RPC 使用介绍

✍️ RPC 简单介绍

> RPC

远程过程调用 `RPC（Remote Procedure Call）`。即两台服务器A和B，一个应用部署在A上想要访问位于B上应用提供的函数、方法，由于不在一个内存空间，不能直接调用，需要通过网络来表达调用的语意以及传达调用的数据。

🪄 RPC框架 - Thrift

> Thrift详解

`Thrift` 是一套包含序列化功能和支持服务通信的RPC框架，主要包含三大部分：**`代码生成、序列化框架、RPC框架`**，大致相当于 `protoc + protobuffer + grpc`，并且支持大量语言，保证常用功能在跨语言间功能一致，是一套全栈式的RPC解决方案。

用户通过 Thrift 的 **`IDL（接口定义语言）`** 来描述接口函数及数据类型，然后通过 Thrift 的编译环境生成各种语言类型的接口文件，用户可以根据自己的需要采用不同的语言开发客户端代码和服务器端代码。

💻 使用介绍

📌 后面 BFF 基本都基于 gulu 搭建，gulu rpc 功能实现基于插件 @gulu/rpc

> Part 1. PingPongServer

安装依赖

```
npm i -S @gulu/rpc
```

配置插件

```
// config/config.default.ts
import { HttpApplication } from '@gulu/application-http';

module.exports = (app: HttpApplication) => {
    const config: Partial<Gulu.AppConfig> = {
        plugin: ['@gulu/rpc'],
        rpc: {
            idl: `${app.root}/idl`, // thrift文件目录
            services: {
                PingPongService: {
                    filename: 'pingpong.thrift', // 服务调用的对应thrift文件，可以在idl目录下找到对应的文件
                    psm: 'p.s.m',
```

```
                },
            },
        },
    };

    return  {
        ...config,
    }
};
```

其中，idl 配置存放 thrift 文件的目录，thrift 文件从后端 idl仓库 拉取。

调用

```ts
// app/controller/home.ts
import { Controller, HttpContext } from '@gulu/application-http';

export default class IndexController extends Controller {
    async index(ctx: HttpContext) {
        const resp = await ctx.rpc.PingPongService.Ping({
          ping: 'ping',
        });

        console.log(resp); // { pong: 'pong' }

        ctx.body = resp;
      }
}
```

`Ping` 为 `PingPongService` rpc服务的一个接口，可以在上文配置的 `pingpong.thrift` 文件中，找到接口定义。类似于下面的定义

```
// idl/pingpong.thrift
struct PingRequest {
    1: string ping
}

struct PingResponse {
    1: string pong
}

service PingPongService {
    PingResponse Ping(1: PingRequest req)
}
```

上面例子，thrift 定义了 rpc 接口请求及响应类型，并且在传输过程中实现了 `序列化和反序列化` 。

> 💡 目前 Node.js 端不需要实现 rpc 服务搭建，只涉及对后端 rpc 接口的调用。

🧐 @gulu/rpc 内部实现

拉取

黄浩扬 2022年10月11日

Gulu ferry-fetch：https://code.byted.org/nodejs/gulu/blob/master/packages/byted-gulu-cli@2/README.md#ferry-fetch

> rpc调用链路

回到上面 gulu 对 rpc 接口的调用，请求上下文 ctx 什么时候挂载了 rpc 实例呢?

👀 通过查看源码，可以发现在 `@gulu/rpc` 插件的 **`beforeStart`** 钩子，rpc 挂载到了 app 实例上。

```ts
// byted-gulu-rpc/app.ts
import { createClient } from '@byted-service/rpc';
import { Consul } from '@byted-service/consul';
import { HttpApplication, TypeGenerator } from '@gulu/application-http';
import { clientSym, createClientForApp } from './lib/rpc';

beforeStart(app: HttpApplication) {
    const options = app.config.rpc;
    const consul = app.config.consul ? new Consul(app.config.consul) : undefined;
    const client = createClient({
        consul,
        ...options,
        enableTypingsCodeGen: false,
    });

    app.rpc = createClientForApp(client, app) as any;
    app.rpc[clientSym] = client;
},
```

而我们前面调用 rpc，是从 ctx 上获取的 rpc 实例。其实是在扩展里去实现挂载的。

```ts
// byted-gulu-rpc/app/extension/context.ts
import * as MetaInfo from '@byted-service/metainfo';
import { HttpContext } from '@gulu/application-http';
import { clientSym, createClientForCtx } from '../../lib/rpc';

export default {
    get rpc() {
        if (!this[guluCachedContextRpcSym]) {
            const ctx = this as any as HttpContext;

            // save metainfo from HTTP header
            if ((!ctx.app.prefix || ctx.app.prefix === 'http') && !(ctx as any)[K_HEADER_METAINFO_SETTED]) {
                /* istanbul ignore next */
                MetaInfo.fromHTTPHeader(ctx, ctx.headers || {});
                (ctx as any)[K_HEADER_METAINFO_SETTED] = true;
            }

            this[guluCachedContextRpcSym] = createClientForCtx(ctx.app.rpc[clientSym], ctx);
        }

        return this[guluCachedContextRpcSym];
    },
};
```

最终 rpc 实例创建以及调用的代码大致如下

> @byted-service/rpc
>
> 
> 黄浩扬 2022年8月11日
> @byted-service/rpc 浅析

```ts
// 创建 RPC Client
const client = createClient({
    idl: [resolve(__dirname, 'idl/pingpong.thrift')],
    servers: [server.address() as string],
    services: {
        PingPongService: {},
    },
});

// 调用 RPC method
const resp = await (client as any)['PingPongService']['Ping']({
    ping: 'ping',
});
```

```
RPC Client -> thrift encode -> TCP Socket -> TCP Server -> thrift decode -> RPC Server
```

🔍 服务发现

在上面的例子中，我们请求 RPC Server 只是指定了 psm，具体需要服务发现为我们找到目标服务 - 服务发现 Consul

〰️ 链路信息

一个请求经过的所有服务和中间件，构成了一个调用链包括 TLB, HTTP 服务, thrift 服务, 消息队列(kafka, redis)等。信息传递可能经过调用链的全部节点或部分节点。提供给整个链路使用的信息，称为 `链路信息` - 链路信息 Metainfo

👍 好的实践

- guluS-插件-rpc接口透传
- 中间件写法

🛠️ 问题排查

Gulu 问题排查

🤔 其他

- 数据校验
- 重试机制

- 超时配置