**[java 利用JAX-RS快速开发RESTful 服务](http://www.cnblogs.com/yjmyzz/p/javaee-jax-rs-tutorial.html)**

JAX-RS(Java API for RESTful Web Services)同样也是JSR的一部分，详细规范定义见[https://jcp.org/en/jsr/detail?id=311](https://jcp.org/en/jsr/detail?id=311" \t "_blank) 。从JAVA EE 6开始，已经内置了该技术，跟.NET中的RESTful WCF类似，只要在常规方法上使用一些简单的注解，就可以对外暴露成RESTful 服务.

注：本文运行环境为Jboss EAP 6.x ，其它容器特别是tomcat，并未自带JAX-RS依赖的jar包，如果采用tomcat，需要自己导入这些相关依赖jar包。

**一、最基础的HelloWorld (无参数)**

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @Path("/")

2 public class MyService {

3

4 final String XMLNS\_NAMESPACE = "http://yjmyzz.cnblogs.com/rest/service";

5 final String ROOT\_NODE = "root";

6

7 @GET

8 @Path("/json/hello")

9 @Produces(MediaType.APPLICATION\_JSON)

10 public JAXBElement<String> getHelloWorldJSON() {

11 JAXBElement<String> result = new JAXBElement<String>(new QName("",

12 ROOT\_NODE), String.class, sayHelloWorld());

13 return result;

14 }

15 。。。

16 ｝

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 private String sayHelloWorld() {

2 return "Hello JAX-RS!";

3 }

解释:  
@GET 表示该服务可以直接在浏览器地址栏里访问(对应HTTP请求中的GET方法)

@Path 这里用了二次，第一次是在Class上，相当服务的基地址；第二次是在方法上，表示具体某个服务方法的URL.

此外web.xml中，还需要增加以下内容：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 <servlet-mapping>

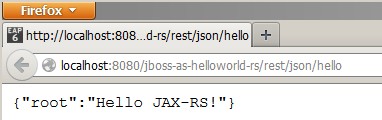
2 <servlet-name>javax.ws.rs.core.Application</servlet-name>

3 <url-pattern>/rest/\*</url-pattern>

4 </servlet-mapping>

@Produces 表示返回消息的格式，配合MediaType.APPLICATION\_JSON说明，本方法返回Json字符串格式

以上几项综合起来，最终可以用类似 http://localhost:8080/jboss-as-helloworld-rs/rest/json/hello 的地址来访问，返回的结果为：

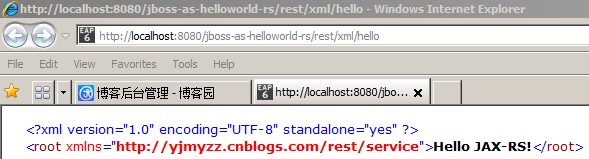


JAXBElement<T> 类似一个包装器(Wrapper)，用于将返回对象做下json/xml格式的包装，比如上图中的{root:}就是它的功劳，感兴趣的朋友，可以把返回类似改成最原始的String对比下差异.

如果需要返回XML格式，只需要把@Produces后面的MediaType.APPLICATION\_JSON换成MediaType.APPLICATION\_XML就行了，即：

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif getHelloWorldXML

效果如下：



**二、带一个基本类型参数的HelloWorld**

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @GET

2 @Path("/json/hi/{name}")

3 @Produces(MediaType.APPLICATION\_JSON)

4 public JAXBElement<String> getHelloToNameJSON(@PathParam("name") String name) {

5 JAXBElement<String> result = new JAXBElement<String>(new QName("",

6 ROOT\_NODE), String.class, sayHelloToName(name));

7 return result;

8 }

9

10 @GET

11 @Path("/xml/hi/{name}")

12 @Produces(MediaType.APPLICATION\_XML)

13 public JAXBElement<String> getHelloToNameXML(@PathParam("name") String name) {

14 JAXBElement<String> result = new JAXBElement<String>(new QName(

15 XMLNS\_NAMESPACE, ROOT\_NODE), String.class, sayHelloToName(name));

16 return result;

17 }

[复制代码](javascript:void(0);)

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 private String sayHelloToName(String name) {

2 return "Hello " + name + ",welcome to the world of JAX-RS!";

3 }

对比一下刚才的代码，有二个变化：

2.1 @Path上多了一个{name}的部分，表示url中，将允许以 /json/hi/xxx 的访问URL响应请求

2.2 方法参数前有一个 @PathParam("name") 的注解，表示如果按/json/hi/xxx的方式访问，则url中xxx的部分，会被映射到name这个参数上(说得更通俗点，就是通过解析url最后的xxx部分直接把name传进来)

运行效果如下:





**三、返回结果为实体Bean**

实际应用中，web服务返回的结果，除了基本类型，还有可能是实体Bean，先来定义一个User类

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package yjmyzz.service.model;

2

3 import javax.xml.bind.annotation.XmlAttribute;

4 import javax.xml.bind.annotation.XmlElement;

5 import javax.xml.bind.annotation.XmlRootElement;

6

7 @XmlRootElement(name = "user")

8 public class User {

9

10 String username;

11 String password;

12 int pin;

13

14 @XmlElement

15 public String getUsername() {

16 return username;

17 }

18

19 public void setUsername(String username) {

20 this.username = username;

21 }

22

23 @XmlElement

24 public String getPassword() {

25 return password;

26 }

27

28 public void setPassword(String password) {

29 this.password = password;

30 }

31

32 @XmlAttribute

33 public int getPin() {

34 return pin;

35 }

36

37 public void setPin(int pin) {

38 this.pin = pin;

39 }

40

41 }

[复制代码](javascript:void(0);)

代码中加的一堆@XML开头的注解，是为了在传输过程中，如果需要以XML格式传输，这些注解可以帮助完成 XML <-> Obj的转换（详情可参见[JAXB相关内容](http://www.cnblogs.com/yjmyzz/p/3532334.html" \t "_blank)）

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @GET

2 @Path("/xml/user")

3 @Produces(MediaType.APPLICATION\_XML)

4 public User getUserInXML() {

5 return getUser("unknown");

6 }

7

8 @GET

9 @Path("/json/user")

10 @Produces(MediaType.APPLICATION\_JSON)

11 public User getUserInJSON() {

12 return getUser("unknown");

13 }

[复制代码](javascript:void(0);)

注意：这里我们没有使用JAXBElement<T>对结果进行包装，辅助方法getUser()代码如下

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 private User getUser(String username) {

2 User user = new User();

3 user.setUsername(username);

4 String pwd = new Long(Math.round(Math.random() \* 100000)).toString();

5 user.setPassword(pwd);

6 int pin = (int) (Math.random() \* 1000);

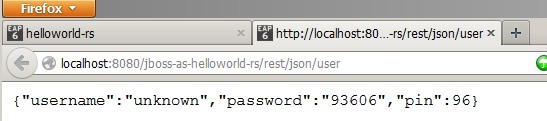
7 user.setPin(pin);

8 return user;

9 }

[复制代码](javascript:void(0);)

运行结果如下：  

当然，也可以给方法增加参数，先从最基本类型的String参数耍起：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @GET

2 @Path("/xml/userByName/{name}")

3 @Produces(MediaType.APPLICATION\_XML)

4 public User getUserInXML(@PathParam("name") String username) {

5 return getUser(username);

6 }

7

8 @GET

9 @Path("/json/userByName/{name}")

10 @Produces(MediaType.APPLICATION\_JSON)

11 public User getUserInJSON(@PathParam("name") String username) {

12 return getUser(username);

13 }

[复制代码](javascript:void(0);)

运行结果如下：  




如果加上包装器JAXBElement<User>后，返回结果会变成什么样呢？ 我也很好奇，so, 嘿-喂狗！

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @GET

2 @Path("/xml/userByName2/{name}")

3 @Produces(MediaType.APPLICATION\_XML)

4 public JAXBElement<User> getUserInXML2(@PathParam("name") String username) {

5 JAXBElement<User> result = new JAXBElement<User>(new QName(

6 XMLNS\_NAMESPACE, ROOT\_NODE), User.class, getUser(username));

7 return result;

8 }

9

10 @GET

11 @Path("/json/userByName2/{name}")

12 @Produces(MediaType.APPLICATION\_JSON)

13 public JAXBElement<User> getUserInJSON2(@PathParam("name") String username) {

14 JAXBElement<User> result = new JAXBElement<User>(new QName("",

15 ROOT\_NODE), User.class, getUser(username));

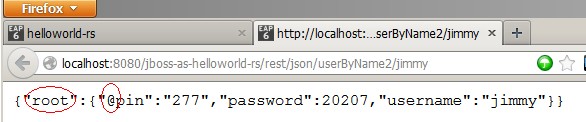
16 return result;

17 }

[复制代码](javascript:void(0);)

结果如下：





个中差异，请自行体会

**四、单个String参数的实体Bean注入**

web服务中方法的传入参数，不可能都是基础类型，同样也可以是实体Bean

http://images.cnblogs.com/OutliningIndicators/ContractedBlock.gif getProductXML & getProductXML2

这里我们用到了一个新的Bean ：Product类，代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package yjmyzz.service.model;

2

3 import javax.xml.bind.annotation.XmlElement;

4 import javax.xml.bind.annotation.XmlRootElement;

5

6 @XmlRootElement(name = "product")

7 public class Product {

8

9 public Product() {

10 System.out.println("construct: Product() is called!");

11 }

12

13 public Product(String name) {

14 System.out.println("construct: Product(String name) is called!");

15 this.productName = name;

16 }

17

18 @XmlElement

19 public String getProductName() {

20 return productName;

21 }

22

23 public void setProductName(String productName) {

24 this.productName = productName;

25 }

26

27 private String productName;

28

29 }

[复制代码](javascript:void(0);)

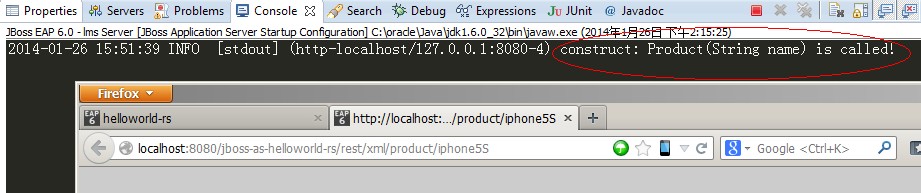
先提个问题：从@Path上看，我们希望用类似 /xml/product/xxx 来请求该web方法，但是方法中的参数是一个Bean实例，而url中的xxx部分只是一个字符串，一个字符串如何被映射成一个Bean实例呢？

关键在于Product中的构造函数：Product(String name) ，方法getProductXML(@PathParam("name") Product product) 被请求时，url中的xxx部分会先映射成参数name，然后name又会被传入到构造函数Product(String name)中，于是一个崭新的Product对象诞生了！(可以通过eclipse Console的输出来印证Product(String name)是否被调用)

运行结果：



eclipse的控制台里，也可以看到构造器，确实被调用了



除了利用“带参数的构造器”来实现“url参数->Bean实例”的注入，JAX-RS机制还允许使用Bean自身提供的static valueOf()方法来实现注入，再来看一下示例：

从Product派生一个子类Book：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 package yjmyzz.service.model;

2

3 import java.util.Calendar;

4 import java.util.Date;

5

6 import javax.xml.bind.annotation.XmlElement;

7 import javax.xml.bind.annotation.XmlRootElement;

8

9 @XmlRootElement(name = "book")

10 public class Book extends Product {

11

12 private String isbn;

13

14 private String author;

15

16 private Date publishDate;

17

18 public Book() {

19 System.out.println("construct: Book() is called!");

20 }

21

22 public static Book valueOf(String isbn) {

23 System.out.println("Book: valueOf(String isbn) is called!");

24

25 Book book = new Book();

26 book.setISBN(isbn);

27 book.setProductName("Java编程思想（第4版）");

28

29 book.setAuthor("[美]埃克尔");

30

31 Calendar calendar = Calendar.getInstance();

32 calendar.set(2007, 6, 1);

33 book.setPublishDate(calendar.getTime());

34 return book;

35 }

36

37 @XmlElement

38 public String getISBN() {

39 return isbn;

40 }

41

42 public void setISBN(String isbn) {

43 this.isbn = isbn;

44 }

45

46 @XmlElement

47 public String getAuthor() {

48 return author;

49 }

50

51 public void setAuthor(String author) {

52 this.author = author;

53 }

54

55 @XmlElement

56 public Date getPublishDate() {

57 return publishDate;

58 }

59

60 public void setPublishDate(Date publishDate) {

61 this.publishDate = publishDate;

62 }

63

64 }

[复制代码](javascript:void(0);)

再定义一个新方法：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @GET

2 @Path("/xml/book/{ISBN}")

3 @Produces(MediaType.APPLICATION\_XML)

4 public Book getBookXML(@PathParam("ISBN") Book book) {

5 return getBookFromServer(book);

6 }

[复制代码](javascript:void(0);)

里面有一个辅助方法，代码如下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

1 private Book getBookFromServer(Book book) {

2 book.setProductName(book.getProductName() + " from server!");

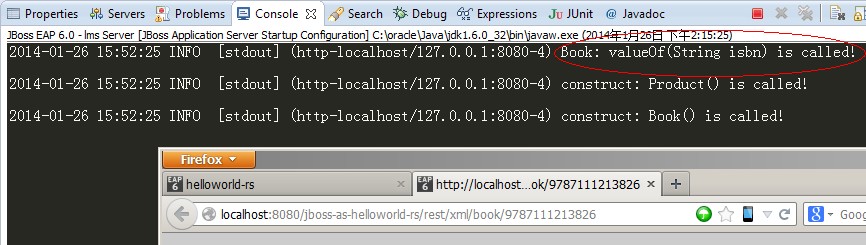
3 return book;

4 }

运行效果如下:



eclipse的控制台信息如下：



**五、POST方式的Bean实例参数注入**

前面提到的都是GET方式的web服务方法，GET方式能传输的数据大小有限，对于较大的参数数据，直接POST的场景更常见，下面是一个示例：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 @POST

2 @Path("/json/add-book")

3 @Produces(MediaType.APPLICATION\_JSON)

4 @Consumes(MediaType.APPLICATION\_JSON)

5 public Book addBookJSON(Book book) {

6 return getBookFromServer(book);

7 }

8

9

10 @POST

11 @Path("/xml/add-book")

12 @Produces(MediaType.APPLICATION\_XML)

13 @Consumes(MediaType.APPLICATION\_XML)

14 public Book addBookXML(Book book) {

15 return getBookFromServer(book);

16 }

[复制代码](javascript:void(0);)

解释一下：

又遇到了一个新的注解：@Consumes(MediaType.APPLICATION\_JSON) 它表示传入的Book实例是json格式。

我们用jQuery以Ajax方式调用这个服务演示一下：

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 function testJsonAddBook(){

2 $(document).ready(function(){

3 $.ajax({

4 type:"POST",

5 url:"rest/json/add-book",

6 dataType: "json",

7 contentType: 'application/json',

8 data:JSON.stringify({"productName":"Java编程思想（第4版）","isbn":"9787111213826","author":"[美]埃克尔","publishDate":1183270161125}),

9 success: function(data, textStatus, jqXHR){

10 alert("以下是从服务器返回的Book对象:\n\n" + JSON.stringify(data));

11 },

12 error: function(jqXHR, textStatus, errorThrown){

13 alert('error: ' + textStatus);

14 }

15 });

16 })

17 }

18

19

20 function testXmlAddBook(){

21 $(document).ready(function(){

22 $.ajax({

23 type:"POST",

24 url:"rest/xml/add-book",

25 dataType: "xml",

26 contentType: 'application/xml',

27 data:"<book><productName>Java编程思想（第4版） from server!</productName><author>[美]埃克尔</author><ISBN>9787111213826</ISBN><publishDate>2007-07-01T13:33:30.782+08:00</publishDate></book>",

28 success: function(data, textStatus, jqXHR){

29 alert("以下是从服务器返回的Book对象:\n\n" + data.documentElement.outerHTML);

30 },

31 error: function(jqXHR, textStatus, errorThrown){

32 alert('error: ' + textStatus);

33 }

34 });

35 })

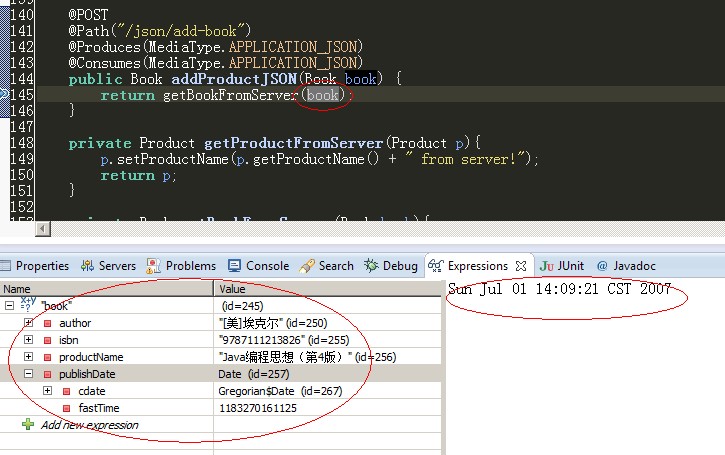
36 }

[复制代码](javascript:void(0);)

运行效果如下：  




如果断点调试ajax，可以看到传过来的json串已经映射成Bean实例了



示例源代码下载：[helloworld-rs.zip](http://files.cnblogs.com/yjmyzz/helloworld-rs.zip)