4. 부호화와 발전

application code 가 바로 적용되지 않는 이유

- code 의 update 방식은 rolling update(stage rollout) 으로 갱신된다.
- client 의 경우 사용자에게 달려있다. 종종 업데이트를 하지 않는 유저도 있기 때문

- 호환성
  - 하위 호환성
    새로운 코드는 예전 코드가 기록한 데이터를 읽을 수 있어야 한다.
    새로운 코드는 기존 데이터에 대해 알기 때문에 큰 문제가 없을 수 있다.
  - 상위 호환성
    예전 코드는 새로운 코드가 기록한 데이터를 읽을 수 있어야 한다.
    새 버전의 코드에 의해 추가된 것을 무시할 수 있어야 하므로 더 어렵다.

데이터 부호화 형식 - Formats for encoding Data

1. 프로그램은 보통 최소 두가지 형태로 표현된 데이터를 사용해 동작한다.
   메모리 객체(object), 구조체(struct), list, array, hash table, tree
   이러한 데이터 구조는 CPU 에서 효율적으로 접근하고 조작할 수 있도록 (포인터를 이용해) 최적화 된다.
2. 2.파일에 쓰거나 네트워크로 전송할때 json 과 같은 타입으로 encoding (부호화) 해야 한다.
   pointer 는 다른 프로세스가 이해할 수 없다.
   바이트열은 메모리에서 사용하는 데이터 구조와 상당히 다르다
   서로 다른 두 타입을 메모리에서 byte로 변경하는 것을 부호화(encoding, serialization or marshalling)이라 하고, 반대를 decoding(parsing, deserialization, unmarshalling) 이라 한다.

언어별 형식 Language-Specific Formats

언어에서 인코딩을 위한 기능이 제공된다. java.io.Serializable
쉽게 인메모리 객체를 저장하고 복원하지만, 심각한 문제점이 많다.

- 다른 언어에서 표현된 데이터를 읽는것은 시스템 통합에 방해가 되기 때문에 각 시스템으로 전송할때 부호화를 수행한다.

- 데이터를 복원하기 위해 복호화 프로세스는 임의의 class를 instance화 할수 있어야 한다. 그런데 이는 보안 문제에 연관되기도 하는데 임의의 인스턴스를 생성하여 원격으로 특정 코드를 실행시킬 수 있기 때문이다.

Json과 XML, 이진 변형 (Json, XML, and Binary Variants)

JSON과 XML을 경쟁자이며 둘다 text 형식으로 압도적으로 인기가 좋다.

xml은 장황하고 복잡하다 비판받고, JSON은 그에 비해 단순하다 또한, 강력하지 않지만 CSV도 있다. 다음은 피상적 문법 외에 차이점과 문제점을 나열한다.

- 수와 숫자에 대한 모호함이 있다.("15", 15를 구분하지 못하는 문제)

◦부동 소수점을을 구별하지 않고 정밀도를 지정하지 않는다.

- 2 53 이 넘어가면 부동 소수점 문제가 발생한다. (twitter는 이 문제를 해결하기 위해 64bit 숫자를 사용한다)

◦XML 도 스키마가 있지만 구현이 난해하며 일반적이지 않다

- XML, JSON은 unicode를 잘 지원한다. 그러나 binary variants 를 지원하지 않는다.

◦binary variant 는 매우 유용하기 때문에 base64로 encoding 하여 전송하여 이 문제를 해결한다.

◦하지만 정공법이 아니고 데이터 크기가 33% 증가한다.

- CSV는 스키마가 없기 때문에 좋지만, 그렇기 때문에 매우 모호하고 이 모호함을 application 에서 해결해야 한다.

이진 부호화 Binary Encoding

조직내에서만 사용하는 데이터라면 최소공통분보 부호화 형식(lowest-common-denominator encoding format)을 사용해야 한다는 부담감이 덜하다. json, xml과 비교해 더 적은 공간, 더 간결하고 더 빠른 파싱인데 data set이 적다면 별 의미가 없지만, terabyte 급이 된다면 이야기가 달라진다.

Json도 이진 형식과 비교하면 더 많은 공간을 사용한다. 이러한 관점이 json(message pack, BSON, BJSON, BISON, smile) 등으로 사용 가능한 이진 부호화 개발이 되었다.(하지만 json 만큼 사용되진 않음)

서비스를 통한 데이터 플로: REST와 RPC Dataflow through Services: REST and RPC

네트워크상 서버를 배치하는 방식

- client와 server로 네트워크를 하고 서버는 network에 API를 공개하고 client는 요청을 만들어 서버에 접근할 수 있다. 이때 API를 **서비스** 라고 한다
- 웹은 HTML, CSS, JS, image 등을 제공하며 GET 요청을 통해 데이터를 받고 POST 요청을 통해 데이터를 전송한다
- 웹에서 웹 브라우저만 유일한 client는 아니다.

◦XMLHttpRequest를 사용해 HTTP를 보내면 XMLHttpRequest가 client일 수 있다.

◦모바일 앱, 데스크탑 앱 등등

- 서버 자체가 다른 서비스의 client 일 수 있다.

◦이러한 방식을 SOA(Service-oriented architecture) 이것이 개선되어 MSA(Microservices architecture)라 불린다.

◦MSA와 같이 대용량 application 의 기능 영역을 소규모 서비스로 나누는 데 사용한다.

서비스와 Database는 여러가지 측면에서 유사하다. 단, 차이는 service 는 비즈니스 로직에 기반하여 입출력을 제한하고, 정해진 입출력만 허용해 API를 공개한다. → 캡슐화

MSA, SOA의 목표는 서비스를 배포와 변경에 독립적으로 만들어 application의 유지보수를 더 쉽게 만드는데 있다. 즉 변경이 잦을것을 대응하기 위한 것이며 새로운 버젼 출시가 빠르기 때문에 API간 호환이 필요하다.

이 장의 핵심 내용이다.

웹 서비스

기본 protocol을 HTTP를 사용하며 이때 웹서비스라 한다. → browser만 의미

- 다른 상황에서도 사용되는 예시

1.HTTP요청을 보내는 client application (mobile앱, ajax) 에서 요청

2.MSA, SOA 아키텍쳐 일부로 같은 조직의 다른 서비스에 요청하는 서비스

3.백엔드간 시스템에서 데이터를 교환하는 일부 (신용카드 처리, 사용자 데이터를 제공하는 OAuth)

이런 웹 서비스에서 대중적인 방법은 REST와 SOAP가 있다.

REST

- HTTP 토대 원칙
- 간단한 data type 을 강조한다
- URL resource를 식별한다
- 캐시 제어, 인증, 콘텐츠 유형 협상을 다룬다

- SOAP이 비해 인기있다.

SOAP

- API 요청을 위한 XML 기반 프로토콜이다.
- HTTP 상에서 일반적으로 사용하지만, HTTP와 독립적이며 HTTP 기능을 사용하지 않는다
  ◦ 그 대신 WS-*라 하여 Web Service Framework 를 제공한다.
- 웹 서비스 기술언어 (Web Service Description Language) 또는 WSDL 이라 하는 XML 기반 언어를 사용해 기술한다.
- 사람이 읽을 수 없도록 설계되어 도구나 IDE에 크게 의존한다

  기업은 RESTful API 를 통한 간단한 접근 방식을 선호한다

원격 프로시저 호출(RPC) 문제 The problems with remote procedure calls(RPCs)

- 웹 서비스는 network 상에서 API를 호출하는 여러 기술중 가장 최신의 형상일 뿐이다.

  이러한 웹 서비스는 원격 프로시저 호출(Remote procedure call, RPC)의 아이디어를 기반으로 한다.

  RPC 모델은 원격 network 서비스 요청을 같은 process 안에서 특정 method를 호출하는것 처럼 사용 가능하게 해준다. (이런 추상화를 location transparency 라 한다)

  이 개념이 매우 편리해 보이지만, 로컬 함수 호출과 다르기 때문에 RPC 방식은 근본적인 결함이 있다.
- 로컬 함수의 경우 예외를 내거나, 값을 반환하지 않을 수 있다.

  ◦ 네트워크는 timeout으로 결과가 없는것 처럼 만들 수 있지만,

  무슨일이 일어났는지 알 방법이 없다.

  ◦ 정말 요청을 제대로 보낸건지 아닌지를 구분하기도 어려워진다
- 실패한 네트워크 요청이 처리를 실행되지만, 응답만 유실된 경우일 수도 있다.

  ◦이때 멱등성 idempotence을 적용하지 않는다면 재시도는 여러 작업이 중복 실행될 수 있다.
- 로컬 함수 호출에 비해 훨씬 시간이 많이 소요되고 그 소요시간을 예측할 수 없다.
- 로컬함수는 pointer를 효율적으로 전달할 수 있다. 네트워크 요청은 부호화 하여 매개변수로 보내야 한다.

  ◦만약 큰 객체를 보내야 하는 상황

- client와 서비스는 다른 언어로 구현할 수 있다. 따라서 RPC 프레임워크는 하나의 언어에서 다른 언어로 데이터 타입을 변환해야 한다.

  이러한 한계와 문제가 있더라도 네트워크 통신을 로컬 함수처럼 사용하려는 수고는 비효율적인 것이 아니다.

RPC의 현재 방향 Current directions for RPC

이러한 문제에도 불구하고 RPC는 사라지지 않았으며, 지금까지 언급한 이진 부호화 위에 RPC 프레임워크가 개발되었다.

- thrift, avro 는 RPC 지원 기능을 내장하고 있으며
- gRPC는 protocol buffer를 이용해 RPC를 구현했다.
- Finagle은 thrift를 사용하고 Rest.li는 HTTP 위에서 json을 사용한다.
- 차세대 framework는 로컬 함수 호출과 네트워크 요청이 다르다는 사실을 분명히 한다.
  - Rest.li는 비동기 작업을 캡슐화 하기 위해 future promise를 사용한다
- 요청 라우팅에서 다루겠지만, service discovery 기능을 제공한다.
  - client가 server를 찾을 수 있는 IP주소, port 번호 제공 기능
- JSON 보다 이진 부호화 형식이 우수한 성능을 제공할수도 있다.

REST

- 웹 브라우저, 커맨드 라인 curl을 사용해 디버깅에 적합하다
- 다양한 도구(서버, 캐시 로드 밸런서, proxy, 방화벽, 모니터링, 디버깅 도구, 테스트 도구)가 있다.

데이터 부호화와 RPC의 발전 Data encoding and evolution for RPC

발전이 의미하는 것은 스키마가 변경되는 것을 의미한다.

발전성을 원한다면, RPC client와 서버를 독립적으로 변경하고 배포할 수 있어야 한다.

"모든 서버를 먼저 갱신하고 client를 갱신해도 문제가 없다"고 가정한다

- RPC의 상하위 호환 속성은 사용된 모든 부호화로부터 상속된다.(모든 부호화의 상하위 호환기능)
  - thrift, gRPC, avro RPC는 각 부호화 형식의에 호환성 규칙이 있고, 이에 따라 발전이 가능하다
  - SOAP 에서 요청과 응답은 XML 스키마로 지정된다. 발전은 가능하지만, 문제가 있다.

    - RESTful API는 응답에 JSON을 사용한다(공식적인 스키마는 없음)
- API 버젼 관리가 반드시 어떤 방식으로 동작해야 한다는 합의는 없으나, 일반적으로 HTTP Aceept 헤더에 버전 번호를 사용하는 방식이 일반적이다.
- 지금까지 하나의 프로세스에서 다른 프로세스로 전달하는 다양한 방식을 살펴봤다.
- REST와 RPC는 하나의 프로세스가 다른 네트워크를 통해 다른 프로세스와 통신을 주고 받으며 빠른 응답을 기대하는 방식이다.

메세지 전달 데이터 플로 Message-Passing Dataflow

RPC와 DB간 비동기 메세지 전달 시스템을 살펴본다(asynchronous message-passing system)

이 시스템은 client 요청(메세지)를 낮은 지연시간으로 다른 프로세스에 전달한다는 점에서 RPC와 동일하다

메세지를 네트워크 연결로 전송하지 않고 임시로 message broker(message queue)나 message-oriented middleware 라는 중간 단계를 거쳐 전송한다는 점이 DB와 유사하다

Message broker를 사용했을때 장점(DB에 바로 데이터를 전송하거나 받아오는 것과 비교했을때)

- 수신자가 사용 불가능하거나 과부하 상태면 message broker는 buffer 역할을 하여 안정성이 향상된다
- 죽었던 process에 메세지를 다시 전달할 수 있어 메세지 유실을 예방한다
- 송신자가 수신자의 IP, port를 알 필요가 없다
- 하나의 메세지를 여러 수신자에게 전송할 수 있다.
- 논리적으로 송신자, 수신자가 분리된다. (누가 publish 하고, 누가 consume 하는지 몰라도 된다)

  메세지 전달 통신은 단방향이다. 이 점이 RPC와 다르다.

  송신 process는 message 응답을 기대하지 않으며, 응답을 전송하는거야 가능하지만, 보통 별도 채널에서 수행한다.

Message broker

최근에는 RabbitMQ, ActiveMQ, NATS, Apache Kafka 같은 오픈소스 구현이 대중화 됐다.

세부적인 시맨틱은 구현과 설정에 따라 다르지만, 일반적으로 다음과 같다.

1.프로세스가 하나의 메세지 이름이 지정된 큐나 토픽으로 전송한다

2.브로커는 해당 큐나 토픽을 소비자(구독자)에게 전달한다

3.동일한 토픽에 여러 producer, consumer 가 있을 수 있다.

4.토픽은 단방향 data flow 만 제공한다.

- 소비자 스스로 메세지를 다른 토픽으로 게시할 수 도 있고
- 원본 메세지 송신자가 소비하는 응답 큐로 게시할 수 있다.

5.특정 data model을 강요하지 않는다. → 단점일 수도 있다.

6.메세지는 모든 부호화 형식을 사용할 수 있도록 설계되어 있다.

선택한 부호화가 상하위 호환성을 모두 가지기만 한다면 message broker 에게 publisher 와 소비자를 변경해 임의의 순서로 배포할 유연성도 얻을 수 있다.

분산 액터 프래임워크 Distrubuted actor framework

actor model은 동시성을 위한 프로그래밍 모델이다.

- thread(경쟁조건, 잠금, 교착 상태 와 관련된 문제)를 직접 처리하는 대신 로직이 actor 에 캡슐화 된다.
- 액터는 로컬 상태를 가질 수 있고, 비동기 메세지의 송수신으로 다른 actor와 통신한다.
- 메세지 전달을 보장하지 않기 때문에 메세지 유실이 발생할 수 있다.
- 액터는 하나의 client나 entity를 나타낸다
- 한번에 하나의 메세지를 처리하는 방식으로 thread 에 대한 걱정을 하지 않아도 된다

  분산 액터 프레임워크는 여러 node 간 application 확장에 사용하는데 송신자, 수신자가 같은 노드이건 아니건 관계없이 동일한 메세지 전달 구조를 사용한다.

  만약 다른 노드이면 부호화되고 network를 통해 전송된다.

  액터 모델은 메세지가 유실된다는 가정을 가지기 때문에 위치 투명성은 RPC 보다 actor 모델에 더 잘 동작한다(로컬과 원격 통신간 불일치를 줄여준다)

  분산 액터 프레임워크는 순회식 upgrade를 수행을 원할때 메세지가 새로운 버젼을 수행하는 노드에서 예전 버젼을 수행하는 노드로 전송하거나 그 반대 경우도 있으므로 상하위 호환에 유의해야 한다.