crazyhzm 265

Dubbo源码解析（二十五）远程调用——hessian协议

java dubbo 阅读约 23 分钟

远程调用——hessian协议

> 目标：介绍远程调用中跟hessian协议相关的设计和实现，介绍dubbo-rpc-hessian的源码。

前言

本文讲解多是dubbo集成的第二种协议，hessian协议，Hessian 是 Caucho 开源的一个 RPC 框架，其通讯效率高于 WebService 和 Java 自带的序列化。dubbo集成hessian所提供的hessian协议相关介绍可以参考官方文档，我就不再赘述。

> 文档地址：http://dubbo.apache.org/zh-cn...

源码分析

（一）DubboHessianURLConnectionFactory

该类继承了HessianURLConnectionFactory类，是dubbo，用于创建与服务器的连接的内部工厂，重写了父类中open方法。

```
public class DubboHessianURLConnectionFactory extends HessianURLConnectionFactory {

    /**
     * 打开与HTTP服务器的新连接或循环连接
     * @param url
     * @return
     * @throws IOException
     */
    @Override
    public HessianConnection open(URL url) throws IOException {
        // 获得一个连接
        HessianConnection connection = super.open(url);
        // 获得上下文
        RpcContext context = RpcContext.getContext();
        for (String key : context.getAttachments().keySet()) {
            // 在http协议头里面加入dubbo中附加值，key为 header+key  value为附加值的value
            connection.addHeader(Constants.DEFAULT_EXCHANGER + key, context.getAttachment(key));
        }

        return connection;
    }
}
```

在hessian上加入dubbo自己所需要的附加值，放到协议头里面进行发送。

（二）HttpClientConnection

该类是基于HttpClient封装来实现HessianConnection接口，其中逻辑比较简单。

```
public class HttpClientConnection implements HessianConnection {

    /**
     * http客户端对象
     */
    private final HttpClient httpClient;

    /**
     * 字节输出流
     */
    private final ByteArrayOutputStream output;

    /**
     * http post请求对象
     */
    private final HttpPost request;

    /**
     * http 响应对象
     */
    private volatile HttpResponse response;

    public HttpClientConnection(HttpClient httpClient, URL url) {
        this.httpClient = httpClient;
        this.output = new ByteArrayOutputStream();
        this.request = new HttpPost(url.toString());
```

（三）HttpClientConnectionFactory

该类实现了HessianConnectionFactory接口，是创建HttpClientConnection的工厂类。该类的实现跟DubboHessianURLConnectionFactory类类似，但是DubboHessianURLConnectionFactory是标准的Hessian接口调用会采用的工厂类，而HttpClientConnectionFactory是Dubbo 的 Hessian 协议调用。当然Dubbo 的 Hessian 协议也是基于http的。

```
public class HttpClientConnectionFactory implements HessianConnectionFactory {

    /**
     * httpClient对象
     */
    private final HttpClient httpClient = new DefaultHttpClient();

    @Override
    public void setHessianProxyFactory(HessianProxyFactory factory) {
        // 设置连接超时时间
        HttpConnectionParams.setConnectionTimeout(httpClient.getParams(), (int) factory.getConnectTimeout());
        // 设置读取数据时阻塞链路的超时时间
        HttpConnectionParams.setSoTimeout(httpClient.getParams(), (int) factory.getReadTimeout());
    }


    @Override
    public HessianConnection open(URL url) throws IOException {
        // 创建一个HttpClientConnection实例
        HttpClientConnection httpClientConnection = new HttpClientConnection(httpClient, url);
        // 获得上下文，用来获得附加值
        RpcContext context = RpcContext.getContext();
        // 遍历附加值，放入到协议头里面
        for (String key : context.getAttachments().keySet()) {
            httpClientConnection.addHeader(Constants.DEFAULT_EXCHANGER + key, context.getAttachment(key));
        }
```

实现了两个方法，第一个方法是给http连接设置两个参数配置，第二个方法是创建一个连接。

（四）HessianProtocol

该类继承了AbstractProxyProtocol类，是hessian协议的实现类。其中实现类基于hessian协议的服务引用、服务暴露等方法。

1.属性

```
/**
 * http服务器集合
 * key为ip：port
 */
private final Map<String, HttpServer> serverMap = new ConcurrentHashMap<String, HttpServer>();

/**
 * HessianSkeleto  集合
 * key为服务名
 */
private final Map<String, HessianSkeleton> skeletonMap = new ConcurrentHashMap<String, HessianSkeleton>();

/**
 * HttpBinder对象，默认是jetty实现
 */
private HttpBinder httpBinder;
```

2.doExport

```
@Override
protected <T> Runnable doExport(T impl, Class<T> type, URL url) throws RpcException {
    // 获得ip地址
    String addr = getAddr(url);
    // 获得http服务器对象
    HttpServer server = serverMap.get(addr);
    // 如果为空，则重新创建一个server，然后放入集合
    if (server == null) {
        server = httpBinder.bind(url, new HessianHandler());
        serverMap.put(addr, server);
    }
    // 获得服务path
    final String path = url.getAbsolutePath();
    // 创建Hessian服务端对象
    final HessianSkeleton skeleton = new HessianSkeleton(impl, type);
    // 加入集合
    skeletonMap.put(path, skeleton);

    // 获得通用的path
    final String genericPath = path + "/" + Constants.GENERIC_KEY;
    // 加入集合
    skeletonMap.put(genericPath, new HessianSkeleton(impl, GenericService.class));

    // 返回一个线程
    return new Runnable() {
        @Override
```

该方法是服务暴露的主要逻辑实现。

3.doRefer

```
@Override
@SuppressWarnings("unchecked")
protected <T> T doRefer(Class<T> serviceType, URL url) throws RpcException {
    // 获得泛化的参数
    String generic = url.getParameter(Constants.GENERIC_KEY);
    // 是否是泛化调用
    boolean isGeneric = ProtocolUtils.isGeneric(generic) || serviceType.equals(GenericService.class);
    // 如果是泛化调用。则设置泛化的path和附加值
    if (isGeneric) {
        RpcContext.getContext().setAttachment(Constants.GENERIC_KEY, generic);
        url = url.setPath(url.getPath() + "/" + Constants.GENERIC_KEY);
    }

    // 创建代理工厂
    HessianProxyFactory hessianProxyFactory = new HessianProxyFactory();
    // 是否是Hessian2的请求 默认为否
    boolean isHessian2Request = url.getParameter(Constants.HESSIAN2_REQUEST_KEY,
Constants.DEFAULT_HESSIAN2_REQUEST);
    // 设置是否应使用Hessian协议的版本2来解析请求
    hessianProxyFactory.setHessian2Request(isHessian2Request);
    // 是否应为远程调用启用重载方法，默认为否
    boolean isOverloadEnabled = url.getParameter(Constants.HESSIAN_OVERLOAD_METHOD_KEY,
Constants.DEFAULT_HESSIAN_OVERLOAD_METHOD);
    // 设置是否应为远程调用启用重载方法。
    hessianProxyFactory.setOverloadEnabled(isOverloadEnabled);
    // 获得client实现方式，默认为jdk
```

该方法是服务引用的主要逻辑实现，根据客户端配置，来选择标准 Hessian 接口调用还是Dubbo 的 Hessian 协议调用。

4.getErrorCode

```
@Override
protected int getErrorCode(Throwable e) {
    // 如果属于HessianConnectionException异常
    if (e instanceof HessianConnectionException) {
        if (e.getCause() != null) {
            Class<?> cls = e.getCause().getClass();
            // 如果属于超时异常，则返回超时异常
            if (SocketTimeoutException.class.equals(cls)) {
                return RpcException.TIMEOUT_EXCEPTION;
            }
        }
        // 否则返回网络异常
        return RpcException.NETWORK_EXCEPTION;
    } else if (e instanceof HessianMethodSerializationException) {
        // 序列化异常
        return RpcException.SERIALIZATION_EXCEPTION;
    }
    return super.getErrorCode(e);
}
```

该方法是针对异常的处理。

5.HessianHandler

```
private class HessianHandler implements HttpHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        // 获得请求的uri
        String uri = request.getRequestURI();
        // 获得对应的HessianSkeleton对象
        HessianSkeleton skeleton = skeletonMap.get(uri);
        // 如果如果不是post方法
        if (!request.getMethod().equalsIgnoreCase("POST")) {
            // 返回状态设置为500
            response.setStatus(500);
        } else {
            // 设置远程地址
            RpcContext.getContext().setRemoteAddress(request.getRemoteAddr(), request.getRemotePort());

            // 获得请求头内容
            Enumeration<String> enumeration = request.getHeaderNames();
            // 遍历请求头内容
            while (enumeration.hasMoreElements()) {
                String key = enumeration.nextElement();
                // 如果key开头是deader，则把附加值取出来放入上下文
                if (key.startsWith(Constants.DEFAULT_EXCHANGER)) {
                    RpcContext.getContext().setAttachment(key.substring(Constants.DEFAULT_EXCHANGER.length()),
                            request.getHeader(key));
```

该内部类是Hessian的处理器，用来处理请求中的协议头内容。

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hessian协议的部分，内容比较简单，可以参考着官方文档了解一下。接下来我将开始对rpc模块关于hessian协议部分进行讲解。

阅读 925 • 更新于 11月8日

crazyhzm
265

关注作者

0 条评论

得票 • 时间

撰写评论 ...

提交评论

推荐阅读

聊聊Dubbo - Dubbo可扩展机制源码解析

摘要： 在Dubbo可扩展机制实战中，我们了解了Dubbo扩展机制的一些概念，初探了Dubbo中LoadBalance的实现，并自己实现了...

猫耳 • 阅读 21