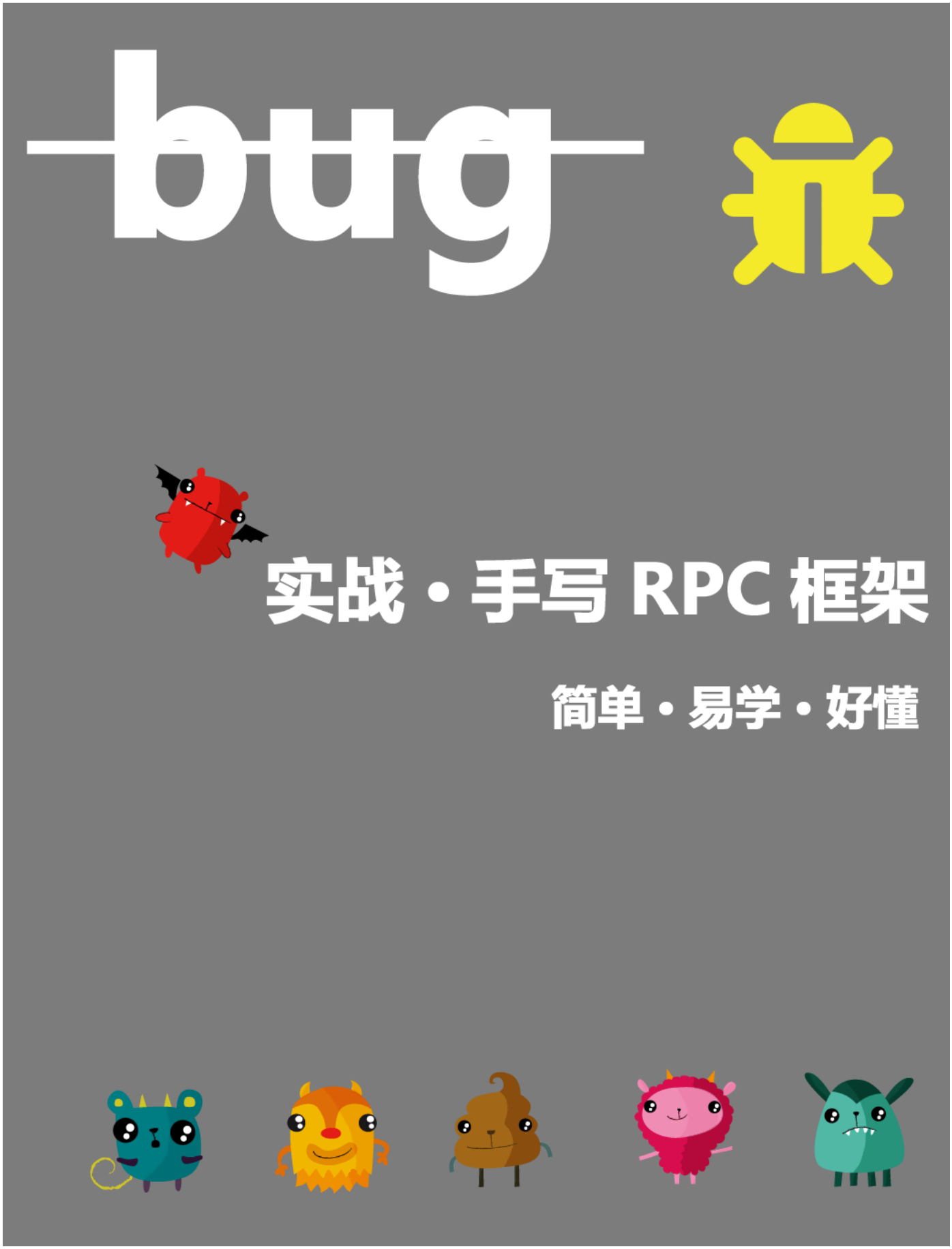
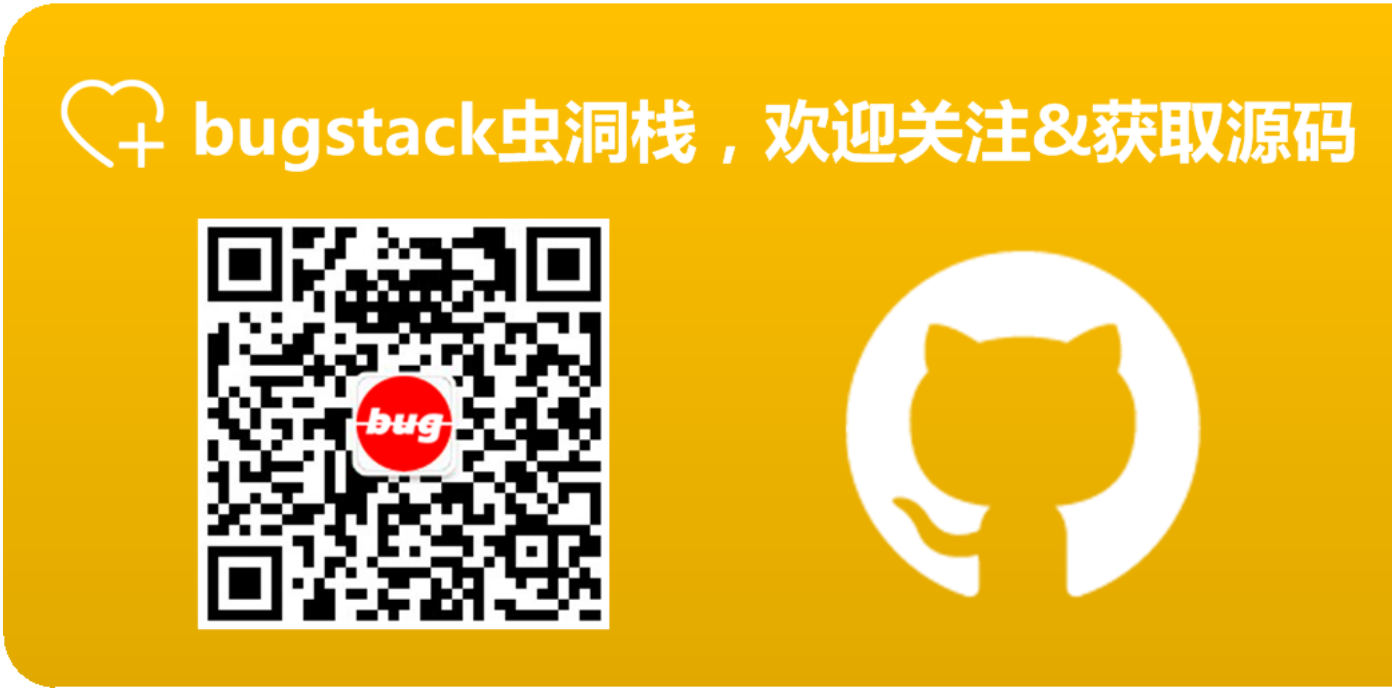


微信公众号：bugstack虫洞栈，欢迎您的关注！不平凡的岁月终究来自你每日不停歇的刻苦，犹如；承遇朝霞，年少正恰，整装戎马，刻印风华。

本公众号会每天定时推送科技资料；专题、源码、书籍、视频、咨询、面试、环境等方面内容。尤其在技术专题方面会提供更多的原创内容，让更多的程序员可以从最基础开始了解到技术全貌，目前已经对外提供的有；《手写RPC框架》、《用Java实现JVM》、《基于JavaAgent的全链路监控》、《Netty案例》等专题。



手写RPC框架

RPC是一种远程调用的通信协议，例如dubbo、thrift等，我们在互联网高并发应用开发时候都会使用到类似的服务。本专题主要通过三个章节实现一个rpc通信的基础功能，来学习RPC服务中间件是如何开发和使用。章节内以源码加说明实战方式来讲解，请尽可能下载源码学习。

公众号：bugstack虫洞栈 | 关注获取源码 | 作者：付政委

章节列表

- 手写RPC框架第一章《自定义配置xml》
- 手写RPC框架第二章《netty通信》
- 手写RPC框架第三章《RPC中间件》

第一章、自定义配置xml

案例介绍 本案例通过三个章节来实现一共简单的rpc框架，用于深入学习rpc框架是如何通信的，当前章节主要介绍如何自定义xml文件并进行解析。想解析自定义的xml首先定义自己的xsd文件，并且实现spring的NamespaceHandlerSupport、BeanDefinitionParser，两个方法进行处理。

> 远程过程调用协议 RPC（Remote Procedure Call）—远程过程调用，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。 RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。
>
> Dubbo是 [1] 阿里巴巴公司开源的一个高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 [2] Spring框架无缝集成。Dubbo是一款高性能、轻量级的开源Java RPC框架，它提供了三大核心能力：面向接口的远程方法调用，智能容错和负载均衡，以及服务自动注册和发现。

环境准备 1、jdk 1.8.0 2、IntelliJ IDEA Community Edition 2018.3.1 x64

代码示例

```
itstack-demo-rpc-01
└── src
    └── main
        ├── java
        │   └── org.itstack.demo.rpc.config
        │       ├── spring
        │       │   ├── bean
        │       │   │   ├── ConsumerBean.java
        │       │   │   ├── ProviderBean.java
        │       │   │   └── ServerBean.java
        │       │   ├── MyBeanDefinitionParser.java
        │       │   └── MyNamespaceHandler.java
        │       ├── ConsumerConfig.java
        │       ├── ProviderConfig.java
        │       └── ProviderConfig.java
        └── resource
```

```
    │       └── META-INF
    │           ├── rpc.xsd
    │           ├── spring.handlers
    │           └── spring.schemas
    └── test
        ├── java
        │   └── org.itstack.demo.test
        │       ├── service
        │       │   ├── impl
        │       │   │   └── HelloServiceImpl.java
        │       │   └── HelloService.java
        │       └── ApiTest.java
        └── resource
            ├── itstack-rpc-consumer.xml
            ├── itstack-rpc-provider.xml
            └── log4j.xml
```

> ProviderConfig.java

```
public class ProviderConfig {

    private String nozzle; //接口
    private String ref;    //映射
    private String alias;  //别名

    //发布
    protected void doExport() {
        System.out.format("生产者信息=> [接口：%s] [映射：%s] [别名：%s] \r\n", nozzle, ref, alias);
    }

    public String getNozzle() {
        return nozzle;
    }

    public void setNozzle(String nozzle) {
        this.nozzle = nozzle;
    }

    public String getRef() {
        return ref;
    }

    public void setRef(String ref) {
        this.ref = ref;
    }

    public String getAlias() {
        return alias;
    }

    public void setAlias(String alias) {
        this.alias = alias;
    }
}
```

> ProviderBean.java

```
public class ProviderBean extends ProviderConfig implements ApplicationContextAware {

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        //发布生产者
        doExport();
    }

}
```

> MyBeanDefinitionParser.java

```
public class MyBeanDefinitionParser implements BeanDefinitionParser {
```

```
    private final Class<?> beanClass;

    MyBeanDefinitionParser(Class<?> beanClass) {
        this.beanClass = beanClass;
    }

    @Override
    public BeanDefinition parse(Element element, ParserContext parserContext) {

        RootBeanDefinition beanDefinition = new RootBeanDefinition();
        beanDefinition.setBeanClass(beanClass);
        beanDefinition.setLazyInit(false);
        String beanName = element.getAttribute("id");
        parserContext.getRegistry().registerBeanDefinition(beanName, beanDefinition);

        for (Method method : beanClass.getMethods()) {
            if (!isProperty(method, beanClass)) continue;
            String name = method.getName();
            String methodName = name.substring(3, 4).toLowerCase() + name.substring(4);
            String value = element.getAttribute(methodName);
            beanDefinition.getPropertyValues().addPropertyValue(methodName, value);
        }

        return beanDefinition;
    }

    private boolean isProperty(Method method, Class beanClass) {

        String methodName = method.getName();
        boolean flag = methodName.length() > 3 && methodName.startsWith("set") && Modifier.isPublic(method.getModifiers()) &&
method.getParameterTypes().length == 1;
        Method getter = null;
        if (!flag) return false;

        Class<?> type = method.getParameterTypes()[0];
        try {
            getter = beanClass.getMethod("get" + methodName.substring(3));
        } catch (NoSuchMethodException ignore) {

        }

        if (null == getter) {
            try {
                getter = beanClass.getMethod("is" + methodName.substring(3));
            } catch (NoSuchMethodException ignore) {

            }
        }

        flag = getter != null && Modifier.isPublic(getter.getModifiers()) && type.equals(getter.getReturnType());

        return flag;

    }

}
```

MyNamespaceHandler.java

```
public class MyNamespaceHandler extends NamespaceHandlerSupport {

    @Override
    public void init() {
        registerBeanDefinitionParser("consumer", new MyBeanDefinitionParser(ConsumerBean.class));
        registerBeanDefinitionParser("provider", new MyBeanDefinitionParser(ProviderBean.class));
        registerBeanDefinitionParser("server", new MyBeanDefinitionParser(ServerBean.class));
    }

}
```

rpc.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns="http://rpc.itstack.org/schema/rpc"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:beans="http://www.springframework.org/schema/beans"
            targetNamespace="http://rpc.itstack.org/schema/rpc"
            elementFormDefault="qualified" attributeFormDefault="unqualified">
    <xsd:import namespace="http://www.springframework.org/schema/beans"/>

    <!-- org.itstack.demo.rpc.config.ServerConfig -->
    <xsd:element name="server">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="beans:identifiedType">
                    <xsd:attribute name="host" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 栈台地点 ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="port" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 栈台岸口  ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>

    <!-- org.itstack.demo.rpc.config.ConsumerConfig -->
    <xsd:element name="consumer">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="beans:identifiedType">
                    <xsd:attribute name="nozzle" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 接口名称 ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="alias" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 服务别名分组信息  ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>

    <!-- org.itstack.demo.rpc.config.ProviderConfig -->
    <xsd:element name="provider">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="beans:identifiedType">
                    <xsd:attribute name="nozzle" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 接口名称 ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="ref" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 接口实现类  ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                    <xsd:attribute name="alias" type="xsd:string">
                        <xsd:annotation>
                            <xsd:documentation><![CDATA[ 服务别名分组信息  ]]></xsd:documentation>
                        </xsd:annotation>
                    </xsd:attribute>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

> spring.handlers

```
http\://rpc.itstack.org/schema/rpc=org.itstack.demo.rpc.config.spring.MyNamespaceHandler
```

> spring.schemas

```
http\://rpc.itstack.org/schema/rpc/rpc.xsd=META-INF/rpc.xsd
```

测试部分

> itstack-rpc-consumer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rpc="http://rpc.itstack.org/schema/rpc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans.xsd
      http://rpc.itstack.org/schema/rpc http://rpc.itstack.org/schema/rpc/rpc.xsd">

    <!-- redis配置，保存链接 -->
    <rpc:server id="consumer_itstack" host="127.0.0.1" port="6379"/>

    <rpc:consumer id="consumer_helloService" nozzle="org.itstack.demo.test.service.HelloService" alias="itStackRpc"/>

</beans>
```

> itstack-rpc-provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rpc="http://rpc.itstack.org/schema/rpc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-
beans.xsd
      http://rpc.itstack.org/schema/rpc http://rpc.itstack.org/schema/rpc/rpc.xsd">

    <rpc:provider id="provider_helloService" nozzle="org.itstack.demo.test.service.HelloService"
                  ref="helloService" alias="itStackRpc" />

</beans>
```

> ApiTest.java

```
package org.itstack.demo.test;

import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/4
 * 本章节主要介绍如何读取自定义配置xml文件字段信息
 */
public class ApiTest {

    public static void main(String[] args) {
        String[] configs = {"itstack-rpc-consumer.xml", "itstack-rpc-provider.xml"};
        new ClassPathXmlApplicationContext(configs);
    }

}
```

测试结果

```
2019-05-07 19:44:24,805 main  INFO [org.springframework.context.support.ClassPathXmlApplicationContext:prepareRefresh:510] -
Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@299a06ac: startup date [Tue May 07 19:44:24 CST
2019]; root of context hierarchy
2019-05-07 19:44:24,872 main  INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader:loadBeanDefinitions:315] -
Loading XML bean definitions from class path resource [itstack-rpc-consumer.xml]
2019-05-07 19:44:24,972 main  INFO [org.springframework.beans.factory.xml.XmlBeanDefinitionReader:loadBeanDefinitions:315] -
Loading XML bean definitions from class path resource [itstack-rpc-provider.xml]
2019-05-07 19:44:25,008 main  INFO
[org.springframework.beans.factory.support.DefaultListableBeanFactory:preInstantiateSingletons:577] - Pre-instantiating
```

```
singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@192b07fd: defining beans
[consumer_itstack,consumer_helloService,provider_helloService]; root of factory hierarchy
服务端信息=> [注册中心地址：127.0.0.1] [注册中心端口：6379]
生产者信息=> [接口：org.itstack.demo.test.service.HelloService] [映射：helloService] [别名：itStackRpc]
```

第二章、netty通信

案例介绍 在我们实现rpc框架的时候，需要选择socket的通信方式。而我们知道一般情况下socket通信类似与qq聊天，发过去消息，什么时候回复都可以。但是我们rpc框架通信，从感觉上类似http调用，需要在一定时间内返回，否则就会发生超时断开。

这里我们选择netty作为我们的socket框架，采用future方式进行通信。

> Netty是由JBOSS提供的一个java开源框架。Netty提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序。也就是说，Netty 是一个基于NIO的客户、服务器端编程框架，使用Netty 可以确保你快速和简单的开发出一个网络应用，例如实现了某种协议的客户、服务端应用。Netty相当于简化和流线化了网络应用的编程开发过程，例如：基于TCP和UDP的socket服务开发。“快速”和“简单”并不用产生维护性或性能上的问题。Netty 是一个吸收了多种协议（包括FTP、SMTP、HTTP等各种二进制文本协议）的实现经验，并经过相当精心设计的项目。最终，Netty 成功的找到了一种方式，在保证易于开发的同时还保证了其应用的性能，稳定性和伸缩性。

环境准备 1、jdk 1.8.0 2、IntelliJ IDEA Community Edition 2018.3.1 x64

代码示例

```
itstack-demo-rpc-02
└── src
    └── main
    │    └── java
    │        └── org.itstack.demo.rpc.network
    │             ├── client
    │             │   ├── ClientSocket.java
    │             │   └── MyClientHandler.java
    │             ├── codec
    │             │   ├── RpcDecoder.java
    │             │   └── RpcEncoder.java
    │             ├── future
    │             │   ├── SyncWrite.java
    │             │   ├── SyncWriteFuture.java
    │             │   ├── SyncWriteMap.java
    │             │   └── WriteFuture.java
    │             ├── msg
    │             │   ├── Request.java
    │             │   └── Response.java
    │             ├── server
    │             │   ├── MyServerHandler.java
    │             │   └── ServerSocket.java
    │             └── util
    │                 └── SerializationUtil.java
    └── test
         └── java
             └── org.itstack.demo.test
                 ├── client
                 │   └── StartClient.java
                 └── server
                     └── StartServer.java
```

> ClientSocket.java

```
package org.itstack.demo.rpc.network.client;

import io.netty.bootstrap.Bootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioSocketChannel;
import org.itstack.demo.rpc.network.codec.RpcDecoder;
import org.itstack.demo.rpc.network.codec.RpcEncoder;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
```

```
 * create by fuzhengwei on 2019/5/6
 */
public class ClientSocket implements Runnable {

    private ChannelFuture future;

    @Override
    public void run() {
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            Bootstrap b = new Bootstrap();
            b.group(workerGroup);
            b.channel(NioSocketChannel.class);
            b.option(ChannelOption.AUTO_READ, true);
            b.handler(new ChannelInitializer<SocketChannel>() {
                @Override
                public void initChannel(SocketChannel ch) throws Exception {
                    ch.pipeline().addLast(
                            new RpcDecoder(Response.class),
                            new RpcEncoder(Request.class),
                            new MyClientHandler());
                }
            });
            ChannelFuture f = b.connect("127.0.0.1", 7397).sync();
            this.future = f;
            f.channel().closeFuture().sync();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            workerGroup.shutdownGracefully();
        }
    }

    public ChannelFuture getFuture() {
        return future;
    }

    public void setFuture(ChannelFuture future) {
        this.future = future;
    }
}
```

MyClientHandler.java

```
package org.itstack.demo.rpc.network.client;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import org.itstack.demo.rpc.network.future.SyncWriteFuture;
import org.itstack.demo.rpc.network.future.SyncWriteMap;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class MyClientHandler extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object obj) throws Exception {
        Response msg = (Response) obj;
        String requestId = msg.getRequestId();
        SyncWriteFuture future = (SyncWriteFuture) SyncWriteMap.syncKey.get(requestId);
        if (future != null) {
            future.setResponse(msg);
        }
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }

}
```

RpcDecoder.java

```
package org.itstack.demo.rpc.network.codec;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.ByteToMessageDecoder;
import org.itstack.demo.rpc.network.util.SerializationUtil;

import java.util.List;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class RpcDecoder extends ByteToMessageDecoder {

    private Class<?> genericClass;

    public RpcDecoder(Class<?> genericClass) {
        this.genericClass = genericClass;
    }

    @Override
    protected void decode(ChannelHandlerContext ctx, ByteBuf in, List<Object> out) {
        if (in.readableBytes() < 4) {
            return;
        }
        in.markReaderIndex();
        int dataLength = in.readInt();
        if (in.readableBytes() < dataLength) {
            in.resetReaderIndex();
            return;
        }
        byte[] data = new byte[dataLength];
        in.readBytes(data);
        out.add(SerializationUtil.deserialize(data, genericClass));
    }

}
```

RpcEncoder.java

```
package org.itstack.demo.rpc.network.codec;

import io.netty.buffer.ByteBuf;
import io.netty.channel.ChannelHandlerContext;
import io.netty.handler.codec.MessageToByteEncoder;
import org.itstack.demo.rpc.network.util.SerializationUtil;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class RpcEncoder extends MessageToByteEncoder {

    private Class<?> genericClass;

    public RpcEncoder(Class<?> genericClass) {
        this.genericClass = genericClass;
    }

    @Override
    protected void encode(ChannelHandlerContext ctx, Object in, ByteBuf out)  {
        if (genericClass.isInstance(in)) {
            byte[] data = SerializationUtil.serialize(in);
            out.writeInt(data.length);
            out.writeBytes(data);
        }
    }

}
```

SyncWrite.java

```
package org.itstack.demo.rpc.network.future;

import io.netty.channel.Channel;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelFutureListener;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

import java.util.UUID;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SyncWrite {

    public Response writeAndSync(final Channel channel, final Request request, final long timeout) throws Exception {

        if (channel == null) {
            throw new NullPointerException("channel");
        }
        if (request == null) {
            throw new NullPointerException("request");
        }
        if (timeout <= 0) {
            throw new IllegalArgumentException("timeout <= 0");
        }

        String requestId = UUID.randomUUID().toString();
        request.setRequestId(requestId);

        WriteFuture<Response> future = new SyncWriteFuture(request.getRequestId());
        SyncWriteMap.syncKey.put(request.getRequestId(), future);

        Response response = doWriteAndSync(channel, request, timeout, future);

        SyncWriteMap.syncKey.remove(request.getRequestId());
        return response;
    }

    private Response doWriteAndSync(final Channel channel, final Request request, final long timeout, final
WriteFuture<Response> writeFuture) throws Exception {

        channel.writeAndFlush(request).addListener(new ChannelFutureListener() {
            public void operationComplete(ChannelFuture future) throws Exception {
                writeFuture.setWriteResult(future.isSuccess());
                writeFuture.setCause(future.cause());
                //失败移除
                if (!writeFuture.isWriteSuccess()) {
                    SyncWriteMap.syncKey.remove(writeFuture.requestId());
                }
            }
        });

        Response response = writeFuture.get(timeout, TimeUnit.MILLISECONDS);
        if (response == null) {
            if (writeFuture.isTimeout()) {
                throw new TimeoutException();
            } else {
                // write exception
                throw new Exception(writeFuture.cause());
            }
        }
        return response;
    }

}
```

SyncWriteFuture.java

```
package org.itstack.demo.rpc.network.future;


import org.itstack.demo.rpc.network.msg.Response;
```

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

public class SyncWriteFuture implements WriteFuture<Response> {

    private CountDownLatch latch = new CountDownLatch(1);
    private final long begin = System.currentTimeMillis();
    private long timeout;
    private Response response;
    private final String requestId;
    private boolean writeResult;
    private Throwable cause;
    private boolean isTimeout = false;

    public SyncWriteFuture(String requestId) {
        this.requestId = requestId;
    }

    public SyncWriteFuture(String requestId, long timeout) {
        this.requestId = requestId;
        this.timeout = timeout;
        writeResult = true;
        isTimeout = false;
    }


    public Throwable cause() {
        return cause;
    }

    public void setCause(Throwable cause) {
        this.cause = cause;
    }

    public boolean isWriteSuccess() {
        return writeResult;
    }

    public void setWriteResult(boolean result) {
        this.writeResult = result;
    }

    public String requestId() {
        return requestId;
    }

    public Response response() {
        return response;
    }

    public void setResponse(Response response) {
        this.response = response;
        latch.countDown();
    }

    public boolean cancel(boolean mayInterruptIfRunning) {
        return true;
    }

    public boolean isCancelled() {
        return false;
    }

    public boolean isDone() {
        return false;
    }

    public Response get() throws InterruptedException, ExecutionException {
        latch.wait();
        return response;
    }

    public Response get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException {
        if (latch.await(timeout, unit)) {
            return response;
```

```
        }
        return null;
    }

    public boolean isTimeout() {
        if (isTimeout) {
            return isTimeout;
        }
        return System.currentTimeMillis() - begin > timeout;
    }
}
```

SyncWriteMap.java

```
package org.itstack.demo.rpc.network.future;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class SyncWriteMap {

    public static Map<String, WriteFuture> syncKey = new ConcurrentHashMap<String, WriteFuture>();

}
```

WriteFuture.java

```
package org.itstack.demo.rpc.network.future;

import org.itstack.demo.rpc.network.msg.Response;

import java.util.concurrent.Future;

public interface WriteFuture<T> extends Future<T> {

    Throwable cause();

    void setCause(Throwable cause);

    boolean isWriteSuccess();

    void setWriteResult(boolean result);

    String requestId();

    T response();

    void setResponse(Response response);

    boolean isTimeout();


}
```

Request.java

```
package org.itstack.demo.rpc.network.msg;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class Request {

    private String requestId;
    private Object result;

    public String getRequestId() {
        return requestId;
    }
```

```
    public void setRequestId(String requestId) {
        this.requestId = requestId;
    }

    public Object getResult() {
        return result;
    }

    public void setResult(Object result) {
        this.result = result;
    }

}
```

Response.java

```
package org.itstack.demo.rpc.network.msg;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class Response {

    private String requestId;
    private String param;

    public String getRequestId() {
        return requestId;
    }

    public void setRequestId(String requestId) {
        this.requestId = requestId;
    }

    public String getParam() {
        return param;
    }

    public void setParam(String param) {
        this.param = param;
    }

}
```

MyServerHandler.java

```
package org.itstack.demo.rpc.network.server;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.util.ReferenceCountUtil;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class MyServerHandler extends ChannelInboundHandlerAdapter{

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object obj){
        Request msg = (Request) obj;
        //反馈
        Response request = new Response();
        request.setRequestId(msg.getRequestId());
        request.setParam(msg.getResult() + " 请求成功，反馈结果请接受处理。");
        ctx.writeAndFlush(request);
        //释放
        ReferenceCountUtil.release(msg);
    }
```

```
    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) {
        ctx.flush();
    }

}
```

ServerSocket.java

```
package org.itstack.demo.rpc.network.server;

import io.netty.bootstrap.ServerBootstrap;
import io.netty.channel.ChannelFuture;
import io.netty.channel.ChannelInitializer;
import io.netty.channel.ChannelOption;
import io.netty.channel.EventLoopGroup;
import io.netty.channel.nio.NioEventLoopGroup;
import io.netty.channel.socket.SocketChannel;
import io.netty.channel.socket.nio.NioServerSocketChannel;
import org.itstack.demo.rpc.network.codec.RpcDecoder;
import org.itstack.demo.rpc.network.codec.RpcEncoder;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class ServerSocket implements Runnable {

    private ChannelFuture f;

    @Override
    public void run() {
        EventLoopGroup bossGroup = new NioEventLoopGroup();
        EventLoopGroup workerGroup = new NioEventLoopGroup();
        try {
            ServerBootstrap b = new ServerBootstrap();
            b.group(bossGroup, workerGroup)
                    .channel(NioServerSocketChannel.class)
                    .option(ChannelOption.SO_BACKLOG, 128)
                    .childHandler(new ChannelInitializer<SocketChannel>() {
                        @Override
                        public void initChannel(SocketChannel ch){
                            ch.pipeline().addLast(
                                    new RpcDecoder(Request.class),
                                    new RpcEncoder(Response.class),
                                    new MyServerHandler());
                        }
                    });

            ChannelFuture f = null;
            f = b.bind(7397).sync();
            f.channel().closeFuture().sync();


        } catch (InterruptedException e) {
            e.printStackTrace();
        } finally {
            workerGroup.shutdownGracefully();
            bossGroup.shutdownGracefully();
        }

    }

}
```

SerializationUtil.java

```
package org.itstack.demo.rpc.network.util;

import com.dyuproject.protostuff.LinkedBuffer;
import com.dyuproject.protostuff.ProtostuffIOUtil;
```

```
import com.dyuproject.protostuff.Schema;
import com.dyuproject.protostuff.runtime.RuntimeSchema;
import org.objenesis.Objenesis;
import org.objenesis.ObjenesisStd;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * Created by fuzhengwei1 on 2016/10/20.
 */
public class SerializationUtil {

    private static Map<Class<?>, Schema<?>> cachedSchema = new ConcurrentHashMap();

    private static Objenesis objenesis = new ObjenesisStd();

    private SerializationUtil() {

    }

    /**
     * 序列化(对象 -> 字节数组)
     *
     * @param obj 对象
     * @return 字节数组
     */
    public static <T> byte[] serialize(T obj) {
        Class<T> cls = (Class<T>) obj.getClass();
        LinkedBuffer buffer = LinkedBuffer.allocate(LinkedBuffer.DEFAULT_BUFFER_SIZE);
        try {
            Schema<T> schema = getSchema(cls);
            return ProtostuffIOUtil.toByteArray(obj, schema, buffer);
        } catch (Exception e) {
            throw new IllegalStateException(e.getMessage(), e);
        } finally {
            buffer.clear();
        }
    }

    /**
     * 反序列化(字节数组 -> 对象)
     *
     * @param data
     * @param cls
     * @param <T>
     */
    public static <T> T deserialize(byte[] data, Class<T> cls) {
        try {
            T message = objenesis.newInstance(cls);
            Schema<T> schema = getSchema(cls);
            ProtostuffIOUtil.mergeFrom(data, message, schema);
            return message;
        } catch (Exception e) {
            throw new IllegalStateException(e.getMessage(), e);
        }
    }

    private static <T> Schema<T> getSchema(Class<T> cls) {
        Schema<T> schema = (Schema<T>) cachedSchema.get(cls);
        if (schema == null) {
            schema = RuntimeSchema.createFrom(cls);
            cachedSchema.put(cls, schema);
        }
        return schema;
    }

}
```

StartClient.java

```
package org.itstack.demo.test.client;

import com.alibaba.fastjson.JSON;
import io.netty.channel.ChannelFuture;
import org.itstack.demo.rpc.network.client.ClientSocket;
```

```
import org.itstack.demo.rpc.network.future.SyncWrite;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class StartClient {

    private static ChannelFuture future;

    public static void main(String[] args) {
        ClientSocket client = new ClientSocket();
        new Thread(client).start();

        while (true) {
            try {
                //获取future，线程有等待处理时间
                if (null == future) {
                    future = client.getFuture();
                    Thread.sleep(500);
                    continue;
                }
                //构建发送参数
                Request request = new Request();
                request.setResult("查询用户信息");
                SyncWrite s = new SyncWrite();
                Response response = s.writeAndSync(future.channel(), request, 1000);
                System.out.println("调用结果：" + JSON.toJSON(response));
                Thread.sleep(1000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

}
```

StartServer.java

```
package org.itstack.demo.test.server;

import org.itstack.demo.rpc.network.server.ServerSocket;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class StartServer {

    public static void main(String[] args) {
        System.out.println("启动服务端开始");
        new Thread(new ServerSocket()).start();
        System.out.println("启动服务端完成");
    }

}
```

测试结果

启动StartServer

```
启动服务端开始
启动服务端完成
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
```

启动StartClient

```
log4j:WARN No appenders could be found for logger (io.netty.util.internal.logging.InternalLoggerFactory).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"3380f061-2501-49b5-998b-21b5956fe60a"}
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"81c51815-4d92-482c-bd05-e4b6dfa4d3b6"}
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"7af01c4f-a438-47a1-b35c-8e2cd7e4a5e7"}
```

```
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"86e38bb1-eccc-4d45-b976-c3b67999e3ab"}
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"7f72002c-3b38-43d9-8452-db8797298899"}
调用结果：{"param":"查询用户信息 请求成功，反馈结果请接受处理。","requestId":"d566a7d4-4b0d-426b-8c09-c535ccf8eb09"}

...
```

第三章、RPC中间件

案例介绍 结合上面两章节，本章将实现rpc的基础功能；提供一给rpc中间件jar给生产端和服务端。 技术点； 1、注册中心，生产者在启动的时候需要将本地接口发布到注册中心，我们这里采用redis作为注册中心，随机取数模拟权重。 2、客户端在启动的时候，连接到注册中心，也就是我们的redis。连接成功后将配置的生产者方法发布到注册中心{接口+别名}。 3、服务端配置生产者的信息后，在加载xml时候由中间件生成动态代理类，当发生发放调用时实际则调用了我们代理类的方法，代理里会通过netty的futuer通信方式进行数据交互。

环境准备 1、jdk 1.8.0 2、IntelliJ IDEA Community Edition 2018.3.1 x64 3、windows redis

代码示例

```
itstack-demo-rpc-03
└── src
    └── main
    │    ├── java
    │    │    └── org.itstack.demo.rpc
    │    │        ├── config
    │    │        ├── domain
    │    │        ├── network
    │    │        │   ├── client
    │    │        │   │   ├── ClientSocket.java
    │    │        │   │   └── MyClientHandler.java
    │    │        │   ├── codec
    │    │        │   │   ├── RpcDecoder.java
    │    │        │   │   └── RpcEncoder.java
    │    │        │   ├── future
    │    │        │   │   ├── SyncWrite.java
    │    │        │   │   ├── SyncWriteFuture.java
    │    │        │   │   ├── SyncWriteMap.java
    │    │        │   │   └── WriteFuture.java
    │    │        │   ├── msg
    │    │        │   │   ├── Request.java
    │    │        │   │   └── Response.java
    │    │        │   ├── server
    │    │        │   │   ├── MyServerHandler.java
    │    │        │   │   └── ServerSocket.java
    │    │        │   └── util
    │    │        │       └── SerializationUtil.java
    │    │        ├── reflect
    │    │        │   ├── JDKInvocationHandler.java
    │    │        │   └── JDKProxy.java
    │    │        ├── registry
    │    │        │   └── RedisRegistryCenter.java
    │    │        └── util
    │    └── resource
    │        └── META-INF
    │            ├── rpc.xsd
    │            ├── spring.handlers
    │            └── spring.schemas
    │
    └── test
        ├── java
        │   └── org.itstack.demo.test
        │       ├── service
        │       │   ├── impl
        │       │   │   └── HelloServiceImpl.java
        │       │   └── HelloService.java
        │       └── ApiTest.java
        └── resource
            ├── itstack-rpc-center.xml
            ├── itstack-rpc-consumer.xml
            ├── itstack-rpc-provider.xml
            └── log4j.xml
```

> ConsumerBean.java

```
package org.itstack.demo.rpc.config.spring.bean;
```

```
import com.alibaba.fastjson.JSON;
import io.netty.channel.ChannelFuture;
import org.itstack.demo.rpc.config.ConsumerConfig;
import org.itstack.demo.rpc.domain.RpcProviderConfig;
import org.itstack.demo.rpc.network.client.ClientSocket;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.reflect.JDKProxy;
import org.itstack.demo.rpc.registry.RedisRegistryCenter;
import org.itstack.demo.rpc.util.ClassLoaderUtils;
import org.springframework.beans.factory.FactoryBean;
import org.springframework.util.Assert;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class ConsumerBean<T> extends ConsumerConfig<T> implements FactoryBean {

    private ChannelFuture channelFuture;

    private RpcProviderConfig rpcProviderConfig;

    @Override
    public Object getObject() throws Exception {

        //从redis获取链接
        if (null == rpcProviderConfig) {
            String infoStr = RedisRegistryCenter.obtainProvider(nozzle, alias);
            rpcProviderConfig = JSON.parseObject(infoStr, RpcProviderConfig.class);
        }
        Assert.isTrue(null != rpcProviderConfig);

        //获取通信channel
        if (null == channelFuture) {
            ClientSocket clientSocket = new ClientSocket(rpcProviderConfig.getHost(), rpcProviderConfig.getPort());
            new Thread(clientSocket).start();
            for (int i = 0; i < 100; i++) {
                if (null != channelFuture) break;
                Thread.sleep(500);
                channelFuture = clientSocket.getFuture();
            }
        }
        Assert.isTrue(null != channelFuture);

        Request request = new Request();
        request.setChannel(channelFuture.channel());
        request.setNozzle(nozzle);
        request.setRef(rpcProviderConfig.getRef());
        request.setAlias(alias);
        return (T) JDKProxy.getProxy(ClassLoaderUtils.forName(nozzle), request);
    }

    @Override
    public Class<?> getObjectType() {
        try {
            return ClassLoaderUtils.forName(nozzle);
        } catch (ClassNotFoundException e) {
            return null;
        }
    }

    @Override
    public boolean isSingleton() {
        return true;
    }


}
```

ProviderBean.java

```
package org.itstack.demo.rpc.config.spring.bean;

import com.alibaba.fastjson.JSON;
import org.itstack.demo.rpc.config.ProviderConfig;
import org.itstack.demo.rpc.domain.LocalServerInfo;
```

```
import org.itstack.demo.rpc.domain.RpcProviderConfig;
import org.itstack.demo.rpc.registry.RedisRegistryCenter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class ProviderBean extends ProviderConfig implements ApplicationContextAware {

    private Logger logger = LoggerFactory.getLogger(ProviderBean.class);

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {

        RpcProviderConfig rpcProviderConfig = new RpcProviderConfig();
        rpcProviderConfig.setNozzle(nozzle);
        rpcProviderConfig.setRef(ref);
        rpcProviderConfig.setAlias(alias);
        rpcProviderConfig.setHost(LocalServerInfo.LOCAL_HOST);
        rpcProviderConfig.setPort(LocalServerInfo.LOCAL_PORT);

        //注册生产者
        long count = RedisRegistryCenter.registryProvider(nozzle, alias, JSON.toJSONString(rpcProviderConfig));

        logger.info("注册生产者：{} {} {}", nozzle, alias, count);
    }

}
```

ServerBean.java

```
package org.itstack.demo.rpc.config.spring.bean;

import org.itstack.demo.rpc.config.ServerConfig;
import org.itstack.demo.rpc.domain.LocalServerInfo;
import org.itstack.demo.rpc.network.server.ServerSocket;
import org.itstack.demo.rpc.registry.RedisRegistryCenter;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.BeansException;
import org.springframework.beans.factory.InitializingBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class ServerBean extends ServerConfig implements ApplicationContextAware {

    private Logger logger = LoggerFactory.getLogger(ServerBean.class);

    @Override
    public void setApplicationContext(ApplicationContext applicationContext) throws BeansException {
        //启动注册中心
        logger.info("启动注册中心 ...");
        RedisRegistryCenter.init(host, port);
        logger.info("启动注册中心完成 {} {}", host, port);

        //初始化服务端
        logger.info("初始化生产端服务 ...");
        ServerSocket serverSocket = new ServerSocket(applicationContext);
        Thread thread = new Thread(serverSocket);
        thread.start();
        while (!serverSocket.isActiveSocketServer()) {
            try {
                Thread.sleep(500);
            } catch (InterruptedException ignore) {
            }
        }
```

```
        logger.info("初始化生产端服务完成 {} {}", LocalServerInfo.LOCAL_HOST, LocalServerInfo.LOCAL_PORT);
    }


}
```

MyClientHandler.java

```
package org.itstack.demo.rpc.network.client;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import org.itstack.demo.rpc.network.future.SyncWriteFuture;
import org.itstack.demo.rpc.network.future.SyncWriteMap;
import org.itstack.demo.rpc.network.msg.Response;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class MyClientHandler extends ChannelInboundHandlerAdapter {

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object obj) throws Exception {
        Response msg = (Response) obj;
        String requestId = msg.getRequestId();
        SyncWriteFuture future = (SyncWriteFuture) SyncWriteMap.syncKey.get(requestId);
        if (future != null) {
            future.setResponse(msg);
        }
    }

    @Override
    public void exceptionCaught(ChannelHandlerContext ctx, Throwable cause) {
        cause.printStackTrace();
        ctx.close();
    }

}
```

MyServerHandler.java

```
package org.itstack.demo.rpc.network.server;

import io.netty.channel.ChannelHandlerContext;
import io.netty.channel.ChannelInboundHandlerAdapter;
import io.netty.util.ReferenceCountUtil;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;
import org.itstack.demo.rpc.util.ClassLoaderUtils;
import org.springframework.context.ApplicationContext;

import java.lang.reflect.Method;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/6
 */
public class MyServerHandler extends ChannelInboundHandlerAdapter {

    private ApplicationContext applicationContext;

    MyServerHandler(ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
    }

    @Override
    public void channelRead(ChannelHandlerContext ctx, Object obj) {
        try {
            Request msg = (Request) obj;
            //调用
            Class<?> classType = ClassLoaderUtils.forName(msg.getNozzle());
            Method addMethod = classType.getMethod(msg.getMethodName(), msg.getParamTypes());
```

```
            Object objectBean = applicationContext.getBean(msg.getRef());
            Object result = addMethod.invoke(objectBean, msg.getArgs());
            //反馈
            Response request = new Response();
            request.setRequestId(msg.getRequestId());
            request.setResult(result);
            ctx.writeAndFlush(request);
            //释放
            ReferenceCountUtil.release(msg);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void channelReadComplete(ChannelHandlerContext ctx) {
        ctx.flush();
    }

}
```

> JDKInvocationHandler.java

```
package org.itstack.demo.rpc.reflect;


import org.itstack.demo.rpc.network.future.SyncWrite;
import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.network.msg.Response;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;

public class JDKInvocationHandler implements InvocationHandler {

    private Request request;

    public JDKInvocationHandler(Request request) {
        this.request = request;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        String methodName = method.getName();
        Class[] paramTypes = method.getParameterTypes();
        if ("toString".equals(methodName) && paramTypes.length == 0) {
            return request.toString();
        } else if ("hashCode".equals(methodName) && paramTypes.length == 0) {
            return request.hashCode();
        } else if ("equals".equals(methodName) && paramTypes.length == 1) {
            return request.equals(args[0]);
        }
        //设置参数
        request.setMethodName(methodName);
        request.setParamTypes(paramTypes);
        request.setArgs(args);
        request.setRef(request.getRef());
        Response response = new SyncWrite().writeAndSync(request.getChannel(), request, 5000);
        //异步调用
        return response.getResult();

    }

}
```

> JDKProxy.java

```
package org.itstack.demo.rpc.reflect;


import org.itstack.demo.rpc.network.msg.Request;
import org.itstack.demo.rpc.util.ClassLoaderUtils;
```

```
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Proxy;

public class JDKProxy {

    public static <T> T getProxy(Class<T> interfaceClass, Request request) throws Exception {
        InvocationHandler handler = new JDKInvocationHandler(request);
        ClassLoader classLoader = ClassLoaderUtils.getCurrentClassLoader();
        T result = (T) Proxy.newProxyInstance(classLoader, new Class[]{interfaceClass}, handler);
        return result;
    }

}
```

RedisRegistryCenter.java

```
package org.itstack.demo.rpc.registry;

import redis.clients.jedis.Jedis;
import redis.clients.jedis.JedisPool;
import redis.clients.jedis.JedisPoolConfig;

/**
 * http://www.itstack.org
 * create by fuzhengwei on 2019/5/7
 * redis 模拟RPC注册中心
 */
public class RedisRegistryCenter {

    private static Jedis jedis;   //非切片额客户端连接

    //初始化redis
    public static void init(String host, int port) {
        // 池基本配置
        JedisPoolConfig config = new JedisPoolConfig();
        config.setMaxIdle(5);
        config.setTestOnBorrow(false);
        JedisPool jedisPool = new JedisPool(config, host, port);
        jedis = jedisPool.getResource();
    }

    /**
     * 注册生产者
     *
     * @param nozzle 接口
     * @param alias  别名
     * @param info   信息
     * @return 注册结果
     */
    public static Long registryProvider(String nozzle, String alias, String info) {
        return jedis.sadd(nozzle + "_" + alias, info);
    }

    /**
     * 获取生产者
     * 模拟权重，随机获取
     * @param nozzle 接口名称
     */
    public static String obtainProvider(String nozzle, String alias) {
        return jedis.srandmember(nozzle + "_" + alias);
    }

    public static Jedis jedis() {
        return jedis;
    }

}
```

ApiTest.java

```
public class ApiTest {

    public static void main(String[] args) {
```

```
        String[] configs = {"itstack-rpc-center.xml", "itstack-rpc-provider.xml", "itstack-rpc-consumer.xml"};
        new ClassPathXmlApplicationContext(configs);
    }

}
```

框架，测试结果

```
2019-....ClassPathXmlApplicationContext:prepareRefresh:510] - Refreshing
org.springframework.context.support.ClassPathXmlApplicationContext@299a06ac: startup date [Tue May 07 20:19:47 CST 2019]; root
of context hierarchy
2019-...ml.XmlBeanDefinitionReader:loadBeanDefinitions:315] - Loading XML bean definitions from class path resource
[spring/itstack-rpc-center.xml]
2019-...ml.XmlBeanDefinitionReader:loadBeanDefinitions:315] - Loading XML bean definitions from class path resource
[spring/itstack-rpc-provider.xml]
2019-...ml.XmlBeanDefinitionReader:loadBeanDefinitions:315] - Loading XML bean definitions from class path resource
[spring/itstack-rpc-consumer.xml]
2019-...upport.DefaultListableBeanFactory:preInstantiateSingletons:577] - Pre-instantiating singletons in
org.springframework.beans.factory.support.DefaultListableBeanFactory@7e0b0338: defining beans
[consumer_itstack,provider_helloService,consumer_helloService]; root of factory hierarchy
2019-...bean.ServerBean:setApplicationContext:25] - 启动注册中心 ...
2019-...bean.ServerBean:setApplicationContext:27] - 启动注册中心完成 127.0.0.1 6379
2019-...bean.ServerBean:setApplicationContext:30] - 初始化生产端服务 ...
2019-...bean.ServerBean:setApplicationContext:41] - 初始化生产端服务完成 10.13.81.104 22201
2019-...bean.ProviderBean:setApplicationContext:35] - 注册生产者：org.itstack.demo.test.service.HelloService itStackRpc 0
```

框架应用 为了测试我们写两个测试工程；itstack-demo-rpc-provider、itstack-demo-rpc-consumer

> itstack-demo-rpc-provider 提供生产者接口

```
itstack-demo-rpc-provider
├── itstack-demo-rpc-provider-export
│   └── src
│       └── main
│           └── java
│               └── org.itstack.demo.rpc.provider.export
│                   ├── domain
│                   │   └── Hi.java
│                   └── HelloService.java
│
└── itstack-demo-rpc-provider-web
    └── src
        └── main
            ├── java
            │   └── org.itstack.demo.rpc.provider.web
            │       └── HelloServiceImpl.java
            └── resources
                └── spring
                    └── spring-itstack-rpc-provider.xml
```

> HelloService.java

```
public interface HelloService {

    String hi();

    String say(String str);

    String sayHi(Hi hi);

}
```

> HelloServiceImpl.java

```
@Controller("helloService")
public class HelloServiceImpl implements HelloService {

    @Override
    public String hi() {
        return "hi itstack rpc";
    }
```

```
    @Override
    public String say(String str) {
        return str;
    }

    @Override
    public String sayHi(Hi hi) {
        return hi.getUserName() + " say: " + hi.getSayMsg();
    }

}
```

> spring-itstack-rpc-provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rpc="http://rpc.itstack.org/schema/rpc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
      http://rpc.itstack.org/schema/rpc http://rpc.itstack.org/schema/rpc/rpc.xsd">

    <!-- 注册中心 -->
    <rpc:server id="rpcServer" host="127.0.0.1" port="6379"/>

    <rpc:provider id="helloServiceRpc" nozzle="org.itstack.demo.rpc.provider.export.HelloService"
                  ref="helloService" alias="itstackRpc"/>

</beans>
```

> itstack-demo-rpc-consumer 提供消费者调用

```
itstack-demo-rpc-consumer
└── src
     ├── main
     │   ├── java
     │   └── resources
     │       └── spring
     │             └── spring-itstack-rpc-consumer.xml
     │
     └── test
         └── java
             └── org.itstack.demo.test
                 └── ConsumerTest.java
```

> spring-itstack-rpc-consumer.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:rpc="http://rpc.itstack.org/schema/rpc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
      http://rpc.itstack.org/schema/rpc http://rpc.itstack.org/schema/rpc/rpc.xsd">

    <!-- 注册中心 -->
    <rpc:server id="consumer_itstack" host="127.0.0.1" port="6379"/>

    <rpc:consumer id="helloService" nozzle="org.itstack.demo.rpc.provider.export.HelloService" alias="itstackRpc"/>

</beans>
```

> ConsumerTest.java

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("/spring-config.xml")
public class ConsumerTest {

    @Resource(name = "helloService")
    private HelloService helloService;
```

```
    @Test
    public void test() {

        String hi = helloService.hi();
        System.out.println("测试结果：" + hi);

        String say = helloService.say("hello world");
        System.out.println("测试结果：" + say);

        Hi hiReq = new Hi();
        hiReq.setUserName("付栈");
        hiReq.setSayMsg("付可敌国，栈无不胜");
        String hiRes = helloService.sayHi(hiReq);

        System.out.println("测试结果：" + hiRes);
    }

}
```

应用，测试结果 测试时启动redis

启动ProviderTest Redis中的注册数据

```
redis 127.0.0.1:6379> srandmember org.itstack.demo.rpc.provider.export.HelloService_itstackRpc
"
{\"alias\":\"itstackRpc\",\"host\":\"10.13.81.104\",\"nozzle\":\"org.itstack.demo.rpc.provider.export.HelloService\",\"port\":222
01,\"ref\":\"helloService\"}"
redis 127.0.0.1:6379>
```

执行ConsumerTest中的单元测试方法

```
log4j:WARN No appenders could be found for logger (org.springframework.test.context.junit4.SpringJUnit4ClassRunner).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
测试结果：hi itstack rpc
测试结果：hello world
测试结果：付栈 say：付可敌国，栈无不胜

Process finished with exit code 0
```

结束！祝福：愿努力拼搏的你，终能收获成绩！犹如；承遇朝霞，年少正恰。整装戎马，刻印风华。