crazyhzm 265

Dubbo源码解析（三十一）远程调用——rmi协议

java dubbo 阅读约 10 分钟

远程调用——rmi协议

> 目标：介绍rmi协议的设计和实现，介绍dubbo-rpc-rmi的源码。

前言

dubbo支持rmi协议，主要基于spring封装的org.springframework.remoting.rmi包来实现，当然最原始还是依赖 JDK 标准的java.rmi.* 包，采用阻塞式短连接和 JDK 标准序列化方式。关于rmi协议的介绍可以参考dubbo官方文档。

> 地址：http://dubbo.apache.org/zh-cn...

源码分析

（一）RmiRemoteInvocation

该类继承了RemoteInvocation，主要是在RemoteInvocation的基础上新增dubbo自身所需的附加值，避免这些附加值没有被传递，为了做一些验证处理。

```
public class RmiRemoteInvocation extends RemoteInvocation {
    private static final long serialVersionUID = 1L;
    private static final String dubboAttachmentsAttrName = "dubbo.attachments";

    /**
     * executed on consumer side
     */
    public RmiRemoteInvocation(MethodInvocation methodInvocation) {
        super(methodInvocation);
        // 添加dubbo附加值的属性
        addAttribute(dubboAttachmentsAttrName, new HashMap<String, String>
(RpcContext.getContext().getAttachments()));
    }

    /**
     * Need to restore context on provider side (Though context will be overridden by Invocation's attachment
     * when ContextFilter gets executed, we will restore the attachment when Invocation is constructed, check
more
     * 需要在提供者端恢复上下文（尽管上下文将被Invocation的附件覆盖
     * 当ContextFilter执行时，我们将在构造Invocation时恢复附件，检查更多
     * from {@link com.alibaba.dubbo.rpc.proxy.InvokerInvocationHandler}
     */
    @SuppressWarnings("unchecked")
    @Override
    public Object invoke(Object targetObject) throws NoSuchMethodException, IllegalAccessException,
            InvocationTargetException {
```

（二）RmiProtocol

该类继承了AbstractProxyProtocol类，是rmi协议实现的核心，跟其他协议一样，也实现了自己的服务暴露和服务引用方法。

1.doExport

```
@Override
protected <T> Runnable doExport(final T impl, Class<T> type, URL url) throws RpcException {
    // rmi暴露者
    final RmiServiceExporter rmiServiceExporter = new RmiServiceExporter();
    // 设置端口
    rmiServiceExporter.setRegistryPort(url.getPort());
    // 设置服务名称
    rmiServiceExporter.setServiceName(url.getPath());
    // 设置接口
    rmiServiceExporter.setServiceInterface(type);
    // 设置服务实现
    rmiServiceExporter.setService(impl);
    try {
        // 初始化bean的时候执行
        rmiServiceExporter.afterPropertiesSet();
    } catch (RemoteException e) {
        throw new RpcException(e.getMessage(), e);
    }
    return new Runnable() {
        @Override
        public void run() {
            try {
                // 销毁
                rmiServiceExporter.destroy();
            } catch (Throwable e) {
                logger.warn(e.getMessage(), e);
```

该方法是服务暴露的逻辑实现。

2.doRefer

```
@Override
@SuppressWarnings("unchecked")
protected <T> T doRefer(final Class<T> serviceType, final URL url) throws RpcException {
    // FactoryBean对于RMI代理，支持传统的RMI服务和RMI调用者，创建RmiProxyFactoryBean对象
    final RmiProxyFactoryBean rmiProxyFactoryBean = new RmiProxyFactoryBean();
    // RMI needs extra parameter since it uses customized remote invocation object
    // 检测版本
    if (url.getParameter(Constants.DUBBO_VERSION_KEY,
Version.getProtocolVersion()).equals(Version.getProtocolVersion())) {
        // Check dubbo version on provider, this feature only support
        // 设置RemoteInvocationFactory以用于此访问器
        rmiProxyFactoryBean.setRemoteInvocationFactory(new RemoteInvocationFactory() {
            @Override
            public RemoteInvocation createRemoteInvocation(MethodInvocation methodInvocation) {
                // 自定义调用工厂可以向调用添加更多上下文信息
                return new RmiRemoteInvocation(methodInvocation);
            }
        });
    }
    // 设置此远程访问者的目标服务的URL。URL必须与特定远程处理提供程序的规则兼容。
    rmiProxyFactoryBean.setServiceUrl(url.toIdentityString());
    // 设置要访问的服务的接口。界面必须适合特定的服务和远程处理策略
    rmiProxyFactoryBean.setServiceInterface(serviceType);
    // 设置是否在找到RMI存根后缓存它
    rmiProxyFactoryBean.setCacheStub(true);
    // 设置是否在启动时查找RMI存根
```

该方法是服务引用的逻辑实现。

后记

> 该部分相关的源码解析地址：https://github.com/CrazyHZM/i...

该文章讲解了远程调用中关于rmi协议实现的部分，逻辑比较简单。接下来我将开始对rpc模块关于thrift协议部分进行讲解。