第2讲 软件服务开发技术之-RPC

郭 勇

哈尔滨工业大学 计算机科学与技术学院

主要内容

1.RPC是什么

2.RPC的发展

3.XML-RPC

4.JSON-RPC

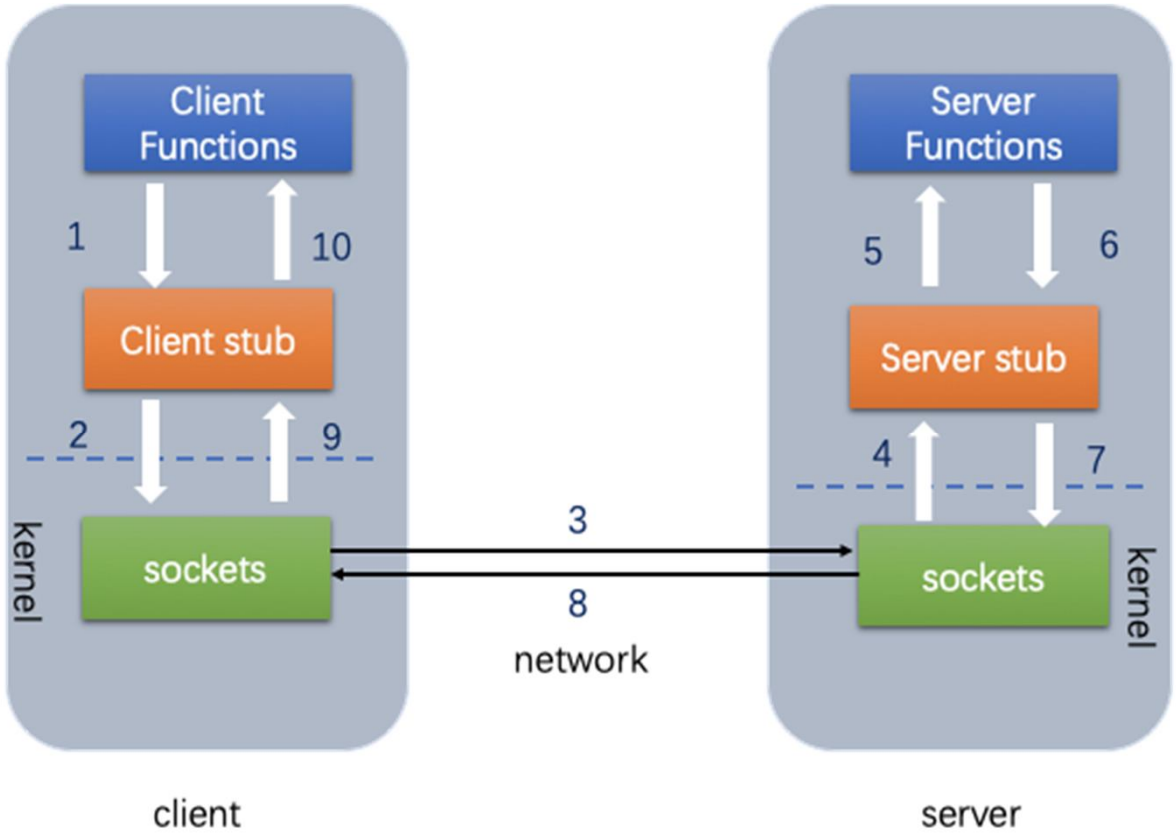
1.1 什么是RPC

RPC(远程过程调用)是什么

- 简单的说，RPC就是从一台机器(客户端)上通过参数传递的方式调用另一台机器(服务器)上的一个函数或方法(可以统称为服务)并得到返回的结果。
- RPC 会隐藏底层的通讯细节(不需要直接处理Socket通讯或Http通讯)
- RPC 是一个请求响应模型。客户端发起请求，服务器返回响应(类似于Http的工作方式)
- RPC 在使用形式上像调用本地函数(或方法)一样去调用远程的函数(或方法)。

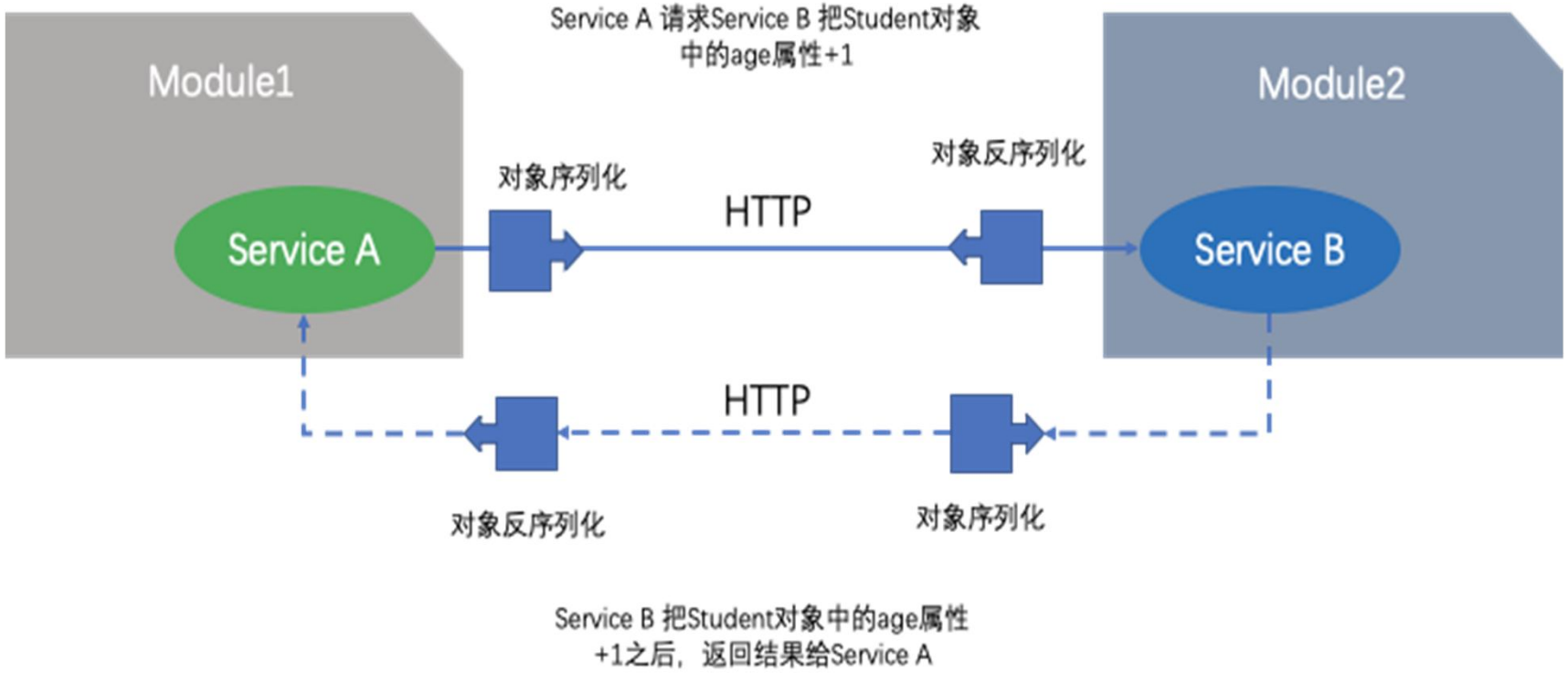
1.1 什么是RPC

RPC过程示例



1.1 什么是RPC

面向服务中RPC示例



1.2 RPC 分类

- 从通信协议的层面大致可以分为
  - 基于HTTP协议的(例如基于文本的SOAP(XML)、Rest(JSON)，基于二进制Hessian(Binary))
  - 基于TCP/IP协议的(通常会借助Mina、Netty等高性能网络框架)
    - Mina(Apache Mina Server): 是一个网络通信应用框架，为开发高性能和高可用性的网络应用程序提供了非常便利的框架。
    - Mina特点：异步的NIO框架,将UDP当成“面向连接”的协议，底层依赖的主要是Java NIO库，上层提供的是基于事件的异步接口
    - Netty:是由JBOSS提供的一个java开源框架，是一个异步的事件驱动网络应用程序框架，
    - Netty特点:基于BIO和NIO的UDP传输，对Java NIO库进行了封装。易于开发的同时还保证了其应用的性能，稳定性和伸缩性。

1.2 RPC 分类

- 从不同的开发语言和平台层面分为
  - 单种语言或平台特定支持的通信技术(例如Java平台的RMI、.NET平台Remoting)
  - 支持跨平台通信的技术(例如HTTP Rest、Thrift等)
- 从调用过程来看，分为：
  - 远程数据共享(例如：共享远程文件，共享数据库等实现不同系统通信)
  - 消息队列
  - RPC(远程过程调用)

1.3主流的Web服务实现方案

- 目前几种主流的Web服务实现方案
  - XML-RPC(Remote Procedure Call)：远程过程调用协议,通过XML将调用函数封装，并使用HTTP协议作为传送机制。
  - JSON-RPC(Remote Procedure Call)：远程过程调用协议,通过JSON将调用函数封装，并使用HTTP协议作为传送机制。
  - SOAP(Simple Object Access Protocol)：简单对象访问协议,是交换数据的一种协议规范,是一种轻量的、简单的、基于XML（标准通用标记语言下的一个子集）的协议，它被设计成在WEB上交换结构化的和固化的信息。
  - REST(Representational State Transfer)：一种软件架构风格，而不是标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等。

1.3主流的Web服务实现方案

- 方案的简单比较
  - RPC出现较早，实现方式多样，但有被SOAP取代的趋势
  - 成熟度上：SOAP在成熟度上优于REST
  - 效率和易用性上：REST更胜一筹
  - 安全性上：SOAP安全性高于REST，因为REST更关注的是效率和性能问题
  - 总体上，因为REST模式的Web服务与复杂的SOAP和XML-RPC对比来讲明显的更加简洁，越来越多的web服务开始采用REST风格设计和实现。

主要内容

1.RPC是什么

2 RPC的发展

3.XML-RPC

4.JSON-RPC

2 RPC的发展

2.1 常见的RPC技术和框架

2.2 第1代 RPC

2.3 第2代 RPC

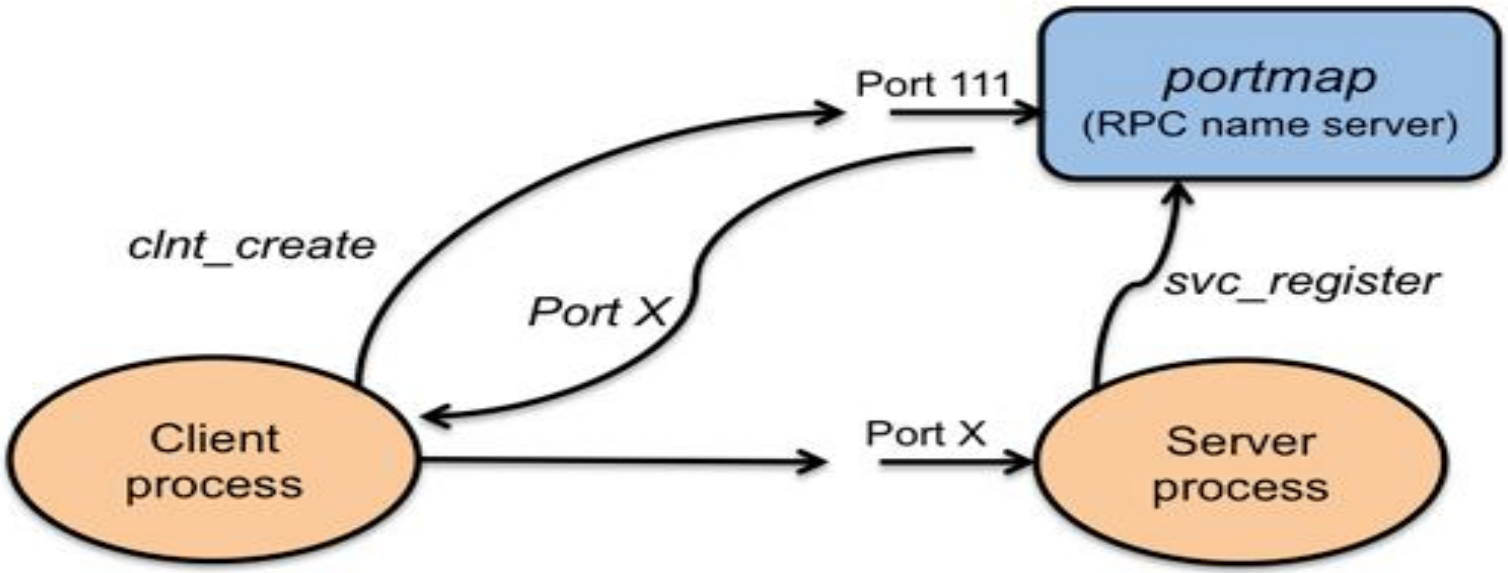
2.4 第3代 RPC

2.1常见的RPC技术和框架

- 互联网时代常见的RPC技术和框架
  - 应用级的服务框架：Dubbo/Dubbox(阿里巴巴)、ZeroC ICE(ZeroC公司, Internet Communications Engine)、GRpc(google的一款语言中立、平台中立、开源的远程过程调用(RPC)系统)、Spring Boot/Spring Cloud(由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程)
  - 基础通信框架：Protocol Buffers(google 的一种数据交换的格式，它独立于语言，独立于平台)、Thrift(由Facebook为“大规模跨语言服务开发”而开发的)
  - 远程通信协议：RMI、Socket、SOAP(HTTP XML)、REST(HTTP JSON)

2.2 第1代 RPC

- 第1代 RPC
  - ONC RPC(以前称为 Sun RPC),在1980年中期 Sun 计算机提供 RPC.
  - 程序员需要写一个客户端程序(client.c),服务器程序(server.c)和 RPC 接口定义(data.x)。



2.3 第2代 RPC

- 第2代 RPC
  - 支持对象(Object)
  - 微软 DCOM(COM+)、CORBA (Common Object Request Broker Architecture通用对象请求代理体系结构) 、Java RMI
  - 第2代 RPC－Java RMI的实现
    - 设计目标:
      - 能够适应语言、集成到语言、易于使用;
      - 支持无缝的远程调用对象;
      - 支持服务器到 applet 的回调;
      - 保障 Java 对象的安全环境;
      - 支持分布式垃圾回收;

2.3 第2代 RPC

❖ 第2代 RPC－Java RMI的实现（续）

- ■ 分布式对象模型与本地 Java 对象模型相似点
  - ● 引用一个对象可以作为参数传递或作为返回的结果；
  - ● 远程对象可以投到任何使用 Java 语法实现的远程接口的集合上；
  - ● 内置 Java instanceof 操作符可以用来测试远程对象是否支持远程接口。
- ■ 不同点在于
  - ● 远程对象的类是与远程接口进行交互，而不是与这些接口的实现类交互；
  - ● 非远程对像(Non-remote object) 作为远程方法调用的参数，是通过复制，而不是通过引用方式进行调用的；也就是使用Java 对象序列化机制将该对象序列化。
  - ● 从远程方法调用返回非远程对象时，将在调用的虚拟机中创建新对象。
  - ● 远程对象是通过引用来传递，而不是复制实际的远程实现；
  - ● 客户端必须处理额外的异常。

2.4 第3代 RPC

- 第3代 RPC －XML RPC及JSON RPC
  - 由于互联网的兴起，Web 浏览器成为占主导地位的用于访问信息的模型。
  - 现在的应用设计的首要任务大多数是提供用户通过浏览器来访问，而不是编程访问或操作数据
  - 传统 RPC 解决方案可以工作在互联网上，但问题是，他们通常严重依赖于动态端口分配，往往要进行额外的防火墙配置。

2.4 第3代 RPC

- 第3代 RPC – XML RPC
  - XML-RPC（RPCXML Remote Procedure Call）是通过HTTP传输XML来实现远程过程调用的RPC
  - 基于HTTP、并且使用XML文本的方式传输命令和数据，所以兼容性更好，能够跨域不同的操作系统、不同的编程语言进行远程过程调用
  - 在兼容性好的同时速度也会慢下来。

2.4 第3代 RPC

- 第3代 RPC －JSON RPC
  - 是一个无状态且轻量级的远程过程调用（RPC）传送协议，其传递内容采用JSON 格式。
  - JSON-RPC 直接在内容中定义了欲调用的函数名称（如{“method”：“getUser”}）。
  - 它能够运行在基于 Socket、HTTP 等诸多不同消息传输环境的同一进程中。其使用 JSON（RFC 4627）作为数据格式。

主要内容

1.RPC是什么

2.RPC的发展

3.XML-RPC

4.JSON-RPC

3.1 XML-RPC的组成及原理

- XML-RPC的组成
  - 一般一个RPC系统包括两个部分：RPC Client和RPC Server
  - Client向Server发送一个请求体为XML的HTTP POST请求，被调用的方法在Server端执行后将执行结果以XML格式返回
  - 与平常的方法调用所不同就是接口“作用域”更大，并且多了一层数据的包装和转换

3.1 XML-RPC的组成及原理

- 工作原理
  - rpcclient的工作原理
    - rpcclient根据URL找到rpcserver
    - 构造命令包
    - 调用rpcserver上的某个服务的某个方法
    - 接收到rpcserver的返回
    - 解析响应包，拿出调用的返回结果。
  - rpcserver的工作原理
    - 启动一个webserver(在使用内置的webserver的情况下)
    - 注册每个能提供的服务，每个服务对应一个Handler
    - 进入服务监听状态。

3.2 XML-RPC的实现

- XML-RPC实现
  - Apache XML-RPC是XML-RPC的一个Java实现，其底层是基于Helma的
  - Helma是一个用来开发快速、稳定的Web应用程序的开源框架
  - XML-RPC Server端
    - 有两种启动XML-RPC的方式
    - 一种是集成在Web Servlet环境中，一般应用在Web环境，要有Tomcat 之类的web服务器
    - 一种是启动独立的内嵌Web Server，内嵌的Web Server可以被嵌入到任意的Java应用中(见例子中的WebServer 类)

3.2 XML-RPC的实现

- XML-RPC Server端(嵌入式 WebServer 实现)

（1）.pom.xml中增加需要的库

```
<dependency>
    <groupId>org.apache.xmlrpc</groupId>
     <artifactId>xmlrpc-server</artifactId>
    <version>3.1.3</version>
</dependency>
```

（2）.创建处理器映射文件MyHandlers.properties，并加入

MyCalculator=webserver.MyCalculator

其中：键 MyCalculator为处理器名，值为类名

3.2 XML-RPC的实现

（3）.创建RPC Server 主程序

```
package webserver;
import org.apache.xmlrpc.server.PropertyHandlerMapping;
import org.apache.xmlrpc.server.XmlRpcServer;
import org.apache.xmlrpc.server.XmlRpcServerConfigImpl;
import org.apache.xmlrpc.webserver.WebServer;
public class Server {
    private static final int port = 9090;
    public static void main(String[] args) throws Exception {
        WebServer webServer = new WebServer(port);
        XmlRpcServer xmlRpcServer = webServer.getXmlRpcServer();
        PropertyHandlerMapping phm = new PropertyHandlerMapping();
        phm.load(Thread.currentThread().getContextClassLoader(), "MyHandlers.properties");
        xmlRpcServer.setHandlerMapping(phm);
        XmlRpcServerConfigImpl serverConfig = (XmlRpcServerConfigImpl)
xmlRpcServer.getConfig();
        serverConfig.setEnabledForExtensions(true);
        serverConfig.setContentLengthOptional(false); //设置是否可以省略“内容长度”标题。
XML-RPC规范要求存在这样的头
        webServer.start();
    }
}
//api文档详见http://ws.apache.org/xmlrpc/apidocs/index-all.html
```

3.2 XML-RPC的实现

（4）.RPC Server 功能函数(即服务)

```
package webserver;
public class MyCalculator {
    public int myAdd(int x, int y) {
        return x + y;
    }
}
```

服务端完成

3.2 XML-RPC的实现

- XML-RPC Client端
  - 客户端有两种调用方式:同步调用和异步调用
  - 如果某个被调用的远程过程执行的很慢，就可能会导致我们的程序处于假死状态，又或者我们只是调用它一下，对其返回结果并不是很关心，这个时候比较适合使用异步调用。

（1）.pom.xml中增加需要的库

```
<dependency>
    <groupId>org.apache.xmlrpc</groupId>
    <artifactId>xmlrpc-client</artifactId>
    <version>3.1.3</version>
  </dependency>
```

3.2 XML-RPC的实现

（2）.客户端程序

```
import java.net.MalformedURLException;
import java.net.URL;
import org.apache.xmlrpc.XmlRpcException;
import org.apache.xmlrpc.client.XmlRpcClient;
import org.apache.xmlrpc.client.XmlRpcClientConfigImpl;
class MyTest {
        public static void main(String[] args) throws IOException {
            logger.info("before xml rpc");
            try {
                XmlRpcClientConfigImpl config = new XmlRpcClientConfigImpl();
                config.setServerURL(new URL("http://127.0.0.1:9090"));
                XmlRpcClient client = new XmlRpcClient();
                client.setConfig(config);
                Object[] params = new Object[]{new Integer(31), new Integer(9)};
                Integer result = (Integer) client.execute("MyCalculator.myAdd", params);
                System.out.println(result);
            } catch (XmlRpcException e) {
                e.printStackTrace();
            } catch (MalformedURLException e) {
                e.printStackTrace();
            }
        }
    }
```

3.2 XML-RPC的实现

- 调用add时客户端请求内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
  <methodName> MyCalculator.myAdd </methodName>
  <params>
    <param>
      <value><int>31</int></value>
    </param>
    <param>
      <value><int>9</int></value>
    </param>
  </params>
</methodCall>
```

3.2 XML-RPC的实现

- 服务器响应的内容如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
  <params>
    <param>
      <value><int>41</int></value>
    </param>
  </params>
</methodResponse>
```

3.2 XML-RPC的实现

3. XML-RPC数据类型

XML Tag Name	Java Type	Describe
<i4> 或 <int>	Integer/int	4字节带符号整数值
<boolean>	Boolean	0==false, 1==true
<string>	String	字符串
<double>	Double	双精度带符号浮点值
<dateTime.iso8601>	java.util.Date	日期/时间
<base64>	byte[]	base64编码的二进制数据
<struct>	java.util.Map	<K,V>对，key必须是string类型，value可以是任意其它类型，struct是可以递归使用的
<array>	java.lang.Object[]| java.util.List	对象数组

主要内容

1.RPC是什么

2.RPC的发展

3.XML-RPC

4.JSON-RPC

4.1 什么是JSON-RPC

- 什么是JSON-RPC
  - 是一个无状态且轻量级的远程过程调用（RPC）传送协议
  - JSON-RPC协议中的客户端一般是为了向远程系统请求执行某个方法向实现了JSON-RPC协议的服务端发送请求
  - 多个输入参数能够通过数组或者对象传递到远程方法，这个远程方法也能返回多个输出数据
  - JSON-RPC序列化采用JSON的形式。（RESTful通常采用http+JSON实现）
  - json rpc 和xmlrpc相比具有很多优点。
    - 首先xmlrpc是以xml作为消息格式，xml具有体积大，格式复杂，传输占用带宽。
    - 程序对xml的解析也比较复杂，并且耗费较多服务器资源。
    - json相比xml体积小巧，并且解析相对容易很多。

4.1 什么是JSON-RPC

- 如何使用JSON-RPC
  - 所有的传输都是单个对象，用JSON格式进行序列化。
  - 请求包含三个特定属性：
    - method方法，是等待调用的远程方法名，字符串类型
    - params参数，对象类型或者是数组，向远程方法传递的多个参数值
    - id 任意类型值，用于和最后的响应进行匹配，也就是这里设定多少，后面响应里这个值也设定为相同的
  - 接收者必须能够给出所有请求的正确的响应
  - 响应也有三个属性：
    - result结果，是方法的返回值，如果方法执行时出现了错误，那么这个值必须为空
    - error错误，当出现错误时，返回一个特定的错误编码，没有错误就为空值
    - id就是请求带的那个id值，用于匹配

4.1 什么是JSON-RPC

- JsonRPC简单说明
  - 调用的Json格式，向服务端传输数据格式如下：
    - {"method": "方法名", "params": ["参数数组"], "id": 方法ID}
    - 说明：
      - 第一个参数： 是方法的名值对
      - 第二个参数： 是参数数组
      - 第三个参数： 是方法ID（可以随意填）
    - 举例： {"method": "doSomething", "params": [], "id": 1234} doSomething 是远程对象的方法， [] 表示参数为空
  - 输出的Json格式
    - { "jsonrpc": "2.0", "id": "1234", "result": null}

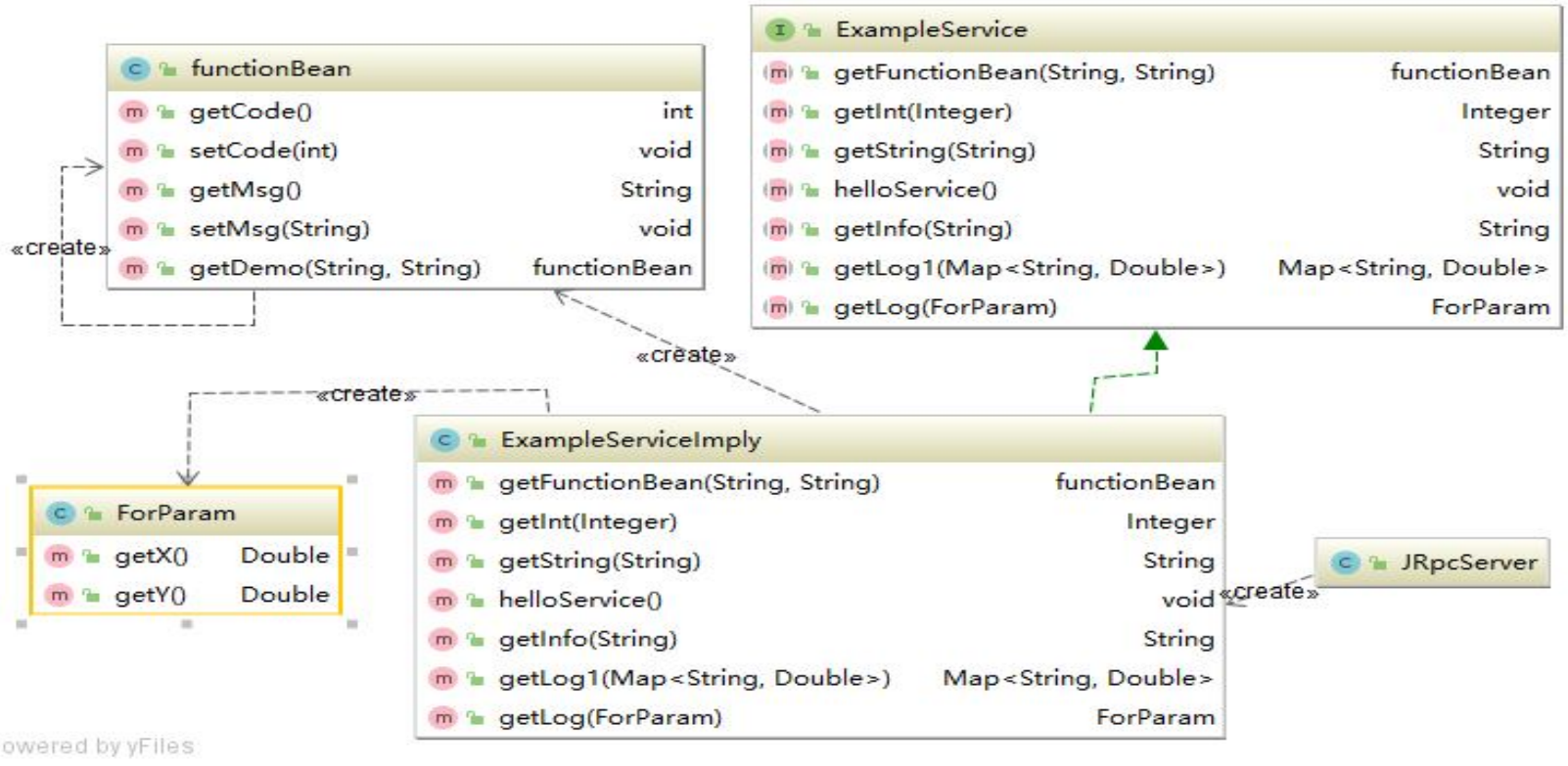
4.2 JSON-RPC实现

- 服务端开发
  - json-rpc是一个轻量级的跨语言远程调用协议，实现及使用简单。仅需几十行代码，即可实现一个远程调用的客户端，方便扩展客户端的实现。服务器端有php、java、python、ruby、.net等语言实现。
  - 主要的依赖包

```
<dependency>
    <groupId>com.github.briandilley.jsonrpc4j</groupId>
    <artifactId>jsonrpc4j</artifactId>
    <version>1.0</version>
</dependency>
```

4.2 JSON-RPC实现

- 服务端开发举例
  - 一个web程序，主要设计了如下几个类
  - 其中RpcServer是一个Serverlet 继承了HttpServlet



4.2 JSON-RPC实现

❖ 服务接口

```
package gy.jsonrpc.server;
import java.util.Map;

public interface ExampleService {
    public functionBean getFunctionBean(String code, String msg);
    public Integer getInt(Integer code);
    public String getString(String msg);
    public void helloService();
    public String getInfo(String sType);

     /*计算以base 为底的value的对数*/
    public Map<String,Double> getLog1(Map<String, Double>jsonParam);
    public ForParam getLog(ForParam param);
}
```

4.2 JSON-RPC实现

❖ 服务接口的实现

```
package gy.jsonrpc.server;
import java.util.HashMap;
import java.util.Map;
public class ExampleServiceImply implements ExampleService {
    private String code;
    private String msg;
    @Override
    public ForParam getLog(ForParam param) {
        ForParam paramres=new ForParam();
        paramres.res=Math.log(param.getX()) / Math.log(param.getY());
        return paramres;
    }
    @Override
    public functionBean getFunctionBean(String code, String msg) {
         functionBean bean1 = new functionBean();
        bean1.setCode(Integer.parseInt(code));
        bean1.setMsg(msg);
        return bean1;
    }

    .......
```

4.2 JSON-RPC实现

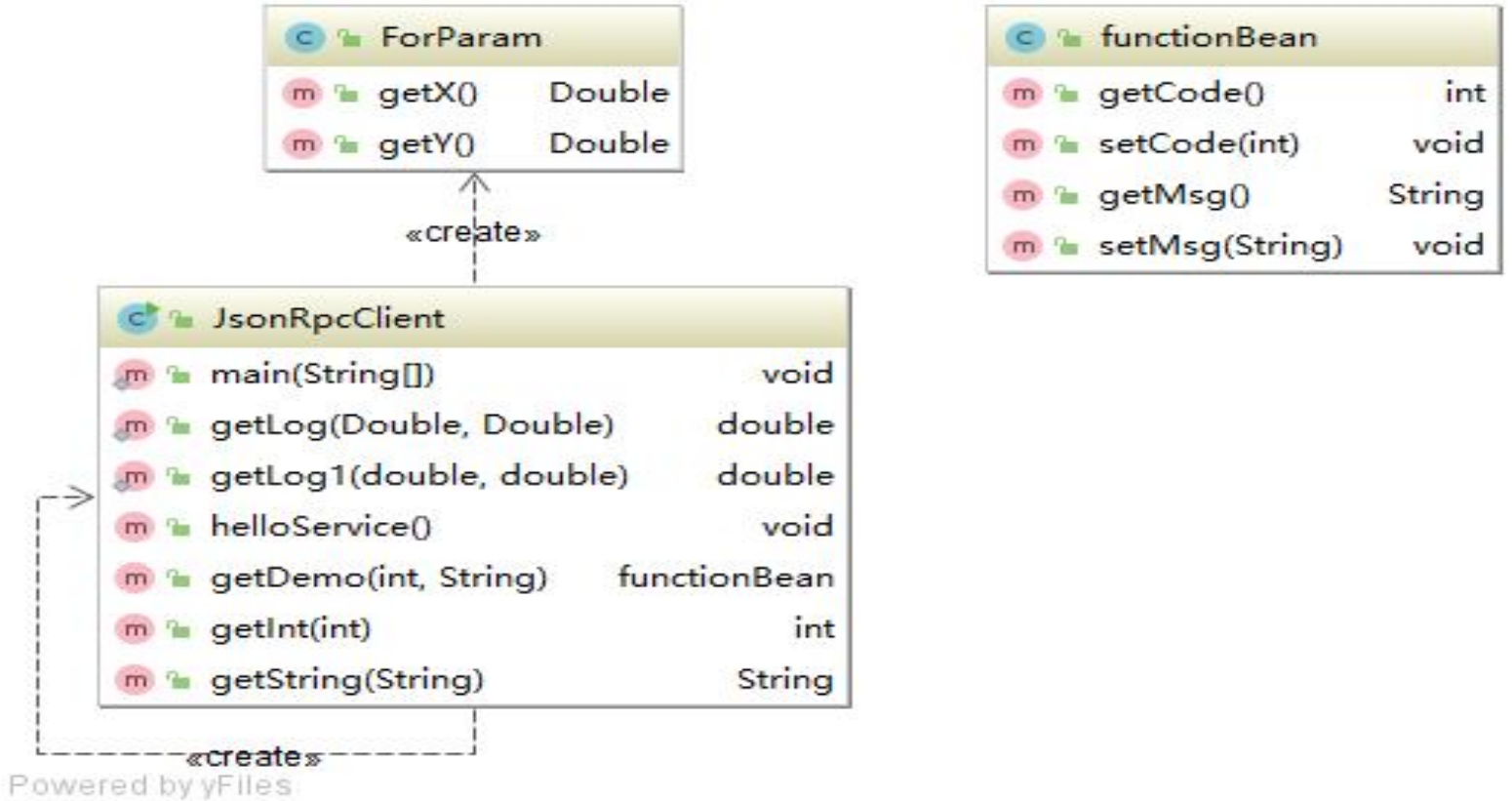
- 服务器web.xml 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         metadata-complete="false" version="2.5">
    <display-name>Archetype Created Web Application</display-name>
    <servlet>
        <servlet-name>JRpcServer</servlet-name>
        <servlet-class>gy.jsonrpc.server.JRpcServer</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>JRpcServer</servlet-name>
        <url-pattern>/rpc</url-pattern>
    </servlet-mapping>
</web-app>
```

4.2 JSON-RPC实现

2. 客户端

- 类图



4.2 JSON-RPC实现

❖ 客户端主要代码

```
public class JsonRpcClient
{
    static JsonRpcHttpClient client;
    public JsonRpcClient() { }
    public static void main(String[] args) throws Throwable {
        // 实例化请求地址，注意服务端web.xml中地址的配置
        try {
            client = new JsonRpcHttpClient(new URL( spec: "http://127.0.0.1:8080/jsonrpc_server/rpc"));
            // 请求头中添加的信息
            Map<String, String> headers = new HashMap<~>();
            headers.put("UserKey", "rpckey");
            // 添加到请求头中去
            client.setHeaders(headers);
            JsonRpcClient test = new JsonRpcClient();
            test.helloService();
            functionBean demo = test.getDemo( code: 1,  msg: "您好！");
            String msg = demo.getMsg();
            System.out.println("test.getString:"+msg);
            System.out.println("The result of log (8,2) is "+getLog(Double.valueOf(8),Double.valueOf(2)));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    .....
```

4.2 JSON-RPC实现

❖ 在客户端类方法中调用服务

```
static public double getLog(Double value, Double base) throws Throwable {
    ForParam param[]=new ForParam[1];
    param[0]=new ForParam(value, base);
    ForParam res=client.invoke( methodName: "getLog", param, ForParam.class);
   return res.res;
}
public void helloService() throws Throwable {
    client.invoke( methodName: "helloService",  argument: null);
}

public functionBean getDemo(int code, String msg) throws Throwable {
    String[] params = new String[] { String.valueOf(code), msg };
    functionBean demo = null;
    demo = client.invoke( methodName: "getFunctionBean", params, functionBean.class);
    return demo;
}
```

4.2 JSON-RPC实现

- 特别要注意客户端URL不能写错，否则无法连到服务器

```
client = new JsonRpcHttpClient(new
URL("http://127.0.0.1:8080/jsonrpc_server/rpc"));
```

- 其中“*jsonrpc_server*”由运行配置中的“*Deployment*”标签下的“*Application context*”决定，“*rpc*”由web.xml 中的“url-pattern”决定。

4.2 JSON-RPC实现

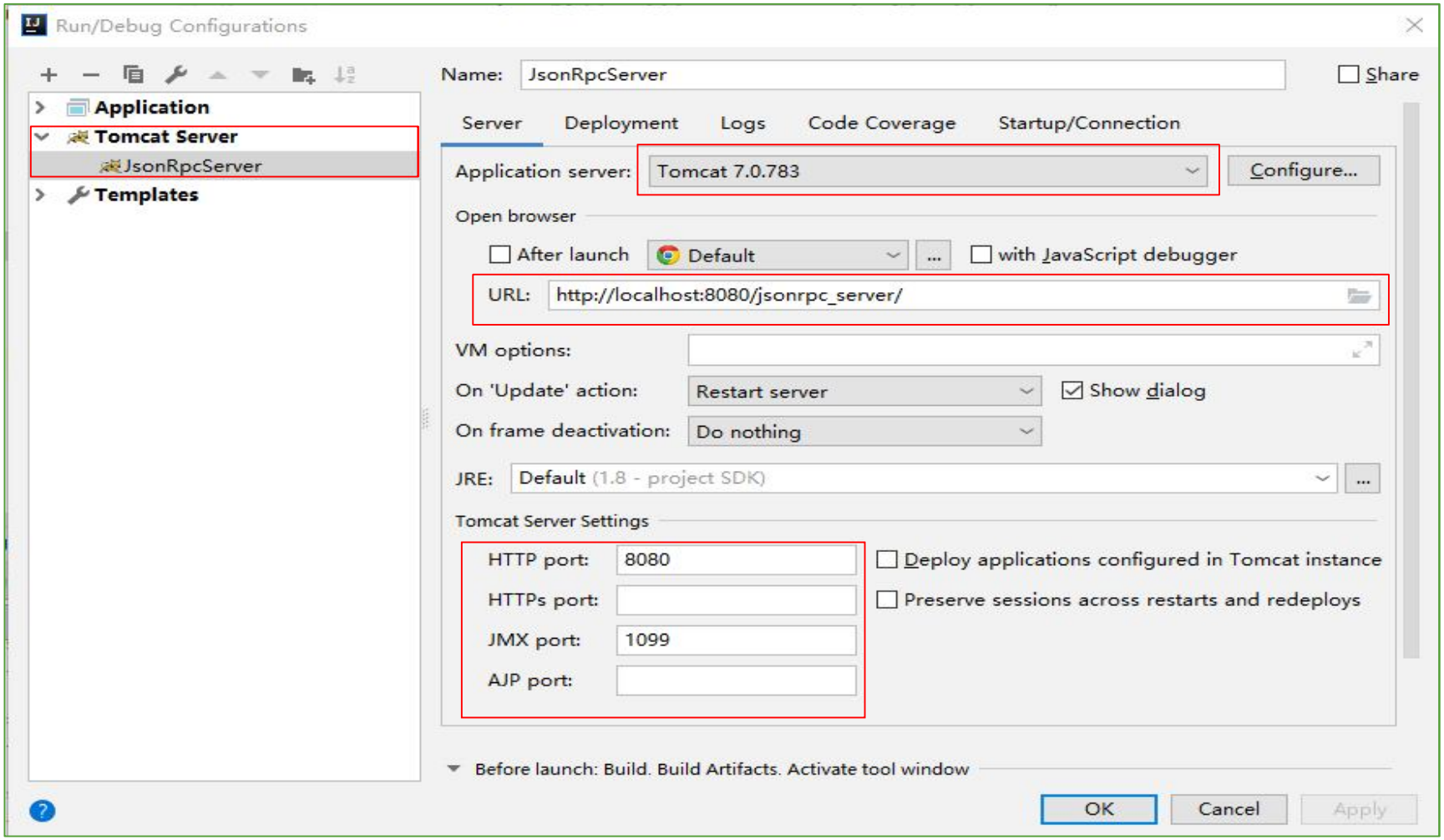
- 用于参数传送的类：对于多个不同类型参数的传递可以定义一个类，简单的同类型参数，可直接用数组

```
package gy.jsonrpc.client;

public class ForParam {
    public double res;
    private Double x;
        private Double y;
        public ForParam() {}
        public ForParam(Double x,Double y) {
            super();
            this.x = x;
            this.y = y;
            res=0;
        }
        public Double getX() { return x; }
        public Double getY() { return y; }
    }
```

4.2 JSON-RPC实现

- 配置Web服务器



4.2 JSON-RPC实现

- 启动JsonRpcServer

```
信息: Server startup in 69 ms
九月 09, 2019 1:12:58 下午 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deploying web application directory C:\Program Files\apache-tomcat-7.0.78\webapps\manager
九月 09, 2019 1:12:59 下午 org.apache.catalina.startup.HostConfig deployDirectory
信息: Deployment of web application directory C:\Program Files\apache-tomcat-7.0.78\webapps\manager has finished in 381 ms
```

- 运行客户端

```
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/Users/ThinkPad/.m2/repository/org/slf4j/slf4j-nop/1.7.2/
SLF4J: Found binding in [jar:file:/C:/Users/ThinkPad/.m2/repository/org/slf4j/slf4j-jdk14/1.5.
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.helpers.NOPLoggerFactory]
test.getString:您好!
The result of log (8,2) is 3.0

Process finished with exit code 0
```

谢谢！