[ Google Protocol Buffer ]

● 이종간 데이터 통신

- 이종간 데이터 통신을 할 경우 서로 다른 환경에 기안한 여러 문제점이 있다.

[ 문제가 될 수 있는 이슈 ]

(1) 특히 서로 다른 OS 및 개발 언어를 사용하는 경우 다양한 문제점을 만나게 된다.

-> OS : Windows / Windows CE / IOS / MAC OS / Android BlackBerry OS / Symbian / RTOS 등

-> OS 종류가 다르면 개발 언어가 달라지고 ( C / C++ / MFC / Java / Android / Objective C )

개발 언어가 다르면 Primitive Variable을 구성하는 Endian / Little Endian이 달라지게 된다.

\*Endian이 다르다는 것

: 데이터를 해석하는 방법이 달라지므로 정확한 값을 주고 받지 못한다. 별도의 Endian을 맞추는작업

을 해야 함.

(2) 개발언어가 달라질 경우 또 다른 문제점,

-> 언어마다 구성하는 객체가 달라 Remote Object등과 같은 객체 지향 통신은 불가능해 지고,

Low Data를 이용해 통신 해야 한다.

- 해결 ?

-> XML 데이터를 이용한 문자열 방식의 통신을 사용할 수도 있는데 호환성은 뛰어나지만 성능적인 측면

에서는 단점이 있다.

-> 구글 프로토콜 버퍼의 경우 다른 지원 방식들과 비교했을 때 성능이 좋은 편이다.

(3) 개발 환경이 다르거나 문자를 처리하는 방식(ASCII, UTF-8, UTF-16, UNICODE)이 서로 상이하여, 언

어 이외의 다국어를 지원하는 경우 문자열 인코딩 방식을 통일하는데 에도 어려움이 있다.

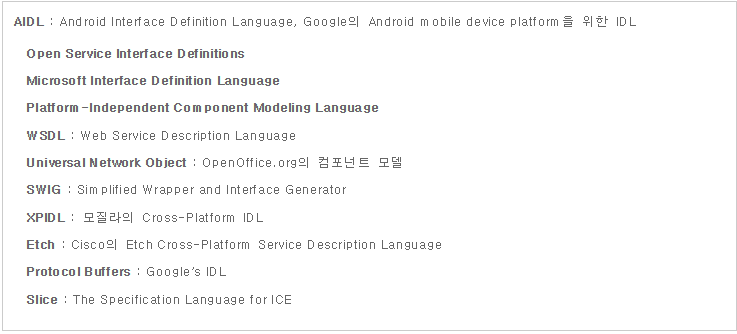
(4) 특히, 데이터 프로토콜에 대한 커뮤니케이션 이슈 문제가 크다.

-> 통신을 주고 받는 호스트들이 데이터의 각 필드의 속성이나 크기 등을 서로 다르게 이해해 혼란을

겪게 되는 경우,

-> 데이터 선언 / 데이터 범위 / 초기화 등의 문제

- 이러한 문제점들을 해결 하기 위한 데이터 구성 방법 -> 인터페이스 정의 언어





- 먼저 Google Protocol Buffer에 대해서 자세히 알아보기 전에,

이러한 환경적인 문제점 즉, 복잡성을 해결하려면 어떠한 데이터 전달 방식을 써야 하나 ?

-> XML 포맷을 이용한 텍스트 형태의 데이터 전달 방식 / ASN.1 or Binary 데이터 방식

(1) XML 포맷

-> 데이터 내용을 XML TAG를 이용해 텍스트 형태로 생성해 해석하여, 데이터를 직접 이해하기도 쉽고,

보편화 되어 있어 각 플랫폼에서 제공하는 XML Parser를 이용하면 데이터 핸들링 하기도 편리하다.

[단점]

-> 데이터 사이즈가 커지거나, 데이터가 큰 경우 Parser의 성능이 저하되는 단점.

-> 또한 데이터 보안에도 취약해 별도 보안책을 마련해야 함.

(2) Binary 데이터 방식

-> 데이터 사이즈가 작고 송수신 처리시 성능을 보장.

-> Binary 데이터 이기 때문에, 데이터만 봐서는 무슨 의미인지 알 수 없어 보안도 향상

-> Google Protocol Buffer는 Google이 Open Source 형태로 제작해, 소스를 포함하여 배포하고 있어,

안정적이고 출처가 한곳이다. 원하는 사람은 소스를 받아 원하는 방식으로 수정도 가능.

[단점]

-> XML에 비해 보편화가 안되있어, 모든 플랫폼에서 사용할 수는 없고,

Serialize / Deserialize 모듈 제공자에 큰 의존성을 갖는다.

● Google Protocol Buffer

(1) 정의

“ Protocol Buffers are a flexible, efficient, automated mechanism for

serialize structured Data ”

“ Protocol Buffers allow you to define simple data structures in a special definition language,

Then compile them to produce classes to represent those structures

in the language of your choice ”

- Google blog의 Protocol Buffer에 대한 정의

- 위에서 처럼, 간단한 구조화 데이터를 원하는 개발 언어에 맞게 효과적으로 직렬화 하는 것이 목적,

구조화된 간단한 데이터를 빈번히 주고받기 위한 성능 위주로 쉽게 구현된 구현체.

(2) 수행 단계

1. Proto 파일 작성

2. Proto Compiler를 통해 각 언어에 맞는 Data Handler class 생성

3. 생성된 class의 set 함수를 통해 데이터 설정

4. writeTo() 함수를 통해 Binary Data로 Serialize

5. serial data 전송

6. serial data 수신

7. parseFrom() 함수를 통해 binary Data를 parsing

8. get 함수를 통해 원하는 데이터 값 조회

(3) 장 / 단점

[장점]

1. 사용이 간단하다.

2. 20 ~ 100배 빠르다

3. 모호하지 않다.

4. 프로그램에서 사용 가능한 클래스를 생성해준다. (C++ / Java / Python 공식 지원)

5. 최대 64MB 크기의 Message까지 지원한다. (성능 보장을 위한 제약)

6. JSON 포맷 전환을 지원한다.

7. BSD License 정책으로 100% 무료이다.

[단점]

1. Repeat 사용시 serialize / parse 시 성능이 저하될 수 있음.

2. 데이터를 눈으로 확인할 수 없다.

3. 표준 프로토콜이 아니다.

4. 단말기 / 세톱박스 등 RTOS에서의 지원이 어렵다.

5. 공식 지원 언어는 c++ / java / python이다. (이외 언어는 3rd party에서 지원)

6. Map / Set을 지원하지 않는다.

(4) Proto 파일 작성

- 이 파일은 구조체와 같이 해당되는 데이터 변수 및 타입을 명시한다. 각 변수는 한정자 조건을 갖는다.

- required : 1개만 존재 가능 ( 무조건 존재 하여야 하는 변수 )

- optional : 0개 혹은 1개 존재 가능 ( 존재하지 않을 수도 있고 존재 할 수도 있는 변수 )

- repeated : 여러 개 존재 가능(0개 포함)

- 데이터 타입



\* Protocol Buffer를 패킷으로 활용할 때 주의 할점

- 패킷으로서 활용되면, 패킷이 뭉쳐서 올 수 있어, 메시지의 길이를 알아낼수 있다거나 끝을 표시해

주는 기능이 필요하다.

syntax = "proto3";

message SearchRequest {

string query = 1;

int32 page\_number = 2;

int32 result\_per\_page = 3;

}

\*syntax를 명시적으로 기입 하지않으면, 컴파일러는 proto2 버전으로 인식한다.

Specifying Field Types

- 위의 예제에서 모든 필드들은 스칼라 타입 : 두개의 integers / 하나의 string 타입이다.

- 하지만, 사용할 필드들을 위한 Composite 타입을 작성할수도있다, enumerations 나 다른 message

타입들 같은 것

Assigning Tags

- 메시지 안에 각 필드들을 정의 할 때 고유한 숫자가 매겨진 태그를 갖는다. 이러한 태그들은 메시지

바이너리 포맷 안에서 각각의 필드를 식별할 때 사용 되고, 지정된 타입의 순서가 변하지 않아야 할 때

쓰인다.

(1~2^29 -1) 범위로 태그 값을 지정할 수 있고, 그리고 19000 ~ 19999는 사용 할 수 없다.

사용불가( FieldDescriptor::kFirstReservedNumber ~ FieldDescriptor::kLastReservedNumber)

프로토콜 버퍼 컴파일러는 이 reserved된 숫자중 하나라도 .proto 파일안에 사용 하였다면, Complain

할 것이다. 또한 이전에 reserved 했던 태그들을 사용 할 수 없다.

Specifying Field Rules

-singular : 0 또는 1개만 존재 가능하다.

-repeated : 0을 포함해 어떤 숫자라도 될 수 있다. 반복된 값들의 순서가 유지 될 것이다.

Repeated 필드들은 이것들이 많이 사용되면 효율적으로 인코딩 될 수 없다.   
 효율적인 인코딩을 위해선 [packed=true]란 새로운 코드를 사용 하여야 한다.

repeated int32 samples = 4 [packed=true];

.