GO RPC

背景

RPC （Remote Procedure Call） 远程过程调用。像调用本地函数一样调用远程服务。

相较Restful API优势:

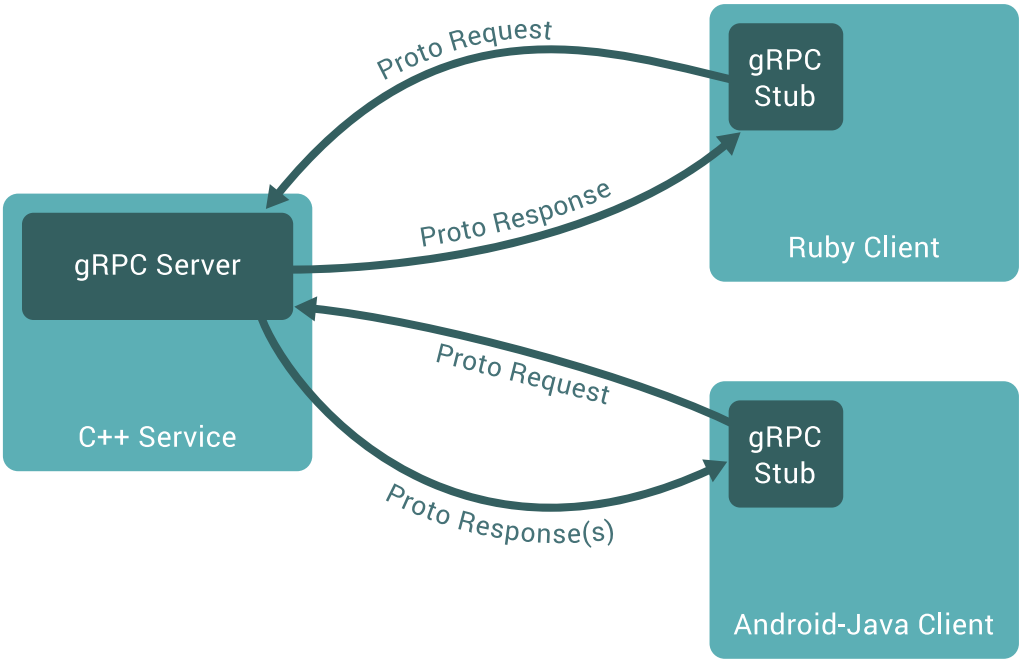
1. Protobuf 减小传输数据大小，提高性能
2. 支持流式通信（像HLS、RTMP协议）

了解Google的gRPC.

gRPC 可以将protocol buffer作其接口定义语言 (**IDL**) 和底层消息交换格式.

在服务端，服务端实现这个接口并运行一个 gRPC 服务器来处理客户端调用。在客户端，客户端有一个存根（在某些语言中仅称为客户端），它提供与服务器相同的方法。

> 设计理念（HERE：2020边缘网关）：
>
> 边缘网关作为平台服务的延伸，平台直接调用边缘网关的能力。边缘网关是gRPC服务端，平台作为客户端。需要考虑：安全、流量统计



使用protocol buffer:

> 以下均是 proto 3

1. proto文件中定义数据结构
2. 定义gprc方法

```
// 定义一个服务
service Greeter {
  // 定义方法
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
```

```
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

3. gRPC 使用`protoc`(protocol buffer编译器)指定的 gRPC 插件生成指定编程语言代码。

   ProtocolBuffer GitHub仓库中提供 Java、C++、Dart、Python、Objective-C、C#、lite-runtime (Android Java)、Ruby 和 JavaScript 版本

四种服务类型：

1. 普通调用

```
rpc GetFeature(Point) returns (Feature) {}
```

2. *服务器端流式 RPC* . 客户端从返回的流中读取数据，直到Eof

```
rpc ListFeatures(Rectangle) returns (stream Feature) {}
```

3. *客户端流式 RPC*

```
rpc RecordRoute(stream Point) returns (RouteSummary) {}
```

4. 双向流式*RPC*

```
rpc RouteChat(stream RouteNote) returns (stream RouteNote) {}
```

GP通过protoc编译后生成（以go为例）

- xxx.pb.go 包括用于填充、序列化和检索请求和响应消息的定义
- xxx_gprc.pb.go 包含了服务端要实现的接口和 客户端调用的方法定义。

proto3语法

.proto Type	Notes	C++ Type	Java/Kotlin Type[1]	Python Type[3]	Go Type	Ruby Type	C# Type	PHP Type	Dart Type
double		double	double	float	float64	Float	double	float	double
float		float	float	float	float32	Float	float	float	double
int32		int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
int64		int64	long	int/long[4]	int64	Bignum	long	integer/string[6]	Int64
uint32		uint32	int[2]	int/long[4]	uint32	Fixnum or Bignum (as required)	uint	integer	int
uint64		uint64	long[2]	int/long[4]	uint64	Bignum	ulong	integer/string[6]	Int64
sint32	带符号，比int32更有效表示负数	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
sint64		int64	long	int/long[4]	int64	Bignum	long	integer/string[6]	Int64
fixed32	固定四个字节。对总是超过288的数更有效	uint32	int[2]	int/long[4]	uint32	Fixnum or Bignum (as required)	uint	integer	int
fixed64		uint64	long[2]	int/long[4]	uint64	Bignum	ulong	integer/string[6]	Int64
sfixed32		int32	int	int	int32	Fixnum or Bignum (as required)	int	integer	int
sfixed64		int64	long	int/long[4]	int64	Bignum	long	integer/string[6]	Int64
bool		bool	boolean	bool	bool	TrueClass/FalseClass	bool	boolean	bool
string	字符串，不能超过232长度	string	String	str/unicode[5]	string	String (UTF-8)	string	string	String
bytes		string	ByteString	str (Python 2) bytes (Python 3)	[]byte	String (ASCII-8BIT)	ByteString	string	

```
message SearchResponse {
  repeated Result results = 1;
}

// 如果在其他文件，通过import引入import "myproject/other_protos.proto";
message Result {
  string url = 1;
  string title = 2;
  repeated string snippets = 3;
}
// 支持嵌套
message SearchResponse {
  message Result {
    string url = 1;
    string title = 2;
    repeated string snippets = 3;
  }
  repeated Result results = 1;
}
```

proto更新说明：

- 不要更改任何现有字段的编号
- 新加字段，旧消息仍能解析。新字段将以默认值出现
- 新加字段，新消息仍能被旧程序解析，程序会忽略新字段
- 字段类型之间有的是兼容的，可以进行修改

应用层传输安全（ALTS）

Go gRPC

https://www.grpc.io/docs/languages/go/quickstart/

Protocolbuffer编译器

Protocol buffer compiler

https://github.com/protocolbuffers/protobuf/releases

设置到环境变量路径中

Go Plugin

下载Go 1.16.10（安装的1.17不能下载）。到Go 1.16bin下面执行如下命令。可执行文件会下载到对应的GOPATH下（确保GOPATH下的bin在系统Path下）、

```
$ go install google.golang.org/protobuf/cmd/protoc-gen-go@v1.26
$ go install google.golang.org/grpc/cmd/protoc-gen-go-grpc@v1.1
```

参考官网例子在helloword中添加一个方法，然后执行

```
protoc --go_out=. --go_opt=paths=source_relative --go-grpc_out=. --go-
grpc_opt=paths=source_relative helloworld/helloworld.proto
```

在对应目录下生成`helloworld/helloworld.pb.go`和`helloworld/helloworld_grpc.pb.go`.

然后在Server中实现扩展的接口，在Client中调用即可。

Server代码：

```
flag.Parse()
lis, err := net.Listen("tcp", fmt.Sprintf("localhost:%d", *port))
if err != nil {
  log.Fatalf("failed to listen: %v", err)
}
var opts []grpc.ServerOption
...
grpcServer := grpc.NewServer(opts...)
pb.RegisterRouteGuideServer(grpcServer, newServer())
grpcServer.Serve(lis)
```

- 启动net服务
- 创建gRPC实例
- 注册我们的服务实现

- 通过`server()`启动组赛服务

Client代码：

```
var opts []grpc.DialOption
...
conn, err := grpc.Dial(*serverAddr, opts...)
if err != nil {
  ...
}
defer conn.Close()
// 创建Client Stub
client := pb.NewRouteGuideClient(conn)

// 调用方法（像使用本地方法一样）
feature, err := client.GetFeature(context.Background(), &pb.Point{409146138,
-746188906})
if err != nil {
  ...
}
```