tRPC-Java快速上手

tRPC知识库

Exported on 06/29/2021

Table of Contents

目录

目录

1 前言

相信现在的你已经迫不及待想要尝试一番了；在使用之前，确保自己对tRPC-Java已经有了一定的了解，并且具备基本的Java开发能力。
下面我们会通过它来搭建一个简单的后台服务，向你展示：

- 快速搭建本地开发环境。
- 用pb方式定义一个简单的trpc服务协议
- 快速生成服务端代码。
- 通过rpc方式，调用服务。
- 部署服务上线

文章中示例源码在git仓库中可以找到，地址为helloworld[1]。

1 https://git.woa.com/trpc-java/trpc-springboot-demo/tree/feature/helloworld

2 1 环境搭建

运行 helloWorld，需要下载 trpc-java 的依赖，首先要把环境搭建[2]完成，如果已经搭建好了 java 和 maven 环境， **则重点是要替换** settings.xml配置[3]。

2 https://iwiki.oa.tencent.com/pages/viewpage.action?pageId=119395381
3 https://iwiki.oa.tencent.com/download/attachments/119395381/settings.xml?api=v2

3 2 服务端开发

3.1 2.1 创建服务仓库

- 建议每个服务单独创建一个git，如：`git.code.oa.com/trpc-java/trpc-java-helloworld-server`，然后拉取仓库到本地。

3.2 2.2 定义服务接口

tRPC采用protobuf[4]来描述一个服务，我们用protobuf定义服务方法，请求参数和响应参数
示例如下：

```
syntax = "proto3";

// package内容格式推荐为trpc.{app}.{server}，以trpc为固定前缀，标识这是一个trpc服务协议，app为你的应用名，server为你的服务进程名
package trpc.test.helloworld;

// 注意：这里java_package指定的是自动生成的java类文件所在目录,保持路径全小写
option java_package="com.tencent.test.helloworld";
// 可以用来指定生成的协议文件是多个类文件还是单个类文件
option  java_multiple_files = false;
// 当java_multiple_files属性为false时，生成的类文件名
option java_outer_classname = "GreeterSvr";

// 定义服务接口
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// 请求参数
message HelloRequest {
  string msg = 1;
}

// 响应参数
message HelloReply {
  string msg = 1;
}
```

如上，这里我们定义了一个`Greeter`服务，这个服务里面有个`SayHello`方法，接收一个包含`msg`字符串的`HelloRequest`参数，返回`HelloReply`数据。
这里需要注意以下几点：

- `syntax`必须是`proto3`，tRPC都是基于proto3通信的。
- `package`内容格式推荐为`trpc.{app}.{server}`，以trpc为固定前缀，标识这是一个trpc服务协议，app为你的应用名，server为你的服务进程名。
- 可以通过`java_multiple_files`属性指定生成的协议文件是多个类文件还是单个类文件
- 当`java_multiple_files=false`且`java_out_classname`没有配置时，service名需要为全小写，多个单词使用下划线分割；否则通过协议生成的文件名会有问题
- `java_package` 定义了**class** 文件输出目录
- `java_out_classname` 定义了**class**的名称

4 https://developers.google.com/protocol-buffers/docs/proto3

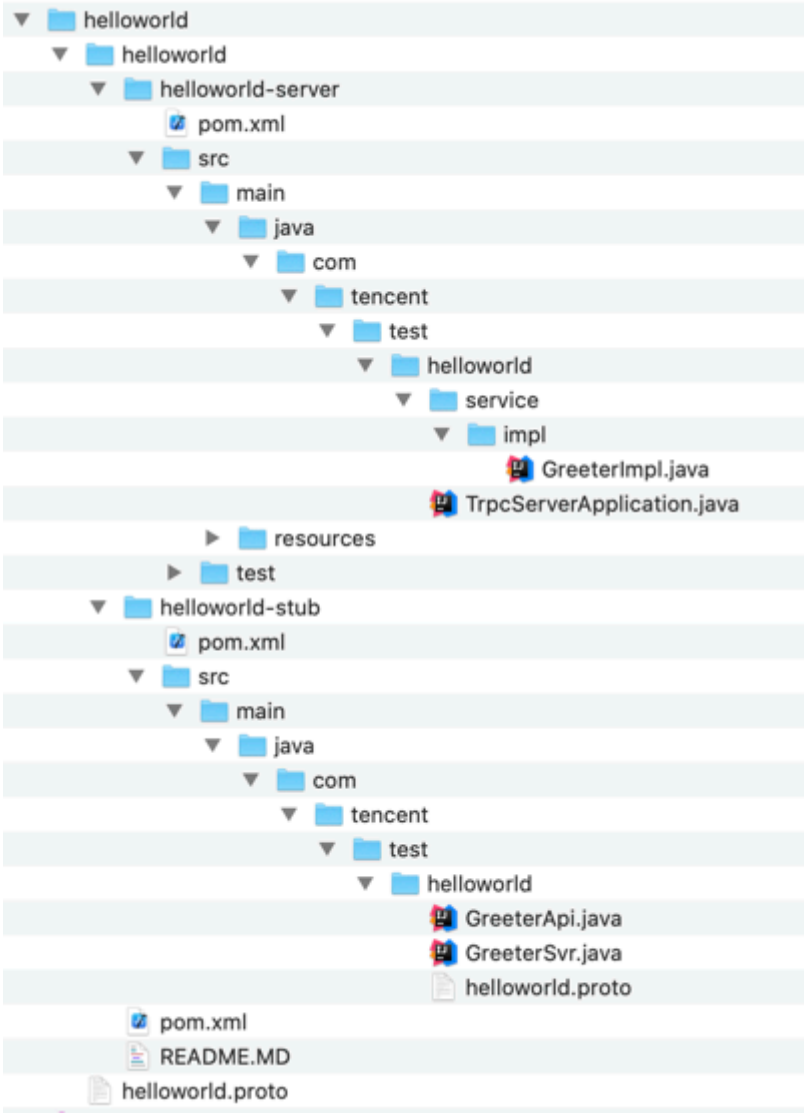
- 定义rpc方法时，一个server（服务进程）可以有多个service（对rpc逻辑分组），一般是一个server一个service，一个service中可以有多个rpc调用。
- 编写protobuf时必须遵循谷歌官方规范[5]。

3.3 2.3 生成服务代码

- 强烈推荐使用rick平台创建协议pb文件及生成服务代码，相关指引见tRPC-Java代码生成[6]。也可通过trpc命令行本地直接生成服务代码，前提是先安装trpc工具[7]：

```
# 首次使用，用该命令生成完整工程，当前目录下不要出现跟pb同名的目录名，如pb名为helloworld.proto，则当前目录不要出现helloworld的目录名
trpc create --protofile=helloworld.proto --lang java
```

- 生成代码目录结构如下：



5 https://developers.google.com/protocol-buffers/docs/style
6 https://iwiki.woa.com/pages/viewpage.action?pageId=50804003
7 https://git.code.oa.com/trpc-go/trpc-go-cmdline

项目拆分为server和stub两个子工程，其中server工程用于业务开发，stub工程包含了用于tRPC通信的协议桩文件；stub可以单独编译成jar放入公共仓库，供其它tRPC-Java服务用于client调用。
下面对每个生成的文件做简要介绍：

- `TrpcServerApplication` 服务启动类，下面框架选型(see page 9) 会重点讲到
- `GreeterImpl` 服务中service实现类，协议中定义的每个service方法的业务实现入口；在这个类里；可以看到我们在协议中定义的SayHello方法，如下为代码：

```
package com.tencent.test.helloworld.service.impl;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Service;
import com.tencent.trpc.core.rpc.RpcContext;
import com.tencent.trpc.core.rpc.RpcServerContext;

import com.tencent.test.helloworld.GreeterApi;
import com.tencent.test.helloworld.GreeterSvr;

import com.tencent.test.helloworld.GreeterSvr.HelloRequest;
import com.tencent.test.helloworld.GreeterSvr.HelloReply;

@Service
public class GreeterImpl implements GreeterApi {
   private static final Logger logger = LoggerFactory.getLogger(GreeterImpl.class);

    @Override
    public GreeterSvr.HelloReply sayHello(RpcContext context,HelloRequest request) {

          RpcServerContext serverContext = (RpcServerContext) context;
          logger.info("[hello server] receive...:{}, context:{}", request,
serverContext);
          //String message = request.getMessage();

          GreeterSvr.HelloReply.Builder rspBuilder =
GreeterSvr.HelloReply.newBuilder();

          //rspBuilder.setMessage(message);
          return rspBuilder.build();      }

}
```

- `GreeterApi` 是一个接口类，和service一一对应，GreeterImpl就是实现了这个接口；在接口及方法上面有@TRpcServcie及@TRpcMethod注解，注解上面的name用于服务内部路由；对应 go 中的 `func=/ + @TrpcService + / + @TRpcMethod`，GreeterImpl正是实现了该接口，才能被识别并注册相应的rpc服务，如下为代码：

```
package com.tencent.test.helloworld;

import java.util.concurrent.CompletionStage;

import com.tencent.trpc.core.rpc.RpcContext;
import com.tencent.trpc.core.rpc.anno.TRpcMethod;
import com.tencent.trpc.core.rpc.anno.TRpcService;

import com.tencent.test.helloworld.GreeterSvr.HelloRequest;
import com.tencent.test.helloworld.GreeterSvr.HelloReply;

/**
```

```
 * server
 *
 * @author
 */
@TRpcService(name = "trpc.test.helloworld.Greeter")
public interface GreeterApi {

    @TRpcMethod(name = "SayHello")
   HelloReply sayHello(RpcContext context, HelloRequest request);
}
```

  - GreeterSvr 包含了tRpc通信所需的请求及响应相关类，HelloRequest及HelloReply就包含在其中。
- 以上代码均为工具自动生成，现在在GreeterImpl类的sayHello方法中，填入你的第一行代码吧，添加 rspBuilder.setMsg("hello world");，接口响应中就能输出hello world了。

3.4 2.4 框架选型

tRPC-Java 支持原生容器和spring容器两种启动方法，两种都是通过 springboot 启动类方式启动的，**区别在于原生版本并没有使用springboot 的能力，包括依赖注入**。
让我们打开上个章节提到的TrpcServerApplication类文件，通过如下方式修改逻辑，我们可以方便的在原生容器模式及spring容器模式下切换，**推荐使用spring容器模式进行服务开发**。

```
package com.tencent.test.helloworld;

import com.tencent.trpc.spring.annotation.EnableTRpc;
import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import com.tencent.trpc.server.main.Main;

// EnableTRpc注解在spring容器模式下需要添加
@EnableTRpc
@SpringBootApplication
public class TrpcServerApplication {

  public static void main(String[] args) {
            new SpringApplicationBuilder()
                    .sources(TrpcServerApplication.class)
                    .web(WebApplicationType.NONE)
                    .run(args);
           // 要直接使用原生容器模式，则不通过 SpringApplicationBuilder.run的方式启动服务，而使用Main.main的方式启动服务
           // Main.main(args);
            System.out.println(">>> start");
  }

}
```

3.5 2.5 修改框架配置

trpc_java.yaml 配置 更多使用[8]

[8] https://iwiki.oa.tencent.com/pages/viewpage.action?pageId=119385319

```
#全局配置
global:
  namespace: Development             #环境类型，分正式Production和非正式Development两种类型

#服务端配置
server:
  app: test #业务的应用名
  server: helloworld #业务的服务名
  local_ip: 127.0.0.1   #本地IP，容器内为容器ip，物理机或虚拟机为本机ip
  jvm_params: -Xms3096m -Xmx3096m -Xmn2048m  #配置jvm相关参数
  # 暴露服务service 配置
  service:   #业务服务提供的service，可以有多个
    - name: trpc.test.helloworld.Greeter    #名字服务配置，用于名字服务寻址，默认使用 tcp 网
络连接协议， trpc 通信协议
      impls: #需要暴露到名字服务上的接口，共享 name 和 port
        - com.tencent.test.helloworld.service.impl.GreeterImpl    #暴露的service协议接口
实现类
      port: 8090 #trpc 服务 port
#客户端配置
client:
  #访问trpc服务配置
  service:
    - name:  greeterClient # 用于通过TRpcProxy查找
      interface: com.tencent.test.helloworld.GreeterApi # 具体的 trpc 客户端接口
      naming_url: ip://127.0.0.1:8090 # 名字服务配置，用于名字服务寻址，此处是 ip 直
连,naming配置,直连 ip://ip:port, 名字服务: polaris://trpc.TestApp.TestServer.Greeter?
key=value&key=value  l5://mid:cmid  cmlb://id
```

框架配置提供了服务启动的基本参数，包括ip、端口、协议等等。这里我们配置了一个监听127.0.0.1:8090的trpc协议的服务，同时我们也配置了一个client客户端，这个在后面章节客户端开发(see page 0)会使用到。

3.6 2.6 本地启动服务

直接在idea编辑器中，通过springboot方式启动服务即可

```
>>> TRpc Container starting!!
>>> TRpc Container started`
>>> start
```

当屏幕上出现以下日志时就说明服务启动成功了

3.7 2.7 自测联调

- 我们可以通过客户端发包工具trpc-cli命令行进行自测，前提是安装trpc-cli工具[9]，
  针对 helloworld 项目，测试命令如下：

  ```
  trpc-cli -func /trpc.test.helloworld.Greeter/SayHello -target ip://
  127.0.0.1:8090 -body '{"msg":"hello,trpc-cli"}'
  ```

  响应如下：

  ```
  maxprocs: Leaving GOMAXPROCS=12: CPU quota undefined
  req json body:{"message":"hello,trpc-cli"}
  ```

9 https://git.code.oa.com/trpc-go/trpc-cli

```
rsp json body:{"message":"hello,trpc-cli"}
req head:request_id:1 timeout:1000 caller:"trpc.app.trpc-cli.service"
callee:"trpc.test.helloworld.Greeter" func:"/trpc.test.helloworld.Greeter/
SayHello" content_type:2
rsp head:request_id:1 content_type:2
node:service:127.0.0.1:8090, addr:127.0.0.1:8090, cost:4.887076ms
err:
```

- 也可以直接在启动服务启动之后，通过另外一个本地的trpc服务以client的方式调用它，在客户端开发 (see page 0)章节会讲到如何使用tRPC-Java发起一个client请求。

4 3客户端开发

我们在实现业务逻辑的同时，需要依赖其它tRPC服务，这个时候可以使用client的方式调用它。
注意：这里需要在trpc_java.yaml文件中添加对应的client配置，上面章节有提到过
##3.1 原生容器模式下客户端开发
原生容器下，通过TRpcProxy获取到客户端代理类，然后和使用本地方法一样使用即可，实现如下：

```
  GreeterApi client = TRpcProxy.getProxy("greeterClient");
  client.sayHello(new RpcClientContext(), HelloRequest.newBuilder().setMsg("hello
world").build());
```

##3.2 spring容器模式下客户端开发
spring容器模式下，我们直接通过注解@TRpcClient直接获取到相应客户端代理，然后直接使用，其中 id 就是配置中 client:service:name
实现如下：

```
@Service
public class TestGreeter {

    @TRpcClient(id = "greeterClient")
    private GreeterApi greeterClient;

     public void TestSayHello() {
         greeterClient.sayHello(new RpcClientContext(),
HelloRequest.newBuilder().setMsg("hello world").build());
      }
}
```

5 4 部署上线

首先大家需要了解，框架是完全独立的，跟任何平台都没有绑定关系，可以支持在任何平台部署。

5.1 4.1 123平台部署

123平台是PCG容器发布平台，PCG员工后续新服务都会统一到这个平台发布服务[10]。
123 生成的配置模板文件中已经包含了各种插件的配置，但是默认只包含服务的注册，当需要使用 client 时，需要添加 selector 配置。具体配置请参考插件配置文档：北极星服务注册与发现[11]。

5.2 4.2 stke部署

可以按需定制流水线在stke进行部署。当我们自己要部署时就需要自己完成所有的配置文件填充，特别是注册中心的配置都需要自己填写，具体参考配置文档：北极星服务注册与发现[12]。
#Q&A

5.3 1.客户端开发中，使用北极星方式进行寻址，我该如何配置？

先在框架配置中添加北极星插件配置，然后将naming_url后填上polaris://{被调服务名} 即可，具体参考框架配置文档tRPC-Java注册与发现配置[13]中服务发现相关知识

5.4 2.windows开发环境下，为什么使用trpc-cli调用本地接口失败？

目前windows平台下，trpc-cli命令有编码问题，如果要调用本地接口，可以使用trpc-ui的方式替代，trpc-ui使用参考文档trpc-ui 使用指南[14]
Owner - zhengxiong(熊政)[15] **Reviewer -** youngwwang(王洋)[16]

10 https://iwiki.oa.tencent.com/pages/viewpage.action?pageId=58492570
11 https://iwiki.woa.com/pages/viewpage.action?pageId=326043584
12 https://iwiki.woa.com/pages/viewpage.action?pageId=500450601
13 https://iwiki.woa.com/pages/viewpage.action?pageId=662870000
14 https://iwiki.woa.com/pages/viewpage.action?pageId=377047500
15 https://iwiki.woa.com/display/~zexinwang
16 https://iwiki.woa.com/display/~youngwwang