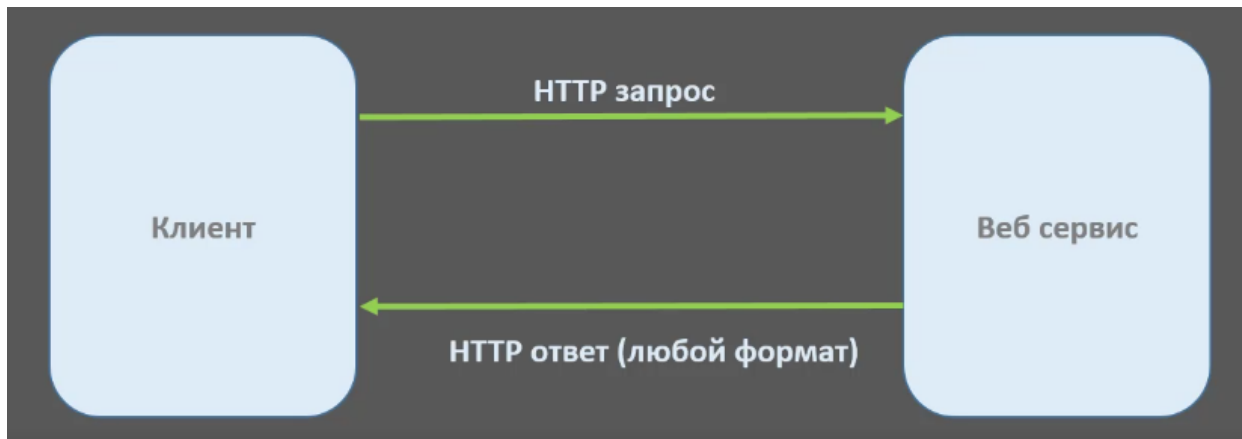


Создание RESTfull вэб-служб (web services)

Как работают механизмы вэб-служб на основе архитектуры REST?



Реализации RESTful

- Apache CXF
- Jersey RI
- <https://jersey.java.net/>

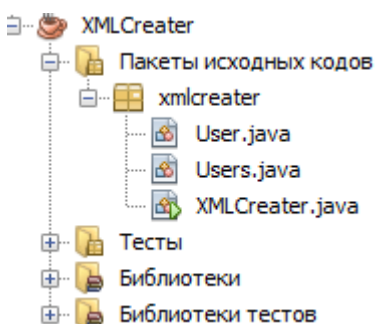
Описание сервиса

- WADL (Web Application Description Language)
- <http://www.w3.org/Submission/wadl/>
- Не является полноценным стандартом W3C
- <http://bitworking.org/news/193/Do-we-need-WADL>

Как создать из класса его xml представление и как восстановить из xml разметки класс java?

Пример 0 XMLCreator создание разметки xml на основании java-классов

Создадим простое консольное приложение на java и добавим в него два класса: User – описывает бизнес-сущность, а класс Users – обертка над коллекцией List (Response запросы служб (их ответы) должны возвращать простые объекты, которые легко переделываются в Json объекты.). У нас должна получится вот такая структура проекта:



Приведем тексты всех классов:

| User.java | Users.java |
|---|--|
| <pre> 1 package xmlcreator; 2 import java.util.ArrayList; 3 import java.util.List; 4 import javax.xml.bind.annotation.XmlElement; 5 import javax.xml.bind.annotation.XmlRootElement; 6 import javax.xml.bind.annotation.XmlAccessType; 7 import javax.xml.bind.annotation.XmlAccessorType; 8 9 @XmlRootElement 10 @XmlAccessorType(XmlAccessType.NONE) 11 public class Users { 12 @XmlElement(name="user") 13 private List<User> users = new ArrayList<User>(); 14 15 public List<User> getUsers() { 16 return users; 17 } 18 19 public void setUsers(List<User> users) { 20 this.users = users; 21 } 22 } </pre> | <pre> 1 package xmlcreator; 2 3 import javax.xml.bind.annotation.XmlAccessType; 4 import javax.xml.bind.annotation.XmlAccessorType; 5 import javax.xml.bind.annotation.XmlAttribute; 6 import javax.xml.bind.annotation.XmlElement; 7 import javax.xml.bind.annotation.XmlRootElement; 8 9 @XmlRootElement 10 @XmlAccessorType(XmlAccessType.NONE) 11 // @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER) 12 // @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER) 13 public class User { 14 @XmlElement 15 private String name; 16 @XmlElement 17 int age; 18 @XmlAttribute(name = "IDENTITY", required = true) 19 int id; 20 @Override 21 public String toString() { 22 return "User{" + "name=" + name + '}'; 23 } 24 public User() { 25 26 } 27 public User(String name) { 28 this.name = name; 29 } 30 public String getName() { 31 return name; 32 } 33 public void setName(String name) { 34 this.name = name; 35 } 36 public void setAge(int age) { 37 this.age = age; 38 } 39 public int getAge() { 40 return age; 41 } 42 public int getId() { 43 return id; 44 } 45 public void setId(int id) { 46 this.id = id; 47 } 48 } </pre> |

Главный класс проекта:

```

1  package xmlcreator;
2
3  import java.io.File;
4  import javax.xml.bind.JAXBContext;
5  import javax.xml.bind.JAXBException;
6  import javax.xml.bind.Marshaller;
7  import javax.xml.bind.Unmarshaller;
8
9  public class XMLCreator {
10
11      public static void main(String[] args) {
12          User customer = new User();
13          customer.setId(100);
14          customer.setName("Вася");
15          customer.setAge(29);
16
17          Users users = new Users();
18          users.getUsers().add(customer);
19          customer.setId(200);
20          users.getUsers().add(customer);
21          try {
22
23              File file = new File("d:\\MyData\\file.xml");
24              JAXBContext jaxbContext = JAXBContext.newInstance(Users.class);
25              Marshaller jaxbMarshaller = jaxbContext.createMarshaller();
26
27              // output pretty printed
28              jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
29
30              jaxbMarshaller.marshal(users, file);
31              jaxbMarshaller.marshal(users, System.out);
32
33          } catch (JAXBException e) {
34              e.printStackTrace();
35          }
36          System.out.println("информация из файла d:\\MyData\\file.xml");
37          loadObjectsFomXmlFile();
38
39      }
40
41      private static void loadObjectsFomXmlFile() {
42          try {
43
44              File file = new File("d:\\MyData\\file.xml");
45              JAXBContext jaxbContext = JAXBContext.newInstance(Users.class);
46
47              Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
48              Users users = (Users) jaxbUnmarshaller.unmarshal(file);
49              System.out.println(users.getUsers());
50
51          } catch (JAXBException e) {
52              e.printStackTrace();
53          }
54      }
55  }

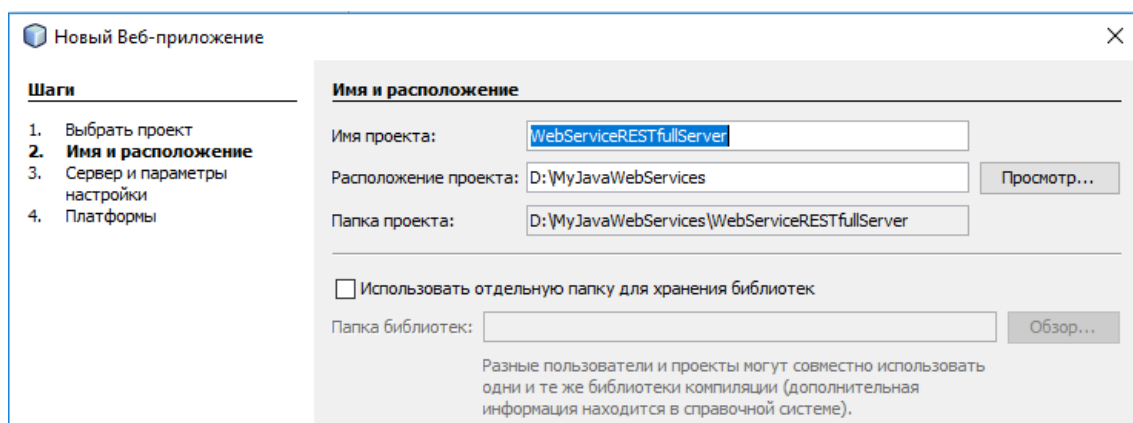
```

Приведем разметку, коротая у нас получилась:

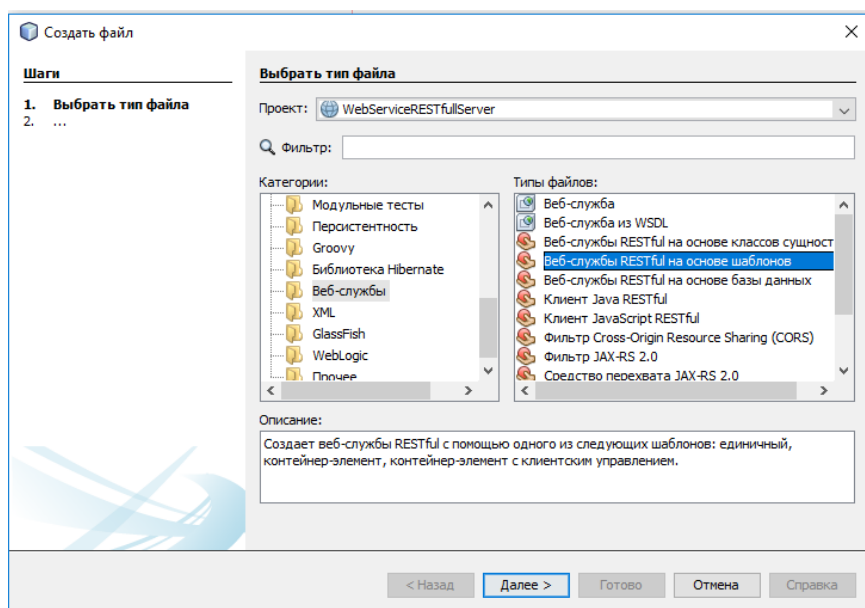
| Тэги в java-классе | Сгенерированная разметка |
|---|---|
| <pre> 9 @XmlElement 10 @XmlAccessorType(XmlAccessType.NONE) 11 // @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER) 12 // @XmlAccessorType(XmlAccessType.PUBLIC_MEMBER) 13 public class User { 14 @XmlElement 15 private String name; 16 @XmlElement 17 int age; 18 @XmlAttribute(name = "IDENTITY", required = true) 19 int id; 9 @XmlElement 10 @XmlAccessorType(XmlAccessType.NONE) 11 public class Users { 12 @XmlElement(name="user") 13 private List<User> users = new ArrayList<User>(); 14 } </pre> | <div> <div>Вывод X</div> <div> <div>Процесс базы данных Java DB x</div> <div>GlassFish Server 4.1.1 x</div> <div>XMLCreator (run) x</div> </div> <div>run:</div> <pre> <?xml version="1.0" encoding="UTF-8" standalone="yes"?> <users> <user IDENTITY="200"> <name>Вася</name> <age>29</age> </user> <user IDENTITY="200"> <name>Вася</name> <age>29</age> </user> </users> </pre> <div>информация из файла d:\MyData\file.xml</div> <div>[User{name=Вася}, User{name=Вася}]</div> <div>СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)</div> </div> |

Пример 1 Создание и использование простого RESTfull сервиса

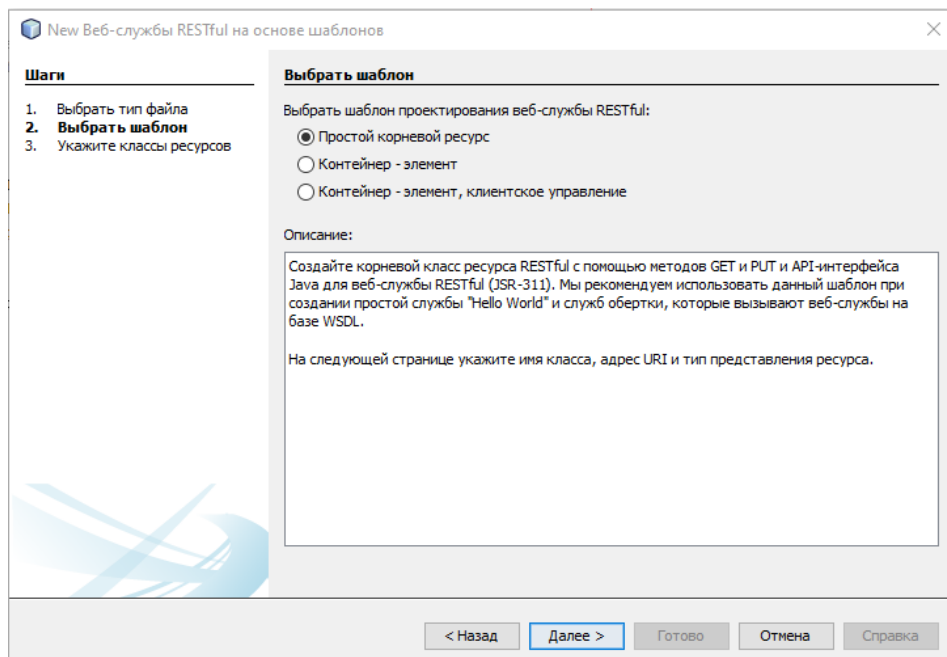
Создаем новое Java Web приложение Java Web Application с именем WebServiceRESTfullServer

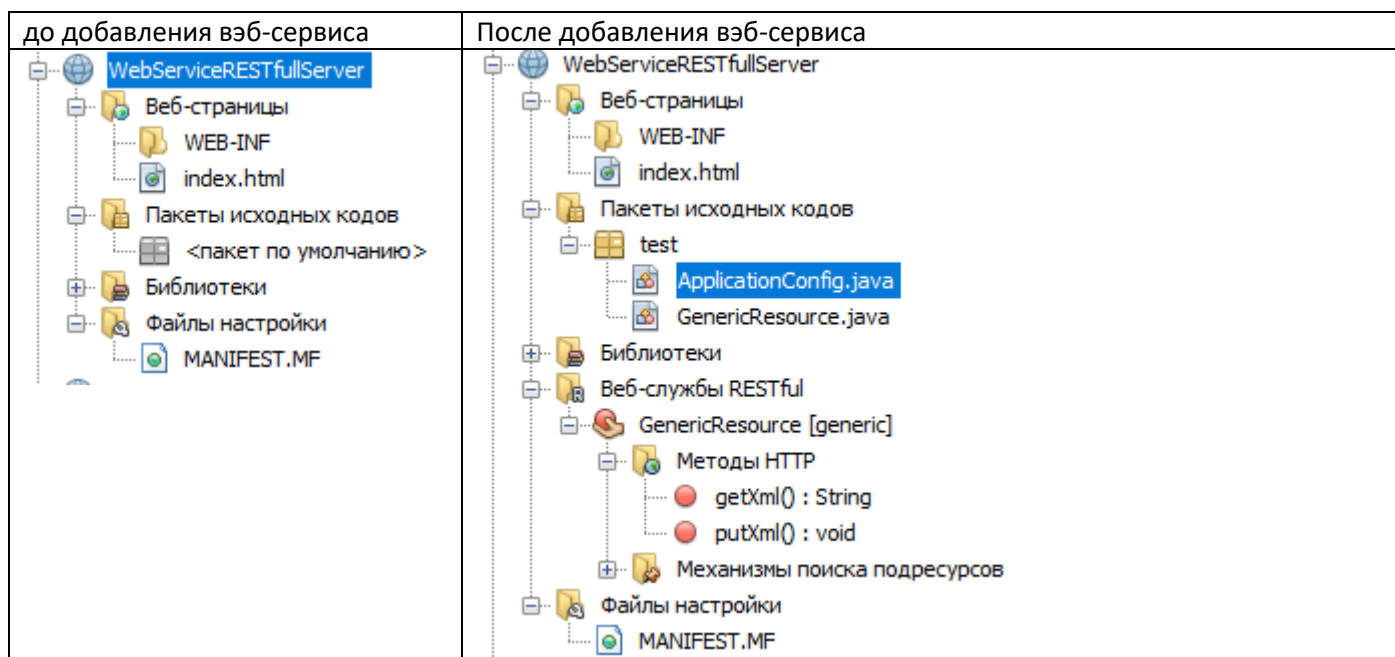
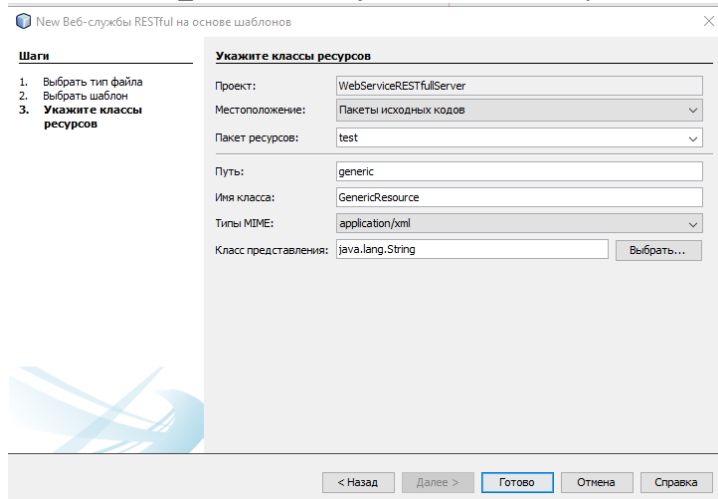


Далее в этом приложении создаем веб-сервис: Web-Service на основе RESTful WebService from Patterns



выбираем Simple root resource (Простой корневой ресурс)





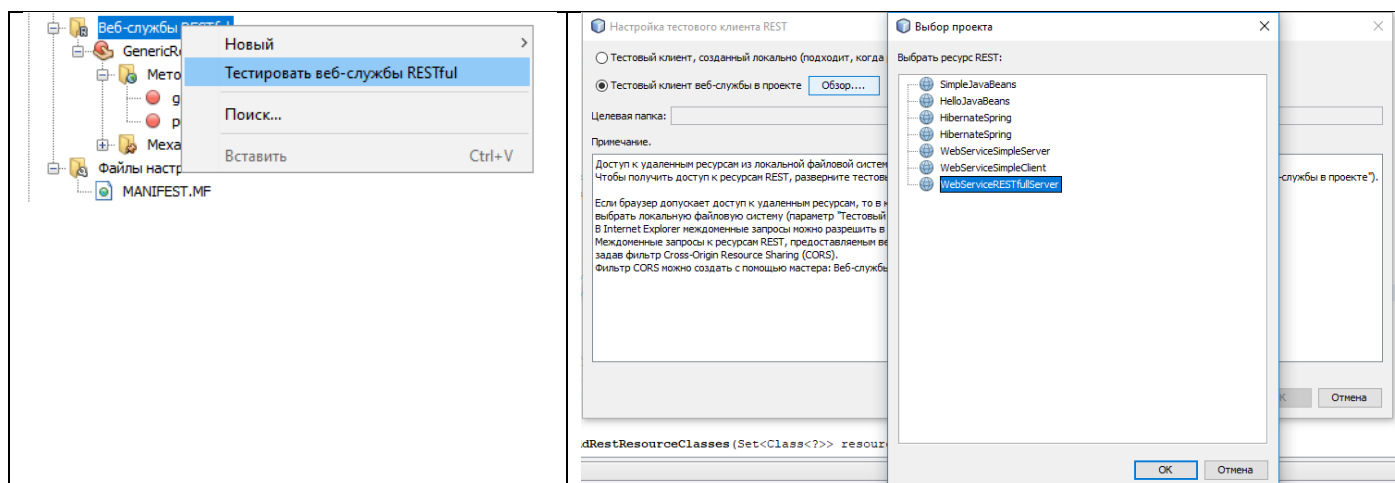
Рассмотрим назначение классов.

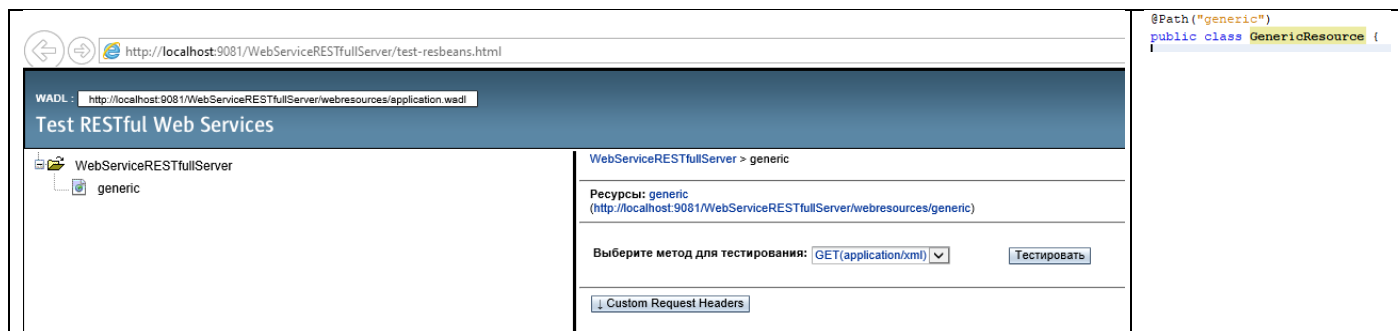
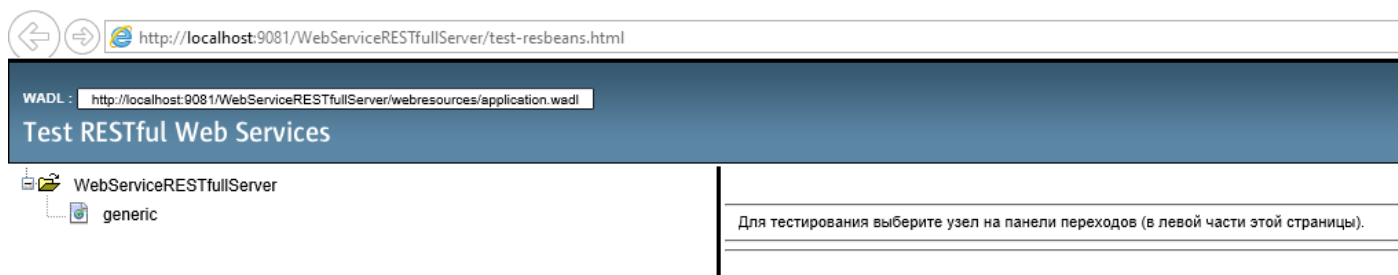
ApplicationConfig – класс описывающий настройки нашей вэб-службы (например, какая библиотека отвечает за разбор или создание json ответов).

Класс GenericResource – содержит код, реализующий логику наше службы.

Тестирование созданного сервиса:

Вызовем контекстное меню для раздела вэб-службы нашего проекта





После нажатия на кнопку «Т» мы видим отображение сведений об исключительной ситуации. Это вызвано текущей реализацией метода для тэга GET внутри класса GenericResource:

| исходный | результат |
|---|--|
| <pre>@GET @Produces(MediaType.APPLICATION_XML) public String getXml() { //TODO return proper representation object throw new UnsupportedOperationException(); }</pre> | <pre>@GET //@Produces(MediaType.APPLICATION_XML) @Produces(MediaType.TEXT_HTML) public String getXml() { //throw new UnsupportedOperationException(); return "Привет from getXml()"; }</pre> |

Создадим в нашем проекте классы с выгрузкой в xml: User, полностью аналогичный классу User из проекта XMLCreator, и класс ResponseList, аналогичный классу Users из проекта XMLCreator. И добавим в класс GenericResource метод getUsers()

| | |
|---|---|
| <pre>1 package test; 2 3 import java.util.ArrayList; 4 import java.util.List; 5 import javax.xml.bind.annotation.XmlElement; 6 import javax.xml.bind.annotation.XmlRootElement; 7 import javax.xml.bind.annotation.XmlAccessType; 8 import javax.xml.bind.annotation.XmlAccessorType; 9 10 @XmlRootElement 11 @XmlAccessorType(XmlAccessType.NONE) 12 public class ResponseList { 13 @XmlElement(name="user") 14 private List<User> users = new ArrayList<User>(); 15 16 public List<User> getUsers() { 17 return users; 18 } 19 20 public void setUsers(List<User> users) { 21 this.users = users; 22 } 23 }</pre> | <pre>@GET @Produces(MediaType.APPLICATION_JSON) @Path("users") //@Path("/") public ResponseList getUsers() { User customer = new User(); customer.setId(100); customer.setName("Вася"); customer.setAge(29); ResponseList responseList = new ResponseList(); responseList.getUsers().add(customer); return responseList; }</pre> |
|---|---|

Внесем изменения к текст метода getClasses класса .В итоге у нас должна получится такая обновленная структура проекта :

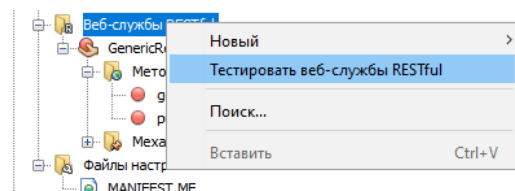
```
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new java.util.HashSet<>();
        // following code can be used to customize Jersey 2.0 JSON provider:
        try {
            Class jsonProvider = Class.forName("org.glassfish.jersey.jackson.JacksonFeature");
            // Class jsonProvider = Class.forName("org.glassfish.jersey.moxy.json.MoxyJsonFeature");
            // Class jsonProvider = Class.forName("org.glassfish.jersey.jettison.JettisonFeature");
            resources.add(jsonProvider);
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(getClass().getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        addRestResourceClasses(resources);
        return resources;
    }
}
```

Приведем отличие между провайдерами:

| | MOxy | JSON-P | Jackson | Jettison |
|-------------------------------------|------|--------|---------|----------|
| POJO-based JSON Binding | Yes | No | Yes | No |
| JAXB-based JSON Binding | Yes | No | Yes | Yes |
| Low-level JSON parsing & processing | No | Yes | No | Yes |

Выполним очищение и повторную сборку проекта. Создадим наши тесты заново



и протестируем работу нашего нового метода:

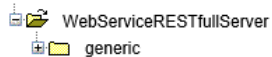
11.05.2017 06_Web Services java часть 6 для студентов.docx

Результат запуска тестовой страницы

← → http://localhost:9081/WebServiceRESTfullServer/test-resbeans.html

WSDL: http://localhost:9081/WebServiceRESTfullServer/webresources/application.wadl

Test RESTful Web Services



WebServiceRESTfullServer

- generic

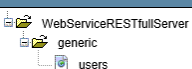
Для тестирования выберите узел на панели переходов (в левой части этой страницы).

Предыдущий тест:

← → http://localhost:9081/WebServiceRESTfullServer/test-resbeans.html

WSDL: http://localhost:9081/WebServiceRESTfullServer/webresources/application.wadl

Test RESTful Web Services



WebServiceRESTfullServer

- generic
 - users

WebServiceRESTfullServer > generic

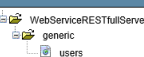
Ресурсы: generic
(http://localhost:9081/WebServiceRESTfullServer/webresources/generic)

Выберите метод для тестирования: GET(text/html)

← → http://localhost:9081/WebServiceRESTfullServer/test-resbeans.html

WSDL: http://localhost:9081/WebServiceRESTfullServer/webresources/application.wadl

Test RESTful Web Services



WebServiceRESTfullServer

- generic
 - users

WebServiceRESTfullServer > generic

Ресурсы: generic
(http://localhost:9081/WebServiceRESTfullServer/webresources/generic)

Выберите метод для тестирования: GET(text/html)

Статус: 200 (OK)

Ответ:

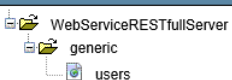
Привет from getXml()

Выделили узел дерева users и нажали на кнопку «Тестировать» в результате увидели:

← → http://localhost:9081/WebServiceRESTfullServer/test-resbeans.html

WSDL: http://localhost:9081/WebServiceRESTfullServer/webresources/application.wadl

Test RESTful Web Services



WebServiceRESTfullServer

- generic
 - users

WebServiceRESTfullServer > generic > users

Ресурсы: generic/users
(http://localhost:9081/WebServiceRESTfullServer/webresources/generic/users)

Выберите метод для тестирования: GET(application/json)

Статус: 200 (OK)

Ответ:

```
{
  "users": [
    {
      "name": "Вася",
      "age": 29,
      "id": 100
    }
  ]
}
```

Рассмотрим теперь как формируется адрес **http://localhost:9091/webresources/generic/users** для нашего сервиса

слово webresources из

```
@javax.ws.rs.ApplicationPath("webresources")
```

```
public class ApplicationConfig extends Application
```

слово generic из

```
@Path("generic")
```

```
public class GenericResource
```

слово users из

```
@GET
```

```
@Produces(MediaType.APPLICATION_JSON)
```

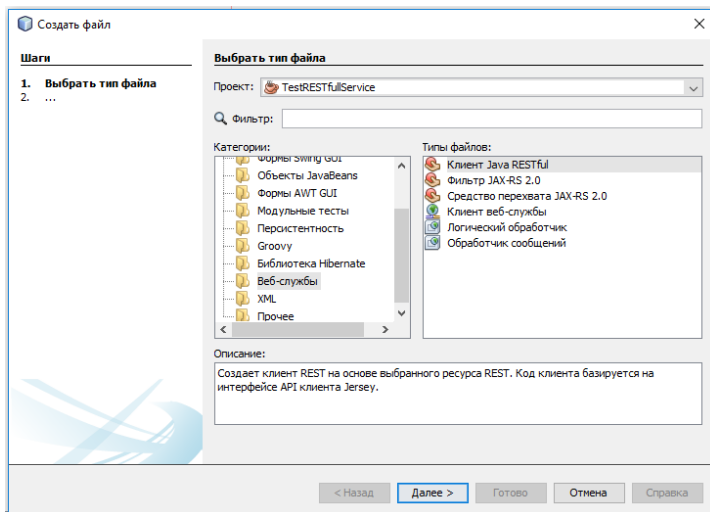
```
@Path("users")
```

```
public ResponseList getUsers()
```

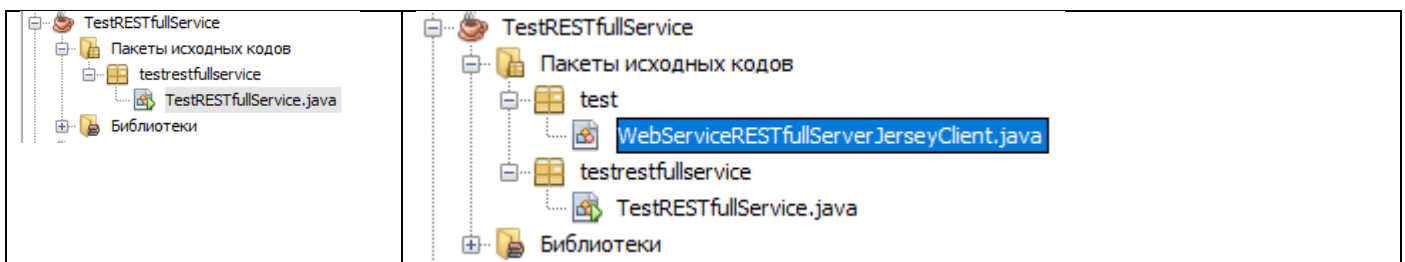
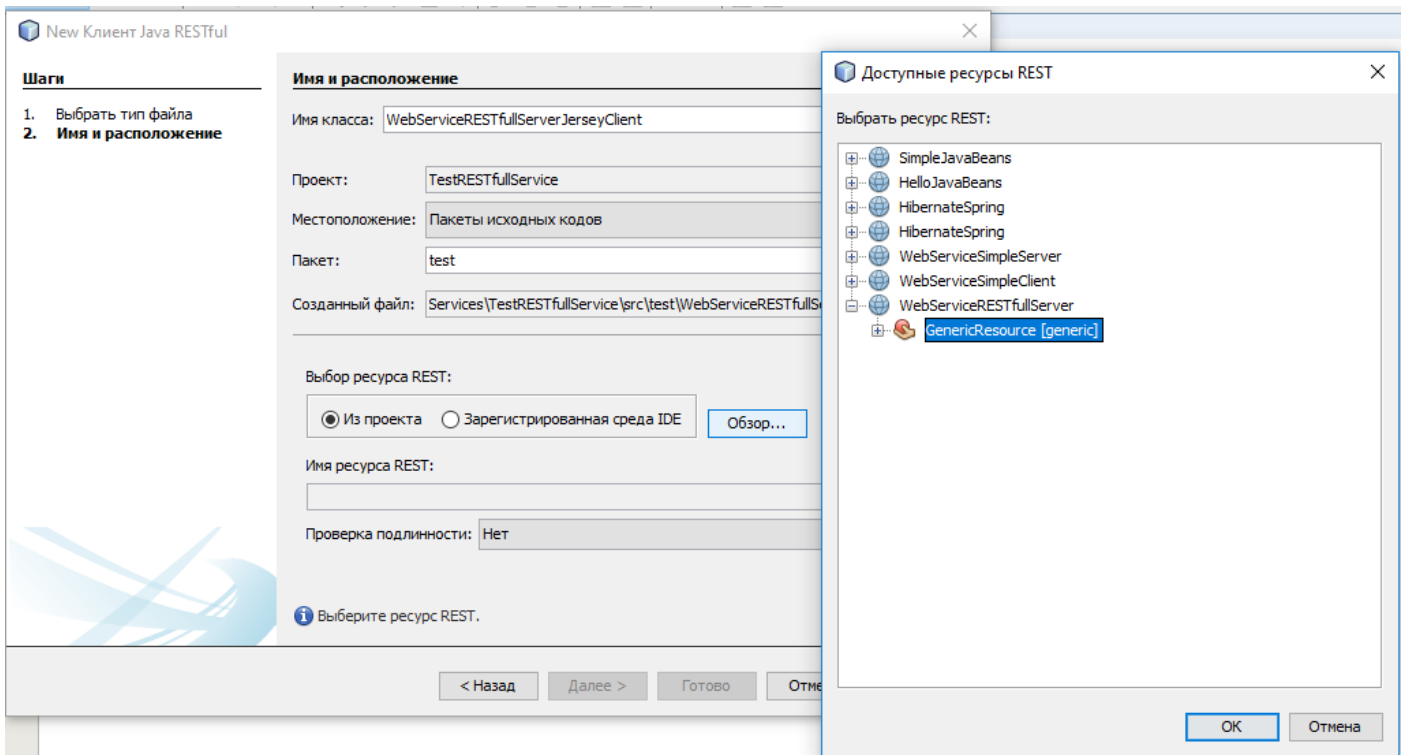
Создание клиента вэб-службы

Создадим обычное консольное java-приложение с именем TestRESTfullService

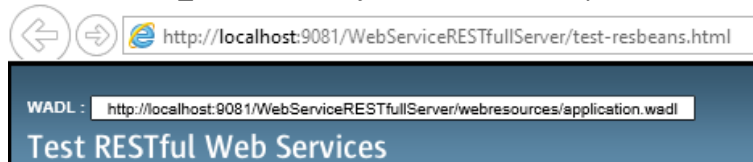
Далее добавляем в него клиент вэб-службы



Далее указываем из какого проекта нам необходимо создать клиентскую службу:



Далее нам необходимо добавить все необходимые классы нашей службы (User и Users). Для этого сгенерируем их из файла wadl (ссылка на который есть в нашем тестовом приложении):



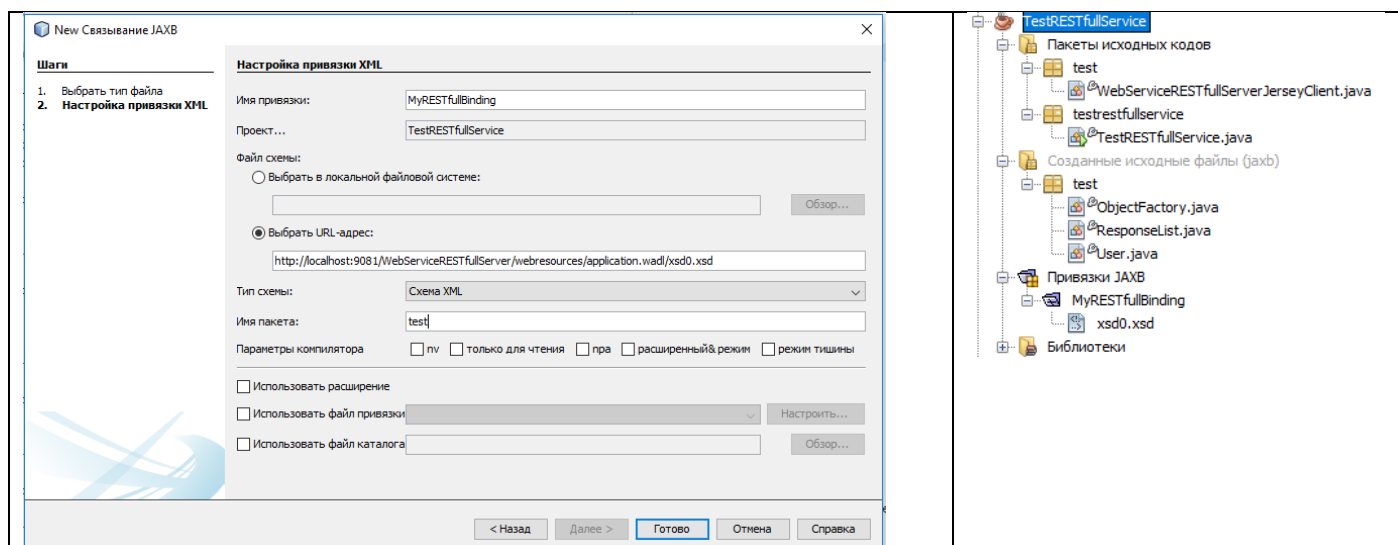
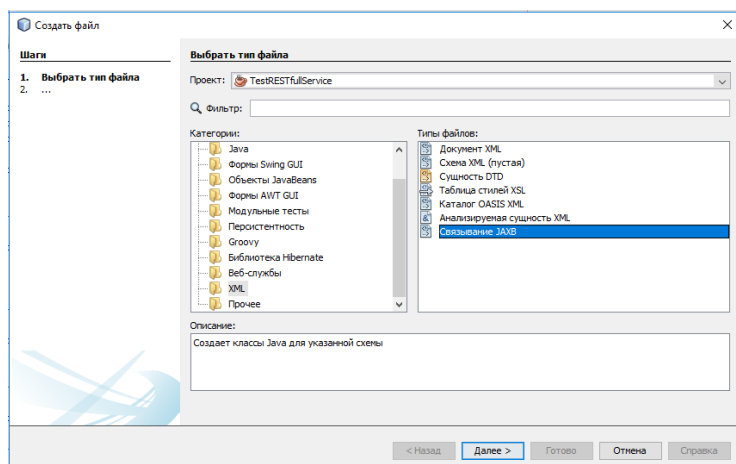
Открываем эту ссылку в измененном виде:

<http://localhost:9081/WebServiceRESTfullServer/webresources/application.wadl/xsd0.xsd>



```
<?xml version="1.0" standalone="true"?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:element type="responseList" name="responseList"/>
  <xs:element type="user" name="user"/>
  - <xs:complexType name="responseList">
    - <xs:sequence>
      <xs:element maxOccurs="unbounded" minOccurs="0" ref="user"/>
    </xs:sequence>
  </xs:complexType>
  - <xs:complexType name="user">
    - <xs:sequence>
      <xs:element type="xs:string" name="name" minOccurs="0"/>
      <xs:element type="xs:int" name="age"/>
    </xs:sequence>
    <xs:attribute type="xs:int" name="IDENTITY" use="required"/>
  </xs:complexType>
</xs:schema>
```

И видим, что для нас создано описание всех задействованных классов. Воспользуемся этим создадим JAXB Binding (JAXB привязку):



Главный класс выглядит так:

```
1 package testrestfullservice;
2
3 import test.ResponseList;
4 import test.User;
5 import test.WebServiceRESTfullServerJerseyClient;
6
7 public class TestRESTfullService {
8
9     public static void main(String[] args) {
10         WebServiceRESTfullServerJerseyClient client = new WebServiceRESTfullServerJerseyClient();
11         ResponseList response = client.getUsers(ResponseList.class) ;
12         for(User user : response.getUser())
13         {
14             System.out.println(user.getName());
15         }
16         client.close();
17     }
18
19 }
```