Protobuf 使用说明

1. 定义报名： package hrv;
2. 导入其他protobuf: import "UserInfo.proto";
3. java包名

option java\_package = "com.sixfoot.hrv.codec.protobuf";

//实体名字

1. option java\_outer\_classname = "LoginRespProto";

基本格式：

如何定义消息：

l  定义一个消息类型

先来看一个非常简单的例子。假设你想定义一个“搜索请求”的消息格式，每一个请求含有一个查询字符串、你感兴趣的查询结果所在的页数，以及每一页多少条查询结果。可以采用如下的方式来定义消息类型的.proto文件了：

|  |
| --- |
| message SearchRequest  {    required string query = 1;    optional int32 page\_number = 2;    optional int32 result\_per\_page = 3;  } |

SearchRequest消息格式有3个字段，在消息中承载的数据分别对应于每一个字段。其中每个字段都有一个名字和一种类型。

1.1指定字段类型

在上面的例子中，所有字段都是标量类型：两个整型（page\_number和result\_per\_page），一个string类型（query）。当然，你也可以为字段指定其他的合成类型，包括枚举（[enumerations](http://www.open-open.com/home/link.php?url=http://code.google.com/intl/zh-CN/apis/protocolbuffers/docs/proto.html%23enum" \t "http://blog.csdn.net/u014308482/article/details/_blank)）或其他消息类型。

1.2  分配标识号

正如上述文件格式，在消息定义中，每个字段都有唯一的一个标识符。这些标识符是用来在消息的二进制格式中识别各个字段的，一旦开始使用就不能够再改 变。注：[1,15]之内的标识号在编码的时候会占用一个字节。[16,2047]之内的标识号则占用2个字节。所以应该为那些频繁出现的消息元素保留 [1,15]之内的标识号。切记：要为将来有可能添加的、频繁出现的标识号预留一些标识号。

最小的标识号可以从1开始，最大到229 - 1, or 536,870,911。不可以使用其中的[19000－19999]的标识号， Protobuf协议实现中对这些进行了预留。如果非要在.proto文件中使用这些预留标识号，编译时就会报警。

1.3 指定字段规则

所指定的消息字段修饰符必须是如下之一：

required：一个格式良好的消息一定要含有1个这种字段。表示该值是必须要设置的；

optional：消息格式中该字段可以有0个或1个值（不超过1个）。

repeated：在一个格式良好的消息中，这种字段可以重复任意多次（包括0次）。重复的值的顺序会被保留。表示该值可以重复，相当于**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/u014308482/article/details/_blank)**中的List。

完整消息如下：

例子1：

package hrv;

import "Response.proto";

import "UserInfo.proto";

option java\_package = "com.sixfoot.hrv.codec.protobuf";

option java\_outer\_classname = "LoginRespProto";

message PConfig

{

// 是否是主要类型

required bool HaveMain = 1;

// 站点名

required string WebName = 2;

// 节点籍列表

repeated AP P = 3;

// 一个节点

message AP

{

// 唯一编号

required int32 IndexID = 1;

// 篇名称

required string PieceName = 2;

// 篇地址

repeated APUrl PUrl = 3;

// 一个节点地址

message APUrl

{

// 类名

required string ClassName = 1;

// 地址

required string Url = 2;

// 是否主要

required bool HaveMain = 3;

}

}

例子2：

package hrv;

option java\_package = "com.sixfoot.hrv.codec.protobuf";

option java\_outer\_classname = "RequestProto";

message Request {

extensions 100 to max;

enum Type {

LOGIN = 0;

CHANGE\_PASSWORD = 1;

START\_SCALE = 2;

STOP\_SCALE = 3;

DATA\_PPG = 4;

DATA\_EP = 5;

DATA\_HRV = 6;

DATA\_IBI = 7;

MARK\_SCALE = 8;

RESOURCE\_LIST = 9;

UPDATE\_USER\_INFO = 10;

GET\_SCALE\_LIST = 11;

GET\_SCALE = 12;

}

required Type type = 1;

optional string timestamp = 2;

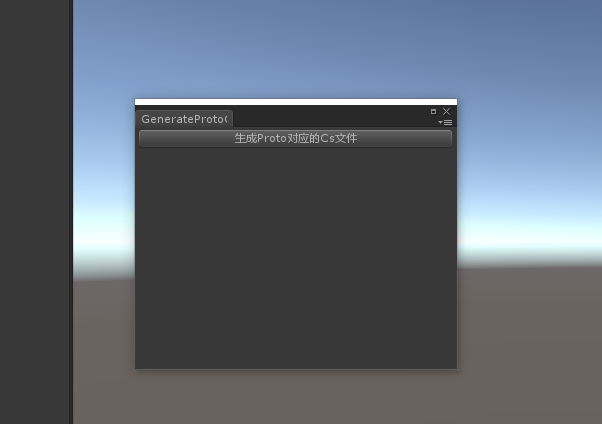
}

2，基本数据类型



1. Unity3d 使用方法：

1.生成CS文件



1. 序列化和反序列化

 // 将消息序列化为二进制的方法

 // < param name="model">要序列化的对象< /param>

 private byte[] Serialize(NetModel model)

 {

  try {

   //涉及格式转换，需要用到流，将二进制序列化到流中

   using (MemoryStream ms = new MemoryStream()) {

    //使用ProtoBuf工具的序列化方法

    ProtoBuf.Serializer.Serialize<NetModel> (ms, model);

    //定义二级制数组，保存序列化后的结果

    byte[] result = new byte[ms.Length];

    //将流的位置设为0，起始点

    ms.Position = 0;

    //将流中的内容读取到二进制数组中

    ms.Read (result, 0, result.Length);

    return result;

   }

  } catch (Exception ex) {

   Debug.Log ("序列化失败: " + ex.ToString());

   return null;

  }

 }

 // 将收到的消息反序列化成对象

 // < returns>The serialize.< /returns>

 // < param name="msg">收到的消息.</param>

 private NetModel DeSerialize(byte[] msg)

 {

  try {

   using (MemoryStream ms = new MemoryStream()) {

    //将消息写入流中

    ms.Write (msg, 0, msg.Length);

    //将流的位置归0

    ms.Position = 0;

    //使用工具反序列化对象

    NetModel result = ProtoBuf.Serializer.Deserialize<NetModel> (ms);

    return result;

   }

  } catch (Exception ex) {

    Debug.Log("反序列化失败: " + ex.ToString());

    return null;

  }

 }