네트웍 코드를 쉽게 테스트하자

중앙대학교 컴퓨터공학과
자바 동호회
JSTORM
http://www.jstorm.pe.kr

Document Information

Document title:	네트웍 코드 테스팅 하기
Document file name:	NetworkCodeTest_JYPark_ 3_final.doc
Revision number:	<1.0>
Issued by:	<JSTORM>
Issue Date:	<2002/09/04 >
Status:	Final

Content Information

Audience	자바 중 , 고급
Abstract	네트웍 작업을 포함하는 코드를 테스팅 하는 방법에 대한 설명.
Reference	(http://www.javaworld.com/javaworld/jw-07-2002/jw-0719-networkunittest_p.html) 를 번역
Benchmark information	

Table of Contents

네트웍 코드를 쉽게 테스트하자

이 문서는 자바월드에 올라온 유닛 테스트에 대한 글입니다. XP 에서는 TDD(Test Driven Development) 라고 하여 테스팅이 중요하다고 부르짖고 있고 Junit 이라는 멋진 테스팅 프레임웍을 만들었습니다. 그러나 Junit은 J2EE쪽에 적용하기에는 어려움이 많고 그 대안으로 cactus나 httpunit 같은 server-side 테스팅 프레임웍도 나와있습니다. 이 문서에서는 프로그램 내부에서 네트웍 작업을 하는 경우 쉽게 테스트 할 수 있는 방법을 제안하고 있습니다.

네트웍 코드 테스팅을 개선하는 두가지 테크닉

(여기서 말하는 네트웍 코드란 말 그대로 프로그램 내부에 다른 서버에 존재하는 프로그램과 통신하는 네트웍 작업이 필요한 코드를 말합니다.)

요약

테스트 슈트(test suite)는 단일 프로세스 상에서 최적으로 동작하므로 네트웍코드는 테스트 하기가 힘들다. Nelson Minar는 네트웍 코드 유닛 테스팅을 위해 2가지 테크닉을 설명한다. 첫번째는 코드를 설계할때 가능한 네트웍 독립적인 로직으루 분할하도록 제안한다. 두번째는 자바의 프로토콜 핸들러 클래스들을 사용해서 실제 네트웍을 거치지 않고 네트웍을 시뮬레이션 하는 방법을 제안한다. 이런 두가지 테크닉을 사용한다면 테스트하기 쉬운 네트웍 코드를 짤 수 있다. 이 문서에는 자바 프로토콜 핸들러의 샘플코드, PrintRSS 데모 프로그램과 테스트 슈트를 포함한다.

좋은 유닛 테스트 슈트는 매번 컴파일 한 후 실행될수 있도록 속도가 빨라야 하고 어떠한 에러라도 잡아낼 수 있도록 신뢰성이 있어야 한다. 그러나 네트웍코드(예를들어 URL에서 읽기 작업을 하는 코드)는 빠르고 견고한 테스트를 하기가 어렵다. 게다가 테스트슈트 내부에서 네트웍 호출이 일어난다면 다른 서버나 네트웍 상태에 따라 느리게 실행되고 신뢰할 수 없게 될것이다.(네트웍 프로그램은 외부 환경에 영향을 많이 받는다.)

웹에서 XML 데이타를 다운받고 포맷팅하고 보여주는(display) 프로그램을 생각해보자. 이런 프로그램을 테스트하기 위해서는 XML 데이타를 보내줄 수 있는 웹서버가 필요할 것이다. 그러나 프로그램의 많은 부분 - XML파서 , formatter , displayer - 은 네트웍이랑 상관없이 테스트 할 수 있다. 이런 생각을 바탕으로 테스트시 네트웍 사용을 피해서 네트웍 코드를 테스트하는 두가지 테크닉을 소개할 것이다. 코드 샘플은 Resources 에서 다운 받을 수 있다.

간단한 네트웍 데모 프로그램인 PrintRSS에 대한 설명을 시작으로 네트웍 커넥션 대신 간단한 Reader와 Writer 객체로 테스팅을 쉽게 할 수 있도록 PrintRSS를 설계하는 방법에 대해 얘기할 것이다.

주의 : 모든 테스트는 Junit 테스팅 프레임웍의 assert() 메소드를 사용한다.

PrintRSS : 데모 프로그램

PrintRSS는 URL에서 데이터를 읽어 처리하는 단순하지만 훌륭한 네트웍 코드 테스팅 데모프로그램이다. PrintRSS는 RSS(RDF Site Summary)형식으로 데이터를 읽는다.

RSS포맷이란 뉴스를 배포할 목적으로 사용되는 단순한 XML 데이타 포맷이다.

이 문서에서 사용하는 RSS 구조는 다음과 같다.

```
<rss><channel>
  <title>Channel Title</title>
  <item><title>Item 1</title></item>
  <item><title>Item 2</title></item> ...
</channel></rss>
```

PrintRSS는 ULR로부터 RSS문서를 다운 받고 내용을 포맷팅하고 title을 System.out으로 출력한다.

Channel Title
Item 1
Item 2

PrintRSS는 4가지 주요 오퍼레이션을 수행한다.

1. Opens a connection to the URL (URL에 커넥션을 연다)
2. Reads in the XML (XML 데이타를 읽는다)
3. Formats the data (읽어온 XML 데이타를 포맷팅한다)
4. Writes to System.out (System.out 으로 출력한다)

PrintRSS 프로그램은 4가지 오퍼레이션을 printURL(URL) 이라는 하나의 메소드로 감춘다. 그러나 이 메소드를 테스트하는 것은 두가지 이유로 어렵다. 첫번째로 이 코드는 URL에서 데이터를 로드하는데 의존한다. URL이 http: URL 이라면 네트웍 사용을 포함한다. 그리고 이 코드는 System.out으로 출력하는 side effect를 숨기고 있다. 이런 상황에서 테스트를 쉽게 하기 위해 PrintRSS를 어떤 방법으로 설계할 수 있을까?

데이타를 캡슐화하는 reader와 writer를 사용하라

PrintRSS의 대부분의 로직은 네트웍에 연결하는 로직이 아니라 단순히 XML을 파싱하고 포맷팅 하는 것이므로 코드를 분리해서 독립적으로 테스트 할 수 있다. 코드를 분리하는 작업은 지루할 수도 있지만 이런 노력을 기울여 모듈화가 증가되고 테스트 가능한 코드가 된다면 훨씬 더 좋은 코드가 결과로 나타난다.(일종의 refactoring)

이런 마음가짐으로 printURL()의 코드파싱 부분과 포맷팅 부분을 새로운 formatReader(Reader , Writer) 메소드로 분리할 수 있다. formatReader 메소드는 XML 데이타를 가진 Reader객체를 가지고 XML데이터를 파싱하고 Writer에게 쓰는 메소드이다.

formatReader(Reader, Writer)를 테스트하는 것은 간단하다.

```
testFormatReaderGoodData():
  String goodRSSData = "<rss><channel>" +
                       "<title>Channel Title</title>" +
                       "<item><title>Item 1</title></item>" +
                       "<item><title>Item 2</title></item>" +
                      "</channel></rss>";
  String goodRSSOutput = "Channel Title\n  Item 1\n  Item 2\n";

  Reader r = new StringReader(goodRSSData);
  Writer w = new StringWriter();
  PrintRSS.formatReader(r, w);

  assertEquals(goodRSSOutput, w.toString());
```

위의 예제는 URL 이나 네트웍 커넥션 없이 reader와 writer로 포맷터와 파서의 로직을 테스트한다. 이 테스트 예제에서 유용한 테스트 테크닉 하나가 숨어있다. 테스트 데이터를 가진 reader 스트림을 생성하는 것은 파일이나 네트웍에서 데이터를 읽어오는 것보다 테스트시에 유용하다.

StringReader와 StringWriter(혹은 ByteArrayInputStream과 ByteArrayOutputStream)은 유닛 테스트 슈트에서 테스트 데이터를 포함하는 것이 효율적이라는 것을 증명한다.

유닛 테스트는 모든 로직이 정상적으로 동작할 때 발생하는 일뿐 아니라 무언가가 잘못됐을 때 발생하는 에러 핸들링 코드를 테스트하는 것도 중요하다.(예외가 발생한 경우 예외 처리가 제대로 되는지도 테스트 해 보아야 한다.) 다음 예는 Junit을 사용해서 잘못된 데이터인 경우 예외를 제대로 던지는지를 테스트하는 예제이다.

```
testFormatReaderBadData():
  String badXMLData = "this is not valid xml data";
  StringReader r = new StringReader(badXMLData);

  try {
    PrintRSS.formatReader(r, new StringWriter());
    fail("should have thrown XML error");
  } catch (XMLParseException ex) {
    // No error, we expected an exception
  }
```

역시 reader와 writer가 데이터를 캡슐화 하지만 중요 차이점은 이 예제에서 사용한 데이터는 formatReader에서 예외를 던진다는 점이다. 만약 예외가 던져지지 않는다면 JUnit의 fail() 메소드가 호출된다.

자바 프로토콜 핸들러를 사용한 네트웍 코드 테스트

네트웍작업이 없는 메소드로 분리하는 것은 PrintRSS 처럼 프로그램을 테스트하기 쉽게 만들지만 이런 노력은 전체 프로그램을 테스트 하는 데에는 불충분하다. 우리는 여전히 네트웍 예외 처리를 포함한 네트웍 코드를 테스트하길 원한다. 때때로 reader 인터페이스를 분리하는 것은 불편하다.

이번절에서는 URL을 가지고 RSS를 읽어들이고 writer에 데이터를 쓰는 formatURL(URL , Wirter) 메소드를 테스트하는 방법에 대해 설명한다.

테스트 코드는 여러 방법으로 URL을 포함할 수 있다. 첫번째로 표준 http: URLs 과 테스트 서버를 사용할 수 있다. 그러나 이런 방법은 테스트 셋팅을 어렵게 하는 콤포넌트로 고정된 웹서버를 필요로 한다.(테스트 내용에 웹서버 셋팅 부분이 들어가야 한다.) 두번째로 file:URLs 과 로컬 파일을 사용할 수 있다. 이러한 접근 방법은 http: URLs 와 같은 문제점을 가진다.(file: 혹은 http: URLs로 테스트 데이터에 접근할 수 있지만 I/O 예외를 발생시키는 네트웍 장애를 시뮬레이션 하기는 불편하다.

URL 코드를 테스트하는 좀 더 나은 방법은 테스트 프로그램이 제어할 수 있는 완전하고 새로운 URL 네임스페이스(testurl: 같은)를 생성하는 것이다.(자바 프로토콜 핸들러를 사용하면 쉽게 할 수 있다.)

testurl 구현하기

자바 프로토콜 핸들러는 프로그래머가 고유한 코드로 각자의 URL 프로토콜을 구현하도록 해준다. Brian Maso의 "A New Era for Java Protocol Handlers" (Sun Microsystems, August 2000) 에서 자바 프로토콜 핸들러에 대한 설명과 소스 코드의 org.monkey.nelson.testurl 패키지에서 예제 구현체를 찾을 수 있다.

프로토콜 핸들러는 간단하다.

첫번째로 프로토콜 핸들러에 쓰기 작업을 하려면 다음 3가지 조건을 만족해야 한다.

- TestURLConnection class : 인풋 스트림과 헤더 등을 리턴하는 실제 메소드를 핸들링 하는 URLConnection 구현체

- Handler class : testurl: URL을 TestURLConnection 으로 변경해준다

- java.protocol.handler.pkgs 속성 : 새로운 URL 네임스페이스 구현체를 찾을수 있도록 자바에게 알려주는 시스템 속성

두번째로 유용한 TestURLConnection 구현체를 제공한다. 이 경우 실제 TestURLConnection 은 라이브러리 사용자에게 숨겨진다. 대신 작업은 헬퍼 클래스인 TestURLRegistry 를 거쳐 실행된다. 이 클래스는 InputStream 과 URL을 연결해주는 TestURLRegistry.register(String , InputStream) 라는 하나의 메소드를 가진다.

```
InputStream inputIS = ...;
TestURLRegistry.register("data", inputIS);
```

URL testurl:data를 연다는 것은 inputIS 라는 input stream을 리턴한다. (URL 입력을 testurl:data로 하면 자바 프로토콜 핸들러가 알아서 inputIS로 연결시켜 준다.) 이 registry를 사용하면 테스트 프로그램에서 출력을 제어하는 URLs 를 쉽게 생성할 수 있다. (일반적으로 input stream 데이터는 복사되는게 아니므로 각 URL 을 한번만 사용할 수 있다.)

올바른 데이터로 테스트하기

첫번째 테스트 -> testurl: 을 사용하여 올바른 입력 데이터로 formatURL() 메소드를 테스트한다.

```
testFormatURLGoodStream():
  InputStream dataIS = new
ByteArrayInputStream(goodRSSData.getBytes());
  TestURLRegistry.register("goodData", dataIS);
  URL testURL = new URL("testurl:goodData");

  Writer w = new StringWriter();
  PrintRSS.formatURL(testURL, w);
  assertEquals(goodRSSOutput, w.toString());
```

이전 테스트에서 사용한 것과 같은 테스트 데이터를 사용한 위의 코드는 testurl:goodData URL과 연결된다. formatURL() 메소드는 URL을 열고 데이터를 읽으므로 테스트 코드는 데이터가 올바르게 읽혀지는가 , 포맷팅 되는가 , Writer로 출력하는가를 테스트할 수 있다.

장애 시뮬레이션 하기

네트웍을 사용하는 프로그램을 작성하기는 쉽지만 네트웍 장애를 적절히 핸들링하는 코드를 작성하는 것은 어렵다. testurl: 은 네트웍 장애를 시뮬레이션 할 수 있다.

일반적으로 외부 웹서버가 작동하지 않을 때 발생하는 URL에 대한 연결이 불가능한 장애를 시뮬레이션 해보자. 우리가 사용하는 testurl: 라이브러리는 testurl:errorOnConnect 같은 URL에 커넥션을 여는 순간 IOException을 발생시키는 특별한 URL을 포함할 수 있다. 이런 URL을 사용해서 URL을 열 수 없는 경우 발생하는 예외를 프로그램이 적절히 핸들링 할 수 있는가를 테스트할 수 있다.

```
testFormatURLNoConnection():
  URL noConnectURL = new URL("testurl:errorOnConnect");
  Writer w = new StringWriter();
  try {
    PrintRSS.formatURL(noConnectURL, w);
    fail("Should have thrown IO exception on connect.");
  } catch (IOException ioex) {
  }
```

위의 테스트는 커넥션이 실패했을 때 formatURL()이 IOException을 던진다는 것을 보장한다. (URL에 대한 커넥션을 열 수 없는 경우 IOException이 발생되는데 위의 경우는 특별한 errorOnConnect 라는 URL을 생성하여 이 URL을 열려고 시도하면 무조건 IOException을 던진다.) 그러나 네트웍 커넥션은 동작하고 트랜잭션이 발생하는 동안 실패한 경우는 어떻게 테스트할 것인가?(쉽게 말해서 연결은 됐는데 중간에 끊어진 경우를 어떻게 테스트 케이스로 만들수 있을까?)

test URL로 연결된 input stream을 제어할 수 있기 때문에 어떠한 input stream 구현체라도 사용할 수 있다. 예를 들어 기본적인 stream을 랩핑해서

특정한 숫자의 바이트만 읽는 read() 메소드를 가진 간단한 BrokenInputStream 클래스를 생성 할 수 있다.

```
testFormatURLBrokenConnection():
  InputStream dataIS = new
ByteArrayInputStream(goodRSSData.getBytes());
  InputStream testIS = new BrokenInputStream(dataIS, 99);
  TestURLRegistry.register("brokenStream", testIS);
  URL testURL = new URL("testurl:brokenStream");

  Writer w = new StringWriter();
  try {
    PrintRSS.formatURL(testURL, w);
    fail("Should have thrown IO exception on read.");
  } catch (IOException ioex) {
  }
```

위의 코드에서 testurl:brokenStream URL은 99 바이트의 올바른 데이터를 리턴하는 input stream과 연결되어 예외를 던진다. 이 경우 예외가 던져지는가를 테스트 할 수 있다. 좀 더 복잡한 프로그램에서 처음 99 바이트를 성공적으로 읽은 후 어떤 작업을 수행했는지를 테스트할 수도 있다. (특정 숫자만큼 데이터를 읽은 후 어떠한 이벤트를 발생시키는 프로그램을 테스트 해야 된다면 이러한 방법으로 테스트 할 수 있을 것이다.)

네트웍 테스팅 라이브러리 확장하기

이 문서에서 네트웍 코드를 테스팅하는 간단한 테크닉을 보였다. 첫번째는 테스트할 수 있도록 네트웍과 분리된 로직으로 코드를 분할하라는 것이고 두번째는 네트웍 장애를 시뮬레이션 할 수 있는 테스트용 URL 네임스페이스와 input stream을 제공하는 간단한 유틸리티 클래스를 사용하는 것이다. 이런 테크닉은 네트웍을 사용하는 코드를 좀 더 쉽게 테스트 할 수 있도록 해 준다.

그러나 이러한 방법은 한계가 있다. Testurl: 라이브러리는 매우 간단하다. TestURLConnection 은 URL에 쓰기(HTTP POST 오퍼레이션을 수행할 필요가 있을 경우)와 헤더(HTTP MIME 타입을 읽어올 필요가 있을 경우)를 지원하지 않는다. 복잡한 URLConnection 구현체는 복잡한 URL 인터액션(웹서비스에서 SOAP call 과 같은) 을 테스트할 수 있을 것이다.

게다가 위의 방법은 실제 네트웍 연결이 아닌 네트웍에 대한 시뮬레이션일뿐이다. 강건한(훌륭한 , 좋은 , 땐땐한) 인터넷 클라이언트는 느린 커넥션 , hung socket , 등과 같은 많은 예기치못한 행동에 대응할 수 있다. 좀 더 복잡한 InputStream 구현으로 이러한 장애를 시뮬레이션 할 수 있지만 모든 네트웍 코드에 대한 테스팅은 실제 소켓에서만 할 수 있다. 그러나 이 문서에서 설명하는 테크닉은 네트웍 프로그램에서 발생할 수 있는 대부분의 상황을 테스트하기에 충분하다.

저자 소개

Nelson Minal는 구글의 소프트웨어 엔지니어로 6년의 자바프로그래밍 경험을 가지고있다. 상업적인 용도로 사용되는 첫번째 분산 컴퓨팅 시스템인 Popular Power , 모바일 에이전트 시스템 , Hive 를 포함해서 자바로 여러 번의 복잡한 네트웍 시스템을 구축한 경험이 있다. Nelson이 유닛 테스트 주의자가 된지 2년이 됐으며 코드가 제대로 테스트 되야지만이 기분 좋게 잠이 든다고 한다.

Resources

To download the sample code package containing the PrintRSS program and a library to handle testurl: connections, go to:
http://www.javaworld.com/javaworld/jw-07-2002/networkunittest/jw-0719-networkunittest.zip

- Read Brian Maso's "A New Era for Java Protocol Handlers" (Sun Microsystems, August 2000) for more details on installing a custom protocol handler:
  http://developer.java.sun.com/developer/onlineTraining/protocolhandlers
- Learn more about JUnit, the unit-testing framework used for this article:
  http://www.junit.org
- Learn more on NanoXML, the ultra-small XML parser used for this article:
  http://nanoxml.sourceforge.net/
- To learn more about the RSS news syndication format, read Rael Dornfest's "RSS: Lightweight Web Syndication" (O'Reilly xml.com,

July 2000):
http://www.xml.com/pub/a/2000/07/17/syndication/rss.html

- Recent *JavaWorld* articles about software testing:
  - "Implement Design by Contract for Java Using Dynamic Proxies," Anders Eliasson (February 2002)
  - "Exceptions: Don't Get Thrown for a Loss," Tony Sintes (February 2002)
  - "Overpower the PreparedStatement," Bob Byron and Troy Thompson (January 2002)
- For more software testing articles, visit the **Testing** section of *JavaWorld*'s Topical Index:
  http://www.javaworld.com/channel_content/jw-testing-index.shtml
- Talk about testing in our **Programming Theory & Practice** discussion:
  http://forums.idg.net/webx?50@@.ee6b806
- Sign up for *JavaWorld*'s free weekly email newsletters:
  http://www.idg.net/jw-subscribe
- You'll find a wealth of IT-related articles from our sister publications at IDG.net