

云社区

可歌可泣的江湖

登录 (https://account.aliyun.com/login/login.htm?from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin) | 注册

博客 (/articles)

问答 (/ask)

聚能聊 (/roundtable)

直播 (/webinar)

云课堂 (/edu/lesson)

论坛 (https://bbs.aliyun.com)

公众号 (/taams)

云计算 (/cloud)

大数据 (/big-data)

云生态 (/marketplace)

阿里开源 (/opensource)

专题 (/topic)

云栖神侠 (/expert)

APP (https://www.aliyun.com/app)

云栖社区 > 博客列表 (/articles) > 正文

一个轻量级分布式RPC框架--NettyRpc

ice.weibo.com/share/share.php?B%80%E4%B8%A7%E8%BD%E9%87%8F%E7%A7%E9%88%8C%E5%B8%83%E5%BC%8FRPC%E6%A1%86%E6%9E%B6-(RPC/tags/type_blog-tagid_10989) Spring (/tags/type_blog-tagid_10990) Netty (/tags/type_blog-tagid_10977) A2Netty%E5%92%8CZookeeper%E6%96%B9%E9%9D%A2

2016-05-20 18:57:11

浏览3074

评论0

zookeeper, (/tags/type_blog-tagid_10989/)

分布式, (/tags/type_blog-tagid_10990/)

ghost丶 (/users/1)

2304篇文章 (/users/1)

关注

摘要：1、背景 最近在搜索Netty和Zookeeper方面的文章时，看到了这篇文章《轻量级分布式 RPC 框架》，作者用Zookeeper、Netty和Spring写了一个轻量级的分布式RPC框架。花了一些时间看了下他的代码，写的干净简单，写的RPC框架可以算是一个简易版的dubbo。

文中提到的云产

1、背景

最近在搜索Netty和Zookeeper方面的文章时，看到了这篇文章《轻量级分布式 RPC 框架 (http://my.oschina.net/huangyong/blog/361751)》，作者用Zookeeper、Netty和Spring写了一个轻量级的分布式RPC框架。花了一些时间看了下他的代码，写的干净简单，写的RPC框架可以算是一个简易版的 dubbo (http://dubbo.io/)。这个RPC框架虽小，但是麻雀虽小，五脏俱全，有兴趣的可以学习一下。

本人在这个简易版的RPC上添加了如下特性：

- * 服务异步调用的支持，回调函数callback的支持
- * 客户端使用长连接（在多次调用共享连接）
- * 服务端异步多线程处理RPC请求

项目地址：<https://github.com/luxiaoxun/NettyRpc> (https://github.com/luxiaoxun/NettyRpc)

2、简介

RPC，即 Remote Procedure Call（远程过程调用），调用远程计算机上的服务，就像调用本地服务一样。RPC可以很好的解耦系统，如WebService就是一种基于Http协议的RPC。

这个RPC整体框架如下：

```
graph TD; Registry["Registry (ZooKeeper)"]; Client["RPC Client"]; Server["RPC Server"]; Client -- subscribe --> Registry; Server -- register --> Registry; Registry -- notify --> Client; Client -- request --> Server; Server -- response --> Client;
```

这个RPC框架使用的一些技术所解决的问题：

- 服务发布与订阅：服务端使用Zookeeper注册服务地址，客户端从Zookeeper获取可用的服务地址。
- 通信：使用Netty作为通信框架。
- Spring：使用Spring配置服务，加载Bean，扫描注解。
- 动态代理：客户端使用代理模式透明化服务调用。
- 消息编解码：使用Protostuff序列化和反序列化消息。

3、服务端发布服务

- 使用注解标注要发布的服务
- 服务注册

分布式关系型数据库

一种稳定、可靠、容量和伸缩的分布式关系型数据库

了解更多 (/go/1/31?pc)

企业级分布式应用

充分利用阿里云现有资源，引入中间件成熟的整套，以应用为中心，帮助企业建并...

了解更多 (/go/1/30?pc)

云服务器ECS (/g

为您提供简单高效、处理的计算服务，帮助您快速全的应用，提升运维效率

了解更多 (/go/1/3?pc)

更多文章共44586

https://yq.aliyun.com/articles/50480

1/10

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Component
public @interface RpcService {
    Class<?> value();
}
```

一个服务接口：

```
public interface HelloService {

    String hello(String name);

    String hello(Person person);
}
```

一个服务实现：使用注解标注

```
@RpcService(HelloService.class)
public class HelloServiceImpl implements HelloService {

    @Override
    public String hello(String name) {
        return "Hello! " + name;
    }

    @Override
    public String hello(Person person) {
        return "Hello! " + person.getFirstName() + " " + person.getLastName();
    }
}
```

服务在启动的时候扫描得到所有的服务接口及其实现：

```
@Override
public void setApplicationContext(ApplicationContext ctx) throws BeansException {
    Map<String, Object> serviceBeanMap = ctx.getBeansWithAnnotation(RpcService.class);
    if (MapUtils.isNotEmpty(serviceBeanMap)) {
        for (Object serviceBean : serviceBeanMap.values()) {
            String interfaceName = serviceBean.getClass().getAnnotation(RpcService.class).value().getName();
            handlerMap.put(interfaceName, serviceBean);
        }
    }
}
```

在Zookeeper集群上注册服务地址：

```


```

当年被马化腾和李里云如今营收超谷
(/articles/58514)

Docker with Spring
(/articles/54035)

[Android] 缓存机制
(/articles/52873)

[数据结构] 数组与转
区别 (/articles/528

[数据结构] 队列 (/a

[数据结构] 栈 (/arti

[数据结构] Hash表
冲突解决 (/articles/

[数据结构] 冒泡排
(/articles/52866)

[数据结构] 选择排
(/articles/52865)

[Android] SQLite的
(/articles/52864)

百款大数据产

1分钱尝鲜

查看详情

(<https://market.aliyun.com>)

让应用更加智能，阿
据产品1分钱尝鲜

版权声明

本社区内容由互联网用户
区不拥有所有权，也不享
。如果您发现本社区中有
，欢迎发送邮件：[zhaoy
ba-inc.com](mailto:zhaoba-inc.com) ([mailto:zha
baba-inc.com](mailto:zhaoba-inc.com)) 进行举报
据，一经查实，本社区将
权内容。

```

public class ServiceRegistry {

    private static final Logger LOGGER = LoggerFactory.getLogger(ServiceRegistry.class);

    private CountDownLatch latch = new CountDownLatch(1);

    private String registryAddress;

    public ServiceRegistry(String registryAddress) {
        this.registryAddress = registryAddress;
    }

    public void register(String data) {
        if (data != null) {
            ZooKeeper zk = connectServer();
            if (zk != null) {
                AddRootNode(zk); // Add root node if not exist
                createNode(zk, data);
            }
        }
    }

    private ZooKeeper connectServer() {
        ZooKeeper zk = null;
        try {
            zk = new ZooKeeper(registryAddress, Constant.ZK_SESSION_TIMEOUT, new Watcher() {
                @Override
                public void process(WatchedEvent event) {
                    if (event.getState() == Event.KeeperState.SyncConnected) {
                        latch.countDown();
                    }
                }
            });
            latch.await();
        } catch (IOException e) {
            LOGGER.error("", e);
        }
        catch (InterruptedException ex){
            LOGGER.error("", ex);
        }
        return zk;
    }

    private void AddRootNode(ZooKeeper zk){
        try {
            Stat s = zk.exists(Constant.ZK_REGISTRY_PATH, false);
            if (s == null) {
                zk.create(Constant.ZK_REGISTRY_PATH, new byte[0], ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.PERSISTENT);
            }
        } catch (KeeperException e) {
            LOGGER.error(e.toString());
        } catch (InterruptedException e) {
            LOGGER.error(e.toString());
        }
    }

    private void createNode(ZooKeeper zk, String data) {
        try {
            byte[] bytes = data.getBytes();
            String path = zk.create(Constant.ZK_DATA_PATH, bytes, ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL_SEQUENTIAL);
            LOGGER.debug("create zookeeper node ({} => {})", path, data);
        } catch (KeeperException e) {
            LOGGER.error("", e);
        }
        catch (InterruptedException ex){
            LOGGER.error("", ex);
        }
    }
}

```

这里在原文的基础上加了AddRootNode()判断服务父节点是否存在，如果不存在则添加一个PERSISTENT的服务父节点，这样虽然启动服务时多了点判断，但是不需要手动命令添加服务父节点了。

关于Zookeeper的使用原理，可以看这里《ZooKeeper基本原理 (<http://www.cnblogs.com/luxiaoxun/p/4887452.html>)》。

4、客户端调用服务

使用代理模式调用服务：



```

public class RpcProxy {

    private String serverAddress;
    private ServiceDiscovery serviceDiscovery;

    public RpcProxy(String serverAddress) {
        this.serverAddress = serverAddress;
    }

    public RpcProxy(ServiceDiscovery serviceDiscovery) {
        this.serviceDiscovery = serviceDiscovery;
    }

    @SuppressWarnings("unchecked")
    public <T> T create(Class<?> interfaceClass) {
        return (T) Proxy.newProxyInstance(
            interfaceClass.getClassLoader(),
            new Class<?>[]{interfaceClass},
            new InvocationHandler() {
                @Override
                public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                    RpcRequest request = new RpcRequest();
                    request.setRequestId(UUID.randomUUID().toString());
                    request.setClassName(method.getDeclaringClass().getName());
                    request.setMethodName(method.getName());
                    request.setParameterTypes(method.getParameterTypes());
                    request.setParameters(args);

                    if (serviceDiscovery != null) {
                        serverAddress = serviceDiscovery.discover();
                    }
                    if (serverAddress != null) {
                        String[] array = serverAddress.split(":");
                        String host = array[0];
                        int port = Integer.parseInt(array[1]);

                        RpcClient client = new RpcClient(host, port);
                        RpcResponse response = client.send(request);

                        if (response.isError()) {
                            throw new RuntimeException("Response error.", new Throwable(response.getError()));
                        } else {
                            return response.getResult();
                        }
                    } else {
                        throw new RuntimeException("No server address found!");
                    }
                }
            }
        );
    }
}

```

这里每次使用代理远程调用服务，从Zookeeper上获取可用的服务地址，通过RpcClient send一个Request，等待该Request的Response返回。这里原文有个比较严重的bug，在原文给出的简单的Test中是很难测出来的，原文使用了obj的wait和notifyAll来等待Response返回，会出现“假死等待”的情况：一个Request发送出去后，在obj.wait()调用之前可能Response就返回了，这时候在channelRead0里已经拿到了Response并且obj.notifyAll()已经在obj.wait()之前调用了，这时候send后再obj.wait()就出现了假死等待，客户端就一直等待在这里。使用CountDownLatch可以解决这个问题。

注意：这里每次调用的send时候才去和服务端建立连接，使用的是短连接，这种短连接在高并发时会有连接数问题，也会影响性能。

从Zookeeper上获取服务地址：



```

public class ServiceDiscovery {

    private static final Logger LOGGER = LoggerFactory.getLogger(ServiceDiscovery.class);

    private CountDownLatch latch = new CountDownLatch(1);

    private volatile List<String> dataList = new ArrayList<>();

    private String registryAddress;

    public ServiceDiscovery(String registryAddress) {
        this.registryAddress = registryAddress;
        ZooKeeper zk = connectServer();
        if (zk != null) {
            watchNode(zk);
        }
    }

    public String discover() {
        String data = null;
        int size = dataList.size();
        if (size > 0) {
            if (size == 1) {
                data = dataList.get(0);
                LOGGER.debug("using only data: {}", data);
            } else {
                data = dataList.get(ThreadLocalRandom.current().nextInt(size));
                LOGGER.debug("using random data: {}", data);
            }
        }
        return data;
    }

    private ZooKeeper connectServer() {
        ZooKeeper zk = null;
        try {
            zk = new ZooKeeper(registryAddress, Constant.ZK_SESSION_TIMEOUT, new Watcher() {
                @Override
                public void process(WatchedEvent event) {
                    if (event.getState() == Event.KeeperState.SyncConnected) {
                        latch.countDown();
                    }
                }
            });
            latch.await();
        } catch (IOException | InterruptedException e) {
            LOGGER.error("", e);
        }
        return zk;
    }

    private void watchNode(final ZooKeeper zk) {
        try {
            List<String> nodeList = zk.getChildren(Constant.ZK_REGISTRY_PATH, new Watcher() {
                @Override
                public void process(WatchedEvent event) {
                    if (event.getType() == Event.EventType.NodeChildrenChanged) {
                        watchNode(zk);
                    }
                }
            });
            List<String> dataList = new ArrayList<>();
            for (String node : nodeList) {
                byte[] bytes = zk.getData(Constant.ZK_REGISTRY_PATH + "/" + node, false, null);
                dataList.add(new String(bytes));
            }
            LOGGER.debug("node data: {}", dataList);
            this.dataList = dataList;
        } catch (KeeperException | InterruptedException e) {
            LOGGER.error("", e);
        }
    }
}

```

每次服务地址节点发生变化，都需要再次watchNode，获取新的服务地址列表。

5、消息编码

请求消息：

□



```
public class RpcRequest {

    private String requestId;
    private String className;
    private String methodName;
    private Class<?>[] parameterTypes;
    private Object[] parameters;

    public String getRequestId() {
        return requestId;
    }

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

    public String getClassName() {
        return className;
    }

    public void setClassName(String className) {
        this.className = className;
    }

    public String getMethodName() {
        return methodName;
    }

    public void setMethodName(String methodName) {
        this.methodName = methodName;
    }

    public Class<?>[] getParameterTypes() {
        return parameterTypes;
    }

    public void setParameterTypes(Class<?>[] parameterTypes) {
        this.parameterTypes = parameterTypes;
    }

    public Object[] getParameters() {
        return parameters;
    }


    public void setParameters(Object[] parameters) {
        this.parameters = parameters;
    }
}
```



响应消息:



```
public class RpcResponse {  
  
    private String requestId;  
    private String error;  
    private Object result;  
  
    public boolean isError() {  
        return error != null;  
    }  
  
    public String getRequestId() {  
        return requestId;  
    }  
  
    public void setRequestId(String requestId) {  
        this.requestId = requestId;  
    }  
  
    public String getError() {  
        return error;  
    }  
  
    public void setError(String error) {  
        this.error = error;  
    }  
  
    public Object getResult() {  
        return result;  
    }  
  
    public void setResult(Object result) {  
        this.result = result;  
    }  
}
```



消息序列化和反序列化工具：（基于 Protostuff 实现）



```

public class SerializationUtil {

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

    private static Objenesis objenesis = new ObjenesisStd(true);

    private SerializationUtil() {
    }

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

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

```

由于处理的是TCP消息，本人加了TCP的粘包处理Handler

```
channel.pipeline().addLast(new LengthFieldBasedFrameDecoder(65536,0,4,0,0))
```

消息编解码时开始4个字节表示消息的长度，也就是消息编码的时候，先写消息的长度，再写消息。

6、性能改进

Netty本身就是一个高性能的网络框架，从网络IO方面来说并没有太大的问题。

从这个RPC框架本身来说，在原文的基础上把Server端处理请求的过程改成了多线程异步：



相关文章

- RESTful_基础知识 (/articles/48854)
- 微服务技术栈选型，看了这个别的可以不用看了 (/articles/62569)
- 一分钟了解阿里云产品：分布式关系型数据库DRDS (/articles/7933)
- 新浪微博千万级规模高性能、高并发的网络架构经验分享 (/articles/66693)

网友评论

登录后可评论，请 [登录](https://account.aliyun.com/login/login.htm?from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin) ([https://account.aliyun.com/login/login.htm?](https://account.aliyun.com/login/login.htm?from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin)

评论

[from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin](https://account.aliyun.com/register/register.htm?from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin)) 或 [注册](#)

[关于我们 \(/www.aliyun.com/about\)](#)

[法律声明 \(/www.aliyun.com/about/law\)](#)

[廉正举报 \(https://jubao.alibaba.com/index.html?site=ALIYUN \(/www.aliyun.com/links\)\)](#)

[https://account.aliyun.com/register/register.htm?](https://account.aliyun.com/register/register.htm?from_type=yqclub&oauth_callback=https%3A%2F%2Fyq.aliyun.com%2Farticles%2F50480%3Fdo%3Dlogin)

 中国站

▼

搜索

阿里巴巴集团 (<http://www.alibabagroup.com/cn/global/home>) 淘宝网 (<http://www.taobao.com/>) 天猫 (<http://www.tmall.com/>) 聚划算 (<http://ju.taobao.com/>) 全球速卖通 (<http://www.aliexpress.com/>) 阿里巴巴国际交易市场 (<http://www.alibaba.com/>) 1688 (<http://www.1688.com/>) 阿里妈妈 (<http://www.alimama.com/index.htm>) 飞猪 (<http://www.alitrip.com/>) 阿里云计算 (<http://www.aliyun.com/compute>) YunOS (<http://www.yunos.com/>) 阿里通信 (<http://aliqin.tmall.com/>) 万网 (<http://wanwang.aliyun.com/>) 高德 (<http://www.autonavi.com/>) UC (<http://www.uc.cn/>) 友盟 (<http://www.umeng.cc>) 虾米 (<http://www.xiami.com/>) 阿里星球 (<http://www.alibabaplanet.com/>) 来往 (<http://www.laiwang.com/>) 钉钉 (<http://www.dingtalk.com/?lwfrom=20150205111943449>) 支付宝 (<https://www.alipay.com/>)

© 2009-2017 Aliyun.com 版权所有 ICP证：浙B2-20080101

 <http://verify.nic.xin/xinDetail/xinAuthInfoDetail?domainName=aliyun.xin>