



# Protocol Buffers

- 虚幻引擎交互设计师班 -



01

概 念 分 析

02

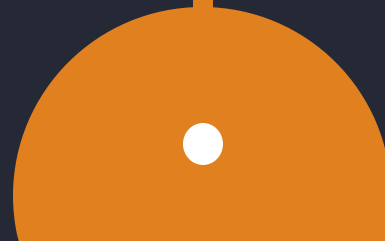
文 本 编 码 格 式

03

J S O N 和 X M L

04

Protocol Buffers



CONTENT



# 概述

- 虚幻引擎交互设计师班 -



火星时代教育

# Protocol Buffers

Protocol Buffers, 是Google公司开发的一种数据描述语言, 类似于XML能够将结构化数据序列化, 可用于数据存储、通信协议等方面。它是一种**独立的数据交换格式**, 可用于多种领域。**它独立于语言, 独立于平台**。谷歌提供了多种编程语言下的序列化和反序列化方案。**由于它是一种二进制格式, 比使用XML进行数据交换包体更小 (小3-10倍), 解析速度更快 (快20-100倍)**。对于PB来说, 它的使用大致分为三步:

1. 定制消息体message (运行前)
2. 编译消息体message, 生成对应的平台代码 (运行前)
3. 应用代码读写结构化数据 (运行中)

# 定义消息体

- 虚幻引擎交互设计师班 -



火星时代教育

# 基本格式

首先需要注意Protocol Buffers需要先定义消息体。它按照给定的格式完成制定，大致如下：

```
message 名称{  
  
}
```

消息体最后需要被保存为.proto的文件，用于对应语言平台的代码。

**我们目前针对的是Proto3进行学习，所有内容针对Proto3说明。**

语法格式

# 样例格式

Syntax标注版本，我们使用的是proto3版本。message标注消息体关键字，结构使用花括号作为分割符号。

```
1  syntax = "proto3";  
2  
3  message MessageName {  
4  
5  }
```



# 注释

注释语法使用 “//” 双分隔符

```
1  syntax = "proto3"; // 协议版本
2
3  message Person { // 消息名称
4
5  }
```

字段

# 字段

proto3当中提供了标量值类型（附表）用来创建字段。字段后=部分非字段默认值，而是字段序号。

```
1  syntax = "proto3"; //协议版本
2
3  message Person{ //消息名称
4      int32 age = 1; //字段规则（暂无）  字段类型，字段名称，字段序号
5      string name = 2;
6  }
```

# 注意事项

- 协议版本不能为空
- 协议名称，非数字开头，敏感大小写
- 字段类型（后表）
- 字段名称，非数字开头即可（禁止中文）敏感大小写
- 字段序号，每个字段的序号必须是唯一的并且为正整数，最小为1最大为 $(2^{29})-1$ ，且禁止是19000~19999的任何一个值（针对频繁出现的消息元素应保证使用1-15编号，因为1-15只需要一个字节编码）

# 字段类型

.proto Type	Notes	C++ Type	Java/Kotlin Type <sup>[1]</sup>	Python Type <sup>[3]</sup>	Go Type	Ruby Type	C# Type	PHP Type
double		double	double	float	float64	Float	double	float
float		float	float	float	float32	Float	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long <sup>[4]</sup>	int64	Bignum	long	integer/string
uint32	Uses variable-length encoding.	uint32	int <sup>[2]</sup>	int/long <sup>[4]</sup>	uint32	Fixnum or Bignum (as required)	uint	integer
uint64	Uses variable-length encoding.	uint64	long <sup>[2]</sup>	int/long <sup>[4]</sup>	uint64	Bignum	ulong	integer/string
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer

.proto Type	Notes	C++ Type	Java/Kotlin Type <sup>[1]</sup>	Python Type <sup>[3]</sup>	Go Type	Ruby Type	C# Type	PHP Type
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long <sup>[4]</sup>	int64	Bignum	long	integer/string
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2 <sup>28</sup> .	uint32	int <sup>[2]</sup>	int/long <sup>[4]</sup>	uint32	Fixnum or Bignum (as required)	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2 <sup>56</sup> .	uint64	long <sup>[2]</sup>	int/long <sup>[4]</sup>	uint64	Bignum	ulong	integer/string
sfixed32	Always four bytes.	int32	int	int	int32	Fixnum or Bignum (as required)	int	integer
sfixed64	Always eight bytes.	int64	long	int/long <sup>[4]</sup>	int64	Bignum	long	integer/string
bool		bool	boolean	bool	bool	TrueClass/FalseClass	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text, and cannot be longer than 2 <sup>32</sup> .	string	String	str/unicode <sup>[5]</sup>	string	String (UTF-8)	string	string
bytes	May contain any arbitrary sequence of bytes no longer than 2 <sup>32</sup> .	string	ByteString	str (Python 2) bytes (Python 3)	[]byte	String (ASCII-8BIT)	ByteString	string

# 默认值

- string类型, 默认值为 "" **注意string类型只接收utf-8格式的编码文本, 其他格式会导致记录问题**
- bytes类型, 默认值为 空字节
- bool类型, 默认值为false
- numeric类型, 默认值为0
- enum类型, 默认值为第一个值, 且必须为0
- message类型, 默认值未设置, 值由语言特性确定

# 字段规则

# 字段规则

Proto3中为添加字段设置了字段规则，字段规则应是以下之一：

- singular：格式良好的消息可以有零个或一个此字段（但不能超过一个）。这是 proto3 语法的默认字段规则。
- repeated：该字段可以在格式良好的消息中重复任意次数（包括零次）。重复值的顺序将被保留。

字段规则中，单一数据（singular）无需指定关键字，数组型需要指定标记（repeated）

```
1  syntax = "proto3"; // 协议版本
2
3  message Person { // 消息名称
4      int32 age = 1; // 字段规则（暂无） 字段类型，字段名称，字段序号
5      string name = 2;
6      repeated string friends = 3; // 重复字段
7  }
```



枚举

# 枚举类型

**枚举可以定在在外部，或是定义在消息体内。** 注意枚举的项值需要从0开始，并且每个枚举都必须包含一个映射到0的常量作为第一个元素。

**定义在外部可以在其他消息中使用，内部无法在其他消息中使用。**

```
1  syntax = "proto3";
2
3  message SearchRequest {
4      string query = 1;
5      int32 page_number = 2;
6      int32 result_per_page = 3;
7      enum Corpus {
8          UNIVERSAL = 0;
9          WEB = 1;
10         IMAGES = 2;
11         LOCAL = 3;
12         NEWS = 4;
13         PRODUCTS = 5;
14         VIDEO = 6;
15     }
16     Corpus corpus = 4;
17 }
```

```
1  syntax = "proto3"; // 协议版本
2  // 定义在外部
3  enum EColor {
4      EC_Blue = 0;
5      EC_Red = 1;
6      EC_Yellow = 2;
7  }
8
9  message Box {
10     EColor color = 1;
11 }
```

# option allow\_alias = true

proto3允许在枚举中将**相同的值**分配不同的**枚举常量来定义别名**。但必须标注allow\_alias选项为true

```
1  syntax = "proto3"; //协议版本
2  //定义在外部
3  enum Enum1 {
4      //允许标记相同值使用不同的名字，例如有两个项都是1，但是名字不同，编译不出错
5      option allow_alias = true;
6      Red = 0;
7      Blue = 1;
8      Green = 1;
9  }
10
11  enum Enum2 {
12      Red = 0;
13      Blue = 1;
14      Green = 1; //编译出错，相同的值出现两次，所以错误
15  }
```

# 注意

- 枚举定义必须要包含一个映射到0的常量作为枚举的第一个元素
- 允许将枚举定义在消息体内，使用范围也在消息体内
- 枚举项常量必须在32位整数范围内
- 禁止将枚举项的值定义为负值，这回导致效率降低

# 嵌套类型

# 嵌套类型

消息内允许定义和使用消息类型，如下图所示。

```
1  syntax = "proto3";
2
3  message Person {
4
5      message Phone {
6          int32 money = 1;
7      }
8
9      Phone phone = 1;
10 }
```

# 外部使用

如果需要在父消息类型之外重用此消息类型，则可以通过使用层级路径来标注：\_Parent\_.Type\_

```
1  syntax = "proto3";
2
3  message Person {
4
5      message Phone {
6          |   int32 money = 1;
7      }
8
9      Phone phone = 1;
10 }
11
12 message Hero {
13
14     Person.Phone phone = 1;
15
16 }
```

# 多层嵌套

消息中不仅仅只支持嵌套一层，你可以根据设计需求嵌套多层也是被允许的

```
1  syntax = "proto3";
2
3  message Person {
4      message Phone { // 第一层
5          message Power { // 第二层
6
7          }
8          int32 money = 1;
9      }
10     Phone phone = 1;
11 }
```



外部导入

# 外部导入

当你希望使用的字段消息类型定义在其他文件中，你又想在当前文件中使用该怎么办？

使用import关键字可以引入已经编写好的proto文件（注意路径层级关系）

**注意尽量不要移动文件位置。如确实需要移动时，建议在原位置放置一个空的proto文件，并使用此文件引入新的地址**

```
1  syntax = "proto3";
2  message Person {
3      enum Gender {
4          NONE = 0;
5          MALE = 1;
6          FEMALE = 2;
7      }
8      Gender gender = 1;
9  }
10
11  //Person是在当前消息同文件夹下的proto文件
12  syntax = "proto3";
13  import "Person.proto";
14  message Team {
15      repeated Person persons = 1;
16  }
```

# 定义层级

# Package层级

借助package可以完成层级划分，用来防止命名污染，以解决协议消息类型之间的命名冲突。

```
1  syntax = "proto3";
2  package Uejoy.Other;
3  message Phone {
4      string name = 1;
5  }
6
7  message Person {
8      Uejoy.Other.Phone phone = 1;
9  }
10
```

map

# map (proto3)

用于创建关联映射作为数据定义的一部分，我们可以使用map关键进行构建。

```
1 syntax = "proto3";  
2  
3 message Person {  
4     |   map<int32, string> test = 1;  
5 }
```

注意：

- map字段不能使用repeated修饰
- map中存储的数据是无序的，因此不能假设map数据的顺序。
- 如合并或解析时，存在重复映射键，则默认使用最后被应用的键（后设置覆盖前设置）

oneof

# oneof (proto3)

如果您有一条包含多个字段的消息，并且最多同时设置一个字段，您可以强制执行此行为并使用 oneof 功能节省内存

```
1  syntax = "proto3";  
2  
3  message Person {  
4      |   oneof test_oneof {  
5          |       string name = 1;  
6          |       int32 age = 2;  
7          |   }  
8  }
```

可以将字段添加到oneof定义中，但是map和repeated字段禁止被添加



# 注意

- 设置oneof字段时，之后最后被设置的字段有值，其他将被清除
- 如果在解析时，遇到同一个成员的多个成员，则只使用最后一个成员
- oneof不能使用repeated
- 反射api适用oneof字段

# 更新规则

# 更新规则

由于业务需求，对于已经编写好的消息可能会涉及修改调整，那么就需要更新消息。更新消息需要遵守以下规则。

- 不要修改任何已经存在的项的ID
- 如果增加了新的项(使用了新的ID)，原有的旧的Message生成的代码进行序列化的信息，依然能被你新的Message生成的代码解析
- 项可以删除，可以修改项名称的方式，或者直接把对应的tagId标记为删除
- int32, uint32, int64, uint64, bool 是彼此兼容的，也就是说这些类型的项可以任何变更到彼此中的任何一个
- sint32, sint64 是彼此兼容的
- string 和 bytes 是彼此兼容的
- fixed32 与 sfixed32, fixed64, sfixed64是兼容的

# 编译消息

- 虚幻引擎交互设计师班 -



火星时代教育

# 下载编译器

写消息主要的目的是定义结构，定义完成需要通过编译工具，生成对应平台的源文件。

将消息文件保存到磁盘中，文件后缀为 “.proto” ，目前支持将消息转换到语言种类有Java、Kotlin、Python、C++、Go、Ruby、Objective-C 或 C# 代码。

首先需要下载编译器。下载路径（v3.20.0发行版，版本更换可从git中找到）：

<https://github.com/protocolbuffers/protobuf/releases/tag/v3.20.0>

找到对应的语言，下载资产包（CPP包）。

# 选取正确编译包

在页面中，我们可以下载到编译器的源码，也可以直接下载已经编译好的编译器包。我们目前只涉猎win平台，所以只下载win平台的包即可。

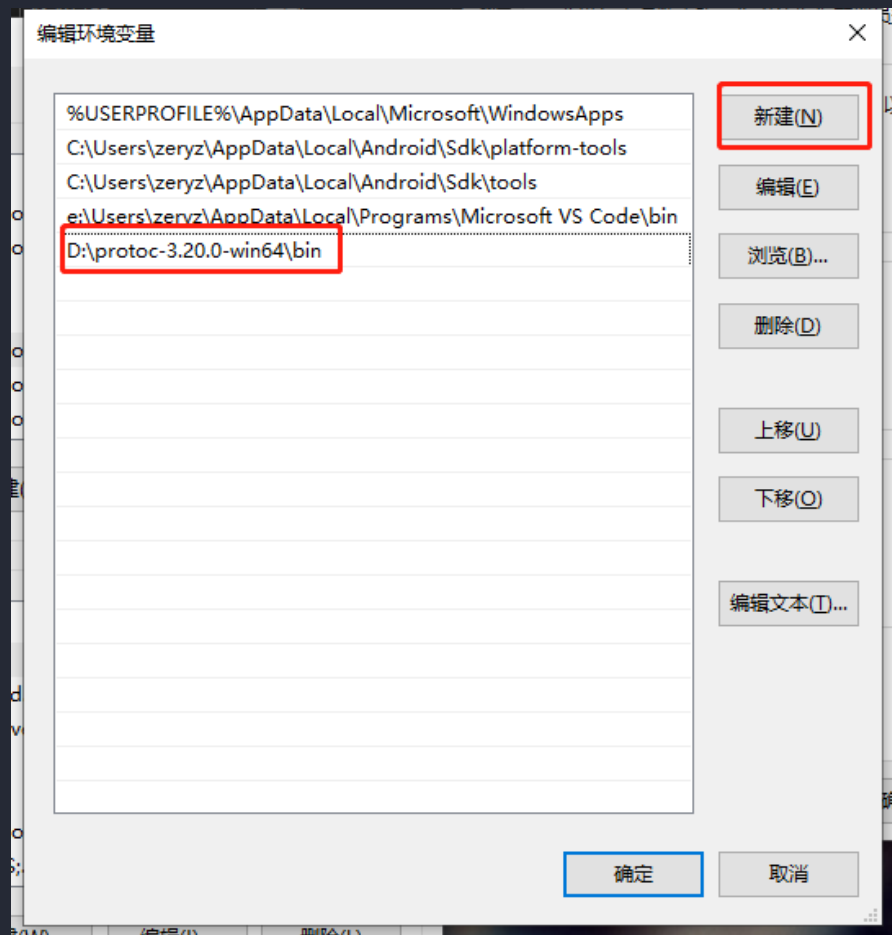
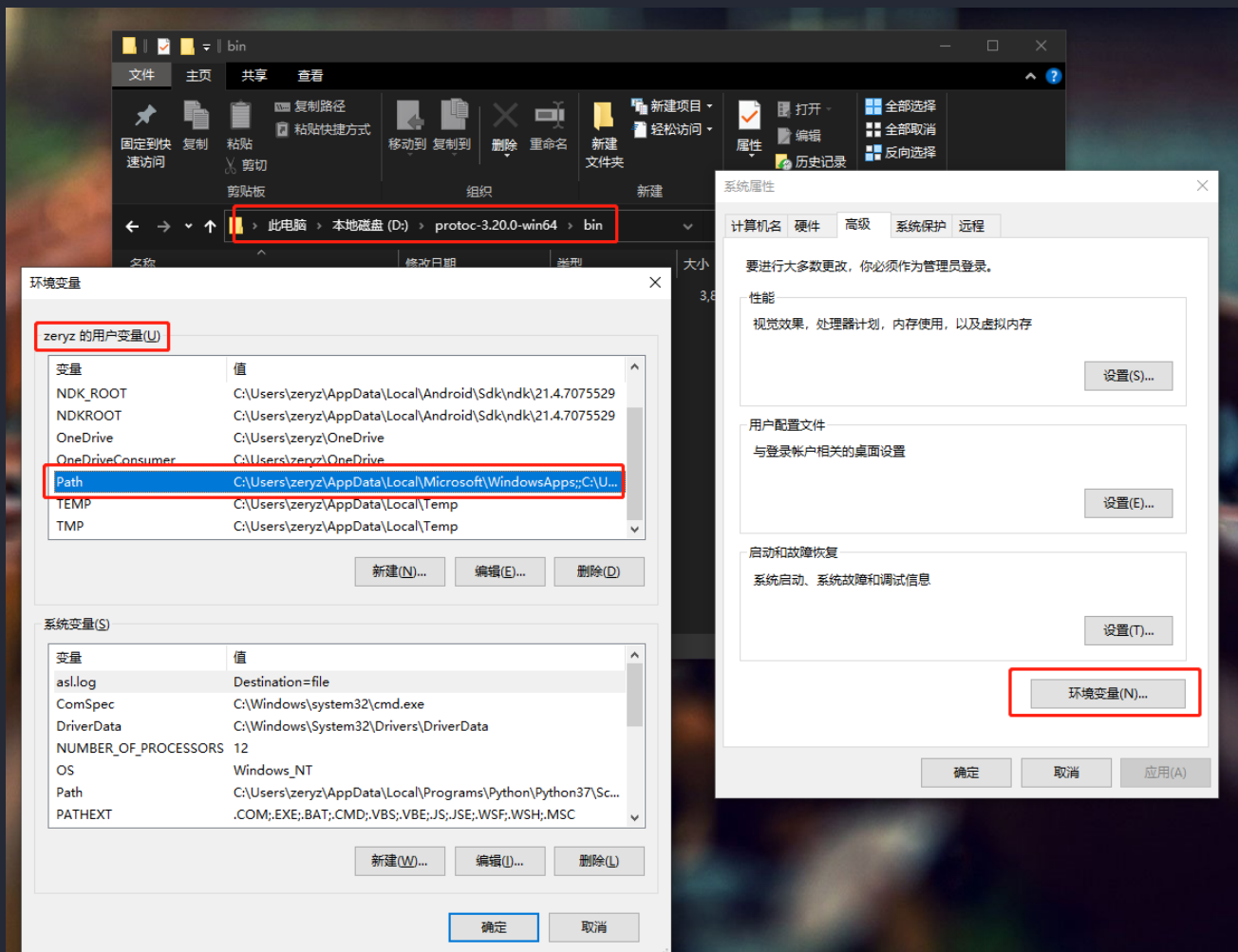
protobuf-csharp-3.20.0.tar.gz	5.33 MB
protobuf-csharp-3.20.0.zip	6.57 MB
protobuf-java-3.20.0.tar.gz	5.29 MB
protobuf-java-3.20.0.zip	6.66 MB
protobuf-js-3.20.0.tar.gz	4.86 MB
protobuf-js-3.20.0.zip	6 MB
protobuf-objectivec-3.20.0.tar.gz	4.99 MB
protobuf-objectivec-3.20.0.zip	6.15 MB
protobuf-php-3.20.0.tar.gz	4.91 MB
protobuf-php-3.20.0.zip	6.03 MB
protobuf-python-3.20.0.tar.gz	4.92 MB
protobuf-python-3.20.0.zip	6.04 MB
protobuf-ruby-3.20.0.tar.gz	4.83 MB
protobuf-ruby-3.20.0.zip	5.89 MB
protoc-3.20.0-linux-aarch_64.zip	1.72 MB
protoc-3.20.0-linux-ppcle_64.zip	1.86 MB
protoc-3.20.0-linux-s390_64.zip	2.01 MB
protoc-3.20.0-linux-x86_32.zip	1.57 MB
protoc-3.20.0-linux-x86_64.zip	1.64 MB
protoc-3.20.0-osx-aarch_64.zip	2.58 MB
protoc-3.20.0-osx-x86_64.zip	2.58 MB
protoc-3.20.0-win32.zip	1.14 MB
protoc-3.20.0-win64.zip	1.47 MB
Source code (zip)	
Source code (tar.gz)	

19

19 people reacted

# 配置环境变量

编译需要使用指令完成，指令执行需要依赖执行文件。为保证指令可执行，需要配置环境变量。



# 检查配置结果

配置完成后，打开win窗口，输入protoc，然后回车，看是否有输出指令帮助。如果有，则配置成功。

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19043.1586]
(c) Microsoft Corporation。保留所有权利。

C:\Users\zervyz>protoc
Usage: protoc [OPTION] PROTO_FILES
Parse PROTO_FILES and generate output based on the options given:
  -IPATH, --proto_path=PATH  Specify the directory in which to search for
                             imports.  May be specified multiple times;
                             directories will be searched in order.  If not
                             given, the current working directory is used.
                             If not found in any of the these directories,
                             the --descriptor_set_in descriptors will be
                             checked for required proto file.
  --version                  Show version info and exit.
  -h, --help                 Show this text and exit.
  --encode=MESSAGE_TYPE     Read a text-format message of the given type
                             from standard input and write it in binary
                             to standard output.  The message type must
                             be defined in PROTO_FILES or their imports.
  --deterministic_output    When using --encode, ensure map fields are
                             deterministically ordered.  Note that this order
                             is not canonical, and changes across builds or
                             releases of protoc.
  --decode=MESSAGE_TYPE     Read a binary message of the given type from
                             standard input and write it in text format
                             to standard output.  The message type must
                             be defined in PROTO_FILES or their imports.
  --decode_raw              Read an arbitrary protocol message from
                             standard input and write the raw tag/value
                             pairs in text format to standard output.  No
```



# 编译指令

启动控制台窗口，将控制台执行路径设置到消息路径下。通过指令完成编译：参照下图

```
C:\Windows\system32\cmd.exe

D:\protobuf>protoc --proto_path=./ --cpp_out=./ Person.proto

D:\protobuf>_
```

`protoc --proto_path=./ --cpp_out=./ Person.proto`

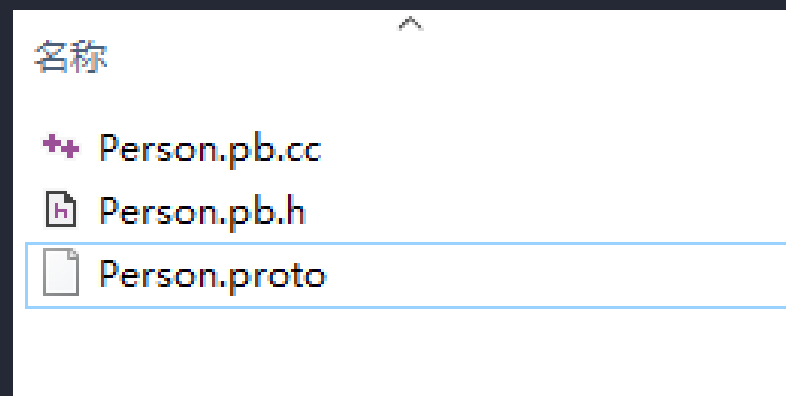
指令说明：

`protoc`：执行应用程序名称

`--proto_path`：引入文件路径，尝试在此路径搜寻

`--cpp_out`：编译cpp文件后输出保存路径

`Person.proto`：被编译的消息文件



# 编译protobuf库

- 虚幻引擎交互设计师班 -



火星时代教育

# C++中使用

我们目前只讨论C++中如何使用protobuf，其他语言不在讨论范围内。

基础要求：win平台，编译器visual studio，语言c++

首先，我们需要先下载protobuf源码用于编译库文件（**下载源码的目的是编译protobuf库**）

地址：<https://github.com/protocolbuffers/protobuf/releases>

▼ Assets 29	
 <a href="#">protobuf-all-3.20.0.tar.gz</a>	7.45 MB
 <a href="#">protobuf-all-3.20.0.zip</a>	9.66 MB
 <a href="#">protobuf-cpp-3.20.0.tar.gz</a>	4.61 MB
 <a href="#">protobuf-cpp-3.20.0.zip</a>	5.61 MB
 <a href="#">protobuf-csharp-3.20.0.tar.gz</a>	5.33 MB
 <a href="#">protobuf-csharp-3.20.0.zip</a>	6.57 MB
 <a href="#">protobuf-java-3.20.0.tar.gz</a>	5.29 MB
 <a href="#">protobuf-java-3.20.0.zip</a>	6.66 MB
 <a href="#">protobuf-js-3.20.0.tar.gz</a>	4.86 MB
 <a href="#">protobuf-js-3.20.0.zip</a>	6 MB

# CMake

下载完成后，我们需要借助CMake工具，编译项目。通过以下地址下载CMake文件。

<https://cmake.org/download/>

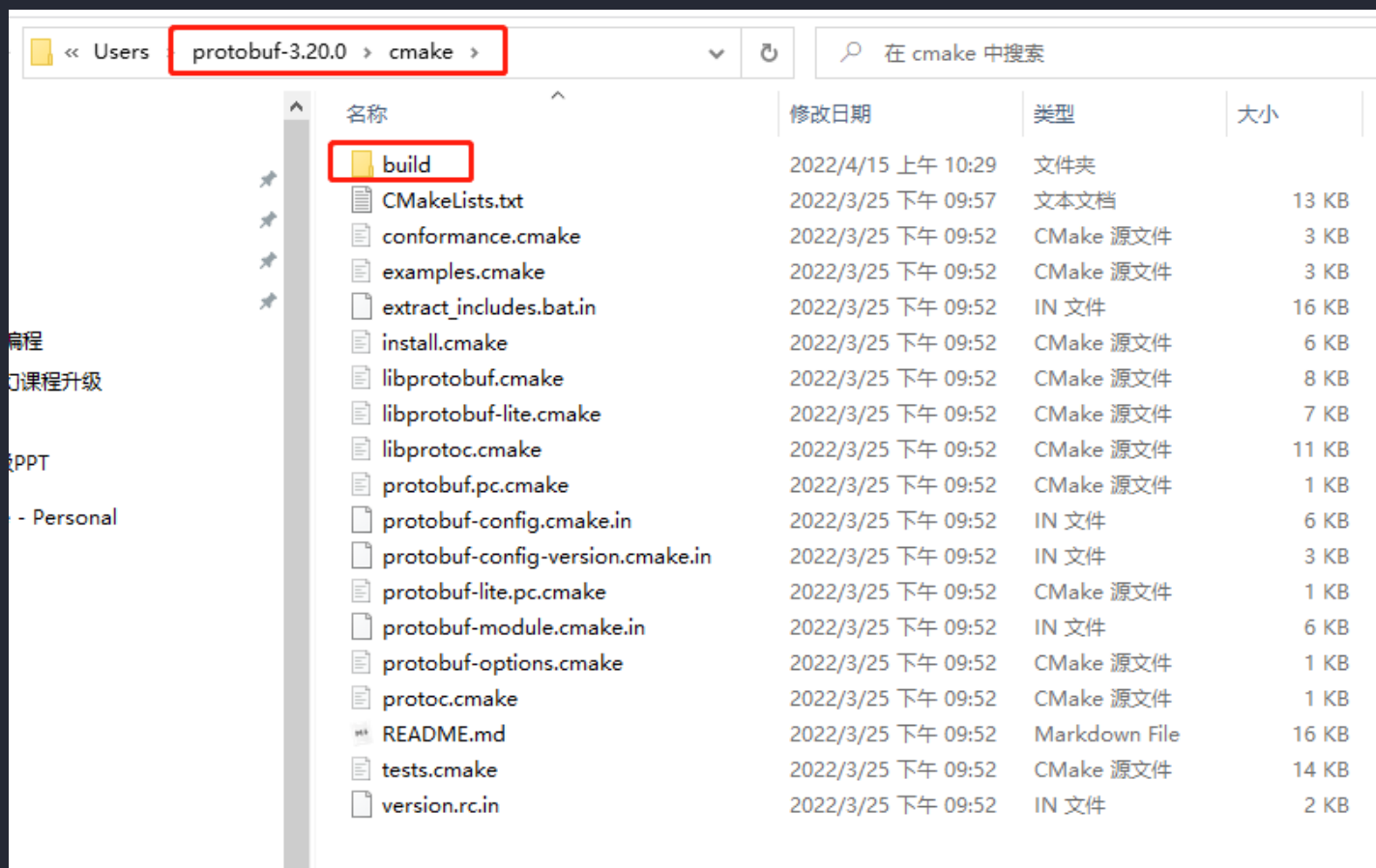
下载完成后，直接一路下一步安装软件，**不用修改安装路径。**

Binary distributions:

Platform	Files
Windows x64 Installer	<a href="#">cmake-3.23.1-windows-x86_64.msi</a>
Windows x64 ZIP	<a href="#">cmake-3.23.1-windows-x86_64.zip</a>
Windows i386 Installer	<a href="#">cmake-3.23.1-windows-i386.msi</a>
Windows i386 ZIP	<a href="#">cmake-3.23.1-windows-i386.zip</a>
macOS 10.13 or later	<a href="#">cmake-3.23.1-macos-universal.dmg</a>
	<a href="#">cmake-3.23.1-macos-universal.tar.gz</a>

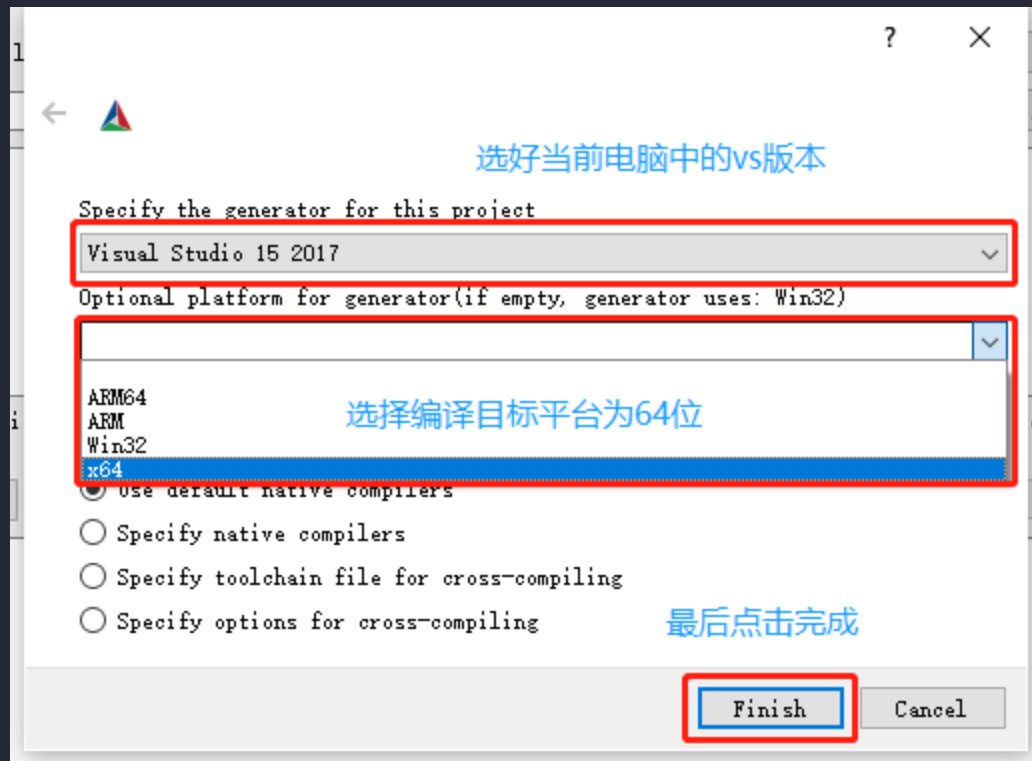
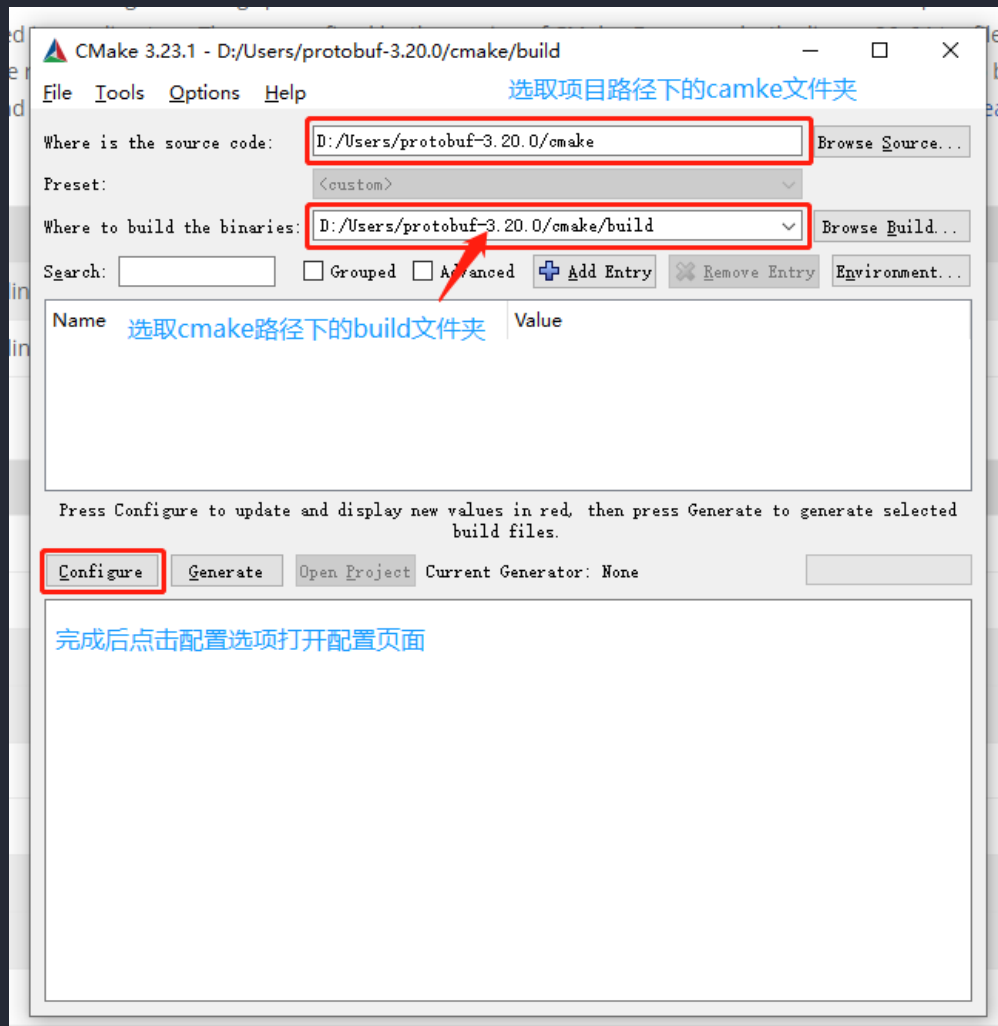
# CMake构建编译环境

解压之前下载好的protobuf压缩包（解压路径不能有中文）。在解压完成的文件夹中找到cmake文件夹，在cmake文件夹中**创建build文件夹**。



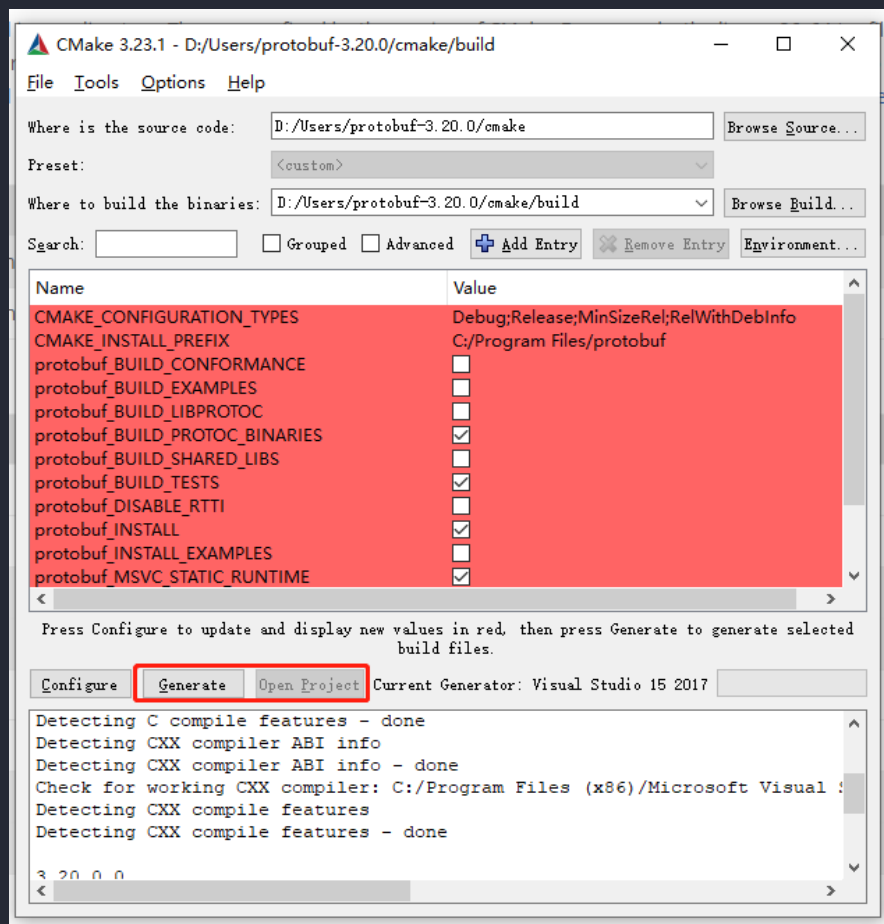
# CMake构建编译环境

打开cmake工具，配置环境关系。



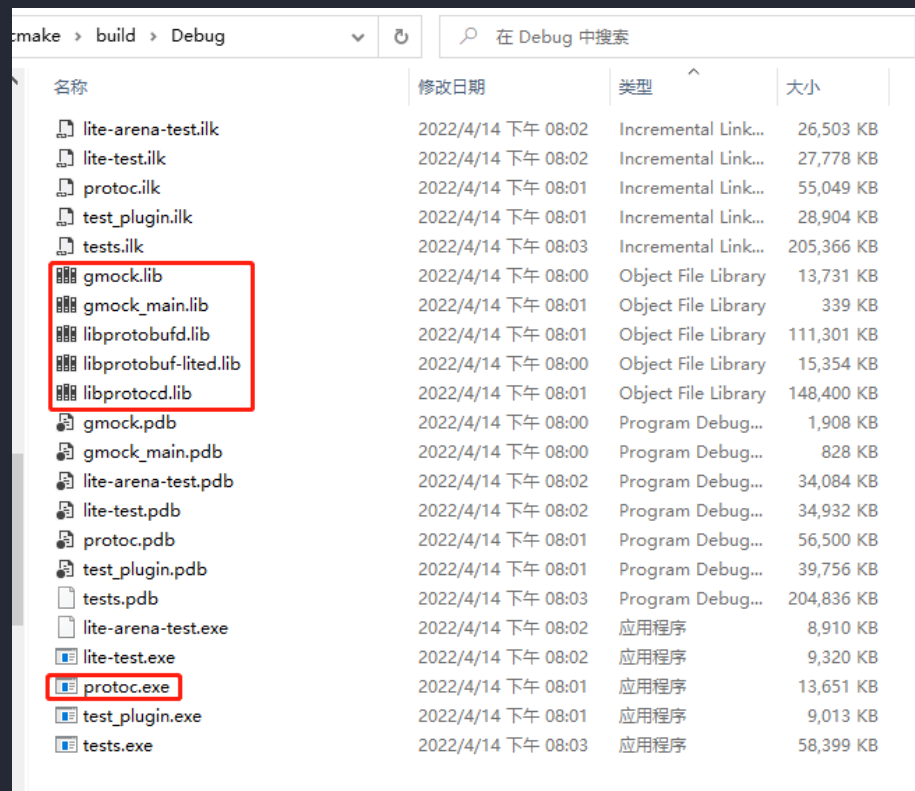
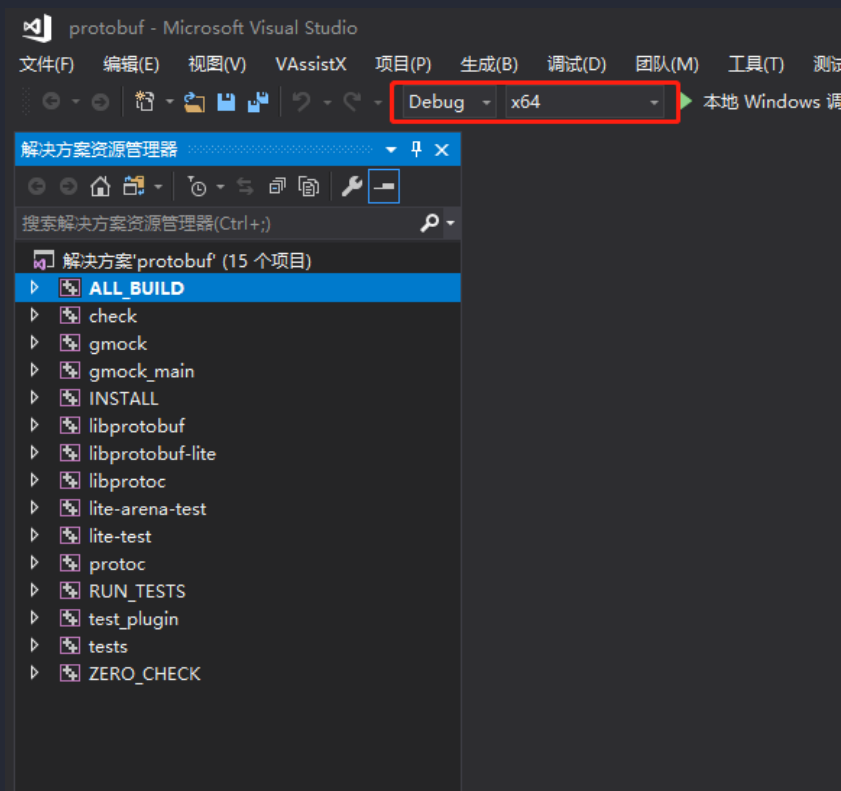
# CMake构建编译环境

配置完成后需要选出构建参数，我们直接使用默认的即可。然后点击生成vs项目配置文件。  
生成配置文件后，点击open project即可打开vs。



# 编译工程

打开VS工程，我们可以选择编译所有工程，也可以独立编译某些工程，为减少大家记忆负担，我们选择编译所有工程。编译完成后会产生对应的库文件，并且会生成protoc编译器（**这个编译器就是我们在git上下载的已经编好的编译器，只不过我们是通过源码编译生成**）





# Protobuf消息

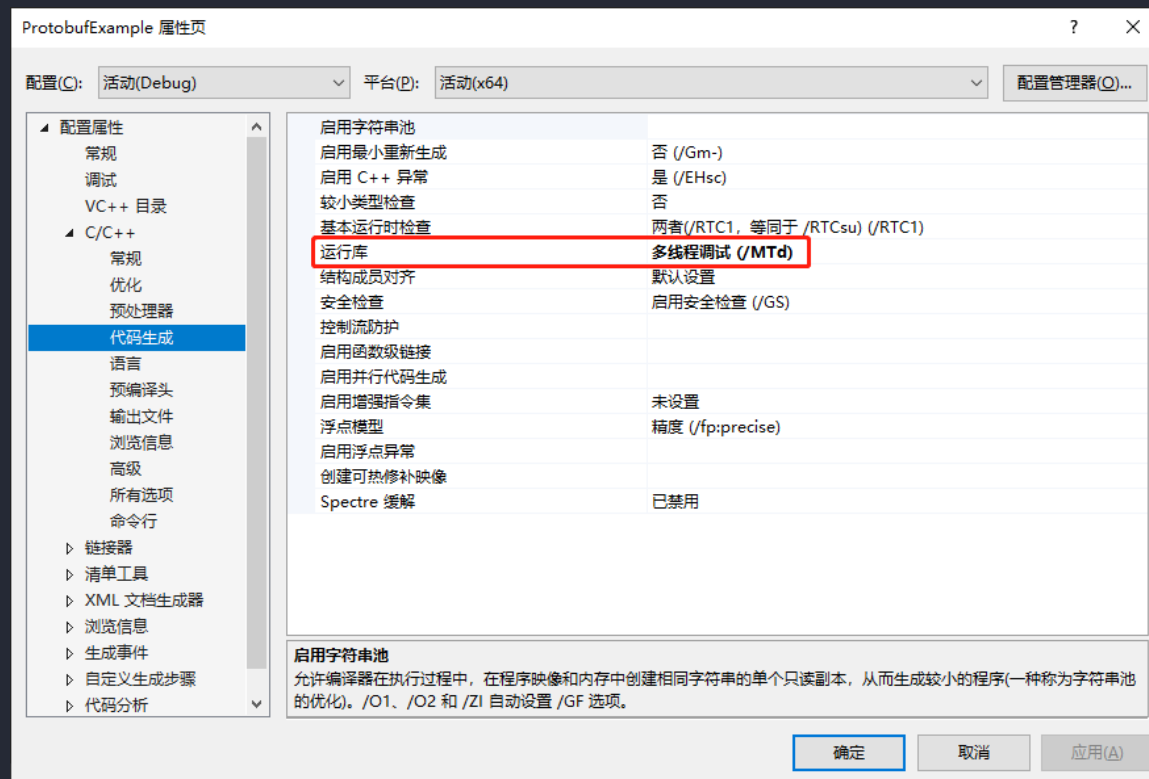
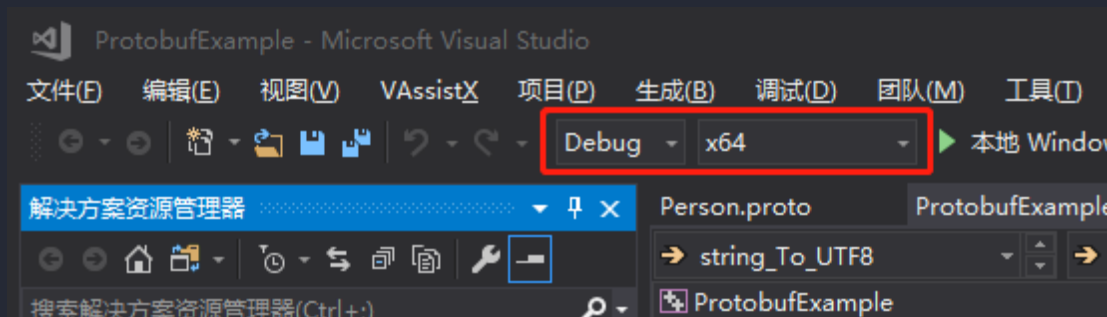
- 虚幻引擎交互设计师班 -



火星时代教育

# 创建C++工程

打开VS，创建C++控制台应用程序。打开后，因为编译的库文件版本是x64，所以需要将当前项目改成x64版本。并将项目配置运行库方式修改为“多线程调试 (/MTd)”（因为库编译运行库方式为多线程调试）



# 配置编译项

我们需要将编译好的库引入到当前项目中。配置动作如下：

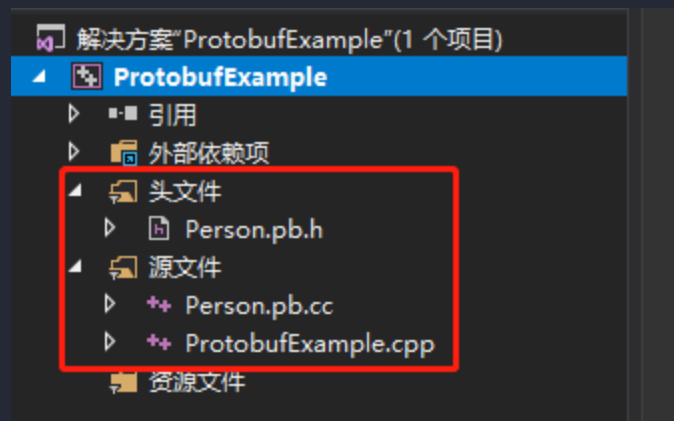
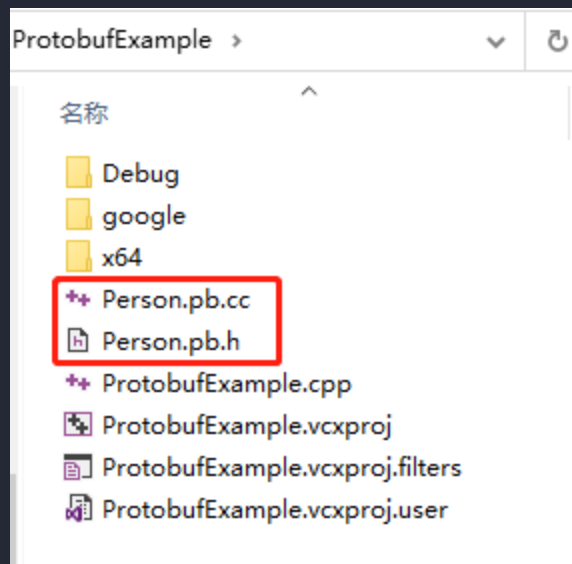
**增加头文件路径：** 配置属性-C/C++-常规-附加包含目录（选择protobuf路径下的src文件夹）

**增加库文件路径：** 配置属性-链接器-常规-附加库目录（路径protobuf/cmake/build/Debug）

**配置附加依赖项：** 配置属性-连接器-输入-附加依赖项（添加libprotobufd.lib, libprotocd.lib库）

# 导入消息文件

将编译好的消息文件，头文件和源文件一并拷贝到项目目录下，并在VS中导入。



# 编写测试代码

参照如下方法可以测试消息结构数据。

```
int main()
{
    //构建消息对象数据
    Person p1;
    //写入数据
    p1.set_name("Jame");
    p1.set_age(50);
    //输出数据
    cout << p1.age() << endl;
    cout << p1.name() << endl;
    //序列化数据, 将内存中的消息对象数据转换为其他数据形式 例如我们序列化到字符串中
    string data;
    //把p1对象转成字符串格式数据
    p1.SerializePartialToString(&data);
    //反向序列化
    Person p2;
    //反向序列化data字符串数据到p2
    p2.ParseFromString(data);
    //输出数据和p1一致
    cout << p1.age() << endl;
    cout << p1.name() << endl;

    std::cout << "Hello World!\n";
}
```

```
1 syntax = "proto3";
2
3 message Person {
4     string name = 1;
5     int32 age = 2;
6 }
```

# 虚幻中的Protobuf

- 虚幻引擎交互设计师班 -



火星时代教育

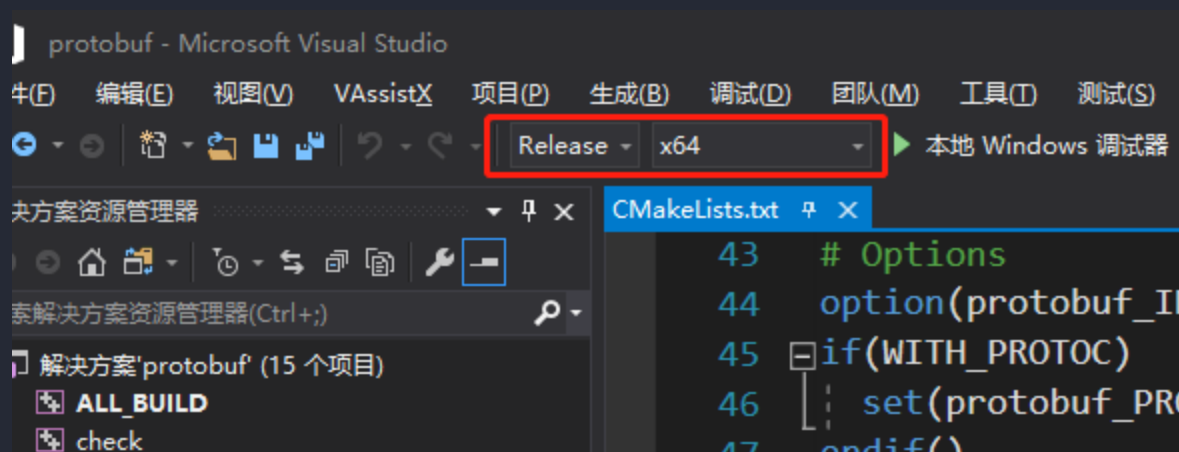
# 前言

库引入方式与普通库静态库相同，但是需要注意以下几点问题：

1. 虚幻引擎是x64平台编译，所以编译库需要保证也是x64版
2. Protobuf库在编译时，需要编译为发行版本，Debug版本中有部分逻辑代码虚幻引擎不支持
3. 注意库编译时修改运行库类型为MD（多线程dll）
4. 只需要编译libprotobuf和libprotoc库即可

# 编译器设置

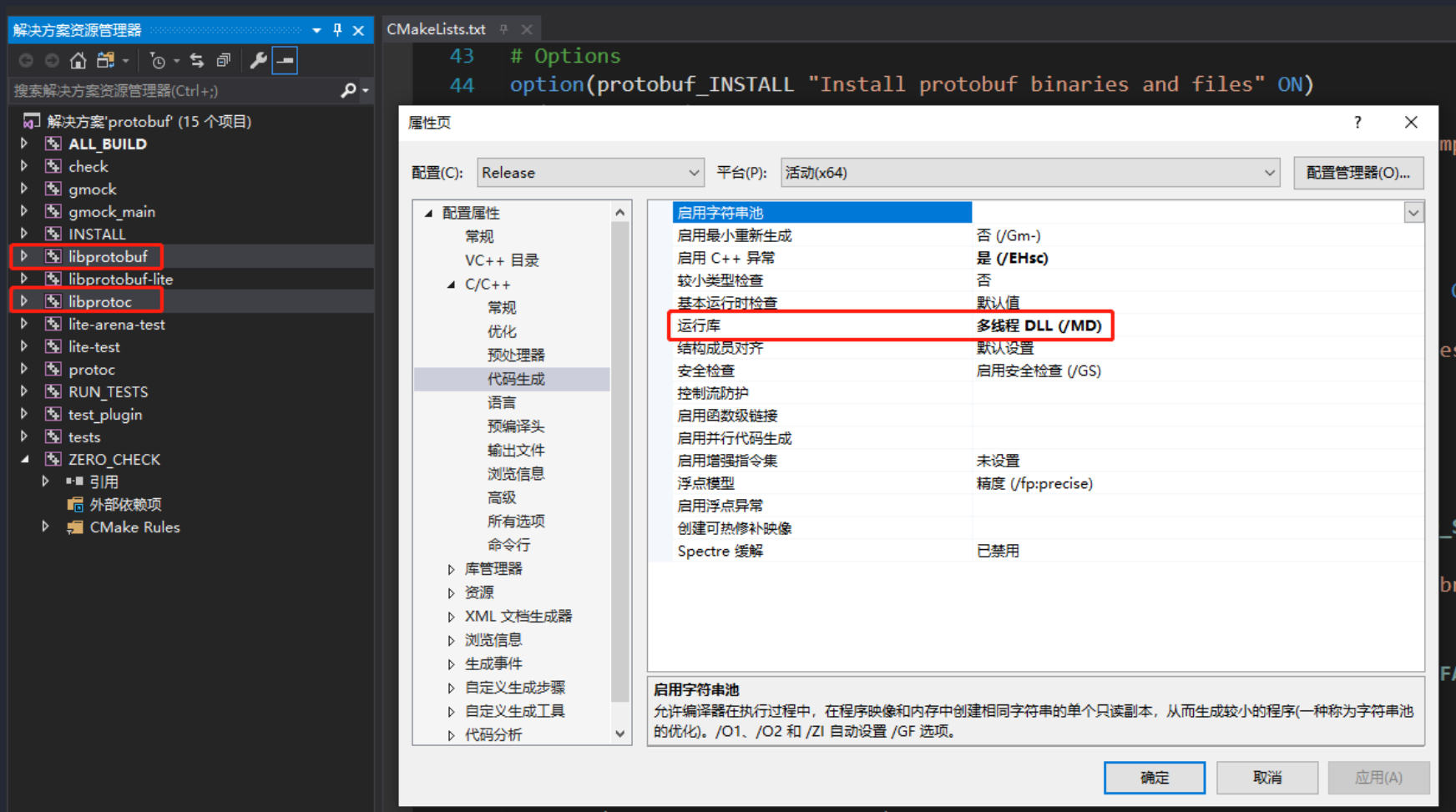
修改编译版本和目标平台





# 修改运行库类型

找到库项目，在属性-配置属性-C/C++-代码生成-运行库，进行调整。**调整后直接右键生成项目即可。**



# 导入引擎

引入方式可以通过两种：模块引入，插件引入。两者基本相同，我们只讨论模块引入。

找到项目路径Source文件夹，新建文件夹，ThirdParty。在ThirdParty文件夹下创建模块文件夹，我创建的模块名字是UEProtobuf，所以文件夹名字也是UEProtobuf。然后在Protobuf文件夹内新建lib和include文件夹。结构如下：

Source|

ThirdParty|

UEProtobuf|

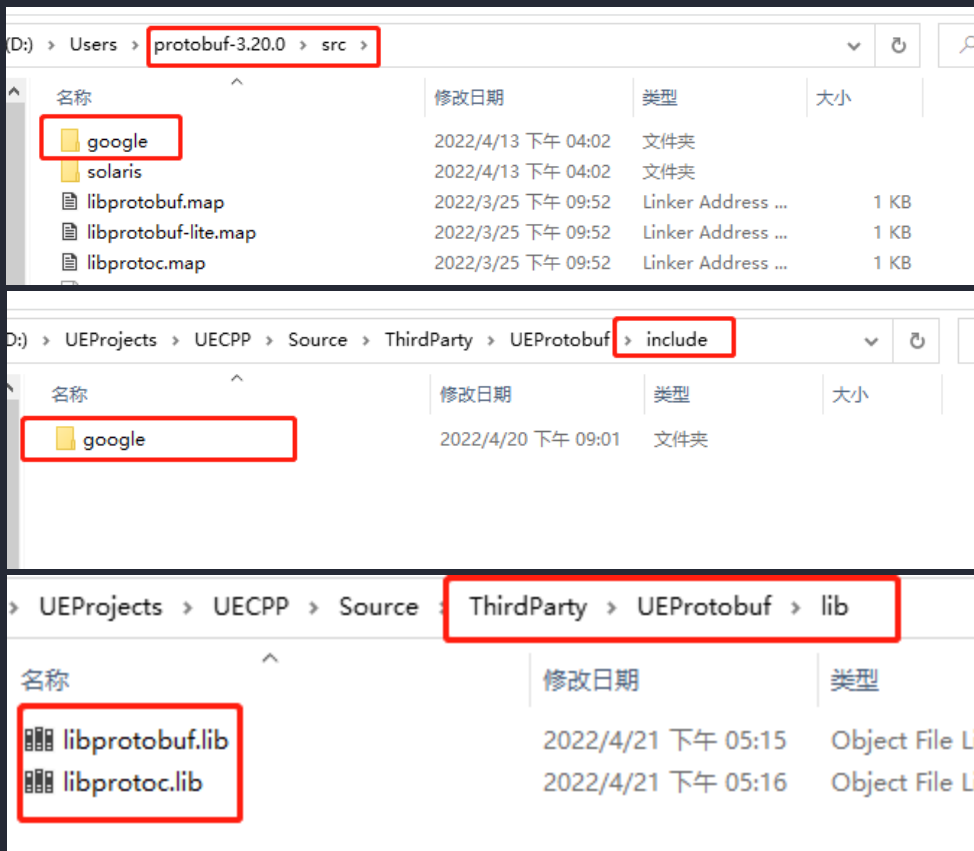
include|

lib|



# 拷贝库文件源码文件

将编译好的库文件拷贝到lib文件夹中，并将protobuf源码中的头文件拷贝到include文件中（**一般源码路径在下载后的protobuf解压后的src中**）



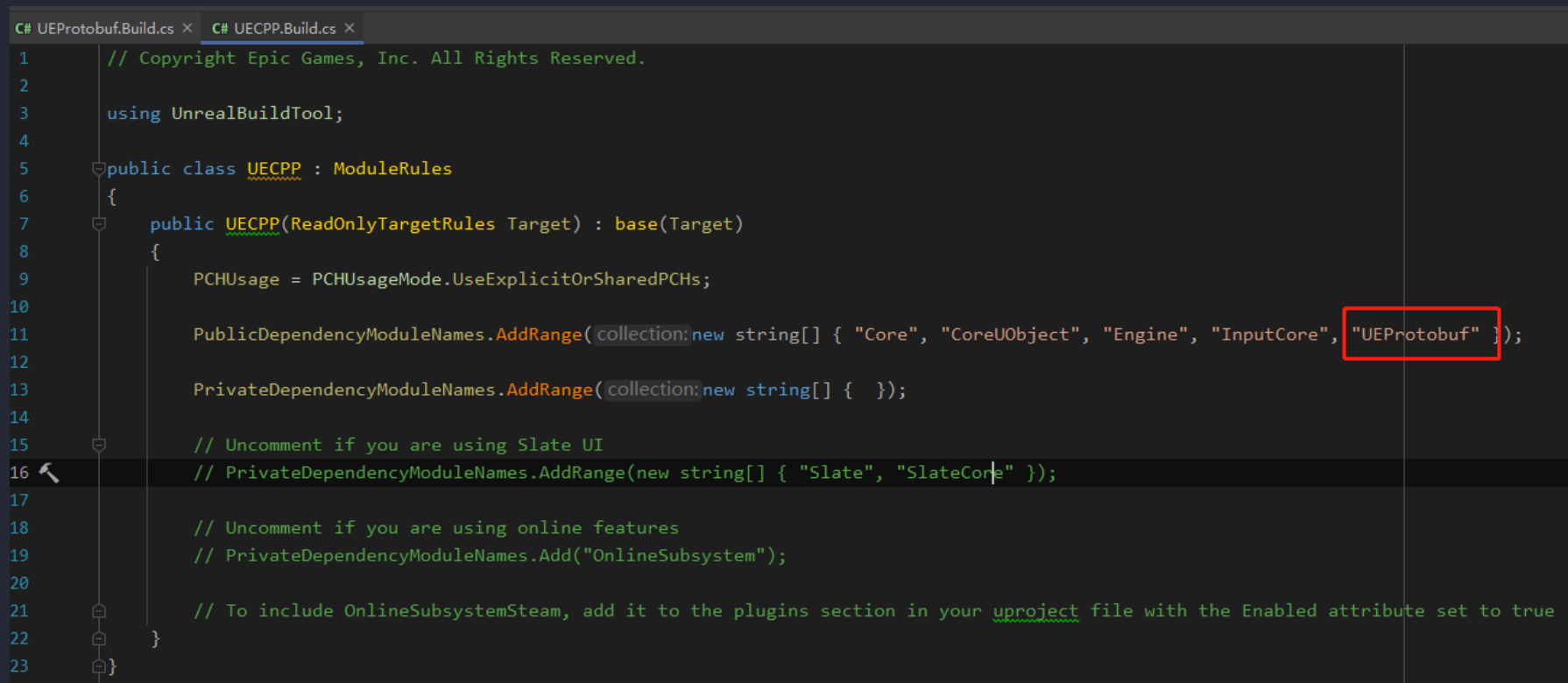
# 创建build文件

在模块文件夹下创建build文件，用于UBT工作时可以识别模块。创建方法，右键创建文本文件，修改名字为：模块名.Build.cs。参照下截图编写build文件内容。需要引入库文件，库路径，

```
C# UEProtobuf.Build.cs x
1 // Copyright 2015-2022 UEJoy 自动生成文件 Version 1.0.3
2
3 using UnrealBuildTool;
4 using System.IO;
5
6 public class UEProtobuf : ModuleRules
7 {
8     public UEProtobuf(ReadOnlyTargetRules Target) : base(Target)
9     {
10         Type = ModuleType.External;
11         PublicSystemLibraryPaths.Add(item: Path.Combine(ModuleDirectory, "lib"));
12         //头文件路径
13         PublicIncludePaths.Add(item: Path.Combine(ModuleDirectory, "include"));
14         //设置依赖库名称
15         PublicSystemLibraries.Add(item: "libprotobuf.lib");
16         PublicSystemLibraries.Add(item: "libprotoc.lib");
17     }
18 }
19 }
```

# 引入模块

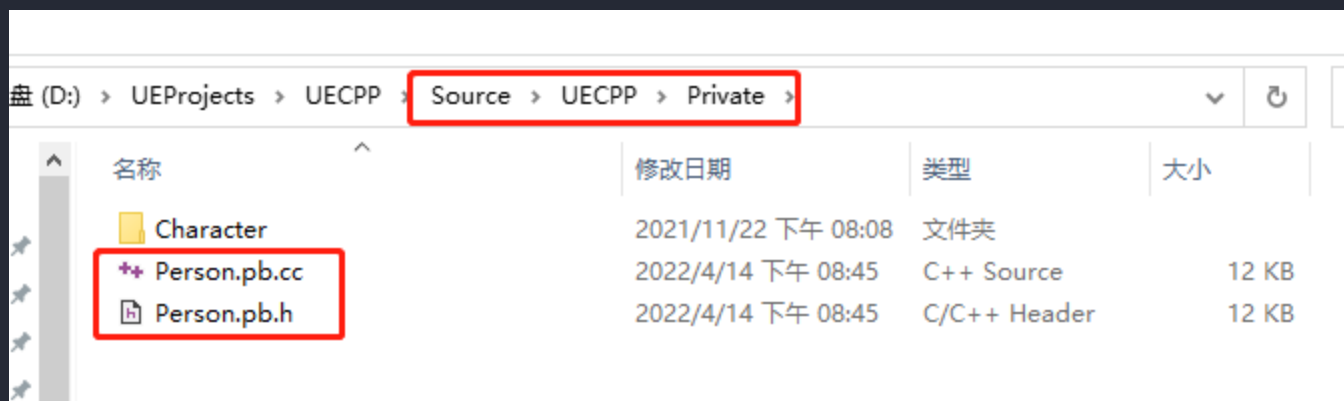
在项目中引入模块，并编译工程。



```
C# UEProtobuf.Build.cs x C# UECPublic.Build.cs x
1 // Copyright Epic Games, Inc. All Rights Reserved.
2
3 using UnrealBuildTool;
4
5 public class UECPublic : ModuleRules
6 {
7     public UECPublic(ReadOnlyTargetRules Target) : base(Target)
8     {
9         PCHUsage = PCHUsageMode.UseExplicitOrSharedPCHs;
10
11         PublicDependencyModuleNames.AddRange(collection:new string[] { "Core", "CoreUObject", "Engine", "InputCore", "UEProtobuf" });
12
13         PrivateDependencyModuleNames.AddRange(collection:new string[] { });
14
15         // Uncomment if you are using Slate UI
16         // PrivateDependencyModuleNames.AddRange(new string[] { "Slate", "SlateCore" });
17
18         // Uncomment if you are using online features
19         // PrivateDependencyModuleNames.Add("OnlineSubsystem");
20
21         // To include OnlineSubsystemSteam, add it to the plugins section in your uproject file with the Enabled attribute set to true
22     }
23 }
```

# 使用Protobuf

将编译好的消息文件头文件和源文件拷贝到项目工程private文件夹下。然后在工程中即可使用。



```
void AUECPPGameModeBase::CallFun()
{
    Person p;
    p.set_age(value: 50);
    p.set_name(arg0: "nihao");
    UE_LOG(LogTemp, Log, TEXT("==%d"), p.age());
}
```

The slide features several decorative orange elements: a small triangle above the 'h' in 'Thanks', a circle above the 'i', a cross to the left of the 'T', a cross to the right of the 's', a cross at the bottom right, and a circle at the bottom left. There are also orange curved lines and a series of four orange arrows pointing left at the top right.

# Thanks

-火星时代游戏设计学院-