JavaSE6WebService之旅 PDF转换可能丢失图片或格式，建议阅读原文
https://www.100test.com/kao_ti2020/145/2021_2022_JavaSE6Web_c104_145029.htm 在过去的几个月里，Sun公司已经为Java平台的第6个版本（Java Platform Standard Edition 6）提供了二进制安装文件，Java文档和Java源代码，就是知名的“ 野马 ”。而且这飞驰的野马要拉住你这车并不算晚。 一个显而易见的问题是，“ 为什么我应该关心？ ”对于这些质疑者，Java SE 6提升的性能包括扩平台性，从开放的程序管理到Java编译器，到系统底层和屏幕组件，到在你的源代码中混合脚本语言（支持JavaScript），到Swing的整洁的外观，到XML数字签名，到智能卡I/O API，到JMX监听线程的升级，为Web服务的服务提供者使用Annotations注释和更加简单的客户端管理在我们要关心的时候，这些名字仅仅是这个平台中的一小部分新功能。（java.net站点提供了关于J2SE 6平台中所有的新功能。）这篇文章中，我们仅仅关注Java SE 6中对于Web Services规范的升级以及JAX-WS（Java API for XML Web Services）2.0规范，这些升级使得我们Web Services的创建和调用变得更加容易。使用这些新功能，我们可以仅仅使用简单的Annotations注释从一个Java类创建Web Services；随后，我们在调用这个服务的时候使用JAX-WS2.0.我们同样可以给这个服务添加一个管理者用来截取这个服务调用而且将截取的SOAP消息传输到System.out从控制台打印出来。 事实上，这些功能曾在JSR181（Java Specification Request 181）和JSR224 （JAX-WS）规范中已经被认可了，只是Java的正式版本拥有这些功能使它更加主流；我们同样期待Java IDE开发平台可以对这些功能

进行良好的支持。 JavaEE5规范中，允许基于标准规范让应用程序开发人员为Web Services提供服务端平台，JSR181和JSR224就是JavaEE5规范中的一部分。理论上讲，若在不改变源代码使用支持这些相同规范的应用程序服务器，在Java SE 6环境中大规模地发布工程这些功能将会破坏早期的Web Services应用程序，这个最初是不希望的。我们的“野马”Web Service：服务端和客户端在我们稳定这匹野马之前，我们先下载这篇文中重提及的zip文件，它主要包括下边四个文件：mustangws.zip包括了这篇文章中提及的Web Service服务端应用程序源代码，构建文件和脚本文件。mustangwsclient.zip包括了客户端的源代码，构建文件和脚本文件。mustangws.jar包含了已经编译过的服务端应用程序。mustangwsclient.jar包含了已经编译过的客户端程序。注意：你需要Java SE 6和Apache Ant这两个软件来运行这篇文章中的例子。 在你解压之后，你将会看到两个文件架，mustangws和mustangwsclient，这两个文件架内分别是Web Services的服务端和客户端。两个项目里面都有同样的src文件夹，里面包含了Java的源文件和Apache Ant软件需要的构建文件build.xml。这里还有附加的wsgenMustang.bat 和wsimportMustang.bat两个批处理文件，里面包含了生成Web Services存根的控制台命令，我们将在下边的章节中讲到。Apache Ant的build.xml文件对于两个应用程序而言都是放在mustangws 和mustangwsclient两个根目录中的。两个文件中有同样的ant任务init,compile,dist,clean和run。 我们Web Services的服务端根目录里面还有一个目录wsdl，里面放了通过wsgen生成的描述Web Service的WSDL（Web Services Description Language）文

件。Web Services Metadata Web Services Metadata详细说明了Annotations的使用，这个在JavaSE5中就被引入了。这些对你而言可能是新概念，Annotations就是能够被源代码使用的注释；一般使用在类定义，函数，函数参数或者函数返回值中。这些Annotations实际上就是对编译器，Java虚拟机和应用程序服务器的一种指示或者暗示，告诉它们应该如何处理这些有注释的代码。Annotations语句使用一般是“ @ ”开始后面跟着Annotation文本内容。比如，@Deprecated注释用于在使用函数的过程中将函数定义时生成的警告去掉，这一点使得Annotations变成了第一个Java类中的构造。 当Web Services Metadata Annotations用于Java源代码中的类定义和函数定义的时，配置Web Services将会相对简单。Web Services Metadata规范将会跟着“ 以Java开始的 ”开发模式，这些意味着你首先定义一个Java类和函数，然后就对他们使用一个Web Services Metadata Annotation。对于Java开发者来说，比起通过WSDL文件创建Web Service来说通过Java类使用Annotation是创建Web Services更加自然的方法。这些Annotations注释用来暗示Web Services的运行时引擎将Java类和它里面的函数解释成为Web Service和相应的Web Service操作。将这个引擎已经绑定在Java SE 6里面了，但是因为Web Services Metadata仍然是Java EE 5里面的一部分，所以你将看到众多厂家还是提供了支持Annotation的应用程序服务器。我们的“ 野马 ”Web服务端 我们不再嗦了，让我们通过“ 野马 ”Web Service的服务端的一部分来看看Web Services Metadata在实践中的使用。浏览mustangws/src/com/techyatra/hellows目录。你将会看到下边的文件：HelloServer Bootstrap TraceHandler HelloException

Person 打开HelloServer文件，里面包含了Web Services Metadata Annotations定义，将一个简单的Java类用作Web Service，下边是一个实现了Service的bean文件：package com.techyatra.hellows.@WebService(name="HelloServer",targetNamespace=http://mustangws.techyatra.com/, serviceName="HelloService")@SOAPBinding(style=SOAPBinding.Style.RPC)public class HelloServer{ @WebMethod(operationName="hello", action="urn:hello") public@WebResult(partName="result")String ping(@WebParam(partName="person",mode=Mode.IN,targetNamespace="http://mustangws.techyatra.com/") Person person) throws HelloException { if (person == null) { System.out.println("function: hello(null)... throwing exception"). throw new HelloException("0001", "Person is null"). } else { System.out.println("function: hello(person.getTitle() person.getName())"). return "Hello. " person.getTitle() person.getName() "!". } } } 在上边的代码中，@WebService注释使得类MustangServer实现了Web Service，而且@WebMethod标识类中ping方法作为WebService的操作，ping除开返回greeting的值就不会做其他的事情。 需要注意的是：不管你愿意不愿意，你都不能任意地将Annotation用于任何一个Java类和函数使其暴露为一个WebService和一个WebService操作。因为Java类若要成为一个实现了WebService的bean，它需要遵循下边这些原则：这个类必须是public类这些类不能是final的或者abstract这个类必须有一个公共的默认构造函数这个类绝对不能有finalize()方法若要成为一个实现了WebService的Bean这

个Java类必须遵循这些原则：这个类必须是public，它的参数、返回值、和异常在每个JAX RPC规范中都描述了Java转化成XML/WSDL映射文件的规则等等。参数和返回值可以是原始类型、数组等等；异常都可以继承Exception；请查阅Java API去看基于XML的远程调用可以知道更多的信息。 现在我们已经拥有了合理规范，让我们接近代码来看看WebService Annotation的作用。注意：这篇文章不是描述所有Annotation和所有的WebService Metadat规范里面的所有成员。如果要了解细节自己去查阅JSR 181规范。 这里有各种类型的Annotation。@WebService和@WebMethod是WSDL映射Annatotion。这些Annotation将描述Web Service的WSDL文档元素和Java源代码联系在一起。@SOAPBinding是一个绑定的annotation用来说明网络协议和格式。@WebService annotation的元素name,serviceName和targetNamespace成员用来描述wsdl:portType，wsdl:service，和targetNameSpace生成WebService中的WSDL文件。@SOAPBinding是一个用来描述SOAP格式和RPC的协议的绑定Annotation。@WebMethod Annotation的operationName成员描述了wsdl:operation，而且它的操作描述了WSDL文档中的SOAPAction头部。这是客户端必须要放入到SQAPHeader中的数值，SOAP 1.1中的一种约束。@WebParam Annotation的partName成员描述了WSDL文档中的wsdl:part。@WebResult Annotation的partName成员描述了wsdl:part用来返回WSDL文档的值。 HelloException类是一个含有属性description和错误代码的异常类，Person是一个只有lastname和firstname 属性的简单类。这些类都没有被任何Annotation注释过。 我们在HelloException中使用@WebFault

确实很诱人，可是并非如此！Annotation是被wsgen用来生成异常bean。更加有趣的是，这个不是JSR 181规范中的一部分，但是是JAX-RPC 1.1中的一部分。 在我们转移到wsgen之前，一个很好的功能需要我们注意的是我们如何重载一个函数使得它成为一个WebService的操作。我们通过不同的@WebMethod的operationName元素值来激活这个功能，这里的重载函数hello()和hello(person)有了不同的operationName元素各自命名为hello和quickHello。 现在为我们的服务应用程序运行Apache Ant的编译工作。利用wsgen通过这样的方式来生成build文件夹用来包含编译过的class文件和一个wsdl文件夹用来放置WSDL文档。wsgen 拥有Web Service Annotation的HelloServer 类无法成功发布因为它包含了额外的没有被注释为WebService的HelloException类或者没有绑定Java XML，如果HelloServer的函数没有抛出异常，我们就不能使用wsgen。尽管如此，在实际应用中，这种做法是不推崇的；因此使用wsgen的目的是为了创建一个能够使用WebService的类。wsgen是一个命令行功能用来生成合适的JAX-WS。它读取WebService的终端类文件，在我们的例子中就是com.techyatra.hellows.HelloServer，同时生成所有用于WebService发布的源代码文件和经过编译过的二进制类文件。它还随意生成WSDL和符合规范的HelloServer类WebService。wsgen从资源文件生成一个完整的操作列表是合法的。 为了方便，这篇文章的源代码mustangws文件夹中包含了一个wsgenHello.bat批处理文件，里面包含了我们的Server项目中所需要的wsgen命令行的操作。 运行mustangws中的mustangws/wsgenHello.bat文件产生HelloExceptionBean源文

件和编译过的class文件放入com.techyatra.hellows.jaxb文件夹中。这些类绑定了关于Java-XML的所有操作。 现在我们有了发布WebService的所有需要的条件，在我们发布这个服务之前，让我们再看一看将运行过程加入到WebService的调用队列。这个相当敏捷万一所有的通用过程都是调用这些服务所需要的过程，如此执行过程对所有的操作者来说都是合法的。操作者允许应用程序介绍自定义和核心商业需求域之外的特殊处理。这个操作者同样会执行安全检查，据使用统计，编码和解码数据，提供给其它模块开发者使用。 