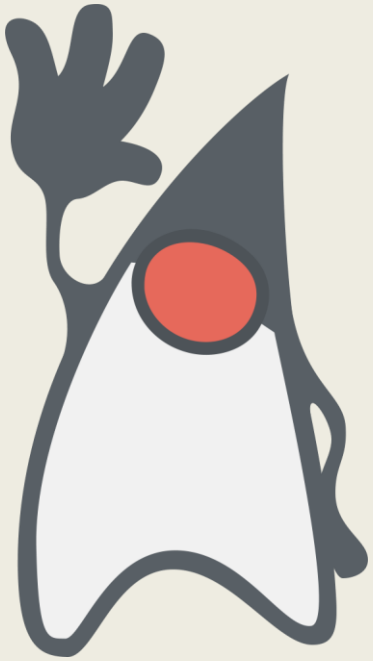
Senior School

Вебинар 09

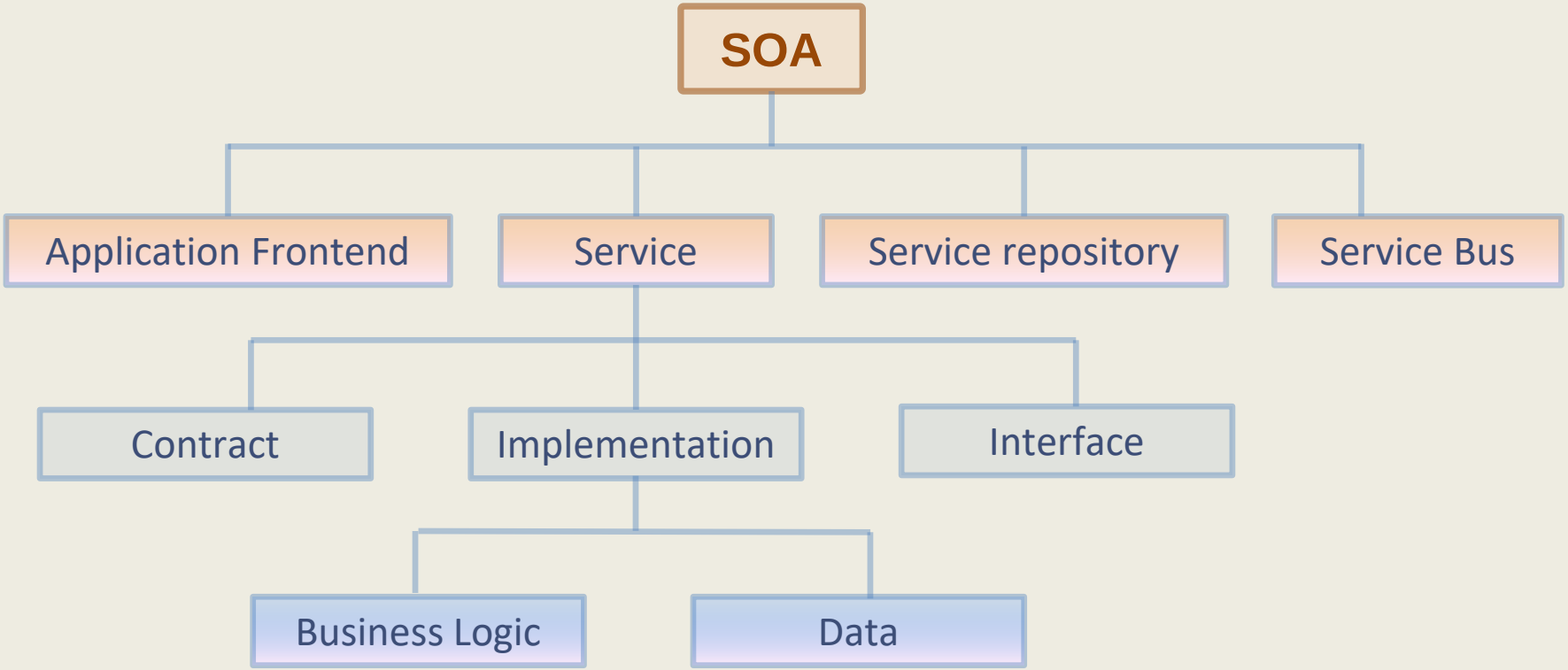


- **WEB-СЛУЖБЫ XML**
- **SOAP, WSDL, UDDI**
- **JAX-WS**
- **APACHE CXF**
- **ДОМАШНЕЕ ЗАДАНИЕ**

Web-служба

- Единица модульности в SOA
- Идентифицируется URL
- Имеет стандартизованный и слабо связанный интерфейс
  (потребитель, не должен знать детали реализации службы: язык, платформа, сигнатуры методов)
- Взаимодействие - протоколы SOAP, XML-RPC, REST и т. д.

Service-oriented architecture



Протоколы

SOAP

(Simple Object Access Protocol)

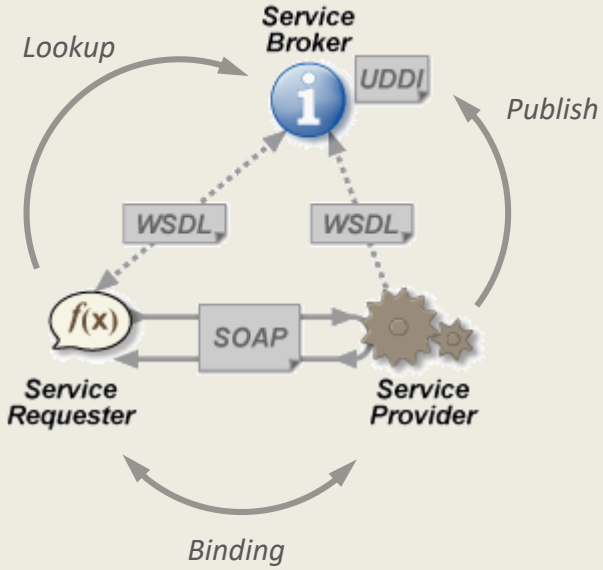
- Обмен сообщениями XML *(запросы, ответы, асинхронные вызовы, оповещения, …)*
- Развитие XML-RPC
- Описание интерфейса – WSDL *(Web Services Description Language)*
- Не зависит от транспортного протокола (HTTP, UDP, JMS, local)

REST

(*Representational State Transfer*)

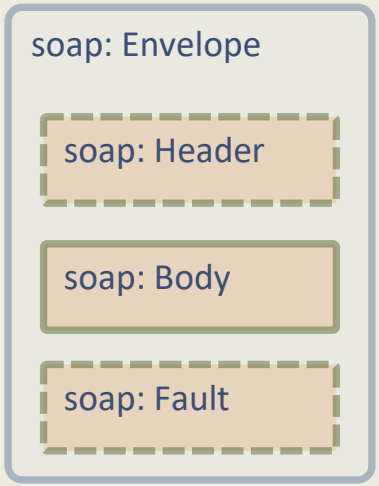
- Основан на HTTP
- Использует *все* возможности HTTP
- Вызовы процедур, Обращение к объектам → GET, POST, PUT, …
- Поставщик и потребитель взаимодействуют подобно серверу и клиенту HTTP
- Данные – XML, JSON

Взаимодействие SOAP



- WSDL
  - контракт веб-службы
  - взаимодействие потребителя и поставщика
- UDDI (Universal Description, Discovery and Integration)
  - реестр, который хранит информацию о службах: *имя, URL, WSDL, описание*
  - для потребителей – средства поиска служб

Сообщение SOAP



Элемент	Назначение
Envelope	Корневой элемент сообщения. Определяет пространство имен
Header	Необязательные атрибуты сообщения или передачи данных
Body	Собственно сообщение
Fault	Информация об ошибках, произошедших при обработке сообщения

- Расширенная спецификация SwA (SOAP with Attachments) – возможность включать бинарные данные (MIME)

Пример сообщений

- Запрос

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
               xmlns:ns1="http://w04.JSS.javajoy.net/">
  <soap:Header/>
  <soap:Body>
     <ns1:validate>
        <card number="12 34567 89011" expiry_date="10/12" control_number="544"/>
     </ns1:validate>
  </soap:Body>
</soap:Envelope>
```

- Ответ

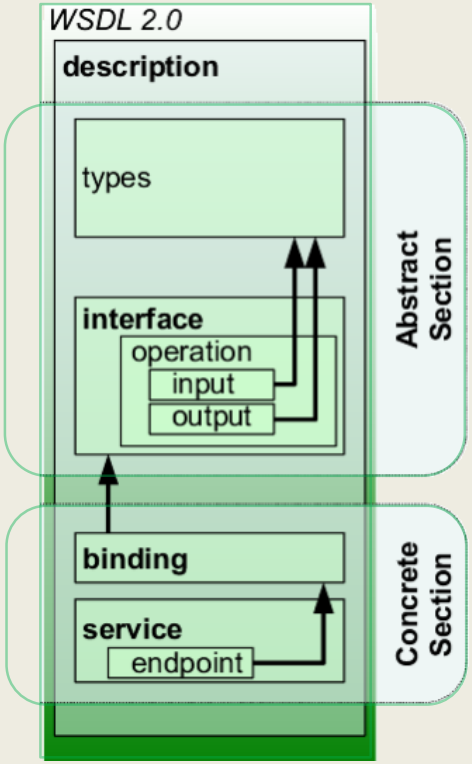
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/
               xmlns:ns1="http://w04.JSD.javajoy.net/">
  <soap:Header/>
  <soap:Body>
     <ns1:validateResponse>
        <return>true</return>
     </ns1:validateResponse>
  </soap:Body>
</soap:Envelope>
```

SOAP и HTTP

```
POST /OrderEntry HTTP/1.1
Host: www.example.com
Content-Type: application/soap;  charset="utf-8"
Content-Length: XXX
<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope" ...>
   ...
   Message information goes here
   ...
</soap:Envelope>
```

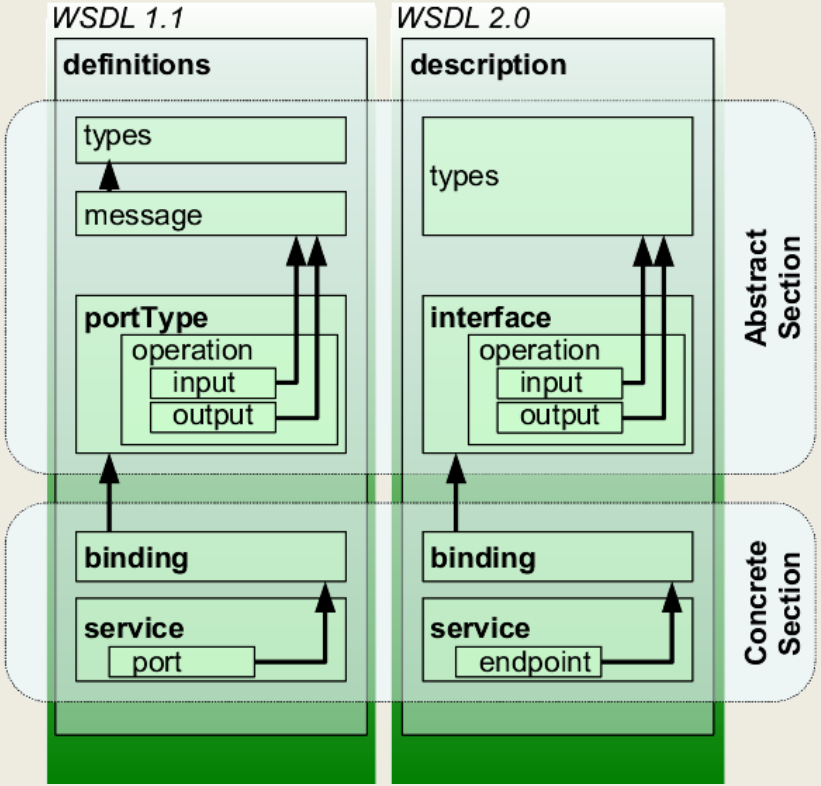
WSDL



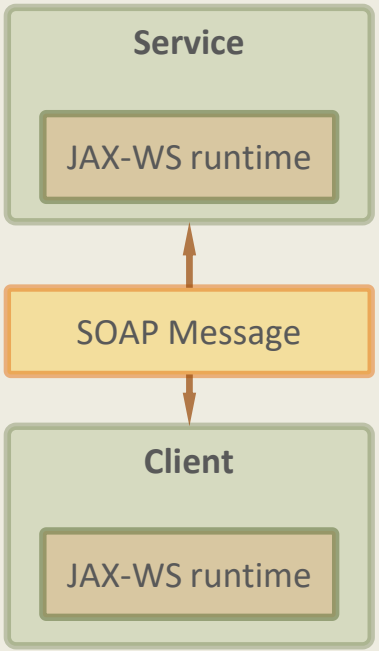
Описывает с помощью XML контракт службы

<description>	Корневой элемент. Содержит глобальные описания пространств имен
<types>	Определяет типы данных, которые будут использованы в сообщениях (XSD)
<interface>	Определяет операции (методы) службы. Каждая операция имеет входное и выходное сообщение
<binding>	Для конкретного интерфейса описывает протокол и форматы данных для операций и сообщений
<service>	Содержит коллекцию элементов <endpoint>
<endpoint>	Адрес (URL) для связывания, конечная точка

Версии WSDL


WSDL 1.1
definitions
types
message
portType
operation
input
output
binding
service
port
WSDL 2.0
description
types
interface
operation
input
output
binding
service
endpoint
Abstract Section
Concrete Section

JAX-WS



- Java API for XML Web Services
  - ***Спецификация*** API и аннотаций для создания XML-веб-служб
  - Часть Java EE
  - Развитие/замена технологии JAX-RPC
  - Включает JAXB (Java Architecture for XML Binding) и SAAJ (SOAP with Attachments API for Java)
- Преимущества
  - Документ-ориентированная модель сообщений
  - Скрывает низкоуровневые детали протокола *(автоматически генерируются дескрипторы, SOAP пакеты, WSDL)*
  - Внедрение ресурсов (Resource injection)
  - Возможность генерации классов WS на основе WSDL

JAX-WS Engines

- GlassFish Metro
  - эталонная реализация (JAX-WS Reference Implementation)
  - часть сервера приложений GlassFish
  - может использоваться отдельно в Tomcat или в Java SE

- Apache Axis2

- Apache CXF – входит в поставку TomEE

- JBossWS Native – входит в поставку WildFly

Установка и настройка

- Загружаем Apache CXF
- Задаем путь к домашней директории CXF – Settings | IDE Settings | Web Services
- Добавляем facet(ы) – ПКМ на модуле | Add Framework Support
  - web
  - Spring Web Services
  - web/WebServices

Создаем простой сервис

- Класс-реализация

```
@WebService(serviceName = "HelloService")
public class MyHelloService {
      public String sayHelloWorldFrom(String from) {
            return "Hello, world, from " + from;
      }
}
```

- Конфигурация CXF

```
<beans ... >
    <jaxws:endpoint
        id="HelloService"
        implementor="package.MyHelloService"
        address="/helloWorld">
    </jaxws:endpoint>
    ...
</beans>
```

- Проверка

web.xml

```
http://localhost:8080
      /context/services

http://localhost:8080
      /context/services
      /helloWorld?wsdl
```

POJO

- ***Plain Old Java Object*** — обычный Java-объект
- Не наследует (не реализует) никаких служебных классов (интерфейсов), кроме классов (интерфейсов) бизнес-модели
- В фреймворках (EJB, JAX-WS, Hibernate, Tapestry) описывают бизнес-логику
- Обязателен только конструктор без параметров
- Для использования в фреймворке – добавляются аннотации

Аннотации

WEB-INF/webservices.xml ← XML

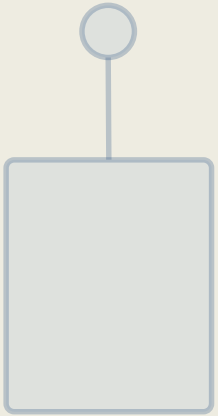
Аннтотация	Отображение	Пример
`@WebService`	Класс/интерфейс → WS	
`@WebMethod`	Метод → Операция WS	
`@WebParam`	Параметр операции → WSDL-part и XML элемент	
`@WebResult`	Возвр. знач. операции → WSDL-part и XML элемент	
`@Oneway`	Односторонняя операция (без выходных параметров)	
`@SOAPBinding`	WS → SOAP	

```
@WebService(serviceName = "My",
    portName = "MyPort",
    targetNamespace =
        "http://some.domain/jaxws/my")
public  class  MyService {
  @WebMethod(operatinName = "add")
  @WebResult(name="JXWsRes")
  String getItem(
      @WebParam(name = "item",
                mode=Mode.IN)
      String item,
      @WebParam(name = "count")
      int count) {
    // ...
  }
}
```

Требования к POJO службы

- Аннотация @javax.jws.WebService или XML-эквивалент в конфиг-файле;
- Public, не final, не abstract;
- public-конструктор по умолчанию;
- без метода finalize() ;
- не сохраняет специфическое для клиента состояние;
- открытые для потребителей методы – @javax.jws.WebMethod,
  а их параметры – должны быть совместимы с JAXB

Service Endpoint Interface - SEI



- Методы, которые может вызвать клиент WS
- Класс `@WebService`, не реализующий интерфейсов – неявный SEI
- Все public-методы отображаются на WSDL-операции,
  кроме аннотированных @WebMethod(exclude = true).
- Аннотация @WebMethod не обязательна и используется для настройки дополнительных параметров

JAXB

Java Architecture for XML Binding - аннотации для преобразования Java-объекта в XML

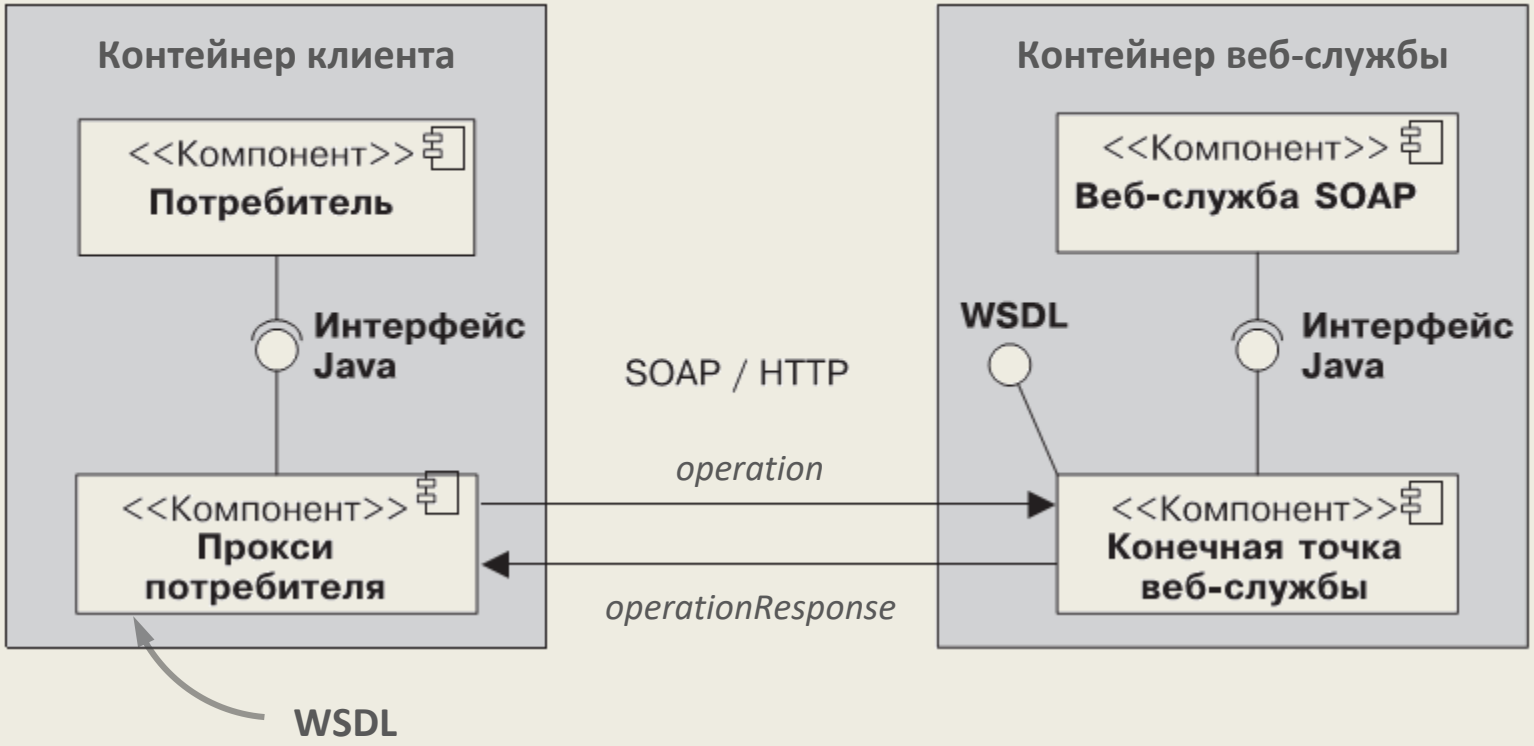
```
@XmlRootElement
public class CreditCard {
   private String number;
   private String expiryDate;
   // ...
   @XmlAttribute(required = true)
   public String getNumber() {  … }
   public void setNumber(String number) {  … }
   @XmlAttribute(name = "expiry_date", required = true)
   public String getExpiryDate();
   public void setExpiryDate(String date) {  … }
   // Конструкторы, другие методы
}
```

Упаковка

.war (или .jar для EJB-endpoint)

- реализация службы компонента и ее зависимые классы;
- endpoint-интерфейсы (не обязательно);
- файл WSDL (или ссылка на него)
  (если используются аннотации, генерируется средой выполнения JAX-WS);
- сгенерированные артефакты для запроса и ответа SOAP
  (не обязательно, генерируются средой выполнения JAX-WS);
- дескриптор развертывания web.xml (опционально)
- конфиг-файл CXF (Spring) - cxf.xml или cxf-servlet.xml.

Вызов веб-службы



Способы вызова

- Программный вызов из Java-кода
- Тестирование с помощью SoapUI
- Вызов сервиса из JavaScript – XMLHttpRequest
  - ограничения безопасности

Вызов службы в java-коде

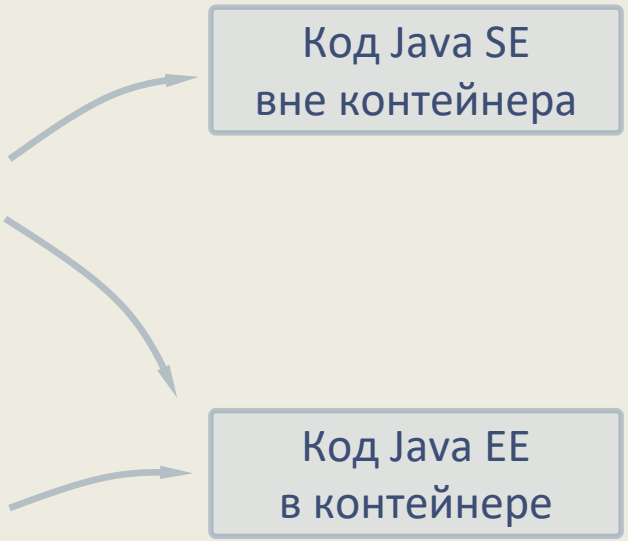
- Программный вызов

```
public static void main(String[] args) {
    CreditCard creditCard = new CreditCard();
    // ...
    CardValidator cardValidator =
       new CardValidatorService().getCardValidatorPort();
    cardValidator.validate(creditCard);
}
```

- Внедрение

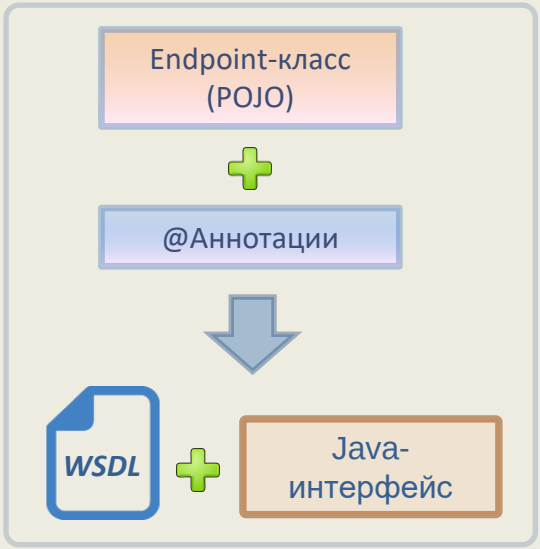
```
@WebServiceRef
private static CardValidatorService cardValidatorService;

public static void main(String[] args) {
   CreditCard creditCard = new CreditCard();
   // ...
   CardValidator cardValidator =
       cardValidatorService.getCardValidatorPort();
   cardValidator.validate(creditCard);
}
```

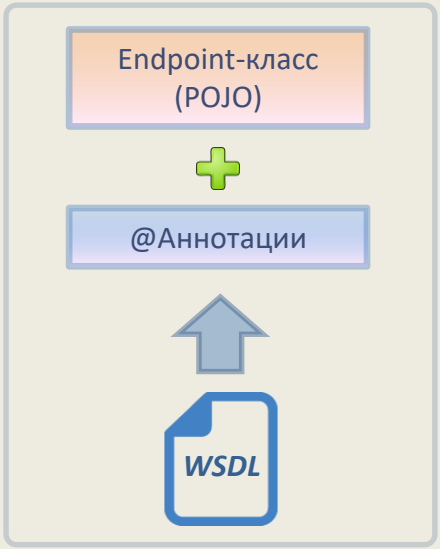


Подходы к созданию службы

Восходящий



Нисходящий



Требует понимания

- WSDL
- XSD (XML Schema Definition) для определения форматов сообщений

Преимущества и недостатки

- Независимость от платформы клиента и сервера.
- Открытые стандарты, простота разработки и отладки
- http-взаимодействие через межсетевой экран (в отличие от CORBA, DCOM, Java RMI)

- Производительность и сетевой трафик
- Безопасность. Необходимо использовать дополнительные технологии (HTTPS, XML Signature, XML Encryption, SAML)

Вопросы

1. В чем принципиальное отличие взаимодействия web-служб и компонентов EJB?

2. Из каких логических частей состоит сервис согласно SOA?

3. Объясните значение терминов WSDL и разницу между ними: interface, binding, service, endpoint?

4. Обязательно ли для службы иметь WSDL, чтобы к ней обращаться и почему?

5. Для чего предназначена аннотация @WebServiceRef?

6. Что такое прокси-объект (proxy)? Каким образом его можно создать и использовать?

7. Как, имея WSDL-документ, сгенерировать endpoint-класс, и наоборот?

Домашнее задание

1. Для созданного на предыдущих занятиях web-приложения реализуйте доступ к бизнес-логике (т.е. основным объектам БД и операциям с ними) в виде web-служб. Необходимо повторно использовать ту же реализацию бизнес логики, которая ранее была доступна через веб-интерфейс.

2. Протестируйте работу служб с помощью клиента SoapUI.

3. Создайте потребителей этих служб в виде объектов Java Bean и используйте их в JSP/JSTL-станицах своего приложения вместо объектов Java Bean, которые вы создавали ранее для доступа к данным.

Что дальше?

- **REST – СЛУЖБЫ**
- **JAX-RS**