
# 자바 직렬화

## (Java Serialization)

Version 1.0 2015/03/27

Sunny Kwak

sunykwak@daum.net

# Agenda

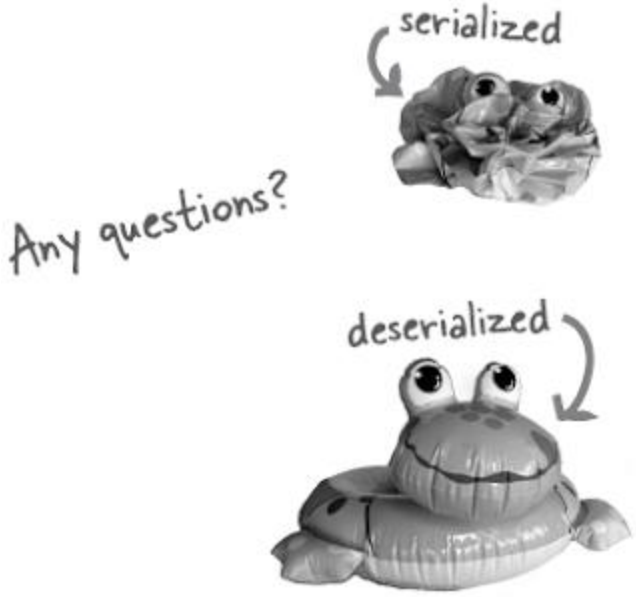
- **직렬화 기술 이해**
  - 직렬화란 무엇인가?
  - 직렬화는 왜 중요한가?
  - 마샬링(marshalling)과의 차이점
  - 분산 컴퓨팅의 흑역사 (COM and CORBA)
  - 직렬화 적용 분야
  - 직렬화 기법 선택 시 고려할 점
  - 직렬화 데이터 형식
  - 구현 기법에 따른 성능 차이
  - 왜 성능이 중요한가?
- **직렬화 프레임워크**
  - JDK's serializable
  - Java externalization
  - Google GSON
  - Jackson
  - BSON for Jackson
  - Protocol Buffers
  - Kryo
  - 성능 벤치마크

What? Why?

# 직렬화 기술 이해

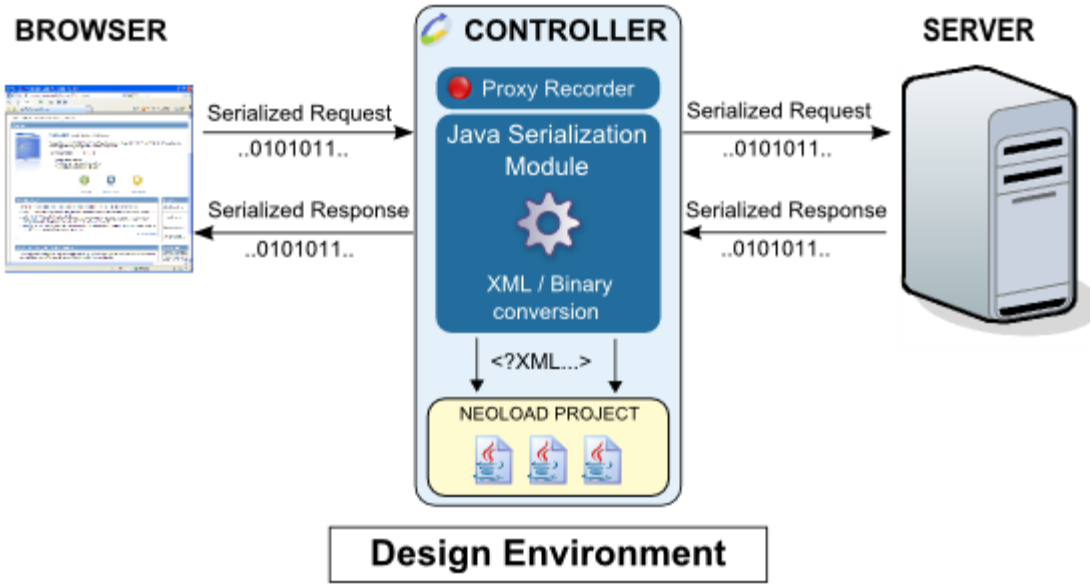
# 직렬화란 무엇인가?

- 컴퓨터 과학, 그중에서 데이터 저장소(data storage)의 맥락에서, **직렬화는 객체의 상태 혹은 데이터 구조를 기록할 수 있는 포맷(예를 들면 파일 또는 메모리 버퍼, 또는 네트워크 연결 링크를 통해 전송될 수 있는 형태)으로 변환**하며, 나중에 동일 혹은 다른 컴퓨터 환경에서 재구성할 수 있게끔 하는 절차이다. (위키피디아 - http://en.wikipedia.org/wiki/Serialization)

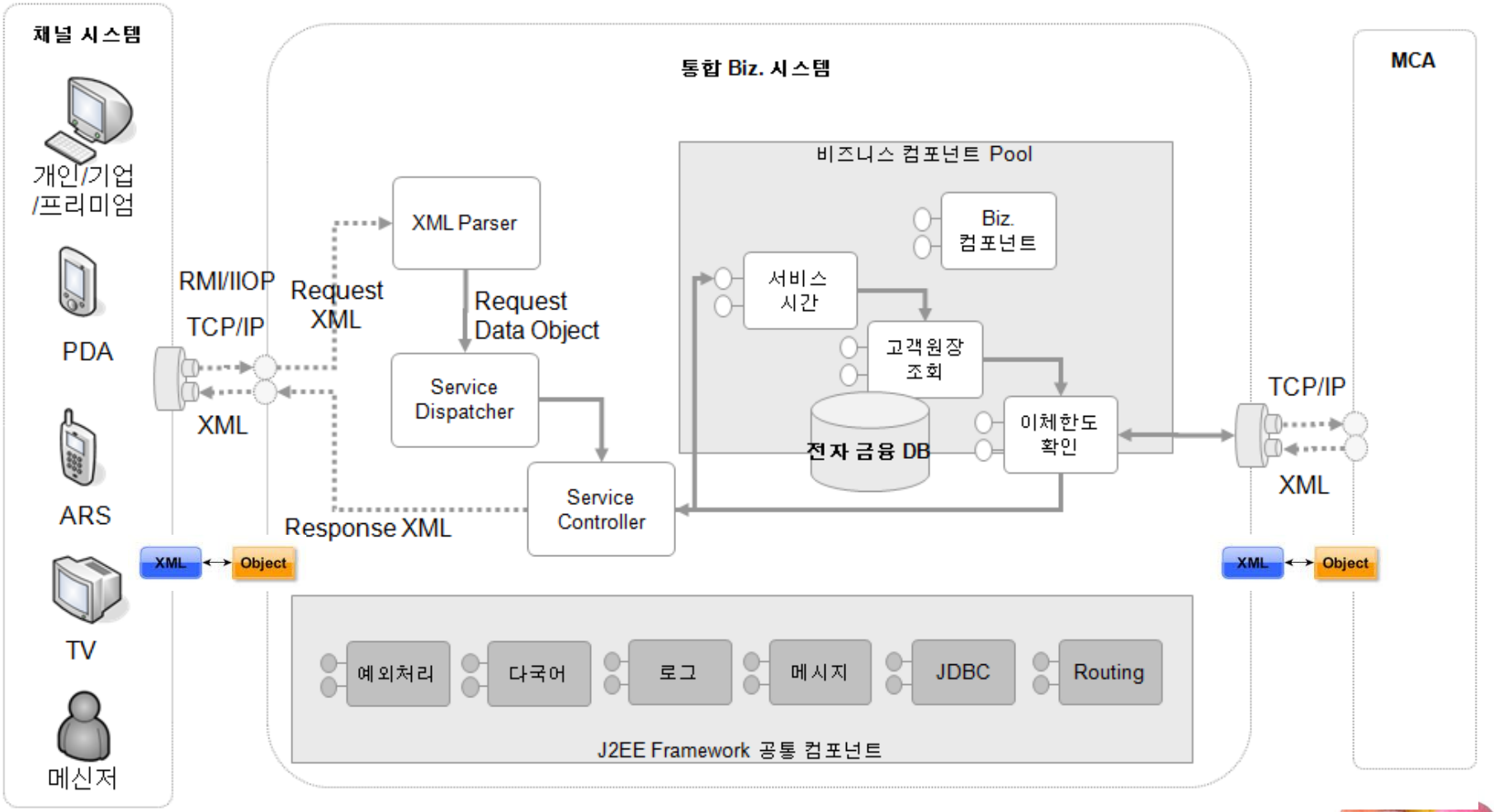



# 직렬화는 왜 중요한가?

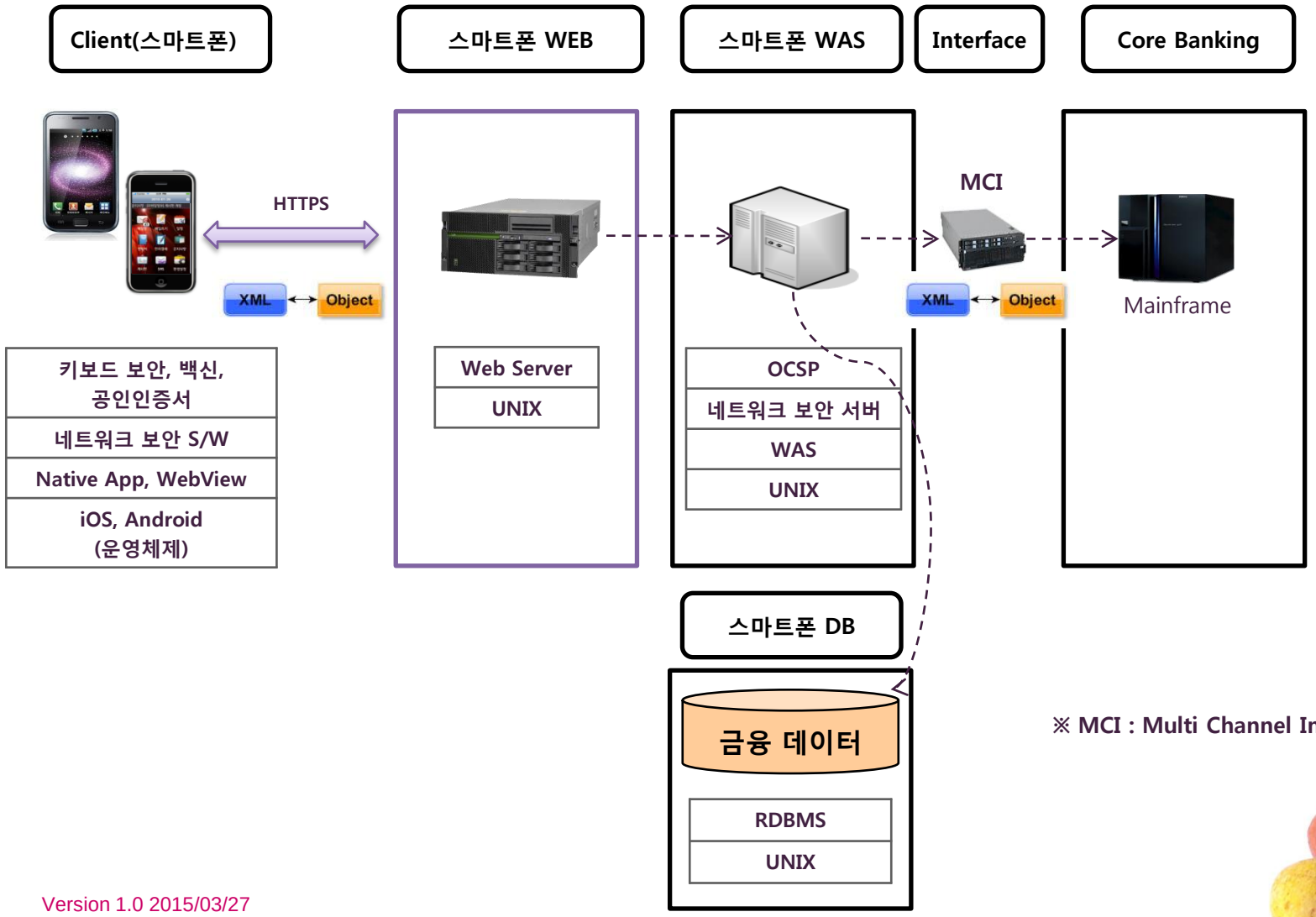
- 직렬화는 인터넷이 보편화된 시대에 빼놓을 수 없는 기술이 되었다.
- 거의 모든 소프트웨어는 네트워크를 통해 데이터를 주고 받고 있으며, 프로그램의 구현하는데 필요한 기반 기술 중에서 빼놓을 수 없게 되었다.
- 직렬화는 네크워크 의존성이 높은 소프트웨어일 경우, 전체 성능을 좌우할 수도 있는 중요한 기술이다.



# 금융 아키텍쳐


채널 시스템
개인/기업
/프리미엄
PDA
ARS
TV
메신저
통합 Biz. 시스템
MCA
RMI/IIOP
TCP/IP
XML
Request
XML
XML Parser
Request
Data Object
Service
Dispatcher
Service
Controller
Response XML
XML
Object
비즈니스 컴포넌트 Pool
Biz.
컴포넌트
서비스
시간
고객원장
조회
이체한도
확인
전자 금융 DB
TCP/IP
XML
XML
Object
예외처리
다국어
로그
메시지
JDBC
Routing
J2EE Framework 공통 컴포넌트

# 금융 모바일

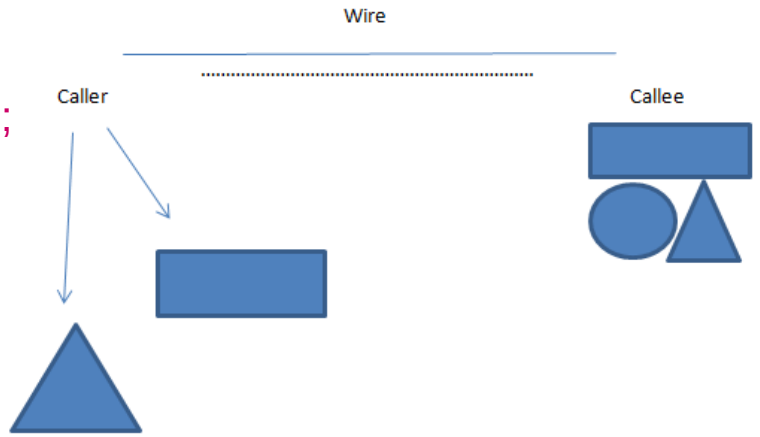



※ MCI : Multi Channel Interface

# 마샬링(marshalling)과의 차이점

- 마샬링과 직렬화는 원격 프로시저 호출(remote procedure call)이라는 맥락에서는 대략적으로 비슷한 단어이다, 의도적인 측면에서는 그 의미가 다르다.
- 마샬링은 이곳에서 저곳으로 인자(parameter)들을 전달하는 작업이고, 직렬화는 구조적인 데이터를 원시형(primitive) 형태에서 바이트 스트림과 같은 형식을 복사하는 작업이다. 이러한 의미에서 직렬화는 마샬링의 pass-by-value 개념을 구현하는 수단이다.

Object response =
RemoteComputer.doSomething(request);



# 분산 컴퓨팅의 흑역사

- **COM (Component Object Model)**
  - : 컴포넌트 객체 모델 (COM)는 마이크로소프트에 의해 1993 년에 도입된 소프트웨어 컴포넌트에 대한 바이너리 인터페이스(binary interface) 표준이다.
  - 다양한 프로그래밍 언어로 작성된 프로세스 간 통신 및 동적 객체 생성을 가능하도록 하는 기술이다. COM 표준은 OLE, OLE 자동화(automation), **ActiveX,** COM+, DCOM, 윈도우 쉘, DirectX, UMDF 및 윈도우 런타임 등 여러 가지 다양한 Microsoft 기술 및 프레임워크의 기초가 된다.

- **CORBA (Common Object Request Broker Architecture)**
  - Common Object Request Broker Architecture (CORBA)는 Object Management Group(OMG)에 의해 정의된 다양한 플랫폼 상에서 동작하는 시스템 간의 통신을 용이하게 하기 위한 표준이다.
  - CORBA는 서로 다른 운영 체제, 프로그래밍 언어, 컴퓨팅 하드웨어 위에서 동작하는 시스템들이 협업(collaboration)할 수 있게 해준다. CORBA는 분산 객체 패러다임의 사례이다.

- **실패한 시도**
  - 네트워크로 연결된 머신(machine)들을 마치 하나인 것처럼 연결하고 싶었으나...
  - 로컬 머신의 프로세스 간 통신이외에는 **망했어요!**

# 직렬화 적용 분야

- **파일 저장소 (File storage)**
  - 프로그램 실행 중에 생성된 데이터를 영구 저장소(파일 시스템) 등에 저장한 후, 이후에 프로그램이 다시 실행되었을 때 저장된 데이터를 메모리 상에 객체 형태로 복구해 사용한다.

- **네트워크 통신 (Network communication)**
  - 네크워크 상에 떨어져 있는 프로그램 간에 데이터를 주고 받기 위해 데이터를 직렬화한 후, 패킷(packet)에 담아 전송한다.

- **데이터베이스 (Database)**
  - 복잡한 형태의 객체를 데이터베이스에 저장할 때 직렬화한 문자열 형태로 테이블의 컬럼에 저장하기도 한다.

- **웹 환경 (Web environment)**
  - 웹 서버에서 브라우저(클라이언트)로 구조화된 데이터를 전송할 때 직렬화한 후 -JSON 형식 등- 전달하는 방식이 점차 많이 사용되고 있다.

# 직렬화 기법 선택 시 고려할 점

- **단순성 (simple)**
  - 사용하기가 복잡하지 않아야 한다.
- **경량 (compact)**
  - 프레임워크(혹은 라이브러리)의 규모가 작아야 한다.
- **유연성 (flexible)**
  - 다양한 데이터 타입을 직렬화할 수 있어야 한다.
- **버전지원 (viersioning)**
  - 객체의 데이터 구조는 설계 및 개발, 나아가 유지보수 단계에서 변화할 수 있다.
- **속도 (fast)**
  - 처리 속도가 빠르면 빠를수록 좋다!
- **확장성 (scalable)**
  - 복잡하거나, 거대한 형태의 데이터를 직렬화 할 수 있어야 한다.
    (deep copy, circular reference ...)

# 직렬화 데이터 형식

- **Binary**
  - 메모리에 저장된 데이터를 최소한의 가공 혹은 가공 없이 바이트의 연속된 형태로 저장하는 방식

- **JSON (JavaScript Object Notation)**
  - 텍스트 형식이므로 사람과 기계 모두 읽기 가능하다. 다양한 프로그래밍 언어에서 읽고 쓸 수 있기 때문에 널리 사용된다.

- **XML (eXtensible Markup Language)**
  - 텍스트 형식이며, JSON에 비해 복잡하다. JSON에 대해 가지는 장점은 스키마(schema)를 적용할 수 있고 무결성 검사가 가능하다.

- **YAML (YAML Ain't Markup Language)**
  - XML에 비해 사람이 읽고 쓰기 쉽도록 고안된 마크업(markup) 언어이다. 문법이 상대적으로 단순하고, 가독성이 높게 설계되어 있다.

# 구현 기법에 따른 성능차

- **Native memory copying using C operations**
  - 객체가 할당되어 있는 메모리 자체를 C 함수를 이용해 복사.
- **“Unsafe” operations**
  - 본래 자바 코어 개발자들이 low-level 프로그래밍 하기 위해 만든 API이다. JNI(Java Native Interface) 와 비슷하거나 유사한 수준의 성능.
- **Ignore object introspection**
  - introspection 은 reflection과 유사한 기술이다. 다만, introspection 은 자바의 instanceof 연산자 처럼 객체의 타입 정보만 조회한다.
- **Direct object-object copying**
  - 자바 코드로 객체 내의 모든 멤버 변수를 복사하는 로직을 구현하는 것.

# 왜 "성능"이 중요한가?

- **CPU 비용**
  - 메모리에 존재하는 바이너리 형태의 객체를 디스크 등에 저장할 수 있는 형태 (텍스트 등)로 저장하기 위해서는 변환(transform) 처리 과정이 필요하며, 반대의 처리 또한 필요하다.

- **메모리 비용**
  - 변환 작업을 수행하는 과정에서 임시 버퍼(temporary buffer)를 할당하고, 네트워크를 통한 송수신 과정에서 스트림 처리 등에 따른 공간 할당이 필요하다.

- **네크워크 비용**
  - 직렬화를 수행하는 대다수의 프로그램 혹은 시스템은 네트워크를 통해 데이터를 주고 받게 된다. 네트워크 송수신에 있어서 패킷(packet)의 크기가 커질수록 전체 성능은 떨어진다.

How

# 직렬화 프레임워크

# JDK's Serializable

- **JDK 의 Serializable 인터페이스**
  - 프로그래밍하기 가장 쉽고, Serializable 인터페이스를 이용해 별도의 라이브러리 없이 즉시 사용할 수 있다.
  - 클래스를 릴리즈(release)한 후에는 구현을 변경하기 어려워 유연성(flexibility)을 감소시킨다.
  - C++, 파이썬(python)등 다른 언어로 구현된 프로그램과 데이터를 교환(exchange)할 수 없다.
  - 기본 연산자의 취약점(hole)으로 인해 불변 값이 손상되거나, 비정상적인 접근이 발생할 수 있다. (invariant corruption and illegal access)
  - 커스터마이징(customization)이 불가능하고, 소스 코드를 수정할 수 있어야 한다.

# Java externalization

- **직렬화 코드를 직접 구현**
  - 객체를 저장(persist) 및 복구(restore)하는 Externalizable 인터페이스를 구현해 직접 직렬화를 구현한다.
  - 인스턴스의 컨텐츠를 저장하고 복구하는 역할을 수행하는 클래스를 구현해야 한다.
  - 클래스의 구조가 변경될 때 마다, 읽고 쓰는 코드를 수정해야 한다.

```
public void writeExternal(ObjectOutput out)
        throws IOException {
    out.writeInt(getId());
    out.writeObject(getName());
    out.writeDouble(getPrice());
}

public void readExternal(ObjectInput in)
        throws IOException,
        ClassNotFoundException {
    setId(in.readInt());
    setName((String) in.readObject());
    setPrice(in.readDouble());
}
```

# Google GSON

- **Google GSON**
  - 자바 객체를 JSON으로 변환하거나 반대의 작업을 수행하는 자바 라이브러리.
  - 직렬화된 객체의 소스 코드를 필요로 하지 않는다.
  - 커스텀 표현(custom representatives)을 지원한다.

```
class Item {
    @Since(2.0)
    private String currentName;
    @Since(1.0)
    private String newName;
    private String name;

    public static void main(String[] args) {
        Item versionItem = new Item();
        Gson gson = new GsonBuilder().
                setVersion(1.0).create();
        String json = gson.toJson(versionItem);
    }
}
```

# Jackson JSON

- **Jackson JSON**
  - 고성능, 인간공학적 JSON 프로세서 자바 라이브러리
  - 광범위한 커스터마이징 툴 지원
  - 혼합 어노테이션 (Mix-in annotations)
  - 실체화된 인터페이스 (Materialized interfaces)
  - 다양한 데이터 포맷 : JSON, CSV, Smile(binary JSON), XML, YAML

# BSON for Jackson

- **BSON for Jackson**
  - 바이너리 인코딩된 JSON (Binary encoded JSON)
  - 몽고 DB의 주된 데이터 교환 포맷
    (Main data exchange format for MongoDB)
  - 확장 프로그램 작성 가능 (Allows writing custom extensions)

# Protocol Buffers

- **Protocol Buffers**
  - 구조적인 데이터를 확장가능하며 효율적인 포맷을 변환하는 방법 제공
  - 구글 내부에서 대부분의 내부 RPC 프로토콜과 파일 포맷에 Protocol Buffers를 사용 중.
  - Java, C++, Python 지원

```
message User {
  required string login = 1;
  repeated Order orders = 2;
}

message Order {
  required int32 id = 1;
  optional string date = 2;
}
```

```
UserProto.Product.Builder builder =
        UserProto.Product.newBuilder();
builder.setName("test");

UserProto.Product product = builder.build();

byte[] data = product.toByteArray();

product = Product.parseFrom(data);
```

# Kryo

- **Kryo**
  - 빠르고 효율적인 객체 그래프 직렬화 자바 프레임워크
  - 구글 코드 상의 오픈 소스 프로젝트
  - 자동화된 깊고 얕은 복사/복제
    (Automatic deep and shallow copying/cloning)
  - 소스 클래스에 대한 코드 작성 요건이 거의 없음
    (Doesn't put requirements on the source classes in most cases)

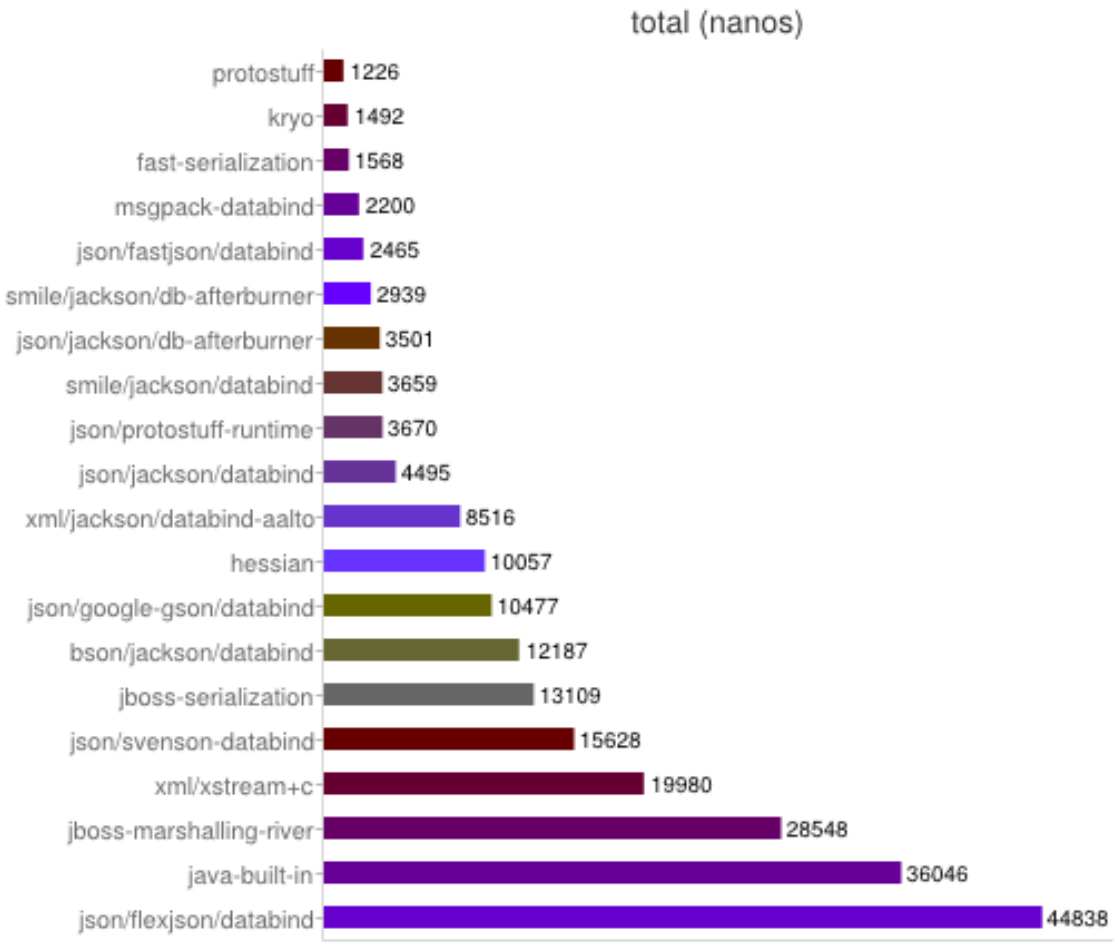
```
Kryo kryo = new Kryo();

kryo.register(User.class);
kryo.register(Order.class);
kryo.register(Product.class);

kryo.register(List.class);
kryo.register(ArrayList.class);

try (Output output = new Output(outputStream)) {
    kryo.writeObject(output, user);
}
```

# 성능 벤치마크


total (nanos)
protostuff 1226
kryo 1492
fast-serialization 1568
msgpack-databind 2200
json/fastjson/databind 2465
smile/jackson/db-afterburner 2939
json/jackson/db-afterburner 3501
smile/jackson/databind 3659
json/protostuff-runtime 3670
json/jackson/databind 4495
xml/jackson/databind-aalto 8516
hessian 10057
json/google-gson/databind 10477
bson/jackson/databind 12187
jboss-serialization 13109
json/svenson-databind 15628
xml/xstream+c 19980
jboss-marshalling-river 28548
java-built-in 36046
json/flexjson/databind 44838

# 이슈와 조언

| Library | Description |
|---|---|
| Gson, Jackson | Crashed when serializing cyclic dependency |
| Simple | Crashed for very big XML file |
| Avro | Bug during deserialization |

| Library | Usage |
|---|---|
| Kryo | Fast and compact serializer for complex objects over network |
| Protocol buffers | Fast serializer for simple objects |
| Jackson(smile) | Jackson-based serializer for Web usage |
| Google JSON | Dirty solution to quickly serialize/deserialize objects |
| Apache Avro | Serialize objects into files with possible schema changes |
| Java | Out-of-the-box trusted solution without additional libraries |

# 참고 자료

- Serialization and Performance in Java (슬라이드쉐어)
  - http://www.slideshare.net/Strannik_2013/serialization-and-performance-in-java
- eishay/jvm-serializers (직렬화 라이브러리 성능 비교 사이트)
  - https://github.com/eishay/jvm-serializers/wiki
- 자바 성능 직렬화 (블로그)
  - http://sunnykwak.tistory.com/95