rpc系列1-10 minute Tutorial

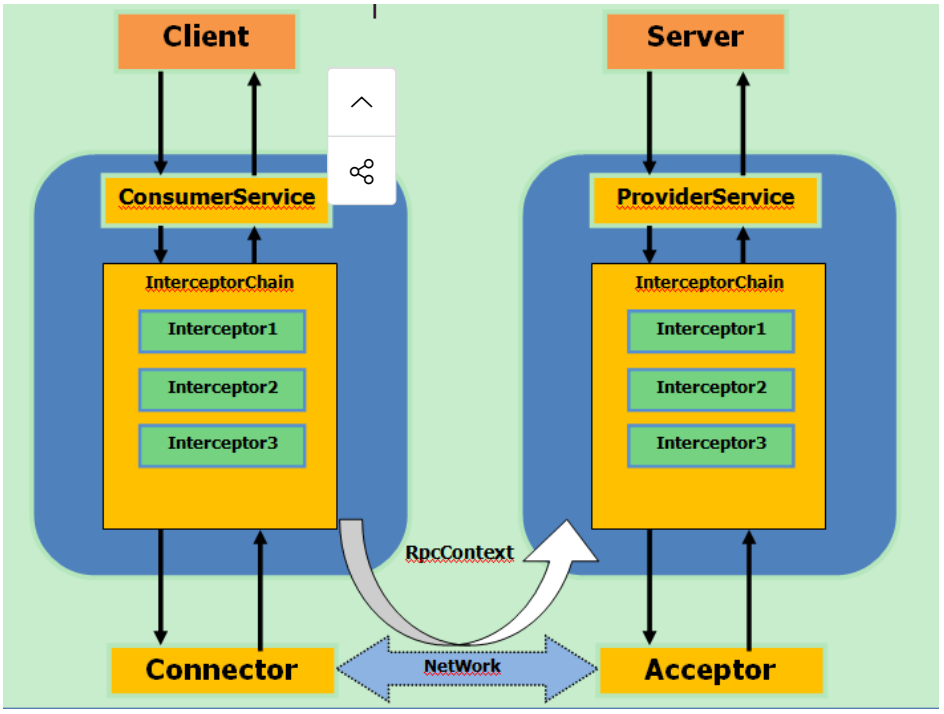
作者 TopGun_Viper (/u/aee2b91e3398) +关注
2016.09.12 09:23 字数 906 阅读 176 评论 1 喜欢 6
(/u/aee2b91e3398)

一个简单的rpc demo

最近在网上看到阿里巴巴2015年的中间件性能挑战赛的一个题目，实现一个简单的RPC框架，于是乎有一种冲动实现一个简单的rpc，要求基本按照竞赛题目的要求，具体如下：

> 1.要成为框架：对于框架的使用者，隐藏RPC实现。
>
> 2.网络模块可以自己编写，如果要使用IO框架，要求使用netty-4.0.23.Final。
>
> 3.能够传输基本类型、自定义业务类型、异常类型（要在客户端抛出）。
>
> 4.支持异步调用，提供future、callback的能力。
>
> 5.要处理超时场景，服务端处理时间较长时，客户端在指定时间内跳出本次调用。
>
> 6.提供RPC上下文，客户端可以透传数据给服务端。
>
> 7.提供Hook，让开发人员进行RPC层面的AOP。

最终预期的框架结构：



rpc-demo结构图

- ConsumerService、ProviderService是提供给client和server端使用的api接口。
- InterceptorChain：提供了RPC层面的AOP功能。日志、白名单过滤、权限认证等等。
- RpcContext：提供上线文，双端可以透明传输数据。
- Connector、Acceptor：网络模块，第一步自己用JavaSocket实现。

要求有了，下面第一步先整一个能跑起来的！

> 第一步先跑起来

先把我们预期能实现的功能摆出来：

> 基本调用链路畅通，能够传输基本类型、自定义业务类型、异常类型（要在客户端抛出）。

测试用的业务接口UserService:

```
/**
 * 测试用业务接口
 *
 * @author wqx
 *
 */
public interface UserService {

    /**
     * 基本链路测试
     *
     * @return
     */
    public String test();

    /**
     * 自定义业务类型测试
     *
     * @param userId
     * @return
     */
    public User queryUserById(int userId);

    /**
     * 异常测试
     *
     * @throws IOException
     */
    public Object exceptionTest() throws RpcException;
}
```

业务实现UserServiceImpl类：

```
/**
 * 测试业务接口实现类
 *
 * @author wqx
 *
 */
public class UserServiceImpl implements UserService {

    public String test() {
        return "hello client, this is rpc server.";
    }

    public User queryUserById(int userId) {
        User parent = new User(100,"小明爸爸");
        User child = new User(101,"小明同学");
        parent.addChild(child);
        return parent;
    }

    public Object exceptionTest() throws RpcException {
        throw new RpcException("exception occur in server! ! ! ");
    }
}
```

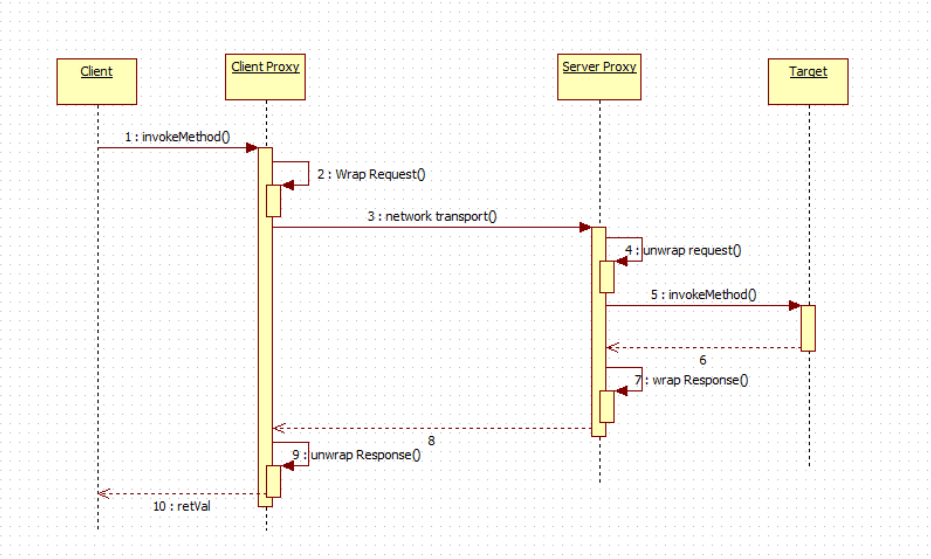
测试用的自定义业务类型User

```
/**
 * 测试用的自定义业务类型
 *
 * @author wqx
 *
 */
public class User implements java.io.Serializable{

    private static final long serialVersionUID = 49339440916323966L;

    private Integer id;

    private String name;

    private List<User> childs;


    public void addChild(User child){
        if(childs == null){
            childs = new ArrayList<User>();
        }
        childs.add(child);
    }
    //。。。getter setter
}
```

> 需求明确。。。键盘飞起。。。

基本步骤很简单，通过Proxy对客户端方法调用进行拦截，在代理对象的回调方法中，发起远程调用，网络模块先采用简单的Java提供的SocketAPI吧，对象的序列化和反序列化也是用JDK自带的功能实现。一切从简从速！！！

基本流程如下图：



rpc调用流程

1. invokeMethod:客户端调用目标对象的方法，被代理对象拦截。
2. wrap Request：封装调用参数（方法名，方法参数等），实现RpcRequest对象，并序列化。
3. network transport：网络传输，将序列化的参数对象传输到目标服务端。
4. unwrap request：对接收到的请求参数进行反序列化过程。
5. invokeMethod：通过反射机制method.invoke(obj,args)调用目标方法。
6. 将结果包装在RpcResponse对象中，进行序列化，为返回做准备。
7. 和步骤3一样，服务端通过网络将结果返回给客户端。

8. unwrap Response：反序列化，得到RpcResponse对象，从中获取结果retVal，并返回客户端。

过程很简单，下面开始实现第一个组件RpcBuilder，主要功能用来生成client端和server端的代理对象。

```
/**
 * rpc服务类
 *
 * @author wqx
 *
 */
public final class RpcBuilder {

    //构建客户端的代理对象
    public static Object buildRpcClient(final Class<?> interfaces,final String host,final
 int port){
        if(interfaces == null){
            throw new IllegalArgumentException("interfaces can not be null");
        }

        return Proxy.newProxyInstance(RpcBuilder.class.getClassLoader(), new Class<?>[]{i
nterfaces},
                new InvocationHandler(){

            //拦截目标方法->序列化method对象->发起socket连接
            public Object invoke(Object proxy, Method method,
                    Object[] args) throws Throwable {

                //创建连接,获取输入输出流
                Socket socket = new Socket(host,port);
                Object retVal = null;
                try{

                    ObjectOutputStream out = new ObjectOutputStream(socket.getOutputStrea
m());
                    ObjectInputStream in = new ObjectInputStream(socket.getInputStream())
;
                    try{
                        //构造请求参数对象
                        RpcRequest request = new RpcRequest(method.getName(), method.getP
arameterTypes(),args);
                        //发送
                        out.writeObject(request);

                        //接受server端的返回信息---阻塞
                        Object response = in.readObject();

                        if(response instanceof RpcResponse){
                            RpcResponse rpcResp  = (RpcResponse)response;
                            if(!rpcResp.isError()){
                                retVal = rpcResp.getResponseBody();
                            }else{
                                return new Throwable(rpcResp.getErrorMsg());
                            }
                        }
                    }finally{
                        out.close();
                        in.close();
                    }
                }finally{
                    socket.close();
                }
                return retVal;
            }
        });
    }

    private static int nThreads = Runtime.getRuntime().availableProcessors() * 2;
    private static ExecutorService handlerPool = Executors.newFixedThreadPool(nThreads);

    public static void buildRpcServer(final Object service, final int port) throws IOExce
ption{
        if (service == null)
            throw new IllegalArgumentException("service can not be null.");

        ServerSocket server = new ServerSocket(port);
        System.out.println("server started!!!");
        while(true){
            Socket socket = server.accept();//监听请求--阻塞

            //交由线程池异步处理
            handlerPool.submit(new Handler(service,socket));
```

```
        }
    }
    static class Handler implements Runnable{

        private Object service;

        private Socket socket;

        public Handler(Object service,Socket socket){
            this.service = service;
            this.socket = socket;
        }
        public void run() {
            try{
                ObjectInputStream in = null;
                ObjectOutputStream out = null;
                RpcResponse response = new RpcResponse();
                try {
                    in = new ObjectInputStream(socket.getInputStream());
                    out = new ObjectOutputStream(socket.getOutputStream());

                    Object req = in.readObject();
                    if(req instanceof RpcRequest){
                        RpcRequest rpcRequest = (RpcRequest)req;
                        Method method = service.getClass().getMethod(rpcRequest.getMethod
Name(), rpcRequest.getParameterTypes());
                        Object retVal = method.invoke(service, rpcRequest.getArgs());
                        response.setResponseBody(retVal);
                        out.writeObject(response);
                    }else{
                        throw new IllegalArgumentException("bad request!");
                    }
                } catch (Exception e) {
                    response.setErrorMsg(e.getMessage());
                    response.setResponseBody(e);
                    out.writeObject(response);
                }finally{
                    in.close();
                    out.close();
                }
            }catch(Exception e){}
        }
    }
}
```

其中每次发送请求包含的信息（方法名、参数名），将封装在RpcRequest中，实现如下：

```
/**
 * 封装请求参数
 *
 * @author wqx
 *
 */
public class RpcRequest implements Serializable
{
    /**
     *
     */
    private static final long serialVersionUID = -7102839100899303105L;

    //方法名
    private String methodName;

    //参数类型
    private Class<?>[] parameterTypes;

    //参数列表
    private Object[] args;

    public RpcRequest(String methodName,Class<?>[] parameterTypes,Object[] args)
    {
        this.methodName = methodName;
        this.parameterTypes = parameterTypes;
        this.args = args;
    }
    //getter and sette
```

服务端返回的执行结果封装在RpcResponse中，如下所示：

```
/**
 * 响应对象
 *
 * @author wqx
 *
 */
public class RpcResponse implements Serializable{

    static private final long serialVersionUID = -4364536436151723421L;

    //响应实体
    private Object responseBody;

    //错误信息
    private String errorMsg;

    public boolean isError(){
        return errorMsg == null ? false:true;
    }
    //getter and setter
}
```

自定义的业务异常RpcException：

```
/**
 * 自定义异常
 *
 * @author wqx
 *
 */
public class RpcException extends RuntimeException {

    private static final long serialVersionUID = -21578721570062083 60L;

    public RpcException(String msg){
        super(msg);
    }
}
```

client端测试代码：

```
// client端
public class ClientTest {

    private static String host = "127.0.0.1";
    private static int port = 8888;

    public static void main(String[] args) {

        UserService userService = (UserService) RpcBuilder.buildRpcClient(UserService.cla
ss, host, port);

        Object msg = null;
        try{
        msg = userService.test();//测试基本链路是否畅通
//        msg = userService.exceptionTest();//异常测试
//        msg = userService.queryUserById(0);//传输自定义业务类型
//            if(msg instanceof User){
//                System.out.println("parent:" + ((User)msg).getName());
//                System.out.println("child:" + ((User)msg).getChilds().get(0).getName())
;
//            }
           System.out.println("msg:" + msg);
        }catch(Exception e){
            System.out.println("errorMsg:" + e);
        }
    }
}
```

server端的测试代码：

```
public class ServerTest {

    private static int port = 8888;

    public static void main(String[] args) throws IOException {
        UserService userService = new UserServiceImpl();
         //暴露服务
        RpcBuilder.buildRpcServer(userService,port);
    }
}
```

测试test方法：预期输出：

```
msg : hello client, this is rpc server.
```

测试exceptionTest方法：

```
输出:
errorMsg:edu.ouc.rpc.RpcException: exception occur in server! ! !
```

测试queryUserById方法：

```
输出:
parent:小明爸爸
child:小明同学
```

done！！！
上述源码托管在github上 (https://github.com/TopGunViper/rpc-race/tree/feature_v1.0)

实现简单但问题多多

- 序列化方案（jdk原生序列化方案性能太差了）
- 网络传输方案（Socket BIO需要）
- 服务端方法执行时间很长怎么办？
- 异步调用如何实现？
- 。。。
  一个一个get！！！
- rpc系列1-10 minute Tutorial (http://www.jianshu.com/p/6016fb568452)
- rpc系列2-提供上下文RpcContext，客户端可以透传数据给服务端。
  (http://www.jianshu.com/p/96f8aa40d3b3)
- rpc系列3-支持异步调用，提供future、callback的能力。
  (http://www.jianshu.com/p/38dc67349799)
- rpc系列4-处理超时场景.及提供hook (http://www.jianshu.com/p/185773c9734a)
- rpc系列5-添加拦截器链，实现rpc层面的AOP
  (http://www.jianshu.com/p/64355d8cb1ee)

TopGun_Viper (/u/aee2b91e3398)
写了 23557 字，被 15 人关注，获得了 29 个喜欢
(/u/aee2b91e3398)
+关注

吾日三省吾code，可以为师矣。。。

如果觉得我的文章对您有用，请随意打赏。您的支持将鼓励我继续创作！

赞赏支持

♡ 喜欢 (/sign_in) | 6

更多分享

(http://cwb.assets.jianshu.io/notes/images/5751752

登录 (/sign_in) 后发表评论

1条评论 只看作者

按喜欢排序 按时间正序 按时间倒序

妮萌的小心思 (/u/e75db34e1cd7)
2楼 · 2016.09.18 20:33
(/u/e75db34e1cd7)
好高大上的感觉，虽然我不懂

赞 回复

被以下专题收入，发现更多相似内容

首页投稿 程序员 程序员的日常