



추가 내용

MM4220 게임서버 프로그래밍
정내훈

게임 엔진

• 게임서버 엔진의 종류

종류	차별성	지원 운영체제	지원 언어
프라우드넷(ProudNet) http://www.proudnet.com	클라이언트-서버 간 네트워킹과 클라이언트 간 직접 네트워킹을 단순하게 사용할 수 있게 한다. 네트워크 암호화, 압축, 흐름 제어 등 기능을 제공한다.	윈도, 리눅스, iOS, 안드로이드, 플레이스테이션 4, WebGL	C++, C#
포톤 서버(Photon Server) https://www.photonengine.com	다양한 플랫폼의 게임 클라이언트를 지원한다. Unity Engine에 특화되어 있다.	윈도, iOS, 안드로이드, 플레이스테이션, Xbox, WebGL	C#
게임스파크(GameSparks) https://www.gamesparks.com	로그인, 매치메이킹, 플레이어 정보 관리 등을 서버 개발 없이도 가능하게 한다.	구매받지 않는다 (SaaS 형태로 되어 있다).	C++, C#, Objective-C, 자바
플레이팩(PlayFab) http://www.playfab.com	게임스파크와 유사	구매받지 않는다 (SaaS 형태로 되어 있다).	C++, C#, Lua, Objective-C, 자바, 자바스크립트
락넷(RakNet) http://www.jenkinssoftware.com	10년 전 즈음 유니티 엔진과 언리얼 엔진에 기본 내장되었던 적이 있다.	윈도, 리눅스, iOS, 안드로이드	C++, C#

프라우드 넷

- 한국에서 개발된 네트워크 엔진
- 초창기에는 P2P를 위한 출판칭으로 유명했음
- 초창기에는 일반 C-API였으나, C++11 친화적으로 진화, C# 지원
- Cross Platform
 - Windows, Linux, Android, iOS, PlayStation4
- 높은 생산성, 고성능
- PIDL(Proudnet Interface Definition Language)을 사용한 프로토콜 관리

프라우드 넷

- 기능
 - 기본 네트워킹. 서버/클라이언트 클래스
 - Wifi/Cellular 간 Handover
 - Remote Procedure Call
 - P2P Communication
- 평가
 - 좋은 엔진이고, 생산성도 높고, 성능도 좋고, 가격도 싸다.
 - MMO서버를 프라우드 넷으로 만드는 경우는 거의 없다. (성능 문제가 아니라 서버 프로그래머의 자존심???)
 - 서버 제작 시 가장 큰 문제는 멀티쓰레드 문제인데 그 문제를 해결해 주지 않는다.

유용한 라이브러리

- 프로토콜 관리 문제 해결
 - 고성능
 - 다른 언어, 다른 운영체제, 다른 CPU사이의 통신 문제 해결
- ProtocolBuffer
 - <https://developers.google.com/protocol-buffers>
 - 플랫폼과 언어에 독립적인 패킷 기술 라이브러리
 - 사용하기 쉽고, Configuration 파일로도 사용하기 좋음
- FlatBuffer
 - <https://google.github.io/flatbuffers/>
 - ProtocolBuffer의 단점인 성능을 보완
 - 사용 편의성을 희생

Advanced Issues

- ASIO : 멀티쓰레드 에서의 구현
- 새로운 네트워크 API
 - Windows : RIO
 - Linux : IO-uring
- 분산 언어를 사용한 분산 서버
 - Elixir
 - Erlang
- Seamless 분산 서버

Protocol Buffers

- <https://developers.google.com/protocol-buffers>
- 정의 : Protocol buffers are a language-neutral, platform-neutral extensible mechanism for **serializing** structured data.

Protocol Buffers

- 구글에서 개발한 언어, 컴퓨터 구조, 운영체제에 독립적인 구조화된 데이터의 직렬화 라이브러리로서, XML과 비슷하지만 더 작고, 빠르고, 간단하다. (tinycloud?)
- 많은 언어를 지원한다.
 - C++, C#, Dart, Go, Java, Python
- 확장 가능 : 수정한 protocol도 호환 가능
 - minor 버전은 호환, major 버전은 호환 불가능으로 이해하면 됨.

Protocol Buffers

- 용도
 - 프로토콜정의
 - Configuration File
 - Save File
 - 구조화된 데이터의 전송/보관
- 비교
 - 전용 Binary 포맷 : 호환성, 유지관리 비용
 - 스트링 : 유지관리 비용, 성능 하락
 - XML : 용량 증가, 성능 하락

Protocol Buffers

- 프로토콜 버퍼에서 데이터를 정의한다.
 - 고유 포맷(언어?)가 있다. .proto 확장자를 가진다.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Protocol Buffers

- message가 데이터의 단위이다.
- 각 메시지는 field를 갖는다.
- field는 이름과 밸류 type을 갖는다.
- 밸류 type은
 - 정수, 실수, Boolean, string, binary이다.
 - 메시지도 올 수 있다.
- 옵션으로 다음 속성을 지정한다.
 - required : 필수, optional : 옵션, repeated : 반복 가능(0도 가능)
 - []에 추가 속성을 넣을 수 있다. 예) default, packed
- field 뒤에 붙는 숫자는 field를 구분하는 index이다.
 - unique해야 하고, 이것이 바뀌면 serialize된 data의 해석이 불가능해진다.
 - 1 - 15가 적당하고 더 큰값을 사용하면 serialize된 data의 size가 커진다.

```
message Person {  
  required string name = 1;  
  required int32 id = 2;  
  optional string email = 3;  
  
  enum PhoneType {  
    MOBILE = 0;  
    HOME = 1;  
    WORK = 2;  
  }  
  
  message PhoneNumber {  
    required string number = 1;  
    optional PhoneType type = 2 [default = HOME];  
  }  
  
  repeated PhoneNumber phone = 4;  
}
```

Protocol Buffers

- 주의
 - required 필드는 호환성을 위해 반드시 유지되어야 한다.
 - 삭제 시 바뀐 프로토콜에서 해석 불가능.
- 설명문 : C/C++ 스타일

```
/* SearchRequest represents a search query, with pagination options to
 * indicate which results to include in the response. */

message SearchRequest {
  required string query = 1;
  optional int32 page_number = 2;  // Which page number do we want?
  optional int32 result_per_page = 3;  // Number of results to return per page.
}
```

Protocol Buffers

- 새 버전 Proto3
 - required와 optional이 사라짐

Protocol Buffers

- 예약 필드 : 필드 번호나 필드 이름
 - delete시 재활용을 못하게 -> 이전 데이터의 잘못된 접근 방지
 - 추후 사용 예약.

```
message Foo {  
    reserved 2, 15, 9 to 11;  
    reserved "foo", "bar";  
}
```

Protocol Buffers

- map

- 정의

```
map<key_type, value_type> map_field = N;
```

- 사용예

```
map<string, Project> projects = 3;
```

- 제한

- repeated나 optional 또는 required로 지정할 수 없다.

- 구현 : 다음과 같다.

```
message MapFieldEntry {  
    optional key_type key = 1;  
    optional value_type value = 2;  
}  
repeated MapFieldEntry map_field = N
```

Protocol Buffers

• 기본 데이터 타입 (proto2)

.proto Type	Notes	C++ Type	Java Type	Python Type ^[2]	Go Type
double		double	double	float	*float64
float		float	float	float	*float32
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
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 ^[3]	*int64
uint32	Uses variable-length encoding.	uint32	int ^[1]	int/long ^[3]	*uint32
uint64	Uses variable-length encoding.	uint64	long ^[1]	int/long ^[3]	*uint64
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	*int32
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long ^[3]	*int64
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2 ²⁸ .	uint32	int ^[1]	int/long ^[3]	*uint32
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2 ⁵⁶ .	uint64	long ^[1]	int/long ^[3]	*uint64
sfixed32	Always four bytes.	int32	int	int	*int32
sfixed64	Always eight bytes.	int64	long	int/long ^[3]	*int64
bool		bool	boolean	bool	*bool
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	unicode (Python 2) or str (Python 3)	*string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	bytes	[]byte

Protocol Buffers

• 기본 데이터 타입 (proto3)

.proto	Notes	C++	Go	C#
double		double	float64	double
float		float	float32	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers - if your field is likely to have negative values, use sint32 instead.	int32	int32	int
int64	Uses variable-length encoding. Inefficient for encoding negative numbers - if your field is likely to have negative values, use sint64 instead.	int64	int64	long
uint32	Uses variable-length encoding.	uint32	uint32	uint
uint64	Uses variable-length encoding.	uint64	uint64	ulong
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int32	int
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	int64	long
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^{28} .	uint32	uint32	uint
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^{56} .	uint64	uint64	ulong
sfixed32	Always four bytes.	int32	int32	int
sfixed64	Always eight bytes.	int64	int64	long
bool		bool	bool	bool
string	A string must always contain UTF-8 encoded or 7-bit ASCII text, and cannot be longer than 2^{32} .	string	string	string
bytes	May contain any arbitrary sequence of bytes no longer than 2^{32} .	String	[]byte	ByteString

Protocol Buffers

- 업데이트시 호환성 유지
 - 필드 숫자 변경 금지
 - 다른 용도로 재사용도 금지
 - required 필드 유지 (삭제나 추가 금지)
 - int32, uint32, int64, uint64, bool은 자유롭게 변경 가능
 - sint32, sint64는 서로만 호환
 - optional과 repeated는 호환 그러나 마지막 값만 접근 가능

Protocol Buffers

- RPC : 자체 RPC 시스템을 내장하고 있어서 RPC에 사용되는 매개변수와 결과 값을 **protocol buffer** 포맷으로 정의할 수 있다.

```
service SearchService {  
    rpc Search (SearchRequest) returns (SearchResponse);  
}
```

- proto3부터는 구글의 gRPC 시스템과의 연동을 추천한다.

Protocol Buffers

- RPC 호출

```
using google::protobuf;

protobuf::RpcChannel* channel;
protobuf::RpcController* controller;
SearchService* service;
SearchRequest request;
SearchResponse response;

void DoSearch() {
    // You provide classes MyRpcChannel and MyRpcController, which implement
    // the abstract interfaces protobuf::RpcChannel and protobuf::RpcController.
    channel = new MyRpcChannel("somehost.example.com:1234");
    controller = new MyRpcController;

    // The protocol compiler generates the SearchService class based on the
    // definition given above.
    service = new SearchService::Stub(channel);

    // Set up the request.
    request.set_query("protocol buffers");

    // Execute the RPC.
    service->Search(controller, request, response, protobuf::NewCallback(&Done));
}

void Done() {
    delete service;
    delete channel;
    delete controller;
}
```

Protocol Buffers

- RPC 서비스

```
using google::protobuf;

class ExampleSearchService : public SearchService {
public:
    void Search(protobuf::RpcController* controller,
               const SearchRequest* request,
               SearchResponse* response,
               protobuf::Closure* done) {
        if (request->query() == "google") {
            response->add_result()->set_url("http://www.google.com");
        } else if (request->query() == "protocol buffers") {
            response->add_result()->set_url("http://protobuf.googlecode.com");
        }
        done->Run();
    }
};

int main() {
    // You provide class MyRpcServer. It does not have to implement any
    // particular interface; this is just an example.
    MyRpcServer server;

    protobuf::Service* service = new ExampleSearchService;
    server.ExportOnPort(1234, service);
    server.Run();

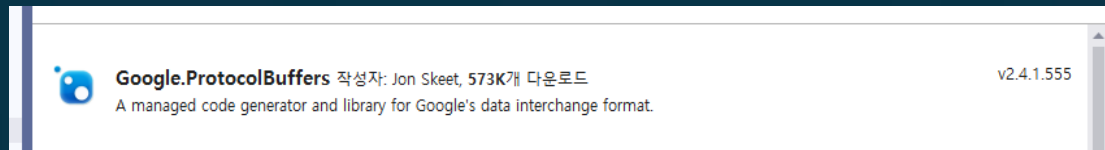
    delete service;
    return 0;
}
```

사용법

- .proto 파일을 protocol buffer compiler로 컴파일 하면 원하는 언어의 소스 파일이 나온다.
 - C++는 .proto.h와 .proto.cc파일을 생성한다.
 - 여기에 message에 1:1 매칭이 되는 class가 정의되어 있다.
 - 위에서 정의된 class에 데이터를 넣은 후 고유 메소드를 호출해서 serialize한다.

사용법

- Nuget Download




- 옛날 버전

- 홈페이지에서 소스 다운

- <https://github.com/protocolbuffers/protobuf/releases/tag/v3.11.4>

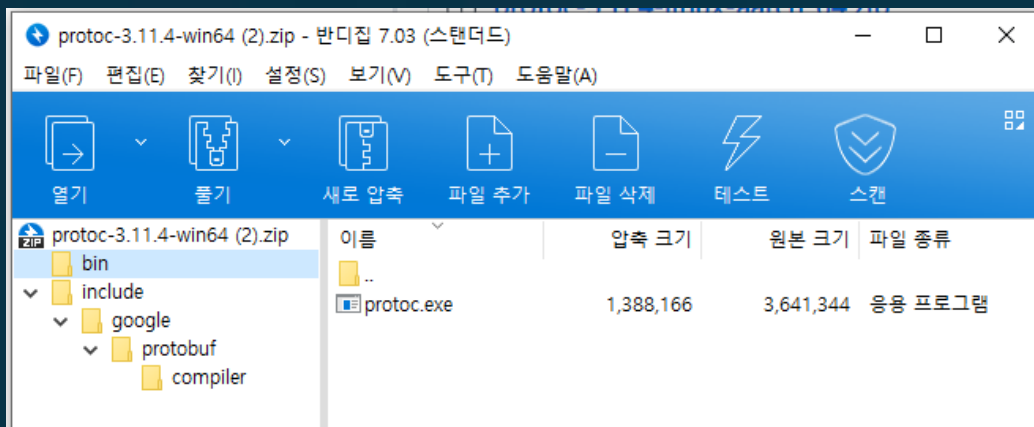
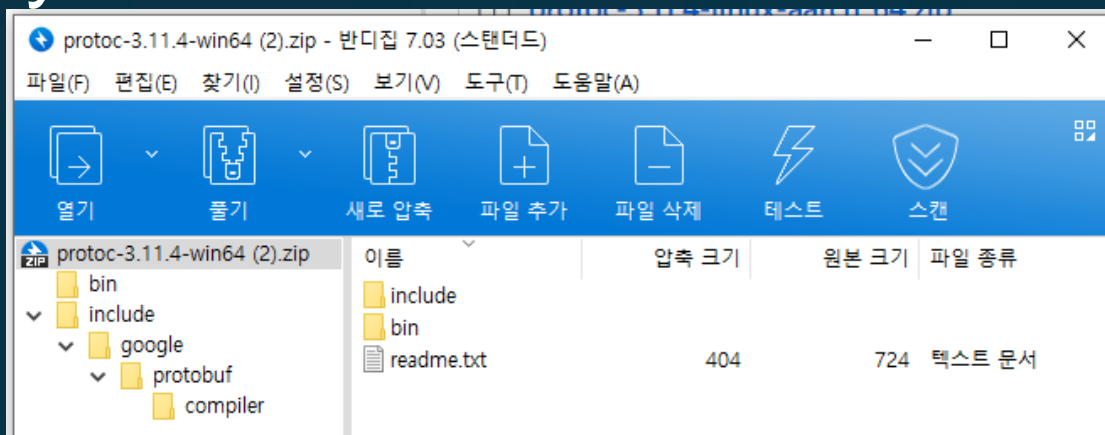
 protobuf-cpp-3.11.4.zip	5.35 MB
 protoc-3.11.4-win64.zip	1.36 MB

- 또는 Binary Down

 protoc-3.11.4-win64.zip	1.36 MB
---	---------

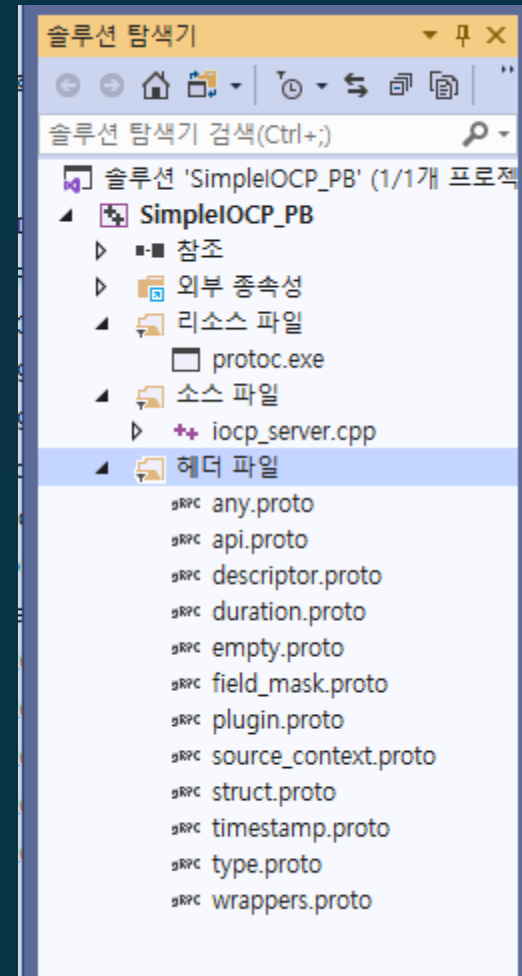
사용법

• Binary



사용법

- 설치
 - protoc.exe를 포함
 - include 파일의 내용 포함



사용법

- protocol.proto 작성

```
syntax = "proto3";
import "google/protobuf/any.proto";

enum Direction {
    UP = 0;
    DOWN = 1;
    LEFT = 2;
    RIGHT = 3;
}

message CS_MOVE {
    Direction dir = 1;
}

message CS_LOGIN {
    string user_name = 1;
}
```

```
message SC_LOGIN_OK {
    int32 id = 1;
    int32 x = 2;
    int32 y = 3;
    string user_name = 4;
}

message SC_POSITION {
    int32 id = 1;
    int32 x = 2;
    int32 y = 3;
}

message SC_ENTER {
    int32 id = 1;
    string user_name = 2;
    int32 x = 3;
    int32 y = 4;
}

message SC_LEAVE {
    int32 id = 1;
}

message packet {
    google.protobuf.Any packets = 1;
}
```

사용법

- protocol.proto 컴파일

```
D:\₩depot₩Graduate₩2020-AGS₩SimpleIOCP_PB>bin₩protoc.exe --cpp_out=PROTOCOL protocol.proto

D:\₩depot₩Graduate₩2020-AGS₩SimpleIOCP_PB>dir PROTOCOL
D 드라이브의 볼륨: 960PRO
볼륨 일련 번호: 0EFF-D7DE

D:\₩depot₩Graduate₩2020-AGS₩SimpleIOCP_PB₩PROTOCOL 디렉터리

2020-04-30 오후 02:05 <DIR>          .
2020-04-30 오후 02:05 <DIR>          ..
2020-04-30 오후 02:05                55,882 protocol.pb.cc
2020-04-30 오후 02:05                48,265 protocol.pb.h
                2개 파일                104,147 바이트
                2개 디렉터리 172,919,676,928 바이트 남음

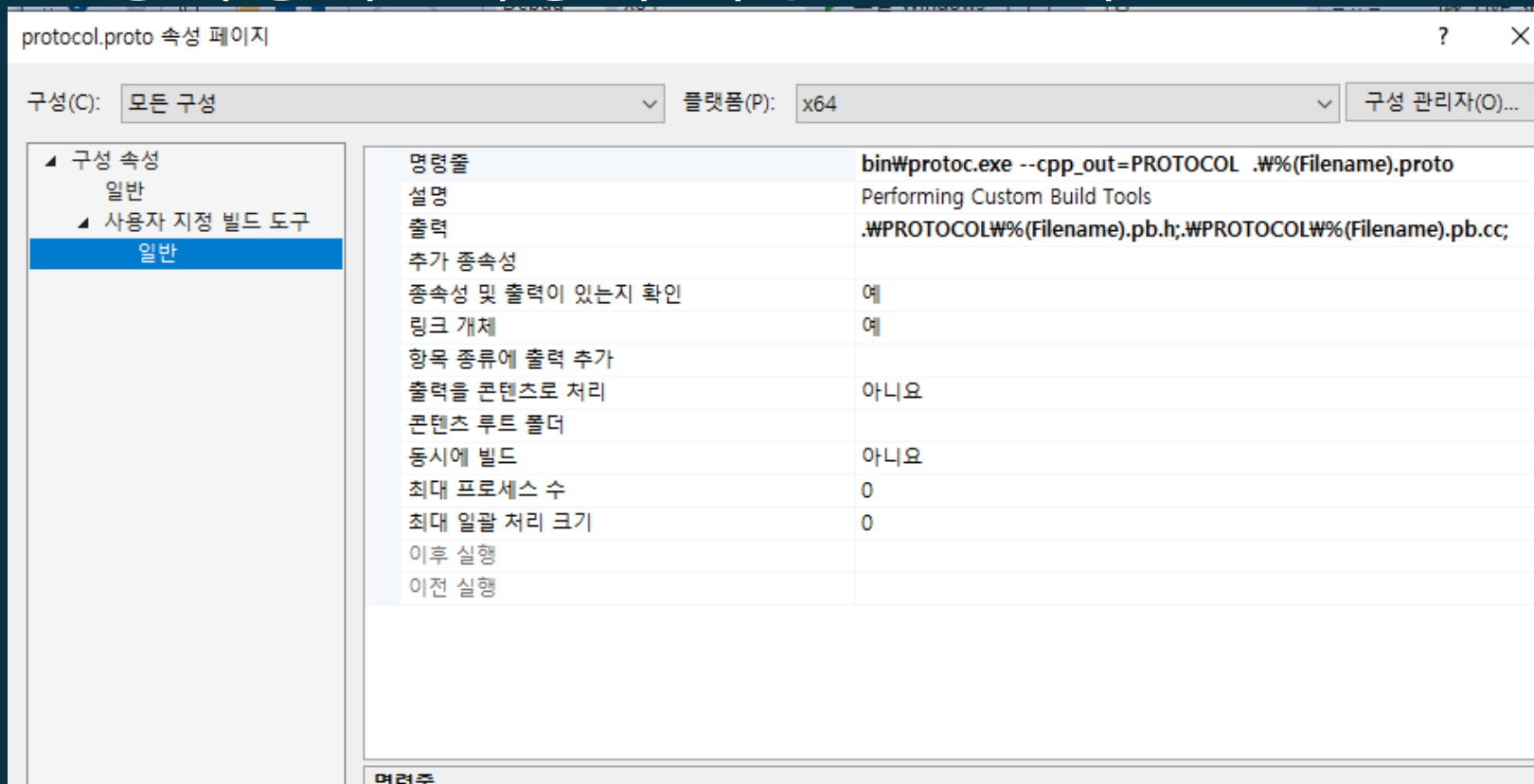
D:\₩depot₩Graduate₩2020-AGS₩SimpleIOCP_PB>
```

사용법

- 중간파일
 - protocol.pb.cc
 - protocol.pb.h
 - 프로젝트에 속해야 한다.
 - 소스컨트롤에는 속해 있지 않아야 한다.
 - protocol.proto가 수정되면 재 작성 되어야 한다.
 - protocol.proto의 속성 수정
 - 항목형식->사용자 지정 빌드 도구

사용법

- protocol.proto의 속성 수정
 - 항목형식->사용자 지정 빌드 도구



실습

- 기존의 IOCP 서버를 Protocol Buffer로 수정해 본다.

실습

- 라이브러리 설치
 - 소스를 다운 받고
 - Cmake를 설치한후
 - CMake 실행(x64 옵션을 줌)
 - 이후 생성된 .sln을 열고 컴파일
 - libprotobuf를 만들 때 코드생성에서 shared library를 선택하자.
 - 생성된 libprotobuf.lib를 프로젝트에 복사
 - #pragma comment(lib, "libprotobuf.lib")

실습

- 시리얼라이즈

```
void send_login_ok_packet(int id)
{
    SC_LOGIN_OK_PB packet;
    packet.set_id(id);
    packet.set_x(clients[id]->x);
    packet.set_y(clients[id]->y);

    char buf[512];
    packet.SerializeToArray(buf, sizeof(buf));
    send_packet(id, buf);
}
```


실습

- 디시리얼라이즈

```
void ProcessPacket(int id, void *buff)
{
    packet any;
    any.ParseFromArray(buff, MAX_BUFFER);
    if (any.packets().Is<CS_LOGIN_PB>()) {
        CS_LOGIN_PB p;
        any.packets().UnpackTo(&p);
        ProcessLogin(id, p.user_name().c_str());
    }
}
```

FlatBuffers

- <https://google.github.io/flatbuffers/>
- 정의
 - FlatBuffers is an **efficient** cross platform serialization library for C++, C#, C, Go, Java, Kotlin, JavaScript, Lobster, Lua, TypeScript, PHP, Python, Rust and Swift. It was originally created at Google **for game development** and other performance-critical applications.

FlatBuffers

- 용도
 - Parsing이나 unpacking이 없는 직렬화
 - 효율적인 메모리 사용
 - 용량 절약, 복사 비용 감소
 - 유연성
 - 과거 버전과의 호환성
 - 작은 코드 기반, 강한 typing, 사용 편의성
 - 크로스 플랫폼

FlatBuffers

- 용도
 - Parsing이나 unpacking이 없는 직렬화
 - 효율적인 메모리 사용
 - 용량 절약, 복사 비용 감소
 - 유연성
 - 과거 버전과의 호환성
 - 작은 코드 기반, 강한 typing, 사용 편의성
 - 크로스 플랫폼

FlatBuffers

- 플랫버퍼에서 데이터를 정의한다.
 - 고유 포맷(언어?)가 있다. .fbs 확장자를 가진다.

```
// example IDL file
namespace MyGame;
attribute "priority";
enum Color : byte { Red = 1, Green, Blue }
union Any { Monster, Weapon, Pickup }
struct Vec3 {
  x:float;
  y:float;
  z:float;
}
table Monster {
  pos:Vec3;
  mana:short = 150;
  hp:short = 100;
  name:string;
  friendly:bool = false (deprecated, priority: 1);
  inventory:[ubyte];
  color:Color = Blue;
  test:Any;
}

root_type Monster;
```

FlatBuffers

- Table

- 데이터 단위이다.
- Field를 가진다.
- 주의

```
table Monster {  
  pos:Vec3;  
  mana:short = 150;  
  hp:short = 100;  
  name:string;  
  friendly:bool = false (deprecated, priority: 1);  
  inventory:[ubyte];  
  color:Color = Blue;  
  test:Any;  
}
```

- Field를 추가할 때 반드시 끝에 추가.(호환성을 위해)
- Field삭제 금지. (호환성을 위해) 필요 없으면 사용 중지
- Table이름과 Field이름 변경은 자유

- Field

- 이름과 타입을 가진다.
- =를 통해 기본값을 설정할 수 있다.

FlatBuffers

- Type

- 기본

- 8비트 : byte, bool
 - 16비트 : short, ushort
 - 32비트 : int, uint, float
 - 64비트 : long, ulong, double

- Built-in

- Vector : [type]
 - string : UTF-8
 - Table, Struct, enum, union

```
table Monster {  
  pos:Vec3;  
  mana:short = 150;  
  hp:short = 100;  
  name:string;  
  friendly:bool = false (deprecated, priority: 1);  
  inventory:[ubyte];  
  color:Color = Blue;  
  test:Any;  
}
```

FlatBuffers

- Struct

- table과 유사
- 변경 불가
 - 적은 메모리 사용

- Enum

- 저장 타입 지정 필요
- 실제 값은 0부터 시작

- 기본값

- 실제 값이 저장되지 않는다. Code에서 처리
- Constant 처럼 사용

- Union

```
enum Color : byte { Red = 1, Green, Blue }
union Any { Monster, Weapon, Pickup }
struct Vec3 {
  x:float;
  y:float;
  z:float;
}
table Monster {
  pos:Vec3;
  mana:short = 150;
  hp:short = 100;
  name:string;
  friendly:bool = false (deprecated, priority: 1);
  inventory:[ubyte];
  color:Color = Blue;
  test:Any;
}
```


FlatBuffers

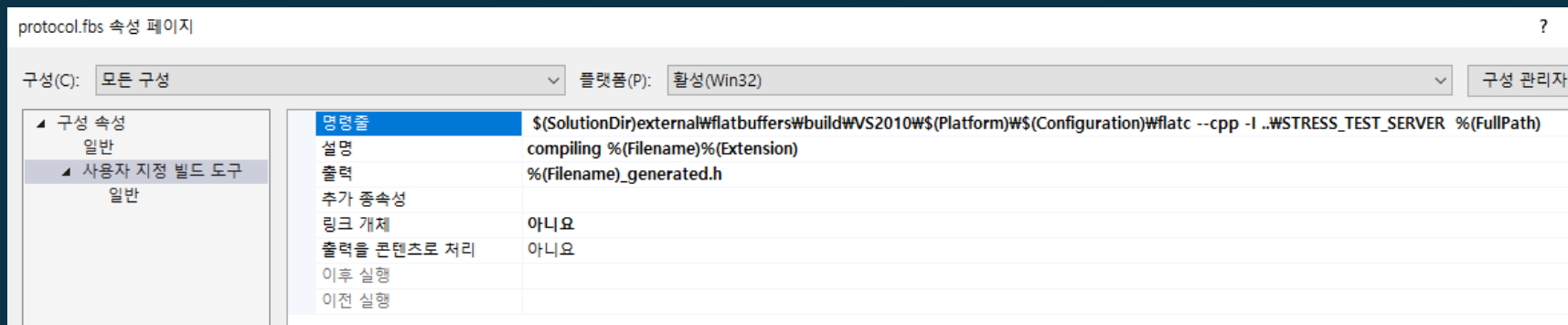
- Namespaces
 - C++과 같은 기능
- Includes
 - #include와 같음
- Attribute
 - 필드에 넣을 추가
정보 정의
- Root type
 - 변경의 기본

```
// example IDL file
namespace MyGame;
attribute "priority";
enum Color : byte { Red = 1, Green, Blue }
union Any { Monster, Weapon, Pickup }
struct Vec3 {
  x:float;
  y:float;
  z:float;
}
table Monster {
  pos:Vec3;
  mana:short = 150;
  hp:short = 100;
  name:string;
  friendly:bool = false (deprecated, priority: 1);
  inventory:[ubyte];
  color:Color = Blue;
  test:Any;
}

root_type Monster;
```

사용법

- xxx.fbs 파일을 flatc.exe로 컴파일
 - 결과 : xxx_generated.h
 - xxx.fbs 파일에 사용자 지정 빌드 추가



사용법

• protocol.fbs 작성

```
enum MsgId_ingame :ubyte {  
    MOVE_C2S = 100,  
    MOVE_S2C = 101,  
    GAME_END_S2C = 102,  
    JOIN_STAGE_C2S = 103  
}  
  
struct Vec3 {x:float;  y:float;  z:float;}  
  
table Move_C2S {  
    position : Vec3;  
    direction : Vec3;  
    anim_stat : short;  
}  
  
table Move_S2C {  
    object_id :int;  
    position : Vec3;  
    direction : Vec3;  
    anim_stat : short;  
}
```

```
table GameEnd_S2C {  
}  
  
table JoinStage_C2S {  
    msgid : MsgId_ingame = JOIN_STAGE_C2S;  
    stageid : string;  
}
```

사용법

- Unserialize

```
auto move_C2S = flatbuffers::GetRoot<Move_C2S>(payload);  
auto position = move_C2S->position();  
auto direction = move_C2S->direction();
```

```
struct Vec3 {x:float; y:float; z:float;}  
  
table Move_C2S {  
  position : Vec3;  
  direction : Vec3;  
  anim_stat : short;  
}
```

사용법

• Serialize

```
flatbuffers::FlatBufferBuilder fbb;

auto new_pos = Vec3(position->x(), position->y(), position->z());
auto new_direction = Vec3(direction->x(), direction->y(), direction->z());

Move_S2CBuilder packet_move_S2C(fbb);
packet_move_S2C.add_position(&new_pos);
packet_move_S2C.add_direction(direction);
packet_move_S2C.add_anim_stat(move_C2S->anim_stat());
auto message = packet_move_S2C.Finish();
fbb.Finish(message);

char header[2];
header[0] = 2 + fbb.GetSize();
header[1] = MsgId_ingame_MOVE_C2S;
send(client_socket, header, 2);
send(client_socket, fbb.GetBufferPointer(), fbb.GetSize());
```