Creator 框架设计

本节课题: **第009课 Protobuf 协议模块设计**

主　　讲: **Blake老师**

【第001节】: ProtoBuf是解决什么问题的;

客户端---》服务器 发送一个数据结构;

JavaScript数据结构---》序列化(二进制数据)　　--->传送---》反序列化---> 服务端变成语言数据结构;

序列化/反序列化:

a: 能够实现功能;

b: 序列化、反序列，跨语言的;

通用方案: Json(主流的编程语言都由json库), Xml; --->文本模式;

数据结构---》json/xml文本---> json/xml文本---》数据结构

{uname="blake", upwd="123456"} --->优点，简单; 缺点: 明文, 体积大,编码解码性能低(40%~60%);

二进制为代表: Protobuf，能够支持多编程语言，实现序列化与反序列化;

Protobuf的原理:

"uname" = "blake", upwd = "123456"，age = 10;

我们自己来做, 二进制; 规定: 第一个字符串是uname, 第二个字符串upwd;

[blake\0123456\0]

encode() { [blake\0123456\0] }　--->decode() { uname = "blake"; upwd "123456"}

优点: 体积小, 不是明文, 缺点: 每个数据结构都要由一个这样的规定;　--->每个数据结构 decode/encode;

编码解码性能高;

每个协议写encode/decode 代码: 不便:

a: 每个协议你都要写，b: 每个编程语言，你都要写;

开发一些底层的代码库(C++/Java/Js/C#等):

1: 完成一些基础数据类型的二进制的编码解码;　----> 多个编程语言都要实现(实现一次就可以了)

float, double, int, string, byte, ...　int = 8;　--->byte = 8;　--->几个bit来进行编码--->节约我们的体积;

2: 自动生成每个协议对应的endode/decode代码 ---》调用这些基础的编码解码的数据;

---> “编译器” + "协议描述文件"

[blake\0123456\0]

```
message Login {
    string uname = 1;
    string upwd = 2;
}
```

step1: 编译器根据， 协议描述 生成---》编程语言等价的数据结构;

struct Login { string uname; string upwd;}

step2: 自动生成编码解码函数:

数据结构---》二进制;
Login login = new Login(); login.uname = "blake"; login.upwd = "123456";
encode(login) ---> [blake\0123456\0] --->序列号---》编码我们的二进制数据
decode (buf) ---> Login login = new Login(); login.uname = 1个数据, login.upwd = 2;

protobuf: 提供了一个数据结构的通用描述文件语法: (描述一次, 按照protobuf的要求的语法)
提供了一个编译器---》自动生成，类型，编码解码函数; (Java/C++/C#等)
提供了一个基础的运行时库, (Java/C++/C#/JS/Python等)

step1: 编写我们的数据结构描述文件; --->协议描述文件;
step2: 使用protobuf 提供的编译器，把协议文件，编译成客户端、服务器的对应编程语言的代码;
step3: protobuf runtime库(Java版, C++版本等) ---> 内置到你的项目中;

step1: 编写我们的数据结构描述文件; --->协议描述文件--->Json格式;
step2: json/xml runtime库(Java版, C++版本等) ---> 内置到你的项目中;

【第002节】: ProtoBuf静态解析与动态解析;

1: "编译型"编程语言: C++, Java, C#, Js
ProtoBuf--->静态解析---》编码解码的代码, 代码体积大一些; --->不带协议描述文件---》项目---》安全;

2: Lua, Js ===>微信小游戏平台;
ProtoBuf ---> 动态解析;--->动态加载协议描述文件---》运行的时候，根据协议描述文件，再解析出来;
动态解析的性能---》静态解析要差一些;代码体积小一些; --->一定要带一个协议描述文件 --->项目---》不安全;

3: step1: 协议编写; ---> 带在项目里面;
step2: 编码解码我们的数据结构; ---> runtime 库;

【第003节】: Cocos Creator Protobuf的使用;
1: protobufjs --->解释型runtime库
2: 资源文件加里面添加一个协议描述文件;

```
2  import { _decorator, Component, Node, TextAsset } from 'cc
3
4  declare const protobuf: any;
5
6  export class ProtoMgr extends Component {
7      public static Instance: ProtoMgr = null as unknown as
8
```

【第004节】: ProtoMgr 的实现;