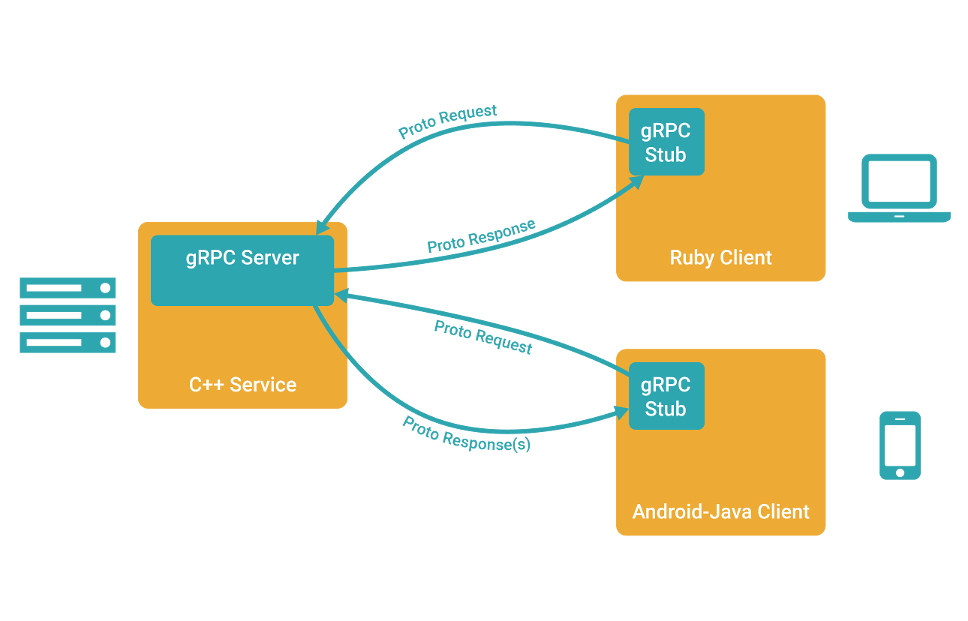
# **[gRPC应用C++](https://www.cnblogs.com/embedded-linux/p/11787696.html)**

## **1．  gRPC简述**

RPC，远程方法调用，就是像调用本地方法一样调用远程方法。

[gRPC](http://www.oschina.net/p/grpc-framework)是Google实现的一种RPC框架，基于HTTP/2标准设计，带来诸如双向流、流控、头部压缩、单 TCP 连接上的多复用请求等特。这些特性使得其在移动设备上表现更好，更省电和节省空间占用。目前提供 C、Java 和 Go 语言版本，分别是：grpc, grpc-java, grpc-go. 其中 C 版本支持 C, C++, Node.js, Python, Ruby, Objective-C, PHP 和 C# 支持。

基本原理：在服务端提供方法，并运行服务器监听客户端请求；客户端调用RPC方法，gRPC客户端向服务端发送请求，并将结果返回给客户端调用函数。



gRPC 默认使用 protocol buffers，这是 Google 开源的一套成熟的结构数据序列化机制（当然也可以使用其他数据格式如 JSON）。推荐使用protobuf v3。

github：<https://github.com/grpc/grpc>

**RPC框架要做到的最基本的三件事：**

1）服务端如何确定客户端要调用的函数；

在远程调用中，客户端和服务端分别维护一个【ID->函数】的对应表， ID在所有进程中都是唯一确定的。客户端在做远程过程调用时，附上这个ID，服务端通过查表，来确定客户端需要调用的函数，然后执行相应函数的代码。

2）如何进行序列化和反序列化；

客户端和服务端交互时将参数或结果转化为字节流在网络中传输，那么数据转化为字节流的或者将字节流转换成能读取的固定格式时就需要进行序列化和反序列化，序列化和反序列化的速度也会影响远程调用的效率。

3）如何进行网络传输（选择何种网络协议）；

多数RPC框架选择TCP作为传输协议，也有部分选择HTTP。如gRPC使用HTTP2。不同的协议各有利弊。TCP更加高效，而HTTP在实际应用中更加的灵活。

**REST与RPC应用场景**

REST和RPC都常用于微服务架构中。

1）HTTP相对更规范，更标准，更通用，无论哪种语言都支持http协议。如果你是对外开放API，例如开放平台，外部的编程语言多种多样，你无法拒绝对每种语言的支持，现在开源中间件，基本最先支持的几个协议都包含RESTful。

2）RPC 框架作为架构微服务化的基础组件，它能大大降低架构微服务化的成本，提高调用方与服务提供方的研发效率，屏蔽跨进程调用函数（服务）的各类复杂细节。让调用方感觉就像调用本地函数一样调用远端函数、让服务提供方感觉就像实现一个本地函数一样来实现服务。

REST调用及测试都很方便，RPC就显得有点繁琐，但是RPC的效率是毋庸置疑的，所以建议在多系统之间的内部调用采用RPC。对外提供的服务，Rest更加合适。

## **2．  安装**

$ git clone https：//github.com/grpc/grpc.git $ cd grpc

$ git submodule update --init

$make

$sudo make install

注：执行submodule时时间较长

IMG_257 install log

安装的文件包含：bin，include，lib，share（根证书）。

## **3．  应用**

examples下有各语言版本的示例，examples/protoc是protoc示例文件。

如下演示的是helloworld示例，examples中不仅支持同步调用，还支持异步调用。

1． proto文件定义RPC方法及参数

helloworld.proto

[IMG_258](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

syntax = "proto3";

option java\_package = "io.grpc.examples";

package helloworld;

// The greeter service definition.service Greeter {

// Sends a greeting rpc SayHello (HelloRequest) returns (HelloReply) {}

}

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

string name = 1;

}

// The response message containing the greetingsmessage HelloReply {

string message = 1;

}

[IMG_259](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

编译生成c++源文件（grpc和应用）

protoc --grpc\_out=. --plugin=protoc-gen-grpc=`which grpc\_cpp\_plugin` helloworld.proto

protoc --cpp\_out=. helloworld.proto

2． server端代码

[IMG_260](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

#include <iostream>

#include <memory>

#include <string>

#include <grpcpp/grpcpp.h>

#ifdef BAZEL\_BUILD

#include "examples/protos/helloworld.grpc.pb.h"#else

#include "helloworld.grpc.pb.h"#endif

using grpc::Server;using grpc::ServerBuilder;using grpc::ServerContext;using grpc::Status;using helloworld::HelloRequest;using helloworld::HelloReply;using helloworld::Greeter;

// Logic and data behind the server's behavior.class GreeterServiceImpl final : public Greeter::Service {

Status SayHello(ServerContext\* context, const HelloRequest\* request,

HelloReply\* reply) override {

std::string prefix("Hello ");

reply->set\_message(prefix + request->name());

return Status::OK;

}

};

void RunServer() {

std::string server\_address("0.0.0.0:50051");

GreeterServiceImpl service;

ServerBuilder builder;

// Listen on the given address without any authentication mechanism. builder.AddListeningPort(server\_address, grpc::InsecureServerCredentials());

// Register "service" as the instance through which we'll communicate with

// clients. In this case it corresponds to an \*synchronous\* service.

builder.RegisterService(&service);

// Finally assemble the server.

std::unique\_ptr<Server> server(builder.BuildAndStart());

std::cout << "Server listening on " << server\_address << std::endl;

// Wait for the server to shutdown. Note that some other thread must be

// responsible for shutting down the server for this call to ever return.

server->Wait();

}

int main(int argc, char\*\* argv) {

RunServer();

return 0;

}

[IMG_261](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

**编译执行：**

g++ -o server helloworld.pb.cc helloworld.grpc.pb.cc server.cc -L/usr/local/lib `pkg-config --cflags protobuf grpc` -std=c++11 `pkg-config --libs protobuf grpc++ grpc` -Wl,--no-as-needed -lgrpc++\_reflection -Wl,--as-needed -ldl

$ ./server

Server listening on 0.0.0.0:50051

**3． client端代码**

[IMG_262](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

#include <iostream>

#include <memory>

#include <string>

#include <grpcpp/grpcpp.h>

#ifdef BAZEL\_BUILD

#include "examples/protos/helloworld.grpc.pb.h"#else

#include "helloworld.grpc.pb.h"#endif

using grpc::Channel;using grpc::ClientContext;using grpc::Status;using helloworld::HelloRequest;using helloworld::HelloReply;using helloworld::Greeter;

class GreeterClient {

public:

GreeterClient(std::shared\_ptr<Channel> channel)

: stub\_(Greeter::NewStub(channel)) {}

// Assembles the client's payload, sends it and presents the response back

// from the server.

std::string SayHello(const std::string& user) {

// Data we are sending to the server. HelloRequest request;

request.set\_name(user);

// Container for the data we expect from the server. HelloReply reply;

// Context for the client. It could be used to convey extra information to

// the server and/or tweak certain RPC behaviors. ClientContext context;

// The actual RPC.

Status status = stub\_->SayHello(&context, request, &reply);

// Act upon its status.

if (status.ok()) {

return reply.message();

} else {

std::cout << status.error\_code() << ": " << status.error\_message()

<< std::endl;

return "RPC failed";

}

}

private:

std::unique\_ptr<Greeter::Stub> stub\_;

};

int main(int argc, char\*\* argv) {

// Instantiate the client. It requires a channel, out of which the actual RPCs

// are created. This channel models a connection to an endpoint (in this case,

// localhost at port 50051). We indicate that the channel isn't authenticated

// (use of InsecureChannelCredentials()). GreeterClient greeter(grpc::CreateChannel(

"127.0.0.1:50051", grpc::InsecureChannelCredentials()));

std::string user("world");

std::string reply = greeter.SayHello(user);

std::cout << "Greeter received: " << reply << std::endl;

return 0;

}

[IMG_263](https://www.cnblogs.com/embedded-linux/p/javascript:void(0);)

**编译执行：**

g++ -o client helloworld.pb.cc helloworld.grpc.pb.cc client.cc -L/usr/local/lib `pkg-config --cflags protobuf grpc` -std=c++11 `pkg-config --libs protobuf grpc++ grpc` -Wl,--no-as-needed -lgrpc++\_reflection -Wl,--as-needed -ldl

$ ./client

Greeter received: Hello world