3. 서버간 통신 서버간 통신 기법에 대해서 알아봅시다

1) json serialize/deserialize가 무엇인지에 대해서 간략하게만 소개해봅시다

2) json serialize/deserialize의 장단점에 대해서 소개해봅시다

3) json serialize/deserialize 말고도 다른 부호화 방식이 존재한다면 소개해주세요

4) gRPC에 대해 소개해봅시다. (얕은 내용 + 사용 방법까지만! 깊은 내용은 나중에 자세하게 다시 다룰겁니다)

5) fetch/axios를 이용한 통신에 비해서 gRPC를 이용하는 서버간 통신이 가지는 장점 //

6) gRPC를 이용한 서버간 통신 구현 (Hello를 주고받는 정도의 서버만 구현하면됩니다)

json serialize

메모리를 디스크에 저장하거나 네트워크 통신에 사용하기 위한 형식으로 변환하는 것을 말한다.

json deserialize

그 반대로 디스크에 저장한 데이터를 읽거나, 네트워크 통신으로 받은 데이터를 메모리에 쓸 수 있도록 다시 변환하는 것이다.

JSON 내장 객체

: 자바스크립트는 JSON데이터 처리를 위한 내장 함수를 제공한다.

1) JSON.stringfy(JSON형식의 객체)

: 객체 => 문자열로 변환하며, 이를 직렬화(serialize)라고 한다.

: 통신할 때는 문자열로 직렬화하여 주고 받는 것이 안전하다.

2) JSON.parse(JSON형식의 문자열)

: 문자열 => 객체로 변환하며, 이를 역직렬화(deserialize)라고 한다.

: 통신으로 받은 문자열은 객체로 역직렬화하여 사용하는 것이 편리하다.

JSON 장점

1. **Readable** 텍스트 형태이기 때문에 쉽게 읽을 수 있으며, 때문에 언제든 텍스트 뷰어로 객체를 열어서 검사할 수 있다.
2. **Self-Contained** JSON은 그 자체로 모든 것을 포함하고 있기 때문에 무언가를 동기화하지 않고, json을 보내고 받아들이는 것 만으로 손쉽게 사용 가능하다.
3. **Extensible** JSON객체는 클라이언트에게 특별한 변화 없이 저장하는 데이터를 쉽게 확장할 수 있다.

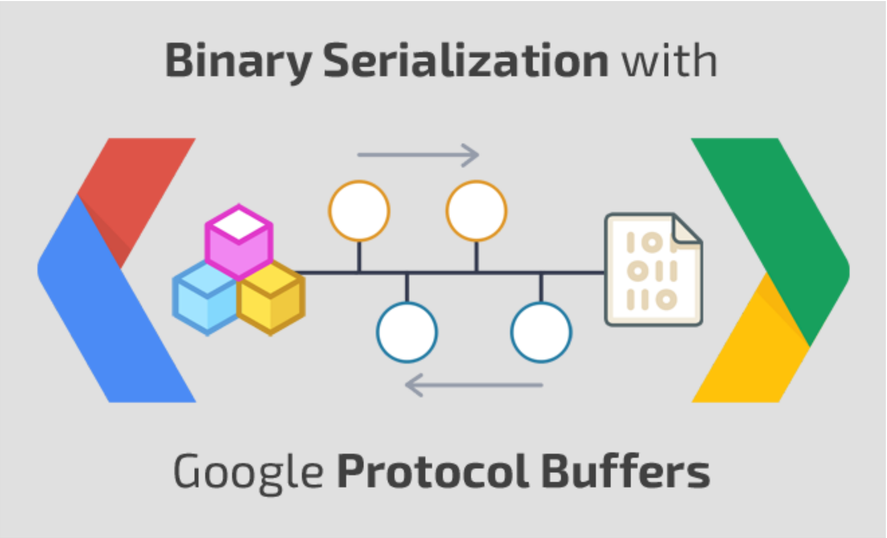
JSON 단점

1. **Expensive**: 대량의 데이터 직렬화시 많은 비용이 발생한다. 또한 역직렬화 객체 역시 상당한 오버헤드를 무시할 수 없다.
2. **Has unclear types**: 따라서 부호 없는 32비트 정수를 저장할 수 없다.
3. **Massive**: 데이터의 양이 JSON은 방대하여 많은 오버 헤드가 발생한다.

*오버헤드: 프로그램의 실행흐름에서 나타나는 현상 중 하나로, 프로그램의 실행흐름 도중에 동떨어진 위치의 코드를 실행시켜야할 때 추가적으로 시간, 메모리, 자원이 사용되는 현상. (어떤 처리를 하기 위해 들어가는 간접적인 처리 시간)

=> **이진 직렬화**를 하는 방법
=> 대표적인 것으로 구글에서 제안한 데이터 직렬화 플랫폼인 **Protocol Buffer**가 있다.

Binary serialization



Protocol buffer는 모든 데이터와 함께 **스키마를 미리 선언**하여 공간과 비용을 아낄 수 있다. 비워져 있는 스키마를 가지고 있는 프로토파일은 실제로 우리가 필요로하는 데이터를 그 안에 채워넣어 protobuf에서 코드를 할당한다. protobuf는 바이너리 형태이기 때문에 휴대하기 간편하고 효율적으로 메세지 교환을 할 수 있다.

프로토콜 버퍼 특징

- 프로토콜 버퍼는 왜 쓰는가?

1. 통신이 빠르기 때문이다.
같은 데이터를 보내더라도 데이터의 크기가 작으니까 같은 시간에 더 많은 데이터를 보낼 수 있다.
물론 더 빠르게 보낼 수도 있다.

2. 파싱을 할 필요가 없다.
JSON포맷으로 온 데이터를 다시 객체로 파싱해서 객체로 사용하는게 보통인데, 프로토콜 버퍼를 쓰면 바이트가 오면 그 바이트 그대로 메모리에 써버리고 객체 레퍼런스가 가리켜 버리면 끝난다. 별도의 파싱이 필요가 없는 것이다. = > 빠름

- 그러면 단점은 없는가?

1. 인간이 읽기 불편하다.
JSON같은 경우는 데이터를 보면 사람이 읽기 정말 편하다. 반대로 프로토콜 버퍼가 쓴 데이터는 사람이 읽기 어렵다. proto파일이 없으면 아예 무슨 의미인지 모른다.
이것은 즉, proto파일이 반드시 있어야 한다는 것이다.

이 문제 때문에 외부 API로 쓰이기에는 문제가 있다. (모든 클라이언트가 proto파일이 있어야하기 때문)
그래서 내부 서비스간의 데이터 교환에서 주로 쓰인다.

2. proto 문법을 배워야 한다.
프로토콜 버퍼에서 쓰이는 스키마 파일 즉, .proto 파일을 작성할 줄 알아야한다.
근데 그 문법이 다른 프로그래밍 문법과 유사하기 때문에 크게 문제되진 않지만, 또 정확하게 작성하려면 문법을 알아야한다. 문서를 보고.
그리고 proto파일을 한 번 정의했다가 필요에 의해서 변경되면 이 proto파일을 쓰는 애플리케이션은 다시 공유되어야 하고 컴파일되어야 한다는 단점이 있다.

뭐 구글 문서에는 proto파일에 속성이 업데이트되는건 무방하다고 나오는데 속성이 삭제되거나 타입이 변경되고 이러면 당연히 proto파일을 사용하는 애플리케이션 모두가 바꿔야 한다는 문제가 있다.

gRPC

RPC (원격 프로시저 호출)는 미리 정의된 형식을 사용하여 원격 서버에서 서비스를 실행하고 동일한 형식으로 응답을 받을 수 있도록 하는 웹 아키텍처입니다.

gRPC is a robust open-source RPC (Remote Procedure Call) framework used **to build scalable and fast APIs**

gRPC는 Google에서 설계, 개발한 Google Remote Procedure Call이다. 데이터를 가볍고 빠르게 요청하기 위해 고려되었다.
gPRC는 일반적인 원격 프로시저 호출보다 진화된 형태이다.
gRPC를 사용하면 클라이언트는 원격 시스템이 마치 로컬인 것처럼 직접 메소드를 호출할 수 있다.
gRPC는 **프로토콜 버퍼(Protobufs)**를 사용하여 데이터 전송을 위해 텍스트 기반의 JSON, XML이 아닌 **바이너리로 직렬화된 데이터로 인터페이스를 정의한다.**
또한 HTTP/1보다 더 안정적이고 **빠른 HTTP/2를 사용하게 된다**.

- 가벼우며 클라이언트측의 리소스가 거의 필요하지 않다. 따라서 전력이 매우 부족한 상황에서 편리하게 이용할 수 있다.
- 효율적이다. gRPC는 통신을 직렬화하는데 중점을 둔 구조화된 데이터 직렬화 방법인 protobufs(이진형식)를 사용한다. => 이진형식으로 인코딩되어 효율적
- 오픈소스 이며 자유롭게 사용, 수정 또는 분기할 수 있다.

https://appmaster.io/ko/blog/grpc-dae-hyusig

개괄적인 비교

다음 표에서는 gRPC와 JSON을 사용하는 HTTP API 간의 고급 기능 비교를 제공합니다.

기능	gRPC	JSON을 사용하는 HTTP API
계약	필수(`.proto`)	선택 사항(OpenAPI)
프로토콜	HTTP/2	HTTP
Payload	Protobuf(소형, 이진)	JSON(대형, 사람이 읽을 수 있음)
규범	엄격한 사양	느슨함. 모든 HTTP가 유효합니다.
스트리밍	클라이언트, 서버, 양방향	클라이언트, 서버
브라우저 지원	아니요(gRPC-웹 필요)	예
보안	전송(TLS)	전송(TLS)
클라이언트 코드 생성	예	OpenAPI + 타사 도구

JSON과 Protocol Buffer 중 어떤 것을 사용하는 것이 좋을까?

언제나 그렇듯 정해진 정답은 없다. 목적에 따라 적절히 선택하는 것이 가장 올바른 방법일 것이다.

데이터 크기가 작고 검사가 필요하고, 확장성이 필요하거나 다양한 형태의 데이터가 필요하다면 JSON이 더 좋다.
small volume/ need to inspect/ messages are varied.

데이터의 크기가 크고 비슷한 유형의 데이터가 많으며, 성능이 정말 중요한 경우라면 protobuf가 더 나은 선택이라고 볼 수 있다.
high volume. similar data. performance matters.

대체로 많은 서비스에서 클라이언트로 전송해야 하는 데이터의 경우 JSON을 사용하고,
service와 service가 통신하는 경우라면 protobuf를 사용하는 경우가 많다.
진행하는 프로젝트의 속성 고려해야 한다.

https://techblog.woowahan.com/2550/
=> JSON, 프로토콜 버퍼 등은 시스템의 고유 특성과 상관없는 대부분의 시스템에서의 데이터 교환 시 많이 사용 됨
=> 자바직렬화 형태의 데이터 교환은 자바 시스템 간의 데이터 교환을 위해 존재함. (자바 시스템 개발에 최적화됨)

gRPC Introduction

https://www.youtube.com/watch?v=XRXTsQwyZSU

https://learn.microsoft.com/ko-kr/aspnet/core/grpc/comparison?view=aspnetcore-7.0

https://giljae.com/2022/08/05/Restful-vs-gRPC-vs-GraphQL.html