**protocol buffer安装和使用**

官网：<https://github.com/google/protobuf/releases/tag/v3.0.2>

golang官网：<https://github.com/golang/protobuf>

# 1 安装编译器protoc

linux下安装

|  |  |
| --- | --- |
| 检查安装需要用到的编译工具 | yum install autoconf automake libtool curl make g++ unzip |
| 新建一个目录 | mkdir -p /usr/local/protoc |
| 进入目录 | cd /usr/local/protoc |
| 下载linux版本的protoc | wget <https://github.com/google/protobuf/releases/download/v3.0.2/protoc-3.0.2-linux-x86_64.zip>  注:可能比较慢，可以进入官网下载 |
| 解压 | unzip protoc-3.0.2-linux-x86\_64.zip |
| 配置环境变量 | echo 'export PATH=/usr/local/protoc/bin:$PATH' >> ~/.bashrc |
| 使环境变量生效 | source ~/.bashrc |

window下安装

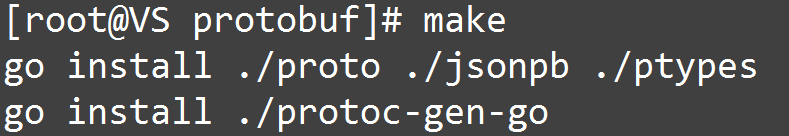
去官网<https://github.com/google/protobuf/releases/tag/v3.0.2下载window>版本文件，把protoc.exe复制到goroot/bin目录下

执行命令protoc --version是否安装成功

# 2 安装编译器插件protoc-gen-go

**go get -u github.com/golang/protobuf/{proto,protoc-gen-go}**

如果无法安装，直接去官网<https://github.com/golang/protobuf>下载源文件，然后复制到对应文件夹，使用命令make安装。



注意：安装完protoc-gen-go执行文放在$goroom/bin目录下，方便编译器protoc找到该插件。

# 3 一个简单例子

在src下新建一些目录和文件，目录结构如下图所示



proto/loginProto文件夹的login.proto的文件内容如下：

|  |
| --- |
| syntax = "proto3";  package login;  // 接口，grpc的客户端和服务端都实现的方法 service Loginer {  rpc Login (LoginRequest) returns (LoginReply) {} }  // grpc客户端请求时传输给服务端的对象 message LoginRequest {  string username = 1; // 用户名  string password = 2; // 密码 }  // grpc服务端返回给客户端的对象 message LoginReply {  bool status = 1; // 登录是否成功  string errorMsg = 2; // 登录失败时的信息 } |

编译\*.proto文件一般有两种，用命令方式或脚本

命令方式：

|  |
| --- |
| protoc -I proto/loginProto proto/loginProto/\*.proto --go\_out=plugins=grpc:proto/loginProto |

脚本方式，genProto.sh文件内容如下

|  |
| --- |
| **#!/usr/bin/env bash** # 获取项目绝对路径 root\_path=$*(cd `dirname $0`; pwd)* # 编译文件位置和生成\*.pb.go文件位置 build\_generate\_path=${root\_path}'/proto/loginProto'  # 编译文件位置 file\_path=${build\_generate\_path}'/\*.proto'  *protoc* -I ${build\_generate\_path} ${file\_path} --go\_out=plugins=grpc:${build\_generate\_path} |

通过protoc编译器根据\*.proto文件自动生成\*.pb.go，到时grpc的客户端和服务端使用时都要import这个\*.pb.go文件。

server文件夹的main.go文件内容如下：

|  |
| --- |
| package main  import (  "log"  "net"   pb "grpc/proto/loginProto"   "golang.org/x/net/context"  "google.golang.org/grpc" )  const (  port = ":50051" )  type server struct { }  func (s \**server*) Login(ctx context.Context, in \*pb.LoginRequest) (\*pb.LoginReply, error) {  rpl := &pb.LoginReply{}  println(in.Username, in.Password)  if in.Username == "grpc" && in.Password == "123456" {  rpl.Status = true  rpl.ErrorMsg = ""  } else {  rpl.Status = false  rpl.ErrorMsg = "failed to verify"  }   return rpl, nil }  func main() {  // 监听服务  lis, err := net.Listen("tcp", port)  if err != nil {  log.Fatalf("failed to listen: %v", err)  }  // 实例化对象  s := grpc.NewServer()  // 注册方法  pb.RegisterLoginerServer(s, &server{})  // 启动gppc服务器  if err := s.Serve(lis); err != nil {  log.Fatalf("failed to serve: %v", err)  } } |

client文件夹的main.go文件内容如下

|  |
| --- |
| package main  import (  "log"  "os"   pb "grpc/proto/loginProto"   "golang.org/x/net/context"  "google.golang.org/grpc" )  const address = "localhost:50051"  func main() {  // 获取系统参数值  var username, password string  if len(os.Args) == 3 {  username = os.Args[1]  password = os.Args[2]  } else {  log.Printf("usage: <program file> <username> <password>")  return  }   // 连接grpc服务器  conn, err := grpc.Dial(address, grpc.WithInsecure())  if err != nil {  log.Fatalf("did not connect: %v", err)  }  defer conn.Close()   // 实例化对象loginerClient，对象实现方法有Login  c := pb.NewLoginerClient(conn)   // 调用方法，输入参数LoginRequest对象，得到结果LoginReply对象  r, err := c.Login(context.Background(), &pb.LoginRequest{Username: username, Password: password})  if err != nil {  log.Fatalf("could not get result: %v", err)  }  log.Printf("Message: %#v", r) } |

在终端切换到server目录下，执行go run main.go

在另一个终端切换到client目录下，执行go run main.go grpc 123456

