10 RPC 框架代码分析之注册中心模块

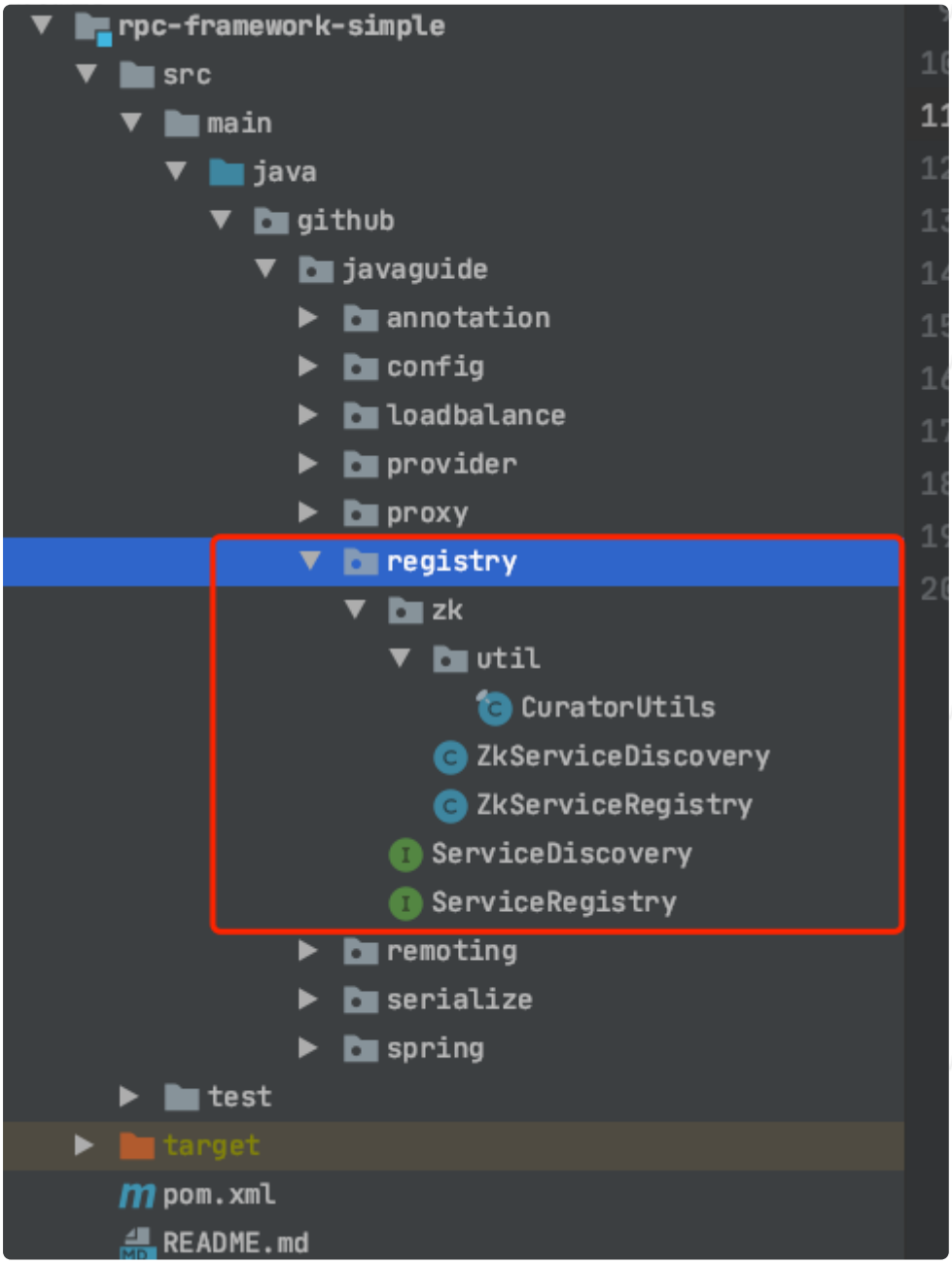
10 RPC 框架代码分析之注册中心模块

我们之前在“如何自己实现一个 RPC 框架？”这篇文章中介绍到说：**注册中心负责服务地址的注册与查找，相当于目录服务。** 服务端启动的时候将服务名称及其对应的地址(ip+port)注册到注册中心，服务消费端根据服务名称找到对应的服务地址。有了服务地址之后，服务消费端就可以通过网络请求服务端了。

简单来说注册中心就像是一个中转站，提供的作用就是根据调用的服务名称找到远程服务的地址（数据保存服务）。

注册中心模块整体结构如下：

公众号JavaGuide



我们定义了两个接口 `ServiceDiscovery.java` 和 `ServiceRegistry.java`，这两个接口分别定义了服务发现和服务注册行为。

`ServiceRegistry.java`

```java
/**
 * 服务注册
 *
 * @author shuang.kou
 * @createTime 2020年05月13日 08:39:00
 */
public interface ServiceRegistry {
    /**
     * 注册服务到注册中心
     *
     * @param rpcServiceName    完整的服务名称 (class name+group+version)
     * @param inetSocketAddress 远程服务地址
     */
    void registerService(String rpcServiceName, InetSocketAddress inetSocketAddress);

}
```

ServiceDiscovery.java

```java
/**
 * 服务发现
 *
 * @author shuang.kou
 * @createTime 2020年06月01日 15:16:00
 */
public interface ServiceDiscovery {
    /**
     * 根据 rpcServiceName 获取远程服务地址
     *
     * @param rpcServiceName 完整的服务名称 (class name+group+version)
     * @return 远程服务地址
     */
    InetSocketAddress lookupService(String rpcServiceName);
}
```

接下来，我们使用 ZooKeeper 作为注册中心的实现方式，并实现了这两个接口。

ZkServiceRegistry.java

```java
/**
 * 服务注册（基于zookeeper实现）
 */
@Slf4j
public class ZkServiceRegistry implements ServiceRegistry {

    @Override
    public void registerService(String rpcServiceName, InetSocketAddress inetSocketAddress) {
        String servicePath = CuratorUtils.ZK_REGISTER_ROOT_PATH + "/" + rpcServiceName + inetSocketAddress.toString();
        CuratorFramework zkClient = CuratorUtils.getZkClient();
        CuratorUtils.createPersistentNode(zkClient, servicePath);
    }
}
```

当我们的服务被注册进 zookeeper 的时候，我们将完整的服务名称 rpcServiceName （class name+group+version）作为根节点，子节点是对应的服务地址（ip+端口号）。

- `class name`：服务接口名也就是类名比如：`github.javaguide.HelloService`。
- `version`：（服务版本）主要是为后续不兼容升级提供可能
- `group`:主要用于处理一个接口有多个类实现的情况。

一个根节点（rpcServiceName）可能会对应多个服务地址（相同服务被部署多份的情况）。

如果我们要获得某个服务对应的地址的话，就直接根据完整的服务名称来获取到其下的所有子节点，然后通过具体的负载均衡策略取出一个就可以了。相关代码如下在 `ZkServiceDiscovery.java` 中已经给出。

`ZkServiceDiscovery.java`

```java
/**
 * 服务发现（基于zookeeper实现）
 */
@Slf4j
public class ZkServiceDiscovery implements ServiceDiscovery {
    private final LoadBalance loadBalance;

    public ZkServiceDiscovery() {
        this.loadBalance = new RandomLoadBalance();
    }

    @Override
    public InetSocketAddress lookupService(String rpcServiceName) {
        CuratorFramework zkClient = CuratorUtils.getZkClient();
        List<String> serviceUrlList = CuratorUtils.getChildrenNodes(zkClient, rpcServiceName);
        if (serviceUrlList.size() == 0) {
            throw new RpcException(RpcErrorMessage.SERVICE_CAN_NOT_BE_FOUND, rpcServiceName);
        }
        // load balancing
        String targetServiceUrl = loadBalance.selectServiceAddress(serviceUrlList);
        log.info("Successfully found the service address:[{}]", targetServiceUrl);
        String[] socketAddressArray = targetServiceUrl.split(":");
        String host = socketAddressArray[0];
        int port = Integer.parseInt(socketAddressArray[1]);
        return new InetSocketAddress(host, port);
    }
}
```

我们根据完整的服务名称便可以将对应的服务地址查出来， 查出来的服务地址可能并不止一个。

所以，我们可以通过对应的负载均衡策略来选择出一个服务地址。

`CuratorUtils.java`

另外，我们还自定义了一个 ZooKeeper Java 客户端 Curtor 的工具类 `CuratorUtils.java` 。关于这个工具类，这里就不再提了。

在《08 Zookeeper 常用命令+ Curtor 使用详解》中已经介绍的非常详细了。