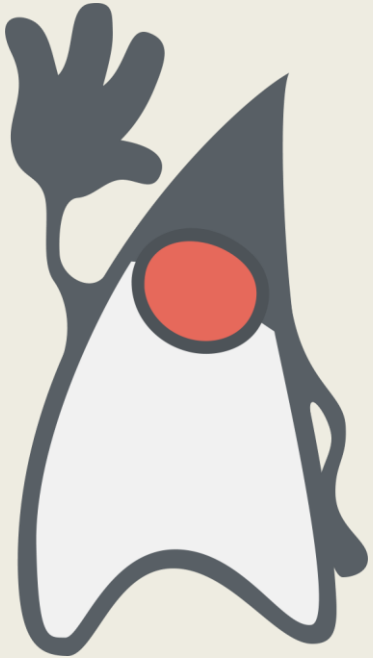
Senior School

Вебинар 10



- **REST**
- **JAX-RS**
- **APACHE CXF**
- **ДОМАШНЕЕ ЗАДАНИЕ**

Транспортные протоколы

SOAP

(Simple Object Access Protocol)

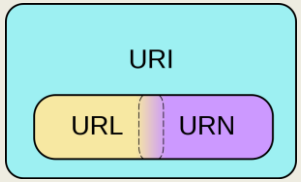
- Обмен сообщениями XML *(запросы, ответы, асинхронные вызовы, оповещения, …)*
- Развитие XML-RPC
- Описание интерфейса – WSDL *(Web Services Description Language)*
- Не зависит от транспортного протокола (HTTP, SMTP, FTP, JMS)

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

Принципы REST



- Каждая информационная единица – ресурс
- Адресуемость – URI = URL + URN + …

  `http://host:port/path/pathParams?queryParams#fragment`
- Единообразный интерфейс – набор http-запросов
- Один URI – несколько представлений
  + согласование содержимого (content negotiating)
- Связность (гипермедийность)
- Состояние приложения отделено от состояния ресурса
- Состояние не сохраняется

REST vs. SOAP

- Не привязан к XML
- Работает быстрее (за счет простоты протокола)
- Упрощение разработки
- Доступны все наработки http
  (кеширование на уровне сервера, масштабирование)

- Работает с ресурсами, а не с операциями

  не годится для
  - сложной бизнес-логики
  - реализации транзакций
  - требования к безопасности операций
- Меньшая стандартизация

Применение REST

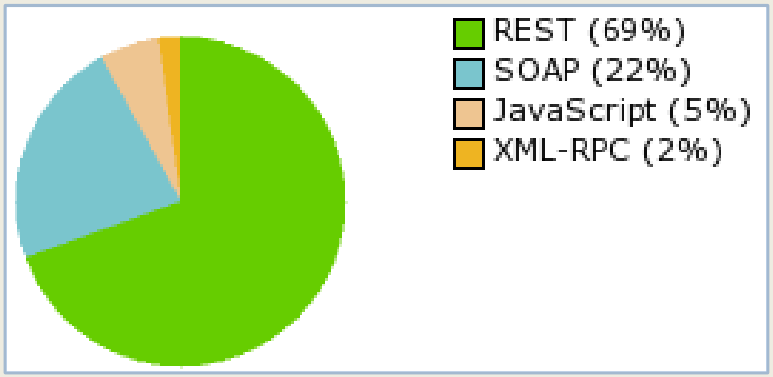
Сервис букмекерской конторы для работы с футбольной статистикой:

- для пользователей – получить список матчей, получить детали о матче;
- для редакторов – редактировать (Create, Edit, Delete) список матчей, редактировать детали матча.

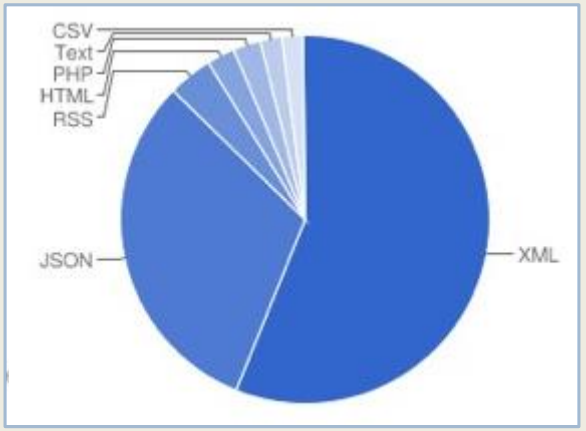
	Что происходит при обращении с использованием методов HTTP			
URL	**GET**	**POST**	**PUT**	**DELETE**
example.com/matches	Получаем список матчей	Создаем заново список матчей	Обновляем список матчей	Мстим директору букмекерской конторы
example.com/matches/28	Получаем детали матча с ID=28	Создаем информацию о матче	Обновляем детали матча	Удаляем матч

Использование в web API

Protocols



Data formats



programmableweb.com, 2013

Используемые запросы

Методы	Назначение
GET	- Получение представления ресурса - Безопасный, идемпотентный *Может вызываться повторно*
POST	- Создание нового ресурса и возвращение URI, … - Не безопасный, не идемпотентный - В случае успеха – код ответа 201
PUT	- Обновление состояния ресурса или создание по URI - Не безопасный, идемпотентный
DELETE	- Удаление ресурса. Идемпотентный
HEAD	- Аналогичен GET, но тело ответа не передается - Проверка правильности URI, размера
TRACE	Запрос отправляется назад без изменений
OPTIONS	Запрос вариантов использования/требований

Пример WADL

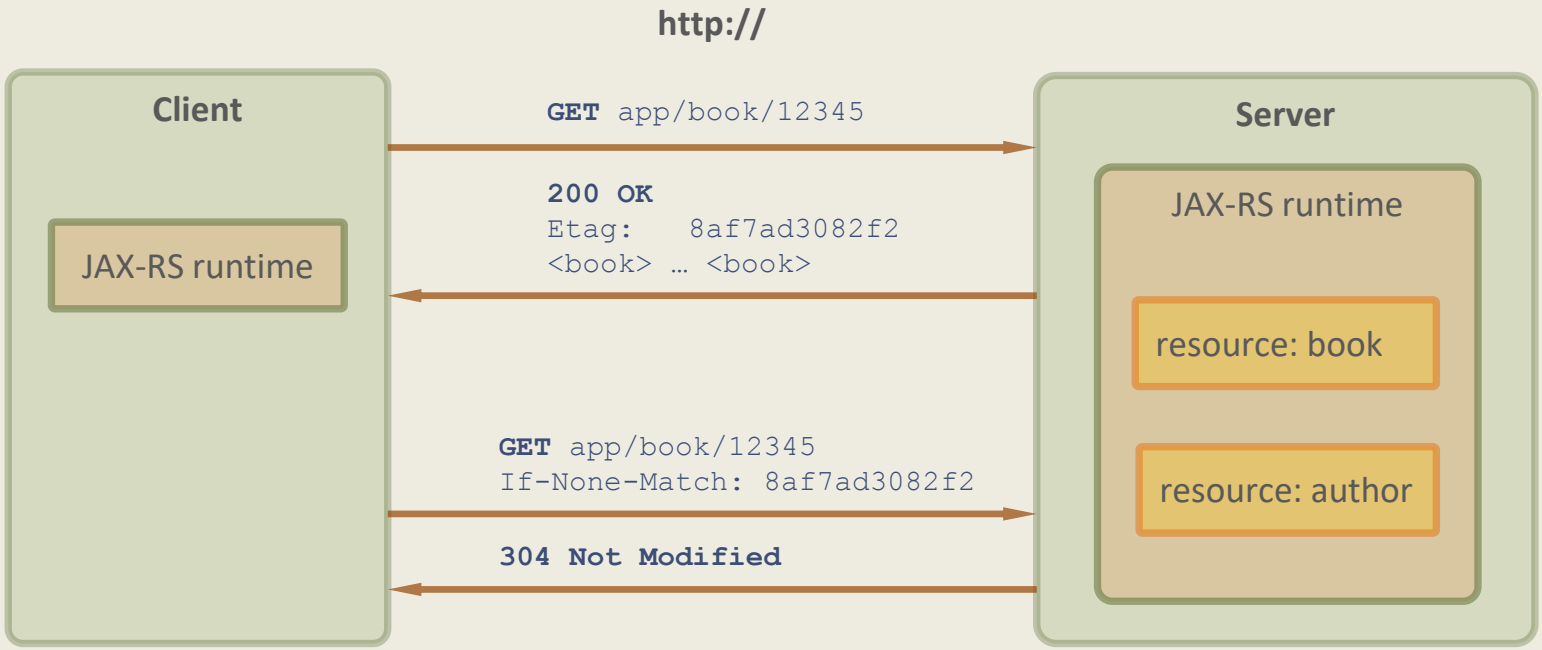
```
<application xmlns="http://wadl.dev.java.net/2009/02">
 <grammars>   <xs:schema>  ...  <xs:schema>   </grammers>
 <resources base="http://locathost:8080/app/">
  <resource path="book/{id}">
    <param name="id" style="template" type="xs:long"/>
    <method name="GET">
      <response>
        <representation element="book" mediaType="application/xml"/>
        <representation element="book" mediaType="application/json"/>
      </response>
    </method>
    <method name="DELETE"/>
  </resource>
  <resource path="book">
    <method name="GET">  ...  </method>
  </resource>
 </resources>
</application>
```

Ресурс

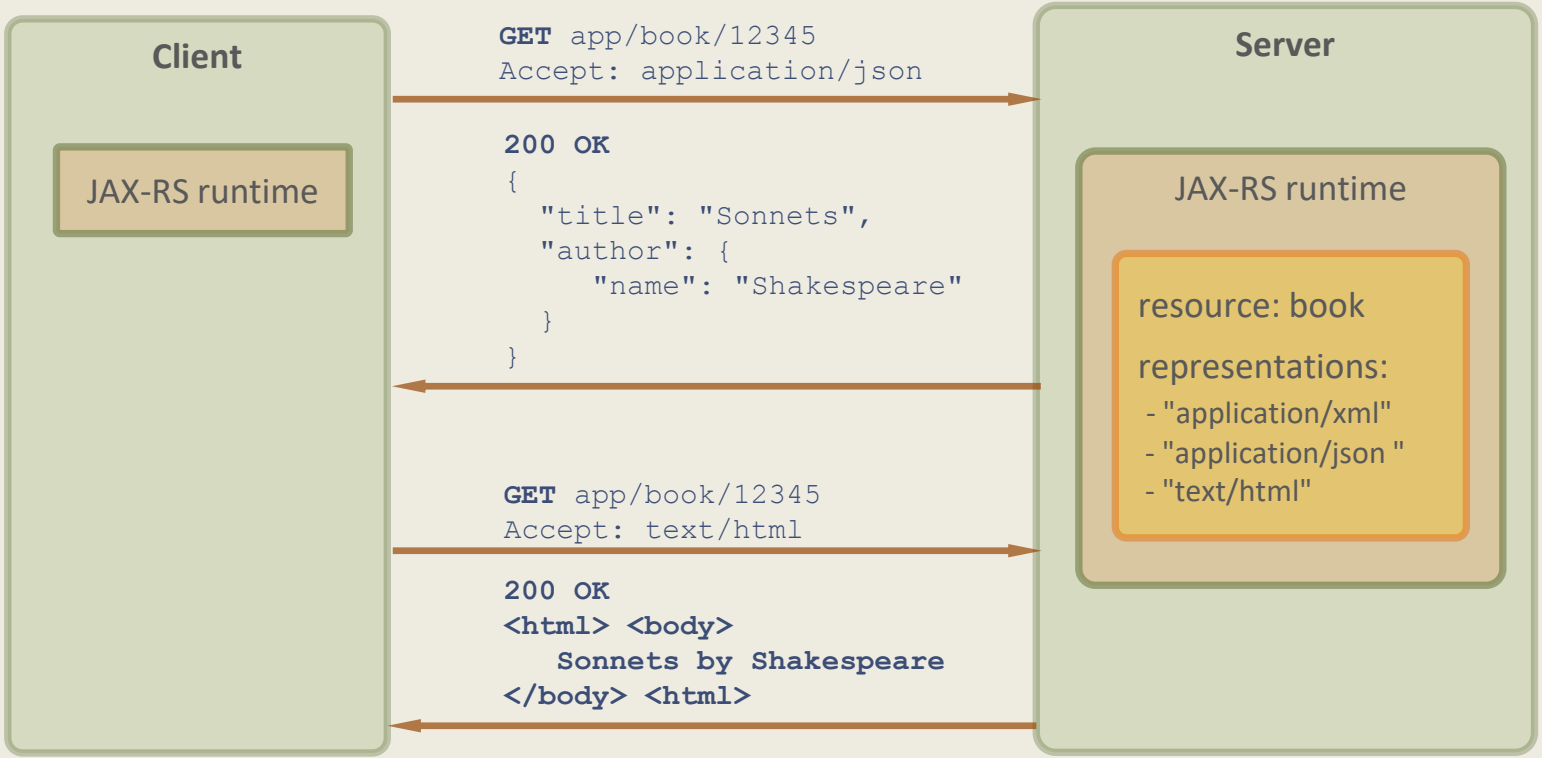
Метод

Ресурс

Условный запрос



Согласование содержимого



Используемые заголовки

Заголовок	Назначение	Пример
Accept	Допустимые типы содержимого	text/plain, application/json, application/xml
Accept-Charset Accept-Encoding Accept-Language	Допустимые наборы символов , варианты кодировки, язык ответа	utf-8 gzip, deflate en-US
Cookie	Файл HTTP-cookie, ранее отосланный сервером	
Content-Length Content-Type	Длина тела запроса в байтах MIME-тип тела запроса	1024 text/xml
Date	Дата и время отправки сообщения	
Etag	Хеш-код (MD5- или SHA1) конкретного состояния ресурса	8af7ad3082f20958
If-Match If-None-Match	Действие производится, только если на сервере есть(нет) ресурс с переданным значением хеша	8af7ad3082f20958
If-Modified-Since If-Unmodified-Since	Если ресурс (не) изменился, сервер отправляет код 304 и пустое тело ответа	
User-Agent	Пользовательский агент	Mozilla/5.0

Спецификации REST

- WADL (*Web Application Description Language*)
  - Машинно-читаемое XML-описание службы, аналог WSDL
  - Не стандартизирован
- Спецификация API – JAX-RS 2.0 → Java EE 7
  - Аннотации
  - Автоматическая генерация/парсинг запросов
  - JAXB, JSON
- Реализации
  - Эталон – Jersey
  - Apache CXF, RESTEasy (JBoss)

Простой ресурс

- Класс-ресурс

```
@Path("/helloworld")
public class HelloResource {
    @GET                        // Method to process HTTP GET requests
    @Produces("text/plain")     // will produce content of the MIME type "text/plain"
    public String getClichedMessage() {
        return "Hello World";
    }
}
```

*http://localhost:8080/{context}/**resources**/**helloworld***

- Запуск сервиса

```
JAXRSServerFactoryBean sf = new JAXRSServerFactoryBean();
sf.setResourceClasses( HelloResource.class );
sf.setAddress("http://localhost:8080/{context}/resources");
Server server = sf.create();
...
server.stop();
```

List<Class>

web.xml:
cxf servlet

Конфигурация CXF

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxrs="http://cxf.apache.org/jaxrs"
       xsi:schemaLocation="
          http://www.springframework.org/schema/beans
          http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
          http://cxf.apache.org/jaxrs http://cxf.apache.org/schemas/jaxrs.xsd">

    <bean id="helloServiceBean" class="package.HelloResource"/>

    <jaxrs:server id="rsService" address="/rs">
        <jaxrs:serviceBeans>
            <ref bean="helloServiceBean"/>
        </jaxrs:serviceBeans>
    </jaxrs:server>

</beans>
```

Тестирование

- `http://{host}:{port}`
  `/{context}/{CXF-servlet}`
  `/[{jaxrs:server path}]?_wadl`
- С помощью GET, POST-запросов из браузера
- REST Client tool window

  Tools | Test RESTful Web Service
- SoapUI

Пример: bookstore

resource	GET	POST	PUT	DELETE
/books	Весь список книг, поиск книг	–	–	–
/book/{id}	Информация о книге	Добавить книгу (id не указывается)	Изменить выходные данные книги	Удалить книгу
/authors	Список всех авторов, поиск авторов	–	–	–
/author/{id}	Информация об авторе	Добавить автора (id не указывается)	Изменить данные автора	Удалить автора
/book/{id}/author	Список авторов книги	Добавить автора для книги	–	Удалить автора из книги
/book/{id}/review	Список отзывов книги	Добавить отзыв по книге	Редактировать отзыв	Удалить отзыв по книге

Аннотации

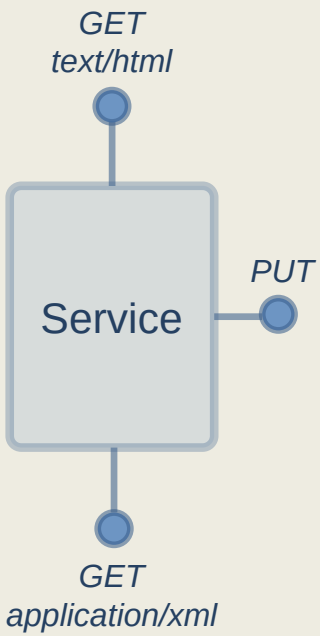
Аннотация	Назначение	Пример
@Path	Шаблон относительного URI ресурса, может содержать параметры	`@Path("/book")` `public class BookService {`
@GET, @POST, @PUT, @DELETE, @HEAD	Помечают методы, обрабатывающие запросы	`@GET` `@Path("/{id}")`
@Consumes @Produces	Content-negotiating на стороне сервера	`@Produces("text/html")` `@Consumes("*/*")`
@PathParam, @QueryParam @MatrixParam	Доступ к параметрам шаблона URI, параметрам GET-запроса	`public String doGetAsHtml(` `  @PathParam("id") String rId,`

@HeaderParam
@CookieParam
@FormParam
	Доступ к заголовкам, кукам и данным формы (Content-Type: application/x-www-form-urlencoded)	`  @CookieParam("user") String uId){`
		`  // ...`
		`}`

@Context	Доступ к доп. данным запроса/ответа	`@GET`
		`public void f(@Context HttpHeaders h){`
		`  map = h.getRequestHeaders();`
		`}`

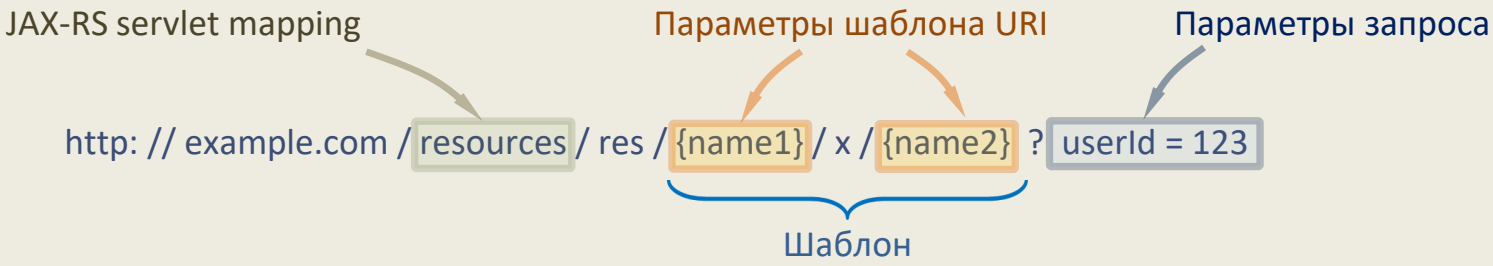
@Provider	Переопределение стандартных классов, (MessageBodyReader, MessageBodyWriter)	

Обработчики запросов



- Соответствие типам запросов – аннотации @GET, @POST, @PUT, @DELETE, @HEAD
- Один тип запросов – множество методов, различающихся типом @Produces или @Consumes
- Реализации методов HEAD и OPTIONS генерируются автоматически
- Можно использовать собственные типы запросов
- Возвращают – void или Response
- ResponseBuilder – средство построения ответа (доп. заголовки, …)
- MessageBodyReader or MessageBodyWriter – преобразование : java-объект - тело сообщения

Параметры




```
@GET
public String doGetAsHtml( @PathParam("name1") String nameFirst,
                           @PathParam("name2") String nameSecond,
                           @DefaultValue("0") @QueryParam ("userId") int id ) { }
```

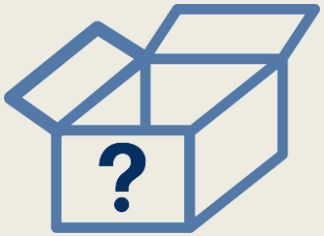
- Значение может быть пустым или содержать пробелы:

```
http://example.com/resources/res//x/Main%20User
```

- По умолчанию – значения должны соответствовать "[^/]+?"
- Можно назначить свой формат:

```
@Path("res/{name1}/x/{name2: [a-zA-Z][a-zA-Z_0-9]*}")
```

Что можно передавать?



- Примитивы и обертки, кроме char, Character
- Любой класс с конструктором:

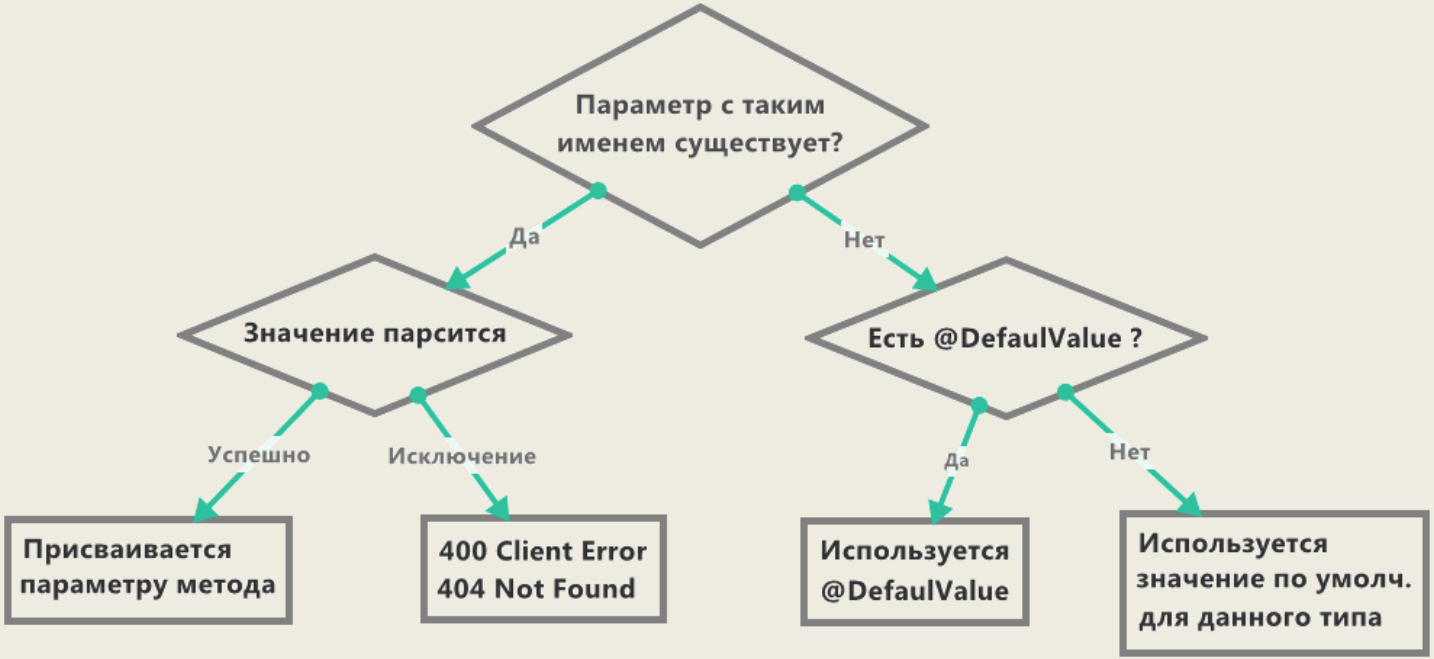
```
public MyClass(String param) { … }
```

- или методом:

```
public static void valueOf(String param) { … }
```

- Коллекцию List<T>, Set<T>, или SortedSet<T>, где T – допустимый тип
- Если передано несколько значений с одним именем,
то тони доступны через List<T>

Преобразование параметров



Данные из формы

1) Отдельные параметры по имени

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("name") String name) {
    // ...
}
```

2) Все параметры в виде Map

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(MultivaluedMap<String, String> formParams) {
    // ...
}
```

Используем @Context

- Информация о URI запроса:

```
@POST
public String doPost(@Context UriInfo ui) {
    MultivaluedMap<String, String> queryParams = ui.getQueryParameters();
    MultivaluedMap<String, String> pathParams = ui.getPathParameters();
}
```

- Информации о заголовках запроса :

```
@GET
public String doGet(@Context HttpHeaders hh) {
    MultivaluedMap<String, String> headerParams = hh.getRequestHeaders();
    Map<String, Cookie> pathParams = hh.getCookies();
```

- Request - content negotiating
- SecurityContext – пользователь, группа, роли, использование https

@Provider

- Объекты, переопределяющие некоторые функции среды выполнения JAX-RS
- Виды:
  - Entity: `MessageBodyReader , MessageBodyWriter`

    HTTP request body → method parameters

    method return value → response entity body
  - Context: `ContextResolver`

    context → @Context
  - Exception: `ExceptionMapper`

    Java exceptions → JAX-RS Response

Создаем MessageBodyWriter

- Создаем класс-Provider

```
@Provider
@Produces("text/html")
public class BookHtmlProvider implements MessageBodyWriter<BookVO>{
    public void writeTo( ... , OutputStream outputStream) { ... }
}
```

- Создаем GET-метод для ресурса

```
@GET
@Produces({"text/html"})
public BookVO getAsHtml(@PathParam("id") long id) { return new BookVO(); }
```

@XmlRootElement

- Добавляем Provider в конфигурацию CXF

```
<jaxrs:server id="customerService" address="/">
   <jaxrs:serviceBeans>  ...  </jaxrs:serviceBeans>
   <jaxrs:providers>
       <bean class="package.resources.BookHtmlProvider"/>
   </jaxrs:providers>
</jaxrs:server>
```

Вопросы

1. Перечислите основные отличия RESTful-служб от служб SOAP.

2. Что такое content negotiating и как он реализуется?

3. В чем разница между методами PUT и POST? Какие проблемы могут возникнуть, если поменять местами аннотации @PUT и @POST?

4. Могут ли ресурсы быть вложенными?

5. Можно ли аннотацией @GET пометить не один, а несколько методов ресурса?

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 REST API.
   Необходимо повторно использовать ту же реализацию бизнес логики, которая ранее была доступна через веб-интерфейс.

2. Протестируйте работу служб с помощью REST Client и SoapUI .

3. Создайте простого потребителя этих служб, который содержит код для обращения к ресурсам сервера: получение в виде списка, добавление, удаление, изменение.

4.* Внедрите фреймворк Jackson для серализации и десериализации пересылаемых сообщений в формат JSON.

5.* Создайте подробную документацию вашего REST API с помощью ……