crazyhzm 265

Dubbo源码解析（二十六）远程调用——http协议

java dubbo 阅读约 19 分钟

远程调用——http协议

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

前言

基于HTTP表单的远程调用协议，采用 Spring 的HttpInvoker实现，关于http协议就不用多说了吧。

源码分析

（一）HttpRemoteInvocation

该类继承了RemoteInvocation类，是在RemoteInvocation上增加了泛化调用的参数设置，以及增加了dubbo本身需要的附加值设置。

```
public class HttpRemoteInvocation extends RemoteInvocation {

    private static final long serialVersionUID = 1L;
    /**
     * dubbo的附加值名称
     */
    private static final String dubboAttachmentsAttrName = "dubbo.attachments";

    public HttpRemoteInvocation(MethodInvocation methodInvocation) {
        super(methodInvocation);
        // 把附加值加入到会话域的属性里面
        addAttribute(dubboAttachmentsAttrName, new HashMap<String, String>
(RpcContext.getContext().getAttachments()));
    }

    @Override
    public Object invoke(Object targetObject) throws NoSuchMethodException, IllegalAccessException,
            InvocationTargetException {
        // 获得上下文
        RpcContext context = RpcContext.getContext();
        // 获得附加值
        context.setAttachments((Map<String, String>) getAttribute(dubboAttachmentsAttrName));

        // 泛化标志
        String generic = (String) getAttribute(Constants.GENERIC_KEY);
        // 如果不为空，则设置泛化标志
```

（二）HttpProtocol

该类是http实现的核心，跟我在《dubbo源码解析（二十五）远程调用——hessian协议》中讲到的HessianProtocol实现有很多地方相似。

1.属性

```
/**
 * 默认的端口号
 */
public static final int DEFAULT_PORT = 80;

/**
 * http服务器集合
 */
private final Map<String, HttpServer> serverMap = new ConcurrentHashMap<String, HttpServer>();

/**
 * Spring HttpInvokerServiceExporter 集合
 */
private final Map<String, HttpInvokerServiceExporter> skeletonMap = new ConcurrentHashMap<String,
HttpInvokerServiceExporter>();

/**
 * HttpBinder对象
 */
private HttpBinder httpBinder;
```

2.doExport

```
@Override
protected <T> Runnable doExport(final T impl, Class<T> type, URL url) throws RpcException {
    // 获得ip地址
    String addr = getAddr(url);
    // 获得http服务器
    HttpServer server = serverMap.get(addr);
    // 如果服务器为空，则重新创建服务器，并且加入到集合
    if (server == null) {
        server = httpBinder.bind(url, new InternalHandler());
        serverMap.put(addr, server);
    }

    // 获得服务path
    final String path = url.getAbsolutePath();

    // 加入集合
    skeletonMap.put(path, createExporter(impl, type));

    // 通用path
    final String genericPath = path + "/" + Constants.GENERIC_KEY;

    // 添加泛化的服务调用
    skeletonMap.put(genericPath, createExporter(impl, GenericService.class));
    return new Runnable() {
        @Override
        public void run() {
```

该方法是暴露服务等逻辑，因为dubbo实现http协议采用了Spring 的HttpInvoker实现，所以调用了createExporter方法来创建创建HttpInvokerServiceExporter。

3.createExporter

```
private <T> HttpInvokerServiceExporter createExporter(T impl, Class<?> type) {
    // 创建HttpInvokerServiceExporter
    final HttpInvokerServiceExporter httpServiceExporter = new HttpInvokerServiceExporter();
    // 设置要访问的服务的接口
    httpServiceExporter.setServiceInterface(type);
    // 设置服务实现
    httpServiceExporter.setService(impl);
    try {
        // 在BeanFactory设置了所有提供的bean属性，初始化bean的时候执行，可以针对某个具体的bean进行配
        httpServiceExporter.afterPropertiesSet();
    } catch (Exception e) {
        throw new RpcException(e.getMessage(), e);
    }
    return httpServiceExporter;
}
```

该方法是创建一个spring 的HttpInvokerServiceExporter。

4.doRefer

```
@Override
@SuppressWarnings("unchecked")
protected <T> T doRefer(final Class<T> serviceType, final URL url) throws RpcException {
    // 获得泛化配置
    final String generic = url.getParameter(Constants.GENERIC_KEY);
    // 是否为泛化调用
    final boolean isGeneric = ProtocolUtils.isGeneric(generic) || serviceType.equals(GenericService.class);

    // 创建HttpInvokerProxyFactoryBean
    final HttpInvokerProxyFactoryBean httpProxyFactoryBean = new HttpInvokerProxyFactoryBean();
    // 设置RemoteInvocation的工厂类
    httpProxyFactoryBean.setRemoteInvocationFactory(new RemoteInvocationFactory() {
        /**
         * 为给定的AOP方法调用创建一个新的RemoteInvocation对象。
         * @param methodInvocation
         * @return
         */
        @Override
        public RemoteInvocation createRemoteInvocation(MethodInvocation methodInvocation) {
            // 新建一个HttpRemoteInvocation
            RemoteInvocation invocation = new HttpRemoteInvocation(methodInvocation);
            // 如果是泛化调用
            if (isGeneric) {
                // 设置标志
                invocation.addAttribute(Constants.GENERIC_KEY, generic);
            }
```

该方法是服务引用的方法，其中根据url配置simple还是commons来选择创建连接池的方式。其中的区别就是SimpleHttpInvokerRequestExecutor使用的是JDK HttpClient，HttpComponentsHttpInvokerRequestExecutor 使用的是Apache HttpClient。

5.getErrorCode

```
@Override
protected int getErrorCode(Throwable e) {
    if (e instanceof RemoteAccessException) {
        e = e.getCause();
    }
    if (e != null) {
        Class<?> cls = e.getClass();
        if (SocketTimeoutException.class.equals(cls)) {
            // 返回超时异常
            return RpcException.TIMEOUT_EXCEPTION;
        } else if (IOException.class.isAssignableFrom(cls)) {
            // 返回网络异常
            return RpcException.NETWORK_EXCEPTION;
        } else if (ClassNotFoundException.class.isAssignableFrom(cls)) {
            // 返回序列化异常
            return RpcException.SERIALIZATION_EXCEPTION;
        }
    }
    return super.getErrorCode(e);
}
```

该方法是处理异常情况，设置错误码。

6.InternalHandler

```
private class InternalHandler implements HttpHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {
        // 获得请求uri
        String uri = request.getRequestURI();
        // 获得服务暴露者HttpInvokerServiceExporter对象
        HttpInvokerServiceExporter skeleton = skeletonMap.get(uri);
        // 如果不是post，则返回码设置500
        if (!request.getMethod().equalsIgnoreCase("POST")) {
            response.setStatus(500);
        } else {
            // 远程地址放到上下文
            RpcContext.getContext().setRemoteAddress(request.getRemoteAddr(), request.getRemotePort());
            try {
                // 调用下一个调用
                skeleton.handleRequest(request, response);
            } catch (Throwable e) {
                throw new ServletException(e);
            }
        }
    }

}
```

该内部类实现了HttpHandler，做了设置远程地址的逻辑。

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http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injvm本地调用部分进行讲解。