# 基本概念

与一般的RPC框架不同，MAC-RPC采用了类似消息队列的远程调用机制，能够减少了阻塞和线程切换开销，提高CPU的使用效率。再加上基于JAVA AIO的全异步的底层网络通讯封装，使得MAC-RPC有着优异的性能表现和与众不同的能力，尤其是在对异步方法调用的支持上。

在使用上，除了必须依赖spring，基本上实现了对应用“零侵入”。

## 服务消费者与提供者

在基于远程方法调用（RPC）的微服务架构设计理念中，一个可供远程调用的方法即被视为一个服务，远程方法调用的发起方（调用方）就是这一**服务的消费者（Consumer）**，被调用方（服务方）就是这一**服务的提供者（Provider）**。

通常一个应用可能会暴露多个可供远程调用的方法，也可能会需要调用多个远程方法。这要求本地应用持有这些远程方法的本地引用。MAC-RPC采用JAVA的动态代理机制，为每一个需要引用的远程方法在本地动态创建对应的“引用”——动态代理对象（Proxy）。

应用是服务的“容器”，应用的实例被称为**节点（Node）**。

## 服务注册与发现

在基于MAC-RPC的微服务环境中，一个应用的一个实例就是一个节点。节点之间通过一个叫做“注册中心”地方交换服务信息，实现服务层面的服务注册与发现。

MAC-RPC采用所谓的“弱中心化”设计，任何一个节点均可以被标记为**主节点（master）**,而主节点在MAC-RPC中就扮演了服务的注册中心的角色，并且可以有多个。普通的RPC节点连接到其中任意一个master节点就可以与其它所有节点连通。

## 服务四要素

以下四个要素用以确定一个服务：

group：服务的分组

name：服务的名称或编号

interface：服务的接口

version：服务的版本

# 配置说明

## 基本配置（config.properties）

必须的配置项如下：

#mac-rpc版本配置，目前固定为1.0.0

rpc.version=1.0.0

#指定本机IP地址，可选，仅当无法正确获取IP时配置

#rpc.ip=20.3.6.172

#指定监听的端口

rpc.port=9901

补充说明：

* 只有当两个版本相同的MAC-RPC应用才可以互相调用
* rpc.ip应为本机要启用的IP地址，默认是自动获取，可以不配。但当本机有多个IP地址时，自动获取的IP可能不是期望的那一个。
* rpc.port是必需的，并且相对同一个IP，必须是唯一的。

## 注册中心配置（nodes.xml）

nodes.xml通常是用来配置被作为注册中心（master）的节点。一个分布式环境至少需要一个master节点。也可以在nodes.xml中配置当前环境中的非master节点。

示例如下：

<nodes>

<node address=*"192.168.1.11:9901"* master=*"true"* />

</nodes>

其中address应该是主节点（master）的IP地址和监听的端口，也可以是本地HOSTS中映射的主机名，也可以是域名或机器名。如果只在本机单机环境运行测试还可以是127.0.0.1或localhost。

## 服务提供者配置（provide.xml）

示例如下：

<services>

<service group=*"cache"* name=*"cacheCatalog"* ref=*"cacheMonitor"*

interface=*"com.boarsoft.cache.config.CacheCatalog"* version=*"1.0.0"*>

</service>

</services>

除了服务四要素外，作为服务提供者配置项还有：

|  |  |  |
| --- | --- | --- |
| **属性** | **含义** | **说明** |
| ref | 实现此服务的spring bean 的id | bean必须实现指定的接口（interface），解析是会生成一个实现此接口的代理类，来调用这个bean |
|  |  |  |

## 服务消费者配置（consume.xml）

### 基本配置

示例如下：

<references>

<reference group=*"cache"* name=*"cacheCatalog"*

interface=*"com.boarsoft.cache.config.CacheCatalog"* id=*"cacheCatalog"*

version=*"1.0.0"* timeout=*"3000"*>

</reference>

</references>

除了服务四要素外，作为服务提供者配置项还有：

|  |  |  |
| --- | --- | --- |
| **属性** | **含义** | **说明** |
| id | 生成的代理对象在  spring中的ID | bean必须实现指定的接口（interface），解析是会生成一个实现此接口的代理类，来调用这个bean |
| timeout | 调用此接口中所有方法的超时时间 | 单位：毫秒 |

### 方法配置

可以在reference标签下添加method标签，以具体修饰某个方法。

method标签不是必须的，无论是否添加method标签，当前接口中的所有方法都会被暴露。这些方法默认是以同步方法调用的它用来调用的（type=SC）。如果需要修改方法的调用方式，或者指定它的超时时间，就需要添加对应的method标签。

arg标签是method的子标签，用于确定方法的签名。对method标签来说，arg标签是必须的，它们需要与方法的参数一一对应，且是有序的。

示例如下：

<reference id=*"demoBc1"* group=*"demo"* name=*"demoBc1"*

interface=*"com.boarsoft.sample.DemoBC"* version=*"1.0.0"* timeout=*"30000"*>

<!-- Sync Call -->

<method name=*"bcTest"* type=*"SC"* timeout=*"2100"*>

<arg type=*"java.lang.String"* />

<arg type=*"int"* />

</method>

<!-- Sync Broadcast -->

<method name=*"bcTest"* type=*"SB"* timeout=*"1200"*>

<arg type=*"java.lang.String"* />

</method>

<!-- Async Call，demoCallback 为本地beanId，可选 -->

<method name=*"asyncCallTest"* type=*"AC"* timeout=*"3000"* callback=*"demoCallback"*>

<arg type=*"java.lang.String"* />

</method>

</reference>

method标签的合法属性有：

|  |  |  |
| --- | --- | --- |
| **属性** | **含义** | **说明** |
| name | 方法名 |  |
| timeout | 此方法的超时时间 | 此设置会覆盖reference中的设置 |
| type | 方法的调用形式 | 详细见“RPC方法类型编号及缩写”章节，默认是同步方法调用（SC） |
| callback | 方法调用完成后，需要执行的回调对象在spring中的ID | 此对象必须实现RpcCallback接口，它可以是一个普通的本地bean也可以是一远程对象的代理（一个在前面已定义的referenece） |

arg标签的合法属性有：

|  |  |  |
| --- | --- | --- |
| **属性** | **含义** | **说明** |
| type | 参数类名 | 参数的java类型 |
|  |  |  |

### 方法类型

MAC-RPC定义的RPC方法类型有以下几种，其中通知类型（NOTICE）总是在服务端异步执行，并返回签收应答，并不返回实际执行的结果。而异步类型（ASYNC）总是在调用方发起调用后，返回本地Future对象而不是远程的执行结果，不会阻塞当前线程。

/\*\* SC：本地：同步调起，阻塞以等待结果；远程：同步执行，返回结果 \*/

**public** **static** **final** **short** ***TYPE\_SYNC\_CALL*** = 0;

/\*\* AC，本地：同步调起，返回Future<Object>；远程：同步执行，返回结果 \*/

**public** **static** **final** **short** ***TYPE\_ASYNC\_CALL*** = 1;

/\*\* SN，本地：同步调起，阻塞以等待应答；远程：异步执行，回声应答 \*/

**public** **static** **final** **short** ***TYPE\_SYNC\_NOTICE*** = 2;

/\*\* AN，本地：同步调起，返回Future<Object>；远程：异步执行，回声应答 \*/

**public** **static** **final** **short** ***TYPE\_ASYNC\_NOTICE*** = 3;

/\*\* SB，本地：依次调用所有服务提供者，等待拿到的所有的Future结束，返回（Map<String, Object>）； 远程：同步执行，返回结果 \*/

**public** **static** **final** **short** ***TYPE\_SYNC\_BROADCAST*** = 4;

/\*\* AB，本地：依次调用所有服务提供者，直接返回拿到的所有的Future（Map<String, Future<Object>>）；远程：同步执行，返回结果 \*/

**public** **static** **final** **short** ***TYPE\_ASYNC\_BROADCAST*** = 5;

/\*\* BN，本地：依次调用所有服务提供者，直接返回拿到的所有的Future（Map<String, Future<Object>>）；远程：异步执行，回声应答 \*/

**public** **static** **final** **short** ***TYPE\_BROADCAST\_NOTICE*** = 6;

注：如果不希望或者不能将异步类型（ASYNC）方法的返回声明为Object，或广播方法的返回声明为Map<String, Object>，可以将服务提供方的接口方法返回值定义为实际的返回结果类型，而将服务消费方的接口方法返回值定义为Future<实际结果类型>。如果也不希望将服务消费方的接口方法返回值修为Future，则需要自行用多线程实现异步。

#### 同步调用（SC）

调用远程方法，并等待对方返回结果。

#### 异步调用（AC）

异步调用方法的返回类型必须声明为Object，在编写实现时，需要返回真实的结果。但调用方在调用时获得的是一个Future<Object>。

#### 同步通知（SN）

调用远程方法，对方会开始异步执行，并立即返回一个签收应答。

#### 异步通知（AN）

异步通知方法的返回类型必须声明为Object。

服务方在收到请求后，会开始异步执行，并立即返回一个签收应答。

调用方调用远程方法，不会阻塞当前线，拿到的返回值是Future<Object>类型，调用其get方法时才会阻塞，以等待实际的返回值——来自服务方的签收应答。

#### 同步广播方法（SB）

同步广播方法也是是需要特殊处理的方法，其返回值必须是Object类型。

在编写服务方的实现时，需要返回真实的结果。但在服务的调用方（服务消费者）拿到的对象却时Map<String, Object>类型。此map的key是执行此方法的节点的地址，形如：IP:PORT，而map的value才是实际的结果。

#### 异步广播方法（AB）

异步广播方法与同步广播方法在处理上基本相同。异步广播方法采用Future模式，返回Map<String, Future<Object>>，因此在获取实际结果前不会阻塞当前线程。

#### 广播通知方法（BN）

广播通知与异步广播方法在处理上相近。不同之处在于，服务端在收到请求后，不是同步执行，返回执行结果，而是异步执行，并立即返回签收应答。

调用方拿到的返回值也是Map<String, Future<Object>>类型，只是从Future中get的只是签收应答，而不是远程方法的实际执行结果。

# QA

**Q: HessianFieldException: com.boarsoft.rpc.bean.RpcFaceConfig.log**

-|ERROR|2017-11-20 17:00:07.299|com.boarsoft.rpc.core.RpcInClerk|145|pool-1-thread-1|Error on deserialize bytes for RPC response.|-

com.caucho.hessian.io.HessianFieldException: com.boarsoft.rpc.bean.RpcFaceConfig.log: com.caucho.hessian.io.ObjectDeserializer: unexpected object java.lang.String (name)

**A: log4j jar包冲突**