자바 리플렉션에 대한 오해와 진실

Posted on 2014/03/15

이 글은 크몽 재능인, socurites 님이 원고를 기고하셨습니다.

내가 그의 이름을 불러 주기 전에는 그는 다만 하나의 몸짓에 지나지 않았다. 내가 그의 이름을 불러 주었을 때, 그는 나에게로 와서 꽃이 되었다. – 김춘수, "꽃"

주변을 자세히 살펴 보자. 이름만 알고 있다면, 그 물체의 모든 정보를 알 수 있는 것들이 많이 존재한다. 내 오른손에 있는 마우스 "MOUSE(MOARUO)"의 경우, 구글과 같은 검색서비스를 이용하면 "Optical, Wheel, Mouse, USB" 라는 정보를 찾아낼 수 있다. 뿐만 아니라, 옥션과 같은 주문서비스를 통해 새로운 "MOUSE(MOARUO)" 마우스를 구매할 수도 있다.

프로그래밍의 세계에서도, 현실의 검색서비스 그리고 주문서비스의 기능을 제공하는데, 이를 "Self Introspection" 또는 "Reflection"이라고 부른다. 클래스(Class)의 이름만으로 클래스의 정보(필드, 메서드)를 찾거나 새로운 객체(Object)를 생성할 수 있다는 얘기다.

Java 는 "java.lang.reflect" 패키지를 통해 Reflection 기능을 제공한다. "Java Reflection" 기능은 간접적으로는 알게 모르게 사용하고 있으며, 또한 필요할 경우 직접적으로 사용할 때도 있다. Reflection 을 직접적으로 사용할 경우, 항상 성능을 염두해야 하지만, 성능만이 고려대상의 전부는 아니다. 수 차례의 SI 프로젝트를 겪으며 터득한 "Java Reflection"에 대한 진실과 오해에 대해 알아보자.

Java Reflection 이란?

오늘은 금요일. 마우스를 제어하는 컨트롤러 프로그램을 개발하기 위해 당신은 두 달여 간을 쉬지 않고 일했다. 오늘은 마우스를 호출하는 작업을 완료하면, 두 달 동안 팽개친 애인에게 점수를 딸 수 있는 유일한 기회이다!

여기 2 개의 마우스 클래스가 있다. 하나는 유선 마우스로 MouseMoaruo 이고, 다른 하나는 MouseMx610 으로 무선 마우스다.

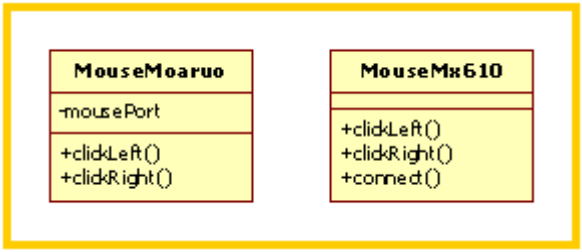

그림 1. 마우스 클래스 Diagram

컨트롤러 프로그램으로부터 받은 마우스 이름을 매개변수로 사용해서, 컨트롤러에 연결할 마우스 클래스의 객체를 생성하는 Factory 클래스를 개발하려고 한다. 매개변수가 “moaruo”일 경우에는 MouseMoaruo 클래스의 객체를 생성하고, “mx610″일 경우에는 MouseMx610 클래스의 객체를 생성해야 한자. 제일 먼저 생각나는 가장 쉬운 접근법은 “if/else”문을 이용하는 것이다.

```
public class MouseFactory {
    public Object getInstance(String name) {
        if ( name.equals("moaruo") ) {
            return new MouseMoaruo();
        } else if ( name.equals("mx610") ) {
            return new MouseMx610();
        }
        return null;
    }
}
```

그림 2. if/else 문을 이용한 Factory 예

개발이 완료될 즈음, 고객이 표준 마우스 모델이 하나 더 추가 되어야 한다고 수정을 요구했다(여러분이라면 이런 상황은 쉽게 상상할 수 있을 것이다). 늘 그렇듯, 고객의 요구를 받아들여 소스코드를 수정하여 if 문을 하나 더 추가하여 요구사항을 해결했다. 이제 퇴근하려는 찰나, 고객은 여러분을 실망시키지 않고 또 다른 마우스를 들고 와서는, 하는 김에 이것도 추가해 달라고 얘기한다. 고객과 애인, 어느 것을 선택할 것 인가!

Factory 클래스의 가장 큰 문제는 if/else 문의 사용이다. 객체 지향 프로그램의 대부분에서 if/else 문은 필요치 않다. 프로그램에 if/else 문(특히, 대량의 if/else 문)을 사용했다면, 객체지향 철학을 어긴 것은 아닌지 의심해 보아야 한다.
애초부터 고객과 애인 중 하나를 선택해야 하는 햄릿의 고민은 할 필요가 없었다. 고객과 애인 모두를 만족시킬 수 있는 방법이 있기 때문이다. Reflection 을 사용한 방법이 그것이다.

```
public class MouseFactoryReflection {
    public Object getInstance(String className)
        throws Exception {
        Class cls = Class.forName(className);
        Constructor constructor = cls.getConstructor();
        return constructor.newInstance();
    }
}
```


그림 3. Reflection 을 이용한 Factory 예

“그림 3″에서 보는 바와 같이, Constructor 객체를 이용하여 마우스 객체를 생성한다. 이 경우, 컨트롤러에서 클래스의 이름을 넘겨주어야 하지만, 새로운 마우스가 생기더라도 Factory 클래스의 수정 없이 유연하게 확장 가능한 코드가 된 것이다. 우리가 일하는 엔터프라이즈 프로젝트 현장에서는 이와 같은 Reflection 이 셀 수 없을 만큼 다양하게 활용하고 있다.

사용처

현장에서 자바 개발자들이 Reflection 을 직접 사용하는 것은 극히 드문 일이다. 그것은 Reflection 이 적용될 수 있고 또한 적용되어야 할 곳은 라이브러리 클래스, 공통 컴포넌트 클래스, 그리고 프레임워크와 같이 Reflection 을 통해 얻는 이득(재사용성, 확장성, 생산성, 유연성 등)이 극대화될 수 있는 곳이어야 하기 때문이다.

1. Java Serialization

객체를 직렬화(Serialization) 해야 할 경우 Serializable 인터페이스를 구현한다. 그리고, 직렬화된 객체를 읽기 위해서는 java.io.ObjectInputStream 클래스의 readObject() 메서드를 이용한다. 필요에 따라 Serializable 인터페이스를 구현한 클래스가 readObject() 메서드를 구현할 수도 있다. 이때, java.io.ObjectInputStream 클래스의 readObject() 메서드는 내부적으로 Reflection 을 이용하여 직렬화된 객체의 readObject() 메서드를 호출한다.

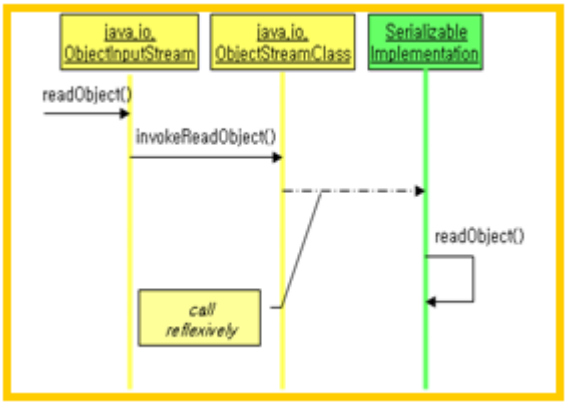

그림 4. Reflection 을 사용한 Serialization

2. Apache Commons BeanUtils library

Struts 프레임워크를 적용한 프로젝트에서 개발한 경험이 있다면, Apache Commons 프로젝트의 BeanUtils 라이브러리 사용을 고민해 본 경험 또한 있을 것이다. Struts 프레임워크는 HttpRequest 객체의 파라미터를 이용하여 ActionForm 클래스의 객체를 생성한다. 이 ActionForm 클래스의 객체를 생성하는 곳에서도 Reflection 이 적용되었다.

Struts 를 이용할 경우, 가장 성가신 부분은 ActionForm 클래스의 객체를 대응하는 VO 클래스의 객체로 변환하는 작업이다. 이 작업은 서비스 레이어를 Struts 에 종속되지 않게 하기 위해 또는, 레이어 분리를 위해 반드시 수행되어야 한다. 만일 지금까지 ActionForm 객체를 서비스 레이어로 바로 넘겼다면 다시 한번 생각해 보라. VO 클래스 사용에 따른 레이어의 분리와, VO 를 사용하지 않음으로써 얻는 개발 생산성 증가, 둘 중 어느 한가지를 택한 것인지.

이때, 사용할 수 있는 것이 Commons BeanUtils 라이브러리이다. Beanutils.copyProperties(Object dest, Object orig)를 이용하여 간단히 ActionForm 객체를 VO 객체로 변환할 수 있다. 규칙은 ActionForm 클래스의 인스턴스 변수명과 VO

클래스의 인스턴수 변수명이 같아야 한다는 것이다. 이 규칙을 따른다면, Reflection 의 마술이 여러분을 위한 모든 작업을 수행해 줄 것이다.

이슈

1. Reflection 을 사용한 코드는 느리다
개발자들 사이에 공공연히 진실로 받아들여지는 이 말은 사실이 아니다. 적절히 사용한 Reflection 은 오히려 성능을 향상시킬 수 있으며, 또한 많은 이득을 제공한다. 뿐만 아니라, 성능만을 고려한 구현이 객체 지향의 설계 원칙들을 역행한다면, 오히려 이는 더욱 나쁜 결과를 낳게 된다.

2. Reflection 을 이용하여 개발한 프로그램은 에러가 발생하기 쉽고 디버깅이 어렵다
Reflection 은 컴파일 시 타입 체킹을 할 수 없다. 따라서, 런타임시 잘못된 파라미터로 인해 런타임 에러가 발생하기 쉽다. 이는 사실이다. 하지만, 적절히 사용된 런타임 에러 메시지를 이용해 충분히 디버깅이 쉬운 환경으로 만들 수 있다.

3. Reflection 을 사용한 코드는 복잡하다
Reflection 을 사용한 코드는 일반적인 객체 생성, 메서드 호출 코드에 비교하면 복잡한 것이 사실이다. 하지만 클래스의 타입을 비교하여 객체를 생성하는 코드의 경우, 대량의 if/else 문을 사용하는 것보다 Reflection 을 이용하여 재사용 가능한 컴포넌트로 만든다면, 오히려 코드를 단순화한다.

4. 성능(Performance) vs. 유연성(Flexibility)
앞서 말한 바와 같이 “Reflection 을 사용한 코드는 느리다”는 사실은 사실이 아니다. 이 말은 Reflection 을 사용할 경우 성능이 떨어지지 않는다라는 얘기가 아니다. 오히려, 성능이 떨어진다는 결과가 다수 존재한다. 아래는 Dennis Sosnoski(5. Java Programming dynamics, Part 2: Introducing reflection)가 측정한 Reflection 에 관한 성능 결과이다. 그림에서 알 수 있듯이, Reflection 을 사용할 경우, 직접(Direct) 또는 참조(Reference)의 경우에 비해 2 ~ 4 배 정도 느리다.

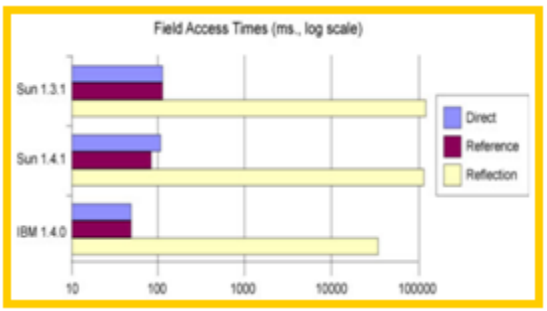

그림 5. 필드 변수 Access 시간

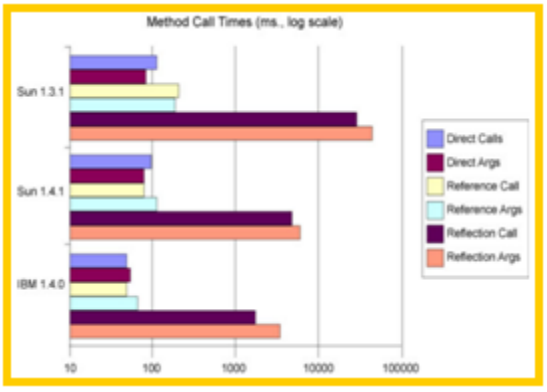

그림 6. 메서드 호출 시간

이 결과를 통해 알 수 있는 사실은 “Reflection 에 따른 성능 저하”가 아니라 “성능 측정 결과, Reflection 을 사용한 **지금 이 경우에는** 성능이 저하되는 것을 검증했다”라는 것이다. 최적화 또는 성능 개선(Optimization)시 유의해야 할 점은 반드시 최적화 이전과 이후의 성능을 측정하여, 성능개선이 가시적으로 보일 때에만 적용해야 한다는 것이다. 만일 최적화가 필요하다고 느낀다면, 아래 규칙을 따르라.

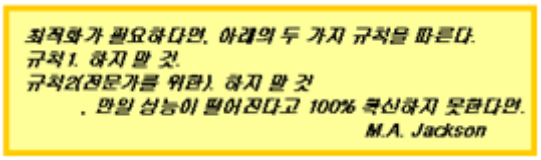

그림 7. 최적화 규칙

Reflection 과 관련된 성능에 관한 논쟁은 오해에서 비롯된 것이다. 이것은 JDK 초기 버전(1.3.0 이전 버전)의 경우 Reflection 의 성능이 현저히 떨어졌다. 하지만 이후의 JDK 버전에서는, Reflection 의 중요성을 인식한 Sun 의 지속적인 노력으로 성능이 개선되고 있으며, 앞으로도 개선의 여지가 남아 있다. 뿐만 아니라 잘 적용한 Reflection 은 많은 이득을 제공한다.

Reflection 을 통해 얻을 수 있는 가장 큰 이득은 시스템 유연성(Flexibility)이다. “그림 3″에서 보는 바와 같이 Mouse 컨트롤러는 미래의 어떤 마우스와도 동작할 수 있다. 이처럼 성능보다 유연성이 더 중요한 상황이 많다

물론 Reflection 적용에 따른 가시적인 성능 저하를 확신한다면, 다른 대안을 생각해 볼 수 있다. 대안에는, Reflection 대신 interface 를 통한 메서드 호출, 코드 자동 생성(Code Generation), 또는 최악의 경우 하드 코딩이 있다.

5. Compile vs. Run-time Type Checking

자바의 경우 컴파일 단계에 강력한 Type Checking 을 지원한다. 아래와 같이 두 개의 클래스가 틀릴 경우, 컴파일이 에러가 발생한다.

```
public void checkType() {
    MouseMoaruo mouseMoaruo
      = new MouseMx610();
}
```

그림 8. 컴파일 에러 예

Reflection 은 실제 클래스 없이 클래스의 이름 또는 메서드의 이름만을 이용한다. 따라서, 아래와 같이, 개발 단계 또는 컴파일 단계에는 Type Checking 을 하지 않는다.

```
public class MouseFactoryReflection {
   public Object getInstance(String className) {
      try {
         Class cls = Class.forName(className);
         Constructor constructor = cls.getConstructor();
         return constructor.newInstance();
      } catch ( ClassNotFoundException cce ) {}
        catch ( NoSuchMethodException nme) {}
        catch ( InvocationTargetException ite ) {}
        catch ( InstantiationException ie ) {}
        catch ( IllegalAccessException iae ) {}
       return null;
   }
}
```

그림 9. Checked Exception 예

대신, 실행시 발생할 수 있는 Exception 을 처리하기 위한 try/catch 문을 추가해야 한다.

자바가 제공하는 Exception 의 종류에는 Checked Exception, Run-time Exception 그리고 error 가 있다. 이중 Checked Exception 의 경우, 위와 같이 컴파일 단계에 catch 하거나 throw 해야만 컴파일 오류가 발생하지 않는다. 이는 Checked Exception 은 예외 상황에서 프로그램적으로 복구할 수 있는 방법이 존재하는 경우를 위한 Exception 이기 때문이다.

하지만 Reflection 사용에 따른 Exception 으로부터 복구할 수 있는 상황이란 거의 없다. 예를 들어, 위의 Class Not Found Exception 이 발생한다면, 이는 클라이언트가 잘못된 클래스 명을 넘겨주었거나 또는 해당 클래스가 없을 2 가지 경우다. 이는 모두 프로그램 에러 상황으로 오히려 Run-time Exception 에 가깝다.

결국, Reflection 을 사용함으로써, 개발단계에서 조기에 발견될 수도 있었을 프로그램 에러들이 런타임시에 발생하게 되는 것이다. 이처럼 에러 상황이 늦게 발견하면 디버깅이 어려워지게 된다. 이와 같은 경우를 위해서, 런타임시 디버깅을 위해 상세한 에러 메시징 기능을 포함하는 것이 좋다.

Drug heals the pain, Overdose kills the gain

Reflection 의 사용에 관한 사실들은 사실 사실이 아니다. 성능, 디버깅, 그리고 복잡성과 관련된 내용들은 잘못 사용된 예에서 파생한 오해들이다.

Reflection 은 염려할 만큼의 성능저하를 가져오지 않는다. 대량의 if/else 문이나 switch 문 대신, 잘 설계된 Reflection 은 객체지향 철학을 어기지 않으면서도 더 좋은 성능을 발휘할 수도 있다.

또한 디버깅이 어려운 것은 컴파일 단계에 처리할 수 있는 오류들이 실행 단계에 발생하기 때문이 아니다. 더 근본적인 이유는 Reflection 을 사용할 경우에 발생할 런타임 에러 메시지를 최대한 상세하게 그리고 친절하게 표시하도록 Exception

전략을 설계하지 못했기 때문이다. 잘 설계된 Exception 처리 전략은 Reflection 뿐만 아니라 시스템의 전체적인 디버깅을 쉽게 만든다.

Reflection 의 복잡성은 사실 개발자 개개인의 초점에 맞추었을 때의 얘기다. 하지만, Reflection 의 사용은 개발자의 관점이 아닌 아키텍트 중심으로 설계되어야 한다. Reflection 이 가장 유용한 곳이 바로 시스템의 아키텍처를 이루는 컴포넌트들이기 때문이다. 오히려 잘 설계된 Reflection 이 제공하는 서비스는 개발을 더욱 단순화 한다.

하지만 지나친 사용은 화를 부를 수 있다. Reflection 을 적절히 사용했고 많은 이득이 따른다 하더라도, 더 간단한 해결책이 존재한다면, Reflection 을 사용하지 말 것을 권한다. 단순한 해결책은 언제 어느 경우에나 최상의 선택이다.

Reflection 의 사용은 양날의 검과 같다. 잘 사용한다면, 이름을 불러주었을 때, 여러분의 꽃이 되어 줄 것이다.

References

1. Glen McCluskey. Using Java Reflection
2. The Java Tutorials. Trail: The Reflection API
3. Joshua Bloch. “Effective Java Programming Language Guide,” Addison-Wesley(2001), pp.158~160, pp.162~164.
4. Yuval A. Java Reflection Performance
5. Dennis Sosnoski, Java Programming dynamics, Part 2: Introducing reflection
6. 로드 존슨 자바 유저스 번역팀 역. “expert
7. one-on-one J2EE 설계와 개발,” 정보문화사(2004), pp.177~185, pp.786~787.
8. Cay S. Hortsmann, Gary Cornell. “Core Java Volume I: Fundamentals” 8th ed., Prentice Hall(2008), pp.217~240.
9. Apache Commons, Commons BeanUtils
10. Guide to Features – Java Platform, Java Object Serialization Specificatoin