


이펙티브 자바 CP.11

🕒 작성 일시	@2023년 4월 2일 오후 9:48
🕒 최종 편집 일시	@2023년 4월 2일 오후 10:56
📄 유형	이펙티브 자바
👤 작성자	 종현 박
👥 참석자	
🗣️ 언어	

11. 직렬화

- 85. 자바 직렬화의 대안을 찾으라
- 86. Serializable을 구현할지는 신중히 결정하라
- 87. 커스텀 직렬화 형태를 고려해라
- 88. readObject 메서드는 방어적으로 작성하라
- 89. 인스턴스 수를 통제해야 한다면 readResolve 보다는 열거 타입을 사용해
- 90. 직렬화된 인스턴스 대신 직렬화 프록시 사용을 검토하라

11. 직렬화

▼ 85. 자바 직렬화의 대안을 찾으라

- 직렬화
 - 장점
 - 프로그래머가 어렵지 않게 분산 객체를 만들 수 있음
 - 단점
 - 보이지 않는 생성자, API와 구현 사이의 모호해진 경계, 잠재적인 정확성 문제, 성능, 보안, 유지보수성 등 대가가 큼.
 - 공격 범위가 너무 넓고 지속적으로 더 넓어져 방어하기 어렵다
 - `ObjectInputStream` 의 `readObject` 메서드를 호출하면서 객체 그래프가 역직렬화되기 때문이다.

- `readObject` 메서드는 클래스 패스 안의 거의 모든 타입의 객체를 만들어 낼 수 있는 생성자이다.
- 바이트 스트림을 역직렬화하는 과정에서 이 메서드는 그 타입들 안의 모든 코드를 수행 할 수 있다. (타입들의 코드 전체가 공격 범위)
- CERT 조정 센터의 기술 관리자 로버트 시커드

자바의 역직렬화는 명백하고 현존하는 위험이다. 이 기술은 지금도 애플리케이션에서 직접 혹은 자바 하부 시스템(RMI - Remote Method Invocation), (JMX - Java Management Extension), (JMS - Java Messaging System) 을 통해 간접적으로 쓰이고 있기 때문이다. 신뢰할 수 없는 스트림을 역직렬화하면 원격 코드 실행, 서비스 거부 등의 공격으로 이어질 수 있다. 아무 잘 못도 없는 애플리케이션이라도 이런 공격에 취약해질 수 있다.

- 가젯
 - 자바 라이브러리와 널리 쓰이는 서드파트 라이브러리에서 직렬화 가능 타입 중 역직렬화 과정에서 호출되 잠재적으로 위험한 동작을 수행하는 메서드를 뜻함.
- 역직렬화 폭탄
 - 역직렬화에 시간이 오래 걸리는 짧은 스트림을 역직렬화하는 것만으로도 서비스 거부 공격에 쉽게 노출될 수 있는 스트림
 - ex) 역직렬화 폭탄 - 이 스트림의 역직렬화는 영원히 계속됨

```
static byte[] bomb() {
    Set<Object> root = new HashSet<>();
    Set<Object> s1 = root;
    Set<Object> s2 = new HashSet<>();
    for (int i = 0; i < 100; i++) {
        Set<Object> t1 = new HashSet<>();
        Set<Object> t2 = new HashSet<>();
        t1.add("foo"); // t1 과 t2 를 다르게 만든다.
        s1.add(t1); s1.add(t2);
        s2.add(t1); s2.add(t2);
        s1 = t1;
        s2 = t2;
    }
    return serialize(root); // 간결하게 하기 위해 이 메서드의 코드는 생략
}
```

- `root` 객체 그래프는 201개의 `HashSet` 인스턴스로 구성되어, 그 각각은 3 개 이하의 객체 참조를 갖는다.
- 역직렬화시 끝나지 않는다.
 - 문제는 `HashSet` 인스턴스를 역직렬화하려면 그 원소들의 해시코드를 계산하는 데 있다. 반복문에 의해 구조의 깊이가 100단계까지 만들어 진다. 따라서 이 `HashSet` 을 역직렬화하려면 `hashCode` 메서드를 2^{100} 번 넘게 호출 될 것이다.
 - 역직렬화가 영원히 지속되는 것도 문제지만, 무언가 잘 못되었다는 신호조차 주지 않는 것도 문제이다.
- 역직렬화의 대처법
 - 아무것도 역직렬화하지 않는 것이 가장 좋은 직렬화 위험을 회피하는 방법이다.
 - **크로스-플랫폼 구조화된 데이터 표현**
 - 객체와 바이트 시퀀스를 변환해주는 다른 매커니즘이 있다.
 - 이 방식들은 자바 직렬화의 위험을 회피하면서 다양한 플랫폼 지원, 우수한 성능, 풍부한 지원 도구, 활발한 커뮤니티 등 수 많은 이점을 제공한다.
 - 이러한 매커니즘들도 직렬화 시스템이라 부르지만, 자바 직렬화와 구분을 해 **크로스-플랫폼 구조화된 데이터 표현**이라고 한다.
- **크로스-플랫폼 구조화된 데이터 표현**
 - 이 표현들의 공통점은 자바 직렬화보다 훨씬 간단하다.
 - 임의 객체 그래프를 자동으로 직렬화/역직렬화 하지 않는다.
 - 대신 속성-값 쌍의 집합으로 구성된 간단하고 구조화된 데이터 객체 사용.
 - 그리고 기본 타입 몇 개와 배열 타입만 지원할 뿐이다.
 - 이런 간단한 추상화만으로도 아주 강력한 분산 시스템을 구축하기에 충분하고, 자바 직렬화의 문제들을 회피 할 수 있다.
- **크로스-플랫폼 구조화된 데이터 표현**의 선두주자
JSON, 프로토콜 버퍼 (Protocol Buffers, protobuf)
 - JSON
 - 브라우저와 서버의 통신용 설계
 - 자바스크립트용
 - 텍스트 기반 사람이 읽을 수 있음

- 오직 데이터 표현
- 프로토콜 버퍼 (Protocol Buffers, protobuf)
 - (구글) 서버 사이에 데이터를 교환하고 저장하기 위해 설계 (gRPC)
 - C++용
 - 이진 표현이라 효율이 높음
 - 문서를 위한 스키마를 제공하고 올바르게 쓰도록 강요
- 어쩔 수 없이 자바 직렬화를 사용시
 - 신뢰할 수 없는 데이터는 절대 역직렬화하지 않는 것이다.
- 직렬화를 피할 수 없고 역직렬화한 데이터가 안전한지 완전히 확신할 수 없다면
 - 객체 역직렬화 필터링을 사용하자
 - 데이터 스트림이 역직렬화되기 전에 필터를 설치하는 기능이다.
 - 클래스 단위로, 특정 클래스를 받아들이거나 거부할 수 있다.
 - 블랙 리스트 (기본 수용 모드)
 - 기록된 잠재적으로 위험한 클래스들은 거부
 - 화이트 리스트 방식 (기본 거부 모드) 추천
 - 기록된 안전하다고 알려진 클래스만 승인
- 86 아이템 부터 직렬화 가능 클래스를 올바르게 안전하고 효율적으로 작성하려는 방법을 제시한다.
- 정리
 - 직렬화는 위험하니 피해야한다.
 - 시스템의 밑바닥 부터 설계 한다면 JSON 이나 프로토콜 버퍼 같은 대안을 사용하자
 - 신뢰할 수 없는 데이터는 역직렬화 하지 말자.
 - 꼭 한다면 객체 역직렬화 필터링을 사용하되, 모든 공격을 막아줄 수 없음을 기억하자.
 - 클래스가 직렬화를 지원하도록 만들지 말고, 꼭 만들어야겠으면 정말 신경써야 한다.(아이템 86 ~ 90)

▼ 86. Serializable을 구현할지는 신중히 결정하라

- ▼ 87. 커스텀 직렬화 형태를 고려해라
- ▼ 88. readObject 메서드는 방어적으로 작성하라
- ▼ 89. 인스턴스 수를 통제해야 한다면 readResolve 보다는 열거 타입을 사용해
- ▼ 90. 직렬화된 인스턴스 대신 직렬화 프록시 사용을 검토하라