CP4. 부호화와 발전

Part 1. 데이터 시스템의 기초

4. 부호화와 발전

▼ 개요

- 대부분의 경우 애플리케이션 기능을 변경하려면 저장하는 데이터도 변경해야 한다. 아마도 새로운 필드나 레코드 유형을 저장해야 하거나 기존 데이터를 새로운 방법으로 제공해야 할지 모른다.
    - 관계형 DB는 일반적으로 DB의 모든 데이터가 하나의 스키마를 따른다.
      스키마가 변경될 수 있지만 특정 시점에는 정확하게 하나의 스키마가 적용된다.
      반면 읽기 스키마 DB는 스키마를 강요하지 않으므로 다른 시점에 쓰여진 이전 데이터 타입과 새로운 데이터 타입이 섞여 포함될 수 있다.
    - 데이터 타입이나 스키마가 변경될 때 애플리케이션 코드에 대한 변경이 종종 발생한다. (ex. 레코드에 새로운 필드가 추가되면 애플리케이션 코드는 해당 필드의 읽고 쓰기를 시작한다.)
      하지만 대규모 애플리케이션에서 코드 변경은 대개 즉시 반영할 수 없다.
        - 서버 측 애플리케이션에서는 한 번에 몇 개의 노드에 새 버전을 배포하고 새로운 버전이 원할하게 실행되는지 확인한 다음 서서히 모든 노드에 실행되게 하는 **순회식 업그레이드(단계적 롤아웃)** 방식이 있다. 순회식 업그레이드는 서비스 정지 시간 없이 새로운 버전을 배포 할 수 있기 때문에 더욱 자주 출시할 수 있다. (좋은 발전성)
        - 클라이언트 측 애플리케이션은 사용자에 전적으로 좌우된다. (업데이트를 안 할 수도 있음)
    - 위 두 가지 내용은 새로운 버전과 이전 버전의 시스템이 동시에 공존할 수 있다는 의미이다.
      시스템이 원할하게 실행되려면 양방향 호환성을 유지해야 한다.
        - 하위 호환성 (새로운 코드는 예전 코드가 기록한 데이터를 읽을 수 있어야 한다.)
        - 상위 호환성 (예전 코드는 새로운 코드가 기록한 데이터를 읽을 수 있어야 한다.)
          하위 호환성 보다 다루기 어려움
    - JSON, XML, 프로토콜 버퍼(Protocol Buffers), 스리프트, 아브로 등 데이터 부호화를 위한 다양한 형식을 확인한다.
        - 어떻게 스키마를 변경하고, old, new 버전의 데이터와 코드가 공존하는 시스템을 어떻게 지원하는지 체크한다.
        - rest, remote procedure call 뿐 아니라 actor 와 메시지 큐 같은 메시지 전달 시스템에서 다양한 데이터 부호화 형식이 데이터 저장과 통신에 어떻게 사용되는지 체크한다.

▼ 데이터 부호화 형식

- 프로그램은 (최소) 두 가지 형태로 표현된 데이터를 사용해 동작한다.
  - 메모리에 객체, 구조체, 목록, 해쉬 테이블 등 데이터가 유지 된다. (이런 데이터 구조는 CPU 에서 효율적으로 접근 조작할 수 있게 최적화(보통 포인터(다른 프로세스가 이해 할 수 없음)) 된다.)
  - 데이터를 파일에 쓰거나 네트워크를 전송하려면 스스로를 포함한 일련 바이트열(ex. JSON) 의 형태로 부호화 해야한다. (다른 프로세스가 이해할 수 있어야 함)
  - 부호화 (직렬화,마샬링) - 인메모리 표현에서 바이트열로 전환
  - 복호화 (파싱, 역직렬화, 언마샬링) - 바이트열에서 인메모리 표현으로 전환
- 언어별 형식
  - 프로그래밍 내장된 부호화 라이브러리는 최소한의 추가 코드로 인메모리 객체를 저장하고 복원할 수 있어 편리하지만, 심각한 문제점 또한 많다. (이팩티브 자바에서는 새로 개발 한다면 그냥 JSON 쓰라고 못 박음)
    - 부호화는 보통 특정 프로그래밍 언어와 묶여 있어 다른 언어에서 데이터를 읽기는 매우 어렵다.
      이런 부호화로 데이터를 저장하고 전송하는 경우 매우 오랜 시간이 될지도 모를 기간 동안 현재 프로그래밍 언어로만 코드를 작성해야 할 뿐 아니라 다른 시스템과 통합하는데 방해된다.
    - 동일한 객체 유형의 데이터를 복원하려면 복호화 과정이 임의의 클래스를 인스턴스화할 수 있어야 한다.
      이것은 종종 보안 문제의 원인이 된다. 공격자가 임의의 바이트열을 복호화할 수 있는 애플리케이션을 얻을 수 있으면 임의의 클래스를 인스턴스화 할 수 있고 공격자가 원격으로 임의 코드를 실행하는 것과 같은 끔직한 일이 발생할 수 있다.
    - 데이터 버전 관리는 보통 부호화 라이브러리에서는 나중에 생각하게 됨.
      데이터를 빠르고 쉽게 부호화하기 위해 상위, 하위 호환성의 불편한 문제가 등한시되곤 한다.
    - 효율성 (부호화, 복호화 시간, 부호화된 크기)도 나중에 생각하게 됨
- JSON, XML, CSV 이진 변형
  - JSON, XML, CSV 부호화
    - 결점
      - 수의 부호화, XML, CSV 에서는 수와 숫자로 구성된 문자열을 구분할 수 없다. JSON 은 문자열과 수를 구분하지만 정수와 부동소수점 수를 구별하지 않고 정밀도도 지정하지 않음. (큰 수를 다룰 때 문제가 일어난다.)
      - JSON 과 XML은 유니코드 문자열(사람이 읽을 수 있는)을 잘 지원한다. 그러나 이진 문자열(문자열에 비해 성능이 훨 뛰어남) 을 지원하지 않는다.
      - 필수는 아니지만 XML, JSON 모두 스키마를 지원한다. 두 부호화 스키마 언어는 상당히 강력하지만 구현하기 난해하다.
        CSV는 스키마가 없으므로 각 로우와 컬럼의 의미를 정의하는 작업이 필요하다.
    - 이러한 결점에도 JSON, XML, CSV는 다양한 용도에 사용하기 충분하다. (특히 데이터 교환 형식)
  - 이진 부호화
    - 큰 데이터 셋인 경우 JSON 같은 부호화 형식일 경우 좋지 못하다. (이진 형식과 비교하면 JSON 같은 부호화는 더 많은 공간을 사용함)
    - JSON 형태는 데이터 안에 속성 값도 포함해야 한다.

```
{
    "userName": "Martin",
    "favoriteNumber": 1337,
    "interests": ["daydreaming", "hacking"]
}
```

- 메시지 팩 형태 (JSON 전용 이진 부호화 형식)

```
83 a8 75 73 65 72 4e 61 6d 65 a6 4d 61 72 74 69 6e ae ... cd 05 3

83 (객체 항목 3)
a8 (문자열 길이 8)
75 73 65 72 4e 61 6d 65 (u s e r N a m e)
a6 (문자열 길이 6)
4d 61 72 74 69 6e (M a r t i n)
ae (문자열 길이 14)
...
cd (부호 없는 16비트 정수)
05 39 (1337)
...
```

  - 첫 번째 바이트 0x83 은 세 개 필드를 가진 객체를 뜻
    두 번째 바이트 0xa8은 이어지는 내용이 8btye 길이의 문자열 뜻
    다음 8바이트는 userName 의 ASCII
    다음 7바이트는 0xa6 문자열 길이 6개 나머지 6개 Martin ASCII

- 스리프트와 프로토 버퍼

▼ 데이터플로 모드

▼ 정리