="http://www.w3.org/1999/xhtml"
xmlns:furniture="http://www.furnitureworld.com">
  <table>
    <tr>
      <td>Elma</td>
      <td>Armut</td>
    </tr>
  </table>
  <furniture:table>
    <furniture:name>Yemek Masasi</furniture:name>
    <furniture:width>80</furniture:width>
    <furniture:length>120</furniture:length>
  </furniture:table>
</data>
```

Default bir namespace tanımı yapılabilir
default namespace'in elemanları
prefix olmadan da kullanılabilir

XML Şema (XSD) Nedir?

- XML dokümanının yapısını tanımlar
- Doküman içerisinde yer alacak **eleman** ve **attribute**'ların neler olabileceğini, bunlar arasındaki yapısal ilişkileri belirtir
- Eleman ve attribute'ların **veri tiplerini**, hangi sabit değerleri alabileceklerini veya default değerlerin neler olabileceğini belirtir
- Bir elemanın **çocuk elemanlarının** neler olabileceğini, sayısını ve sırasını belirtir
- Bir elemanın **içeriğinin olup olamayacağını**, text değer içerip içeremeyeceğini belirtir

XML Dokümanlarının Yapısı

- Bir XML dokümanını oluşturan **temel yapı taşları**:
 - Element
 - Attribute
 - PCDATA: parsed text data
 - CDATA: non-parsed text data
 - Entity: special chars (<,>,&,",')

XML Şema Örneği

```
<schema ...>

  <complexType name="vet">
    <sequence>
      <element name="firstName" minOccurs="1" maxOccurs="1"
type="string"/>
      <element name="lastName" minOccurs="1" maxOccurs="1"
type="string"/>
      <element name="graduationYear" minOccurs="1" maxOccurs="1"
type="integer"/>
      <element name="specialty" minOccurs="0"
maxOccurs="unbounded" type="ws:specialty"/>
    </sequence>
  </complexType>

  <complexType name="specialty">
    <sequence>
      <element name="name" minOccurs="1" maxOccurs="1"
type="string"/>
    </sequence>
  </complexType>

</schema>
```

XML Şema Tanımının Yapısı

Bir XSD tanımının kök elemanı her zaman için <schema>'dır

```
<?xml version="1.0" encoding="UTF-8"?>
<schema
```

Şema içerisinde kullanılacak eleman ve veri tiplerinin XMLSchema namespace ve ön eki tanımlanır.

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

```
targetNamespace="http://www.java-egitimleri.com/vets"
```

```
xmlns="http://www.java-egitimleri.com/vets"
```

```
elementFormDefault="qualified">
```

```
...
</schema>
```

Bu şema tarafından tanımlanacak XML elemanlarının ait olacakları namespace'i tanımlar

Target namespace ve ön eki tanımlanır

Burada tanımlanmış elemanların herhangi bir XML dokümanı içerisinde kullanılması durumunda namespace kalifikasyonuna sahip olmaları gerektiğini anlatır

Simple Element Örnekleri

```
<element name="firstName" type="string"/>
```

```
<element name="lastName" type="string"/>
```

```
<element name="graduationYear" type="integer"/>
```

```
<element name="color" type="string" default="red"/>
```

```
<element name="color" type="string" fixed="red"/>
```

Fixed veya
default değerler
de tanımlanabilir

www.java-egitimleri.com

Complex Element Örneği

Complex element tanımlarken diğer bir yöntemde type tanımını elemanın Dışında bir yerde yapmaktır. Buna global type adı verilir.

```
<element name="vet" type="tns:vetType"/>

<complexType name="vetType">
  <sequence>
    <element name="firstName" type="string" />
    <element name="lastName" type="string" />
    <element name="graduationYear" type="integer" />
    <element name="specialty" type="tns:specialtyType"
      minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
</complexType>

<complexType name="specialtyType">
  <simpleContent>
    <extension base="string"/>
  </simpleContent>
</complexType>
```

Elemanın kaç defa tekrarlayabileceğini belirtir

Complex element'in sadece text içerik barındırmasını sağlar

Attribute ve Mixed Content

```
<complexType name="vetType">
```

```
  <attribute name="id" type="string"/>
```

<attribute> elemanı ile attribute tanımlanır. İsmi, tipi, default değeri vs belirtilir

```
  <sequence>
```

```
    <element name="firstName" type="string" />
```

```
    <element name="lastName" type="string" />
```

```
    <element name="graduationYear" type="integer" />
```

```
  </sequence>
```

```
</complexType>
```

```
<xs:element name="letter">
```

```
  <xs:complexType mixed="true">
```

```
    <xs:sequence>
```

```
      <xs:element name="name" type="xs:string" />
```

```
      <xs:element name="orderid" type="xs:positiveInteger" />
```

```
      <xs:element name="shipdate" type="xs:date" />
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```

Element içerisinde hem çocuk elemanların Hem de text içeriğin birlikte olabileceğini belirtir

XML Dosyadan XSD'ye Referans

- XML içerisinde yapısını tanımlayan XSD'nin lokasyon bilgisi de belirtilebilir

`xmlns:xsi` genellikle XML ve XSD dosyalarının içerisinde kullanılan built-in attribute'lara ait namespace tanımıdır

```
<data
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:html="http://www.w3.org/1999/xhtml"
xmlns:furniture="http://www.furnitureworld.com"

xsi:schemaLocation="
http://www.w3.org/1999/xhtml http://www.w3.org/2002/08/xhtml/xhtml1-strict.xsd
http://www.furnitureworld.com furnitures.xsd">
```

...

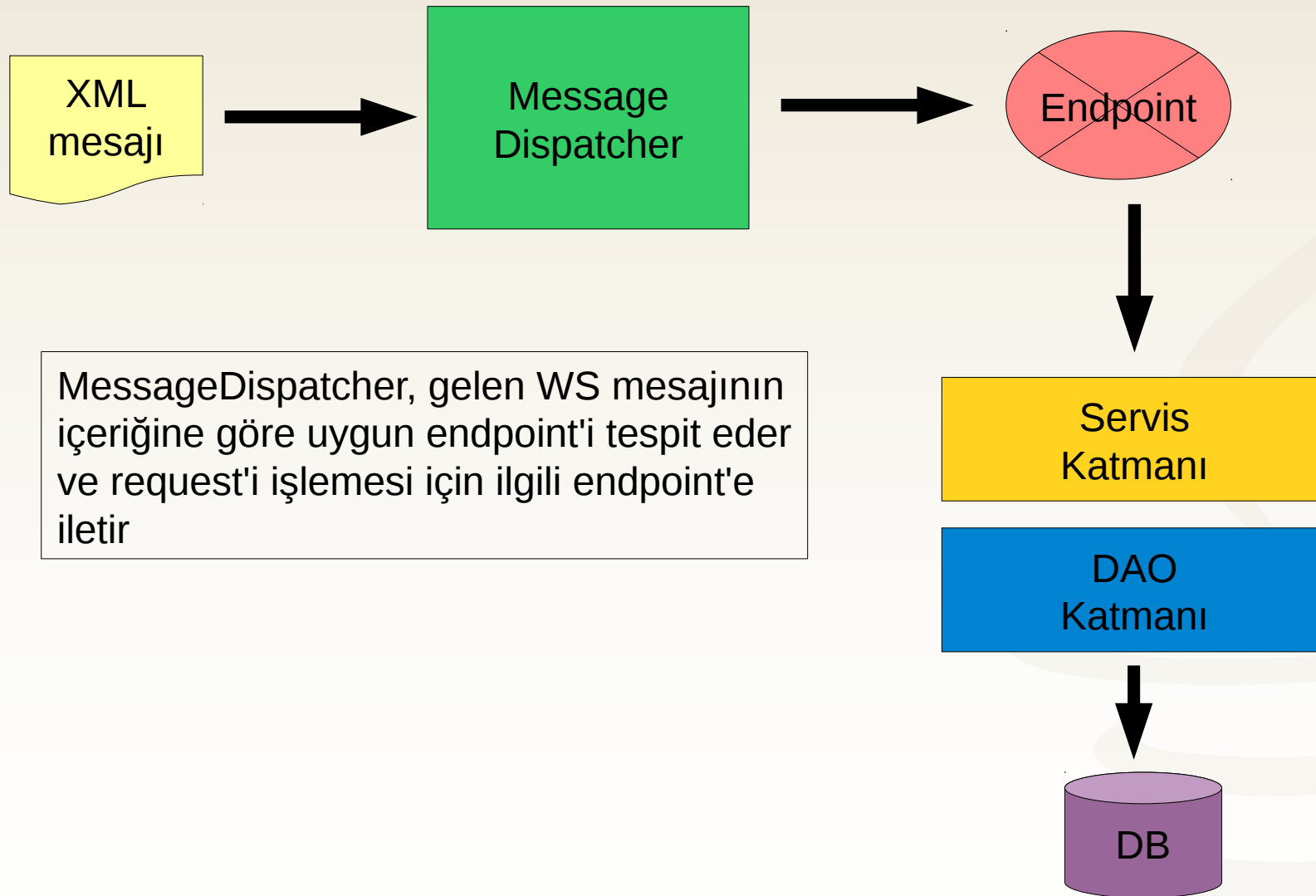
```
</data>
```

XML dokümanının yapısını tanımlayan şema dosyalarının lokasyonu da built-in **schemaLocation** ile belirtilir. Böylece XML **parser** veya **editor** XML dokümanını XSD'ye göre **validate** etme imkanına kavuşur

MessageDispatcherServlet

- **MessageDispatcherServlet** Spring WS'in çalışma zamanındaki **giriş noktası**dır
- Web servis çağrılarını **HTTP** üzerinden ele almayı sağlar
- **DispatcherServlet**'e benzer
- Asıl işi **MessageDispatcher**'a delege eder
- MessageDispatcher da XML mesajlarının **endpoint'lere dispatch** edilmesini sağlar

Spring WS MessageDispatcher Mimarisi



Spring WS Endpoint

- WS çağrısı ile gönderilen SOAP **mesajlarının işlendiği** yerdir
- SOAP mesajı işlenir ve servis katmanındaki **iş mantığı** çalıştırılır
- Servis katmanından **dönen sonuç** da SOAP cevabına dönüştürülerek istemci tarafına iletilir
- Metot parametreleri ve return değerinin dönüşüm işleminde genellikle **XML – nesne transformasyonu** söz konusudur

MessageDispatcherServlet Konfigürasyonu

```
<web-app>
  <servlet>
    <servlet-name>spring-ws</servlet-name>
    <servlet-class>
org.springframework.ws.transport.http.MessageDispatcherServlet
    </servlet-class>

    <init-param>
      <param-name>transformWsdLocations</param-name>
      <param-value>true</param-value>
    </init-param>

    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>spring-ws</servlet-name>
    <url-pattern>/ws/*</url-pattern>
  </servlet-mapping>
</web-app>
```

MessageDispatcherServlet Konfigürasyonu

- Ayrıca **WEB-INF/spring-ws-servlet.xml** isimli Spring ApplicationContext dosyası oluşturulmalıdır

```
<?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:web-services="http://www.springframework.org/schema/web-services"
xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.0.xsd
http://www.springframework.org/schema/web-services
http://www.springframework.org/schema/web-services/web-services.xsd">

<context:component-scan base-package="com.javaegitimleri.petclinic.ws" />

<web-services:annotation-driven />

</beans>
```

Spring WS namespace'inin
<web-services:annotation-driven/>
elemanı ile anotasyon tabanlı Spring
WS konfigürasyonu devreye alınmış olur

Otomatik WSDL Publish

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/greeting"
xmlns:tns="http://www.example.org/greeting" elementFormDefault="qualified">
```

```
<element name="helloWorldRequest">
```

Request soneki WS metot çağrısının input argümanlarını tanımlar

```
  <complexType>
```

```
    <sequence>
```

```
      <element name="name" minOccurs="1" maxOccurs="1" type="string" />
```

```
      <element name="age" minOccurs="1" maxOccurs="1" type="int" />
```

```
    </sequence>
```

```
  </complexType>
```

```
</element>
```

Response soneki ise WS metot çağrısının return tipini tanımlar

```
<element name="helloWorldResponse">
```

```
  <complexType>
```

```
    <sequence>
```

```
      <element name="greeting" minOccurs="1" maxOccurs="1" type="string"/>
```

```
    </sequence>
```

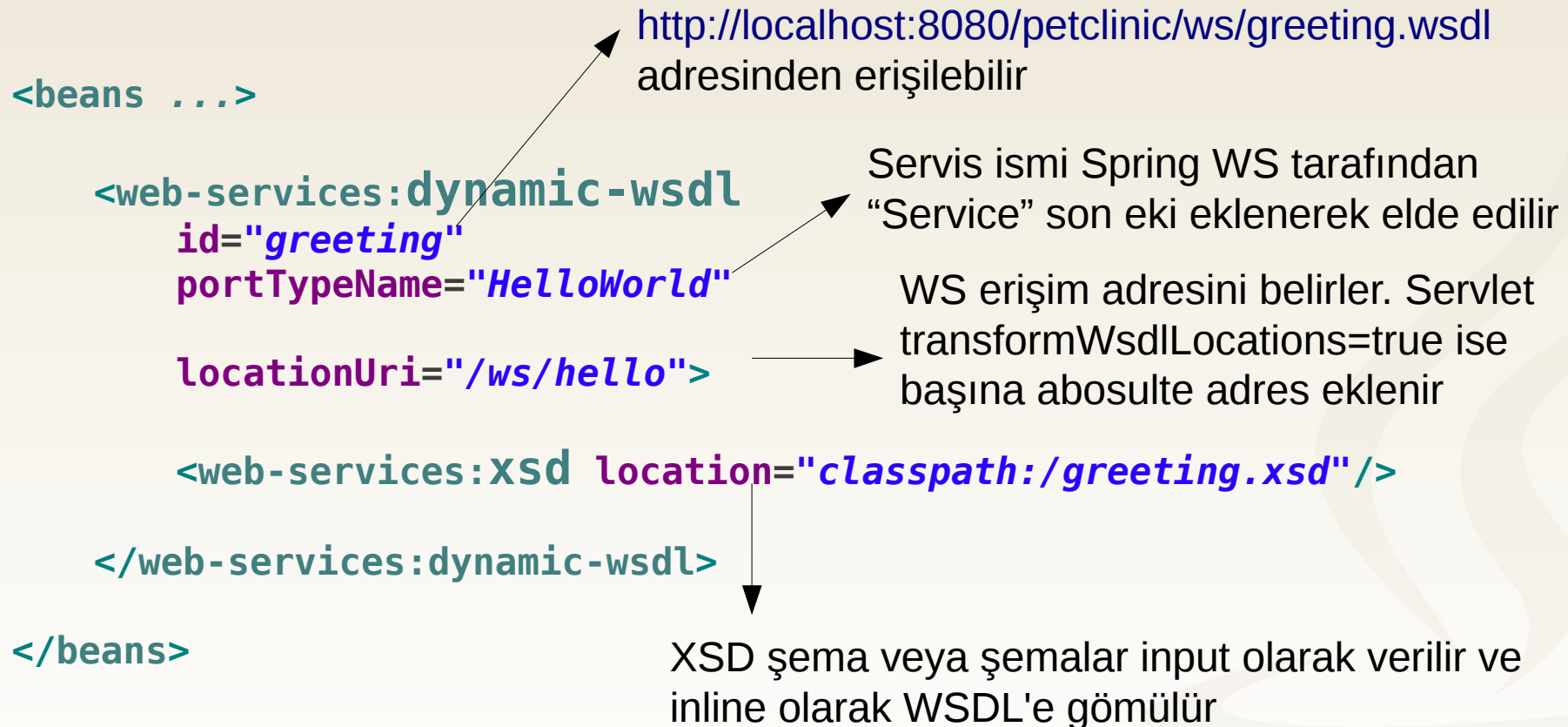
```
  </complexType>
```

```
</element>
```

```
</schema>
```

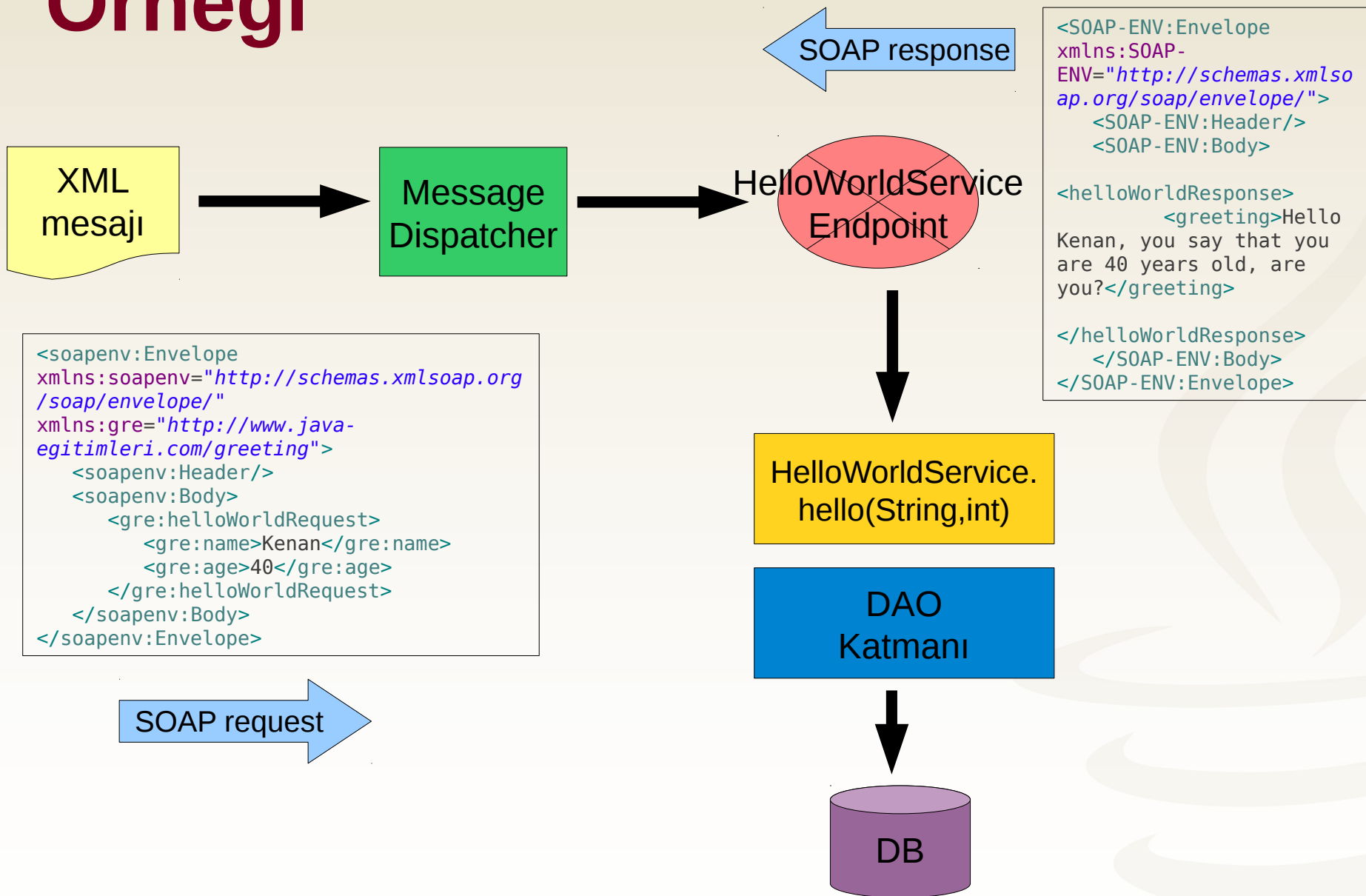
Request ve Response son eklerinden önceki kısım WS metot ismini oluşturur

Otomatik WSDL Publish



`<web-services:dynamic-wsdl>` elemanının `requestSuffix`, `responseSuffix`, `serviceSuffix` gibi attribute'ları ile wsdl auto publish davranışını özelleştirmek mümkündür

Spring WS Endpoint Örneği



Spring WS Endpoint Örneği

@Endpoint

@Component anotasyonundan türer, bean'in bir web servis endpoint olduğunu belirtir

```
public class HelloWorldServiceEndPoint {  
    @Autowired  
    private HelloWorldService helloWorldService;
```

Metodun bir web servis request'inin handler'ı olduğunu belirtir. Bir endpoint'de birden fazla handler olabilir

```
@PayloadRoot(  
    localPart = "helloWorldRequest",  
    namespace = "http://www.java-egitimleri.com/greeting")
```

javax.xml.transform.Source sınıfıdır. SOAP mesaj response'unu oluşturmakta kullanılır

```
public @ResponsePayload Source
```

```
    hello(@RequestPayload Element request) {
```

org.w3c.dom.Element sınıfıdır. XML mesajına Erişim sağlar

```
        String name = request.getElementsByTagNameNS(  
            "http://www.java-egitimleri.com/greeting", "name")  
            .item(0).getTextContent();  
        String age = request.getElementsByTagNameNS(  
            "http://www.java-egitimleri.com/greeting", "age")  
            .item(0).getTextContent();
```

```
        return new StringSource("<helloWorldResponse><greeting>"  
            + helloWorldService.hello(name, Integer.parseInt(age))  
            + "</greeting></helloWorldResponse>");
```

```
    }
```

```
}
```

SOAP mesaj içeriğinin metod input arg veya return değeri olmasını sağlar

Spring WS Endpoint Örneği

@Endpoint

```
public class HelloWorldServiceEndPoint {
```

```
@Autowired
```

```
private HelloWorldService helloWorldService;
```

```
@PayloadRoot(
```

```
    localPart = "helloWorldRequest",
```

```
    namespace = "http://www.java-egitimleri.com/greeting")
```

```
@Namespace(prefix = "ns", uri = "http://www.java-egitimleri.com/greeting")
```

```
public @ResponsePayload Source
```

```
    hello(@XPathParam("//ns:name") String name,
```

```
          @XPathParam("//ns:age") double age) {
```

```
    return new StringSource("<helloWorldResponse><greeting>"
```

```
        + helloWorldService.hello(name, (int)age)
```

```
        + "</greeting></helloWorldResponse>");
```

```
}
```

```
}
```

Namespace tanımlı yapmayı sağlar. Metot, sınıf
veya paket düzeyinde tanımlanabilir

SOAP mesaj payload içerisinde
XML eleman veya attribute
değerlerini doğrudan extract etmeyi
ve metoda parametre geçmeyi sağlar

Metot input argümanlarında sadece Xpath tarafından desteklenen veri tipleri kullanılabilir:
Double/double, Boolean/boolean, String, org.w3c.dom.Node ve org.w3c.dom.NodeList

Spring WS Endpoint Örneği

```
@Endpoint
public class HelloWorldServiceEndPoint {

    @Autowired
    private HelloWorldService helloWorldService;

    @PayloadRoot(
        localPart = "helloWorldRequest",
        namespace = "http://www.java-egitimleri.com/greeting")
    public @ResponsePayload HelloWorldResponse
        hello(@RequestPayload HelloWorldRequest request) {
        String message =
            helloWorldService.hello(request.getName(), request.getAge());
        HelloWorldResponse response = new HelloWorldResponse();
        response.setGreeting(message);
        return response;
    }
}
```



XML şema kullanılarak JAXB transformer ile üretilen Java sınıflarıdır. Bu sınıflarda JAXB anotasyonları mevcuttur. HelloWorldRequest ve HelloWorldResponse sınıfları XmlRootElement anotasyonu ile işaretlenmiştir. JAXB dönüşümü IDE içerisinde veya build aracı ile tetiklenebilir.

İstemci Tarafında Spring WS Kullanımı

- Spring WS kullanarak istemci tarafında web servis çağrıları yapmak için **WebServiceTemplate** ana sınıfı mevcuttur
- WebServiceTemplate **Source** nesnelerini gönderen ve response mesajlarını **Source** veya **Result** olarak işleyen metotlara sahiptir
- Ayrıca XML – nesne **dönüşümü** de yapabilir

WebServiceTemplate Konfigürasyonu

WebServiceTemplate çalışabilmesi için bir MessageFactory nesnesine ihtiyaç duyar. SaajSoapMessageFactory veya AxiomSoapMessageFactory implemantasyonlarından birisi kullanılabilir.

```
<bean id="messageFactory"  
      class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory"/>  
  
<bean id="webServiceTemplate"  
      class="org.springframework.ws.client.core.WebServiceTemplate">  
    <constructor-arg ref="messageFactory"/>  
    <property name="defaultUri"  
      value="http://localhost:8080/petclinic/ws/hello"/>  
</bean>
```

Default bir URI belirtilebilir. Ayrıca web service isteği gönderilirken de URI parametre olarak verilebilir


MessageSender ile Konfigürasyon

```
<bean id="webServiceTemplate"
class="org.springframework.ws.client.core.WebServiceTemplate">

    <constructor-arg ref="messageFactory"/>

    <property name="defaultUri" value="http://localhost:8080/petclinic/ws"/>

    <property name="messageSender">
        <bean
class="org.springframework.ws.transport.http.HttpComponentsMessageSender">
            <property name="credentials">
                <bean class="org.apache.http
.auth.UsernamePasswordCredentials">
                    <constructor-arg value="user1:secret"/>
                </bean>
            </property>
        </bean>
    </property>
</bean>
```



Custom bir MessageSender implemantasyonu property olarak enjekte edilebilir. Örneğin credentials bilgisinin HTTP request'inde gönderilebilmesi için HttpComponentsMessageSender kullanılabilir

WebServiceTemplate ve OXM Konfigürasyonu

```
<bean id="messageFactory"  
class="org.springframework.ws.soap.saaj.SaajSoapMessageFactory"/>
```

```
<bean id="WebServiceTemplate"  
class="org.springframework.ws.client.core.WebServiceTemplate">  
  <constructor-arg ref="messageFactory"/>  
  <property name="defaultUri"  
    value="http://localhost:8080/petclinic/ws/hello"/>
```

```
  <property name="marshaller" ref="jaxbMarshaller"/>  
  <property name="unmarshaller" ref="jaxbMarshaller"/>  
</bean>
```

```
<oxm:jaxb2-marshaller id="jaxbMarshaller"  
  context-path="com.javaegitimleri.petclinic.ws.model"/>
```

context-path attribute'una XML binding yapılmış Java sınıflarını içeren Paketlerin isimleri ":" ile ayrılarak verilir

WebServiceTemplate Kullanımı

```
String message =  
    "<helloWorldRequest xmlns=\"http://www.java-egitimleri.com/greeting\">" +  
    "  <name>Kenan</name>" +  
    "  <age>39</age>" +  
    "</helloWorldRequest>";
```

```
Source requestPayload = new StringSource(message);
```

```
Result responseResult = new StreamResult(System.out);
```

```
webServiceTemplate.sendSourceAndReceiveToResult(  
    requestPayload, responseResult);
```

veya

```
webServiceTemplate.sendSourceAndReceiveToResult(  
    "http://localhost:8080/petclinic/ws/hello",  
    requestPayload, responseResult);
```


WebServiceTemplate Kullanımı

```
HelloWorldRequest request = new HelloWorldRequest();  
request.setName("Kenan");  
request.setAge(39);  
  
HelloWorldResponse response = (HelloWorldResponse)  
    webServiceTemplate.marshallSendAndReceive(request);  
  
System.out.println(response.getGreeting());
```



XML – nesne dönüşümünün yapılabilmesi için WebServiceTemplate nesnesine uygun **Marshaller** ve **Unmarshaller** bean'lerinin tanıtılmış olması gerekir.

Web Servis Çağrılarının Güvenliği

- Web servis çağrılarının ve mesaj içeriğinin güvenliğinin sağlanmasında **üç temel nokta** vardır
 - **Kimliklendirme**
 - Mesajın **imzalanması**
 - Mesaj içeriğinin **şifrelenmesi**
- Spring WS bu üç alanda da çözümler sunmaktadır

EndpointInterceptor ile Güvenlik

- Spring WS'deki güvenlik mekanizması **EndpointInterceptor** üzerine kuruludur
- **İki farklı implemantasyonu vardır**
 - Wss4jSecurityInterceptor: Apache WSS4J tabanlı
 - XwsSecurityInterceptor: Sun XWSS tabanlı
- **Transport protokol bağımsız** çalışma imkanı sunarlar
- Ancak kullanımları **karmaşıktır**, EndpointInterceptor yerine **HTTP Basic Auth** tabanlı yöntem daha çok tercih edilmektedir

Wss4jSecurityInterceptor

- Apache **WSS4J**'i temel alır
- Harici **konfigürasyon dosyası** kullanmaz
- **Property**'ler ile **konfigüre** edilir
- İstemci tarafında **securementActions**, sunucu tarafında ise **validationActions** property'lerine **securement** ve **validation** action'ları tanımlanarak konfigüre edilir
- Bu action'lar hem istemci, hem de sunucu tarafında **aynı sıra ile tanımlanmalıdır**

İstemci Tarafı için Securement Action'ları

- **NoSecurity:** herhangi bir securement action uygulanmaz
- **UsernameToken:** username token ekler
- **UsernameTokenSignature:** username token ve bunun secret key'ini ekler
- **Timestamp:** timestamp ekler
- **Encrypt:** mesajı kriptolar
- **Signature:** mesajı imzalar

Sunucu Tarafı için Validation Action'ları

- **NoSecurity:** herhangi bir işlem yapmaz
- **UsernameToken:** username token'ı validate eder
- **Timestamp:** timestamp'i validate eder
- **Decrypt:** kriptolu mesajı çözümler
- **Signature:** imzayı validate eder

Wss4JSecurityInterceptor Konfigürasyonu - İstemci

```
<bean id="securityInterceptor"  
class="org.springframework.ws.soap.security.wss4j.Wss4JSecurityInterceptor">  
    <property name="securementActions" value="UsernameToken"/>  
    <property name="securementUsername" value="user1"/>  
    <property name="securementPassword" value="secret"/>  
    <property name="securementPasswordType"  
value="PasswordText"/>  
    <property name="securementUsernameTokenElements"  
value="Nonce Created"/>  
</bean>
```



Alabileceği değerler :PasswordText, PasswordDigest
(default)

Wss4JSecurityInterceptor Konfigürasyonu - İstemci

```
<bean id="webServiceTemplate"  
class="org.springframework.ws.client.core.WebServiceTemplate">  
...  
  <property name="interceptors">  
    <array>  
      <ref bean="securityInterceptor"/>  
    </array>  
  </property>  
</bean>
```



Bir önceki adımda tanımlanan securityInterceptor bean'i
WebServiceTemplate'ın **ClientInterceptor**'leri arasına
eklenmelidir

Wss4JSecurityInterceptor Konfigürasyonu - Sunucu

```
<bean id="securityInterceptor" class="org.springframework.ws.soap
    .security.wss4j.Wss4jSecurityInterceptor">
    <property name="validationActions" value="UsernameToken"/>
    <property name="validationCallbackHandler"
        ref="validationCallbackHandler"/>
</bean>
```

Sunucu tarafındaki konfigürasyon
ValidationCallbackHandler'a ihtiyaç duyar. Bu callback handler'ın
farklı implemantasyonları mevcuttur

```
<bean id="validationCallbackHandler"
class="org.springframework.ws.soap.security.wss4j.callback.SimplePasswordValid
ationCallbackHandler">
    <property name="users">
        <props>
            <prop key="user1">secret</prop>
            <prop key="user2">123456</prop>
        </props>
    </property>
</bean>
```

Wss4JSecurityInterceptor Konfigürasyonu - Sunucu

```
<web-services:interceptors>
```

```
  <ref bean="securityInterceptor"/>
```

```
</web-services:interceptors>
```



Bir önceki adımda tanımlanan securityInterceptor bean'i sunucu tarafında Spring WS'e endpoint interceptor olarak eklenmelidir

Spring WS – Spring Security Entegrasyonu

- Bu yaklaşımda **kimliklendirme** Spring Security tarafından gerçekleştirilir
- Spring Security'ye özel **ValidationCallbackHandler**'lardan uygun olan birisi kullanılmalıdır
- Kimliklendirme işlemi başarılı ise **AuthenticationToken SecurityContextHolder**'a set edilir
- Web servis request'inin sonunda da **SecurityContext** temizlenmektedir

Spring WS – Spring Security Entegrasyonu

```
<bean id="securityInterceptor" class="org.springframework.ws.soap.security
    .wss4j.Wss4jSecurityInterceptor">
    <property name="validationActions" value="UsernameToken" />
    <property name="validationCallbackHandler" ref="validationCallbackHandler" />
</bean>
```

```
<bean id="validationCallbackHandler" class="org.springframework.ws.soap.security
    .wss4j.callback.SpringSecurityPasswordValidationCallbackHandler">
    <property name="userDetailsService" ref="userDetailsService"/>
</bean>
```



Kullanılan ValidationCallbackHandler'a
göre **UserDetailsService** veya
AuthenticationManager enjekte edilebilir

```
<security:user-service id="userDetailsService">
    <security:user name="user1" password="secret" authorities="" />
    <security:user name="user2" password="123456" authorities="" />
</security:user-service>
```

Spring WS ve HTTP Basic Auth

- EndpointInterceptor yaklaşımına alternatif diğer yöntem **HTTP BASIC Auth** kullanılmasıdır
- Konfigürasyonu ve **kullanımı basittir**
- **HTTPS** ile birlikte kullanıldığı vakit pek çok kullanım durumundaki gereksinimi karşılar
- Spring Security ile birlikte **kolayca konfigüre edilebilir**
- Ancak **transport protokol bağımlıdır**
- **WS HTTP üzerinden** erişiliyor ise kullanılabilir

Spring WS ve HTTP Basic Auth

– Sunucu Tarafı

<security:http-basic/> elemanı basic authentication'ı aktive eder.
<security:intercept-url> tanımı ile de WS request'lerinin kimliklendirmeye tabi tutulması sağlanır.

```
<security:http auto-config="false">
  <security:csrf disabled="true"/>
  <security:http-basic/>
  <security:intercept-url pattern="/ws/**"
    access="isFullyAuthenticated()"/>
</security:http>

<security:authentication-manager>
  <security:authentication-provider
    user-service-ref="userDetailsService"/>
</security:authentication-manager>

<security:user-service id="userDetailsService">
  <security:user name="user1" password="secret" authorities=""/>
  <security:user name="user2" password="123456" authorities=""/>
</security:user-service>
```

Spring WS ve HTTP Basic Auth

– İstemci Tarafı

İstemci tarafında WebServiceTemplate kullanılıyor ise default MessageSender yerine Apache Commons HttpClient'i kullanan HttpComponentsMessageSender konfigüre edilmelidir

```
<bean id="webServiceTemplate"
class="org.springframework.ws.client.core.WebServiceTemplate">
...
<property name="messageSender">
    <bean class="org.springframework.ws.transport
        .http.HttpComponentsMessageSender">
        <property name="credentials">
            <bean class="org.apache.http
                .auth.UsernamePasswordCredentials">
                <constructor-arg value="user1:secret"/>
            </bean>
        </property>
    </bean>
</property>
</bean>
```

İletişim

- **Harezmi** Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- <http://www.java-egitimleri.com>
- info@java-egitimleri.com



harezmi
bilişim çözümleri

JAVA
Eğitimleri 