crazyhzm 265

Dubbo源码解析（二十七）远程调用——injvm本地调用

dubbo java 阅读约 12 分钟

远程调用——injvm本地调用

> 目标：介绍injvm本地调用的设计和实现，介绍dubbo-rpc-injvm的源码。

前言

dubbo是一个远程调用的框架，但是它没有理由不支持本地调用，本文就要讲解dubbo关于本地调用的实现。本地调用要比远程调用简单的多。

源码分析

（一）InjvmExporter

该类继承了AbstractExporter，是本地服务的暴露者封装，其中实现比较简单。只是实现了unexport方法，并且维护了一份保存暴露者的集合。

```
class InjvmExporter<T> extends AbstractExporter<T> {

    /**
     * 服务key
     */
    private final String key;

    /**
     * 暴露者集合
     */
    private final Map<String, Exporter<?>> exporterMap;

    InjvmExporter(Invoker<T> invoker, String key, Map<String, Exporter<?>> exporterMap) {
        super(invoker);
        this.key = key;
        this.exporterMap = exporterMap;
        exporterMap.put(key, this);
    }

    /**
     * 取消暴露
     */
    @Override
    public void unexport() {
        // 调用父类的取消暴露方法
        super.unexport();
```

（二）InjvmInvoker

该类继承了AbstractInvoker类，是本地调用的invoker实现。

```
class InjvmInvoker<T> extends AbstractInvoker<T> {

    /**
     * 服务key
     */
    private final String key;

    /**
     * 暴露者集合
     */
    private final Map<String, Exporter<?>> exporterMap;

    InjvmInvoker(Class<T> type, URL url, String key, Map<String, Exporter<?>> exporterMap) {
        super(type, url);
        this.key = key;
        this.exporterMap = exporterMap;
    }

    /**
     * 服务是否活跃
     * @return
     */
    @Override
    public boolean isAvailable() {
        InjvmExporter<?> exporter = (InjvmExporter<?>) exporterMap.get(key);
        if (exporter == null) {
```

其中重写了isAvailable和doInvoke方法。

（三）InjvmProtocol

该类是本地调用的协议实现，其中实现了服务调用和服务暴露方法，并且封装了一个判断是否是本地调用的方法。

1.属性

```
/**
 * 本地调用  Protocol的实现类key
 */
public static final String NAME = Constants.LOCAL_PROTOCOL;

/**
 * 默认端口
 */
public static final int DEFAULT_PORT = 0;
/**
 * 单例
 */
private static InjvmProtocol INSTANCE;
```

2.getExporter

```
static Exporter<?> getExporter(Map<String, Exporter<?>> map, URL key) {
    Exporter<?> result = null;

    // 如果服务key不是*
    if (!key.getServiceKey().contains("*")) {
        // 直接从集合中取出
        result = map.get(key.getServiceKey());
    } else {
        // 如果 map不为空，则遍历暴露者，来找到对应的exporter
        if (map != null && !map.isEmpty()) {
            for (Exporter<?> exporter : map.values()) {
                // 如果是服务key
                if (UrlUtils.isServiceKeyMatch(key, exporter.getInvoker().getUrl())) {
                    // 赋值
                    result = exporter;
                    break;
                }
            }
        }
    }

    // 如果没有找到exporter
    if (result == null) {
        // 则返回null
        return null;
    } else if (ProtocolUtils.isGeneric(
```

该方法是获得相关的暴露者。

3.export

```
@Override
public <T> Exporter<T> export(Invoker<T> invoker) throws RpcException {
    // 创建InjvmExporter 并且返回
    return new InjvmExporter<T>(invoker, invoker.getUrl().getServiceKey(), exporterMap);
}
```

4.refer

```
@Override
public <T> Invoker<T> refer(Class<T> serviceType, URL url) throws RpcException {
    // 创建InjvmInvoker 并且返回
    return new InjvmInvoker<T>(serviceType, url, url.getServiceKey(), exporterMap);
}
```

5.isInjvmRefer

```
public boolean isInjvmRefer(URL url) {
    final boolean isJvmRefer;
    // 获得scope配置
    String scope = url.getParameter(Constants.SCOPE_KEY);
    // Since injvm protocol is configured explicitly, we don't need to set any extra flag, use normal refer
process.
    if (Constants.LOCAL_PROTOCOL.toString().equals(url.getProtocol())) {
        // 如果是injvm，则不是本地调用
        isJvmRefer = false;
    } else if (Constants.SCOPE_LOCAL.equals(scope) || (url.getParameter("injvm", false))) {
        // if it's declared as local reference
        // 'scope=local' is equivalent to 'injvm=true', injvm will be deprecated in the future release
        // 如果它被声明为本地引用 scope = local'相当于'injvm = true'，将在以后的版本中弃用injvm
        isJvmRefer = true;
    } else if (Constants.SCOPE_REMOTE.equals(scope)) {
        // it's declared as remote reference
        // 如果被声明为远程调用
        isJvmRefer = false;
    } else if (url.getParameter(Constants.GENERIC_KEY, false)) {
        // generic invocation is not local reference
        // 泛化的调用不是本地调用
        isJvmRefer = false;
    } else if (getExporter(exporterMap, url) != null) {
        // by default, go through local reference if there's the service exposed locally
        // 默认情况下，如果本地暴露服务，请通过本地引用
        isJvmRefer = true;
```

该方法是判断是否为本地调用。

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injvm本地调用的部分，三种抽象的角色还是比较鲜明的，服务暴露相关的exporter、服务引用相关的invoker、以及协议相关的protocol，关键还是弄清楚再设计上的意图，以及他们分别代表的是什么。那么看这些不同的协议实现会很容易看懂。接下来我将开始对rpc模块关于memcached协议部分进行讲解。

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

Quartz调度源码分析

上一篇文章Quartz数据库表分析介绍了Quartz默认提供的11张表，本文将具体分析Quartz是如何调度的，是如何通过数据库的方式...