Open API Client 개발

2012-10-22

정상혁

목차

1. 예제
2. 개발 원칙
3. 라이브러리

1. 예제

1.1 예제 위치

1.2 공통 모듈

1.3 검색 API

1.4 단축 URL API

1.1 예제 위치

SVN, Maven, Eclipse 활용

- SVN
  - https://dev.naver.com/svn/naverapis/trunk/naver-java-client-samples/
  - ID : anonsvn
  - Password : anonsvn
- Maven 프로젝트
- Checkout후 mvn eclipse:eclipse로 이클립스 프로젝트 생성

1.2 공통 모듈

URL 연결 담당 클래스

- ResourceConnector.java : GET 요청 공통 interface
- SimpleConnector.java : JDK의 URLConnection 활용
- HttpClient3Connector.java: Apache HttpClient 3.x 활용
- HttpClient4Connector.java: Apache HttpComponent 4.x 활용

1.3 검색 API

RSS 라이브러리 ROME 활용

- SearchApiClient.java
- SearchApiClientTest.java
  - 파일로 파싱 테스트
- SearchApiClientIntegrationTest
  - 실제 API 서버와 연결해서 테스트

1.3 검색 API

RSS 라이브러리 ROME 활용

- Spring RestTemplate + ROME 활용
  - NaverSearchHttpMessageConverter.java
    - 메시지 파싱 모듈
  - NaverSearchRestTemplateTest.java

1.4 단축 URL API

JAXB2 활용

- ShortUrlApiClientJaxbImpl.java
- ShortUrlApiClientJaxbImplTest.java
  - 파일을 이용해서 파싱만을 테스트
- ShortUrlApiClientJaxbImplIntegrationTest.java
  - 실제 서버와 연결 테스트

1.4 단축 URL API

JacksonJson Object mapping 활용

- ShortUrlApiClientJsonMapperImpl.java
- ShortUrlApiClientJsonMapperImplTest.java
- ShortUrlApiClientJsonMapperImplIntegrationTest.java
- LinkProcessorJsonMapperImpl.java
  - API의 결과 값과 매핑하는 목표 객체의 형태가 다를 때
- LinkProcessorJsonMapperImplTest.java
- ShortUrlRestTemplateTest
  - Spring rest template 활용

1.4 단축 URL API

JacksonJson Streaming 활용

- LinkProcessorJsonStreamImpl.java
- LinkProcessorJsonStreamImplTest.java

2. 개발 원칙

2.1 어느 클라이언트 코드

2.2 예외에 대비하라

2.3 성능을 높이고 메모리를 아껴라

2.4 쓰레드 안정성을 늘 의식해라

2.5 모듈별 역할과 책임을 구분하라

2.6 다시 보는 어느 클라이언트 코드

2.1 어느 클라이언트 코드

> 무엇을 걱정해야 할까?

```
public  User findUser(String userId) {
    String response = HttpUtils.callGet(API_URL + userId);
    String username = ParseUtils.getTagValue(response, "username");
    String address = ParseUtils.getTagValue(response, "address");
    return new User(userId, username, address);
}
```

2.2 예외에 대비하라

Timeout을 설정하라

- 전체 Application의 스레드 풀을 고갈시킬 수도 있다.
  - (501 error)
- 설정 예
  - Tomcat max thread가 500개, max TPS(Transaction per second)가 100일 때 API timeout을 4초로 설정

2.2 예외에 대비하라

Timeout을 설정하라

- JDK URLConnection

```
public InputStream open(String url) throws IOException {
    URLConnection con = new URL(url).openConnection();
    con.setConnectTimeout(this.connectTimeoutMilsec);
    con.setReadTimeout(this.readTimeoutMilsec);
    return con.getInputStream();
}
```

2.2 예외에 대비하라

Timeout을 설정하라

- Apache HttpClient 3.x

```
HttpConnectionManagerParams params = new HttpConnectionManagerParams();
params.setConnectionTimeout(connectTimeoutMilsec);
params.setSoTimeout(readTimeoutMilsec);
params.setDefaultMaxConnectionsPerHost(100);
params.setMaxTotalConnections(100);

connManager = new MultiThreadedHttpConnectionManager();
connManager.setParams(params);
this.httpClient = new HttpClient(connManager);
```

2.2 예외에 대비하라

Timeout을 설정하라

- Apache HttpComponent 4.x

```
connManager = new ThreadSafeClientConnManager();
connManager.setMaxTotal(100);
connManager.setDefaultMaxPerRoute(100);

HttpClient client = new DefaultHttpClient(connManager);
client.getParams().setIntParameter(CoreConnectionPNames.CONNECTION_TIMEOUT, connectTimeoutMilsec);
client.getParams().setIntParameter(CoreConnectionPNames.SO_TIMEOUT, readTimeoutMilsec);

this.httpClient = client;
```

2.2 예외에 대비하라

예외 상황을 테스트하라

- 예외메시지 파싱을 테스트하라.
  - 예외 메시지 파일을 따로 저장해서 테스트
  - Text based protocol의 장점을 활용
- 통합 테스트도 자동화하면 도움이 된다.
  - 매일 돌리다보면 정기 점검일 때, 서버 상태가 이상할 때를 만날 수도 있다.
  - 때로는 스펙에 명시되지 않은 동작까지 알 수 있다.

2.3 성능을 높이고 메모리를 아껴라

불필요한 중간 객체를 생성하지 마라

- 과도한 로깅
- 파싱 모듈에서도 Stream을 바로 활용해라.

2.3 성능을 높이고 메모리를 아껴라

명시적으로 DOM을 쓰지 마라

- Event based인 SAX에 비해서 비효율적.
- XStream은 Stream based의 처리
- JAXB는 보다 high level의 추상화 계층
- SAX, StAX(Streaming API for XML) 등 활용 가능

2.3 성능을 높이고 메모리를 아껴라

Connection Pool을 시도해라

- Apache HttpClient 3.x
  - MultiThreadedHttpConnectionManager 활용
  - 'DefaultMaxConnectionsPerHost' 값 주의
  - Default는 2. 성능에 악영향.
  - Concurrent user * 2 값이 바람직
- Apache HttpComponent 4.x
  - ThreadSafeClientConnManager 활용
  - defaultMaxPerRoute 속성의 default값 주의

2.3 성능을 높이고 메모리를 아껴라

여러 API를 함께 쓴다면 병렬 실행을 고려해라

- JDK concurrent API (ExecutorService)
- Apache HttpClient의 HttpAsyncClient
- Jetty HttpClient
- 얼마만큼 효과가 있을지는 해봐야 안다.

2.4 쓰레드 안정성을 늘 의식해라

쓰레드 안전하지 않은 객체를 파악하라

- HttpClient 3.x
  - MultiThreadedHttpConnectionManager를 사용하지 않을 때의 HttpClient
  - HttpMethod, HttpState, HostConfiguration
- HttpComponent 4.x
  - HttpGet, HttpPost

2.4 쓰레드 안정성을 늘 의식해라

쓰레드 안전하지 않은 객체를 파악하라

- JAXB2
  - Marshaller, Unmarshaller
- JacksonJson
  - JsonParser

2.4 쓰레드 안정성을 늘 의식해라

> **쓰레드 안전한 객체는 매번 생성하지 말라.**

- Apache HttpClient 3.x
  - MultiThreadedHttpConnectionManager를 사용할 때의 HttpClient
- Apache HttpComponent 4.x
  - DefaultHttpClient
  - SingleClientConnManager, ThreadSafeClientConnManager

2.4 쓰레드 안정성을 늘 의식해라

> **쓰레드 안전한 객체는 매번 생성하지 말라.**

- JAXB2
  - JAXBContext
- JacksonJson
  - Thread-safe after configuration : ObjectMapper, JsonFactory

2.4 쓰레드 안정성을 늘 의식해라

쓰레드 안전한 객체는 매번 생성하지 말라.

- 사례 : Spring의 RestTemplate을 매번 생성한다면?
  - RestTemplate의 기본 생성자는 StringHttpMessageConverter를 생성
  - StringHttpMessageConverter는 생성자에서 encoding을 위해 시스템이 지원하는 character set을 확인하게 됨
  - charsets.jar 파일 안의 객체를 동적 로딩하게 되는데, 동적 로딩을 하는 jdk 코드 내 synchronized로 감싼 코드로 인해 locking
  - 길지 않은 Lock구간이지만 대량 요청 시에는 문제가 됨

2.4 쓰레드 안정성을 늘 의식해라

> **쓰레드 안전한 객체는 매번 생성하지 말라.**

- 사례 : Spring의 RestTemplate을 매번 생성한다면?
  - Thread dump

```
at java.nio.charset.Charset$1.getNext(Charset.java:317)
at java.nio.charset.Charset$1.hasNext(Charset.java:332)
at java.nio.charset.Charset$4.run(Charset.java:551)
at java.security.AccessController.doPrivileged(Native Method)
at java.nio.charset.Charset.availableCharsets(Charset.java:546)
at org.springframework.http.converter.StringHttpMessageConverter.
(StringHttpMessageConverter.java:52)
```

2.4 쓰레드 안정성을 늘 의식해라

라이브러리를 늘 의심해라

- 문서에 스레드 안정성에 대한 언급이 없으면 안전하지 않다고 가정해라.
- 먼저 검색해봐라.

2.4 쓰레드 안정성을 늘 의식해라

라이브러리를 늘 의심해라

- 사례 : XStream 1.3.1의 버그
  - 'Infinite loop due to unsafe collection usage'
    - http://jira.codehaus.org/browse/XSTR-584
  - WeakHashMap을 Threadsafe하지 않게 접근
  - 무한루프 -> CPU 100%
  - 네할렘 서버 교체 시기에 나타남
  - 1.4.0에서 패치됨

2.5 모듈별 역할과 책임을 구분하라

추상화 계층을 활용해라

- 구현체를 갈아 끼울 수 있는 계층 활용
- 예) Spring OXM
- 직접 정의한 추상화 계층
- 여러 프로젝트에서 공유될 수 있도록 해라

2.5 모듈별 역할과 책임을 구분하라

통신모듈과 파싱 모듈을 구분해서 구현해라.

- 변화 대응 속도가 빨라진다.
  - 예) 통신 모듈 사용 라이브러리를 바꿀 때.
    - Apache HttpClient 3.x => HttpComponent 4.x
- 테스트 용이성
  - 예) 예상되는 결과를 파일에서 읽어서 파싱 모듈만을 테스트하기

2.6 다시 보는 어느 클라이언트 코드

무엇을 걱정해야 할까?

```
public  User findUser(String userId) {
    String response = HttpUtils.callGet(API_URL + userId);
    String username = ParseUtils.getTagValue(response, "username");
    String address = ParseUtils.getTagValue(response, "address");
    return new User(userId, username, address);
}
```

비정상적인 응답은 어떻게 테스트할까?

String response대신 Stream으로 넘길 수도 있지 않을까?

ParseUtils에서는 반복적으로 처음부터 문자열을 검색하지는 않을까?

3. 라이브러리

3.1 Java URLConnection

3.2 Apache HttpClient 3.x

3.3 Apache HttpComponent 4.x

3.4 Jetty HttpClient

3.5 Spring RestTemplate

3.1 Java URLConnection

기본 JDK 포함

- 의존성 추가가 없음.
- HTTP의 용어와 직관적으로 대응되지는 않음.
  - POST 요청
    - URLConnection.setDoOutput(true);
  - Header 설정
    - URLConnection.setRequestProperty("Content-Type", "text/plain");

3.2 Apache HttpClient 3.x

많이 쓰였던 라이브러리

- 레거시 코드에 많이 보임
- Dependency

```
<dependency>
    <groupId>commons-httpclient</groupId>
    <artifactId>commons-httpclient</artifactId>
    <version>3.1</version>
</dependency>
```

3.3 Apache HttpComponent 4.x

3.x의 새 얼굴

- API 대거 개선. 3.x대와 호환성이 없음
- Dependency

```
<dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpclient</artifactId>
   <version>4.1.3</version>
</dependency>

<dependency>
   <groupId>org.apache.httpcomponents</groupId>
   <artifactId>httpasyncclient</artifactId>
   <version>4.0-beta3</version>
</dependency>
```

3.3 Apache HttpComponent 4.x

3.x의 새 얼굴

- 명확한 Thread safety 문서화

```
@NotThreadSafe
public class HttpGet extends HttpRequestBase {

@ThreadSafe
public class DefaultHttpClient extends AbstractHttpClient {

@ThreadSafe
public class SingleClientConnManager implements ClientConnectionManager {
```

3.4 Jetty HttpClient

Async 실행을 기본적으로 지원

- dependency

```
<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-client</artifactId>
    <version>8.1.7.v20120910</version>
</dependency>
```

3.5 Spring RestTemplate

여러 통신 모듈과 파싱 모듈의 추상화 계층.

- 통신 모듈에 Apache HttpClient 3, 4와 URLConnection 사용
- 파싱 모듈에 JAXB, JacksonJson 등의 다양한 기본 구현체 제공
- 확장 가능
- Spring Android에서도 제공
  - Android 버전에 따라서 통신 라이브러리를 권장하는 것으로 알아서 선택해줌.
    - 진저브레드(2.3) 전에는 구글에서 Apache HttpComponents 권장

정리

검증된 라이브러리로 안전하게, 효율적으로, 유연하게

- 안전하게
  - 쓰레드 안정성, Timeout, 예외 테스트
- 효율적으로
  - Stream활용, Connection pool, Async 검토
- 유연하게
  - 역할과 책임 구분, 추상화 계층

정리

검증된 라이브러리로 안전하게, 효율적으로, 유연하게

- 추천 라이브러리
  - Http Component 4.x, Jetty HttpClient
  - Spring RestTemplate
  - JAXB2, JacksonJson