영속성 관리를 위한 오픈 소스 적용 연구

정광선, 백종현
대우정보시스템 정보기술연구팀
ksjung76@disc.co.kr, baegjh@disc.co.kr

Applying Open Source for Managing Persistence

Jung Kwang Sun, Baeg Jong Hyun
Information Technology Team, Daewoo Information Systems Co., Ltd.

요 약

J2EE 어플리케이션에서 영속성을 관리하기 위한 데이터 계층에 많은 기술들이 제공되고 있다. 특히 ORM(Object Relation Mapping)을 위해 비영리적이면서도 성숙도가 높은 기술들이 많은 관심을 받고 있다. 그러나 중대형 규모의 어플리케이션에서는 아직 활발하게 적용되고 있지 못한 것이 현실이다. 이것은 새로운 기술 적용에 대한 위험 부담과 기술 자체에 대한 신뢰성이 아직 따라 주지 못하고 있기 때문이다.

논문에서는 대표적인 오픈 소스 ORM 기술인 iBatis와 Hibernate에 대해 알아보고, 적용 전략과 가능성을 살펴본다. 이를 위해 몇 가지 시나리오를 가지고 반응 시간과 메모리 활용을 측정하기 위한 테스팅을 수행하고, 이를 통해 각 기술들이 가지는 장단점을 연구한다.

1. 서 론

이제 웹 어플리케이션에서 3계층의 구조는 물리적이든 논리적이든 일반적으로 받아들여지고 있으며, 각 계층을 지원하기 위한 기술들이 체계적으로 제공되고 있다. 그 중에서 영속성을 관리하기 위한 데이터 계층에도 많은 기술들이 제공되고 있다. JAVA 에서는 데이터베이스에 영향을 덜 받으면서 영속성을 관리하기 위해 JDBC를 표준으로 제공하였으며, 점차 어플리케이션이 대형화 됨에 따라 분산 환경에서의 트랜잭션 처리를 위해 EJB라는 표준을 제공하게 되었다. 그러나 비용대비 효과, 기술적인 복잡성 등 여러 가지 단점들이 일부 들어났으며, 이를 보완 혹은 대체하기 위한 시도가 다양하게 진행되고 있다.

그 한가지 현상으로서 자바 개발자들 사이에서는 비영리적으로 배포되는 오픈 소스의 기술들이 각광을 받고 있다. ORM 이라 하여 객체들을 관계 데이터베이스에 매핑하여 영속성을 관리하기 위한 기술들이 발전해 가고 있다. 그러나 이러한 기술들이 실제로 중대형 기업 어플리케이션에 적용되는 사례는 쉽게 찾아볼 수 없다. 표준이 아니라는 점 외에도 많은 이유들이 있겠으나 새로운 기술적용에 따르는 위험부담과 기술에 대한 신뢰성이 아직은 충분히 확보되지 못하고 있다는 것이 큰 이유 중 하나일 것이다.

따라서 본 논문에서는 대표적인 오픈 소스 ORM[1] 기술 중에서 iBaits와 Hibernate에 대해 몇 가지 시나리오를 가지고 반응 시간과 메모리 활용 측면의 테스트를 수행할 것이다. 이를 통해 일반적인 웹 어플리케이션 개발 시에 활용되는 JDBC와 비교보고 각각의 장단점과 활용 전략을 마련해 볼 것이다.

2. 본 론

2 장에서는 JDBC, iBatis, 그리고 Hibernate 기술을 가지고 개발생산성 및 성능적인 측면에 대해 논의 할 것이다. 그 전에 각 기술에 대해 살펴 본다.

2.1 영속성 관리를 위한 기술

JDBC는 자바로 데이터를 관리하기 위한 가장 일반적인 기술이다. JDBC의 경우 개발되는 모듈의 품질은 개발자의 능력에 크게 의존하게 된다. JDBC API를 통해 기본적인 기능들만을 제공받게 되며 데이터의 캐싱이나 트랜잭션 관리, 리소스의 관리 등이 전적으로 개발자의 몫이 되기 때문이다. 또한 개발되는 코드에는 매번 DB와의 연동을 위한 반복적인 코드들이 들어가게 되며, 특히 객체 모델과의 연동을 위해서 각각의 모델에 데이터를 담기 위해 번거롭고 반복적인 코딩이 수반된다.

iBatis[2][3]는 두 가지 프레임워크로 구성되는데, 그 중에서 DB와의 상호작용 제어에 가까운 프레임워크는 SqlMap 이라는 프레임워크이다. 이것은 자바 소스에서 SQL 문장들을 XML로 이전시키면서 SqlMap 프레임워크의 API를 사용하여 DB와의 상호작용을 지원하게 된다. 즉, DB와의 연동을 위한 어떠한 코드도 소스에 존재하지 않는다. 심지어 SQL 문장에서 사용될 파라미터와 결과값까지도 자바 프리미티브 타입이나 객체, 리스트, 혹은 XML 형태로 다루게 된다. 따라서 반복 코딩이 전혀 없으며, SQL 문 작성에도 유연하게 된다. 또한, SqlMap 프레임워크에서는 결과값 캐싱이나 SQL 문장의 캐싱으로 응답시간을 줄이고 DB 서버에 대한 부하를 상당히 줄일 수 있다. 뿐만 아니라 iBatis는 XML 설정 파일을 통해서 분산 환경에서도 적합하게 운용될 수 있도록 설계되었다.

Hibernate[4][5]는 iBatis 보다 더욱 ORM에 충실한 프레임워크이다. XML 파일에 객체 모델과 DB와의 연관 관계를 명시하여 운용되는 형태로서 DB에 맞는 SQL 문을 최적화 하여 생성시켜 준다. 따라서 기본적인 기능들에 대해서는 SQL문을 작성할 필 요 조차 없으며, 상황에 따라 유연하게 적용할 수 있다. Hibernate 역시 XML 파일의 설정을 통해 분산환경에서의 영속성 관리를 지원할 수 있으며,

1차, 2차 캐싱 및 최적화된 SQL문의 생성, 그리고 SQL 문장의 캐싱으로 개선된 성능의 효과를 기대할 수 있다[4]. iBatis와 마찬가지로 DB와의 연동을 위한 반복적인 코드가 소스에 존재 하지 않게 된다.

2.2 테스트 시나리오

시나리오는 2가지로 나누었다. 하나는 데이터를 입력, 조회, 수정, 삭제하는 일반적인 기능으로 이루어졌으며, 다른 하나는 상당량의(몇 천건 이상) 데이터를 조회하는 기능으로 구성되었다.

시나리오 1. 소규모 CRUD의 동시 접근

a. 임의의 객체 하나 생성하여 한건 입력(insert)
b. 랜덤하게 선택된 ID값을 가지는 객체 한건 조회(select)
c. 랜덤하게 생성된 객체의 정보 수정하여 업데이트(update)
d. 임의로 생성하여 입력한 객체 삭제(delete)

- 대상 테이블에서는 이미 2000건의 데이터가 존재
- 요청은 웹 단에서부터 수신
- 10 명의 동시접근자를 유지하여 반복적으로 데이터를 조작 총400개의 요청이 동일 테이블에 발생(10 명 * 10번 * 4개 요청)
- a,b,c,d 각각의 기능들은 다른 트랜잭션상에 존재

시나리오 2. 대규모 데이터의 동시 조회

- 2000건의 데이터를 조회하며, 기타 조건은 위와 동일

서버 사양은 2.6G 1 CPU, 1GB RAM이며, WAS는 Tomcat 5.0.28 그리고 운영체제는 리눅스를 사용하였다. 시나리오 1은 일반적인 어플리케이션의 데이터 조작의 경우를 고려한 것이며, 시나리오 2는 대량의 데이터를 처리하는 시스템의 경우를 고려한 것이다. 어플리케이션은 전형적인 3계층의 구조를 이루고 있으며, 응답 시간을 측정한 위치는 서버 측에서 요청을 받는 부분에서부터 측정하여, 네트워크 시간의 간섭을 배제하였다.

2.3 테스트 수행 결과 및 분석

표 1,- 시간 단위는 millisecond

표 2,- 시간 단위는 millisecond

표 3에서는 JDBC와 iBatis, 그리고 Hibernate를 이용하여 구현된 DAO(Data Access Object)를 통한 응답시간 측정치를 확인할 수 있다. JDBC와 iBatis는 거의 유사한 형태로 결과가 나왔지만, Hibernate에서는 큰 차이를 보이고 있다. 입력의 경우 나머지 두 개는 1.4초대의 평균시간을 나타내고 있지만, Hibernate의 경우 1ms를 나타내고 있다. 이것은 삽입을 위해 최적화 된 SQL 문장을 작성하였으며 세션 단위의 1차 캐시를 효과적으로 활용하였기 때문에 나타난 결과이다. 또한 조회, 수정, 삭제 시에 JDBC와 iBatis는 2.4초대의 평균 시간을 나타내는 것과 달리 평균 0초대의 시간을 보이는데, 이 역시 최적화 된 SQL 수행을 통해 나온 결과이다.

시간 측정 방법에 있어서 JVM의 시스템 milisecond를 획득하여 측정하여 그 보다 세밀한 측정값들이 나오지 않아서 0초로 체크된 것이므로 시간이 걸리지 않았다고 생각해서는 안되겠다.

iBatis는 SQL문을 XML로 빼 놓은 것이기 때문에 JDBC와 비교할 때 성능적으로 괄목할 만큼의 차이는 보이지 않았지만, Hibernate의 경우 스스로 SQL문을 만들어 최적화 함으로서 단순 CRUD의 수행 시에도 상당한 차이를 보여주는 것을 확인할 수 있다.

표 1. JDBC 모듈을 이용한 시나리오 1 의 응답시간 결과

	삽입	조회	수정	삭제
수행 횟수	100	100	100	100
총 응답시간	145956	246953	241131	253545
평균 응답시간	1459	2469	2411	2535

- 시간 단위는 millisecond

표 2. iBatis 모듈을 이용한 시나리오 1 의 응답시간 결과

	삽입	조회	수정	삭제
수행 횟수	100	100	100	100
총 응답시간	145111	255973	264166	241250
평균 응답시간	1451	2559	2641	2412

- 시간 단위는 millisecond

표 3. Hibernate 모듈을 이용한 시나리오 1 의 응답시간 결과

	삽입	조회	수정	삭제
수행 횟수	100	100	100	100
총 응답시간	187	32	15	15
평균 응답시간	1	0	0	0

- 시간 단위는 millisecond

위의 각 응답 시간 테스트를 하면서 함께 측정한 메모리 사용량은 **오류! 참조 원본을 찾을 수 없습니다.,오류! 참조 원본을 찾을 수 없습니다..**그림 3 에서 확인할 수 있다. 40초에서 시작하여 3분 정도에 모두 수행을 마쳤으며, 대부분 22MB에서 25MB사이의 메모리를 사용하고 있음을 알 수 있다. 비슷한 수준이지만, 25MB를 소모한 Hibernate가 다소 메모리를 더 사용하고 있음을 알 수 있으며, 이는 Hibernate가 제공하는 1차 캐시 등이 영향을 미친 것이다.

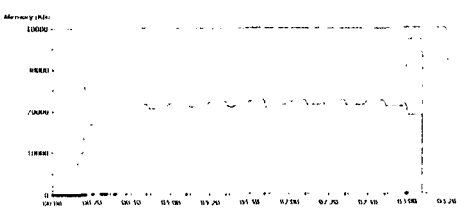

그림 1. JDBC 모듈을 이용한 시나리오 1 메모리 사용량

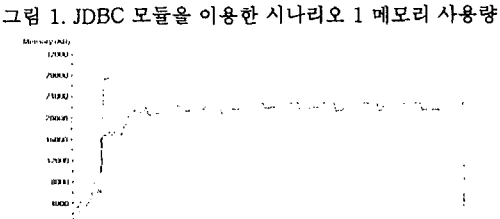

그림 2. iBatis 모듈을 이용한 시나리오 1 메모리 사용량

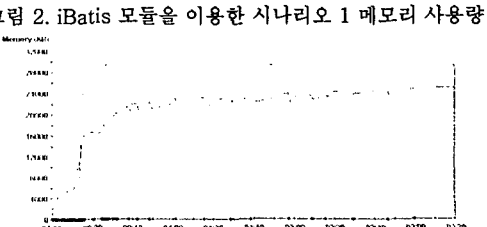

그림 3. Hibernate 모듈을 이용한 시나리오 1 메모리 사용량

다음으로 대량의 데이터 조회의 경우를 테스트한 시나리오 2의 결과는 표 4, 표 5 와 같다. 우선 2000건의 데이터를 단순 조회한 결과 1.7초 안팎의 유사한 응답 시간들을 보이고 있다. 하지만, JDBC와 달리 iBatis와 Hibernate의 캐시 기능을 적용하게 되면 동일한 시나리오에 대해 표 5와 같이 iBatis는 61ms, Hibernate는 262ms 정도로 1초 미만의 빠른 응답시간을 나타내고 있다.

즉, Read-Only의 단순 조회와 같은 경우 ORM 기술들이 제공하고 있는 캐시 기능을 적용하게 되면 JDBC에 비교할 때 상당히 높은 성능을 기대할 수 있다.

표 4. 시나리오 2 의 응답시간 결과

	JDBC	iBatis	Hibernate
수행 횟수	100	100	100
총 응답시간	174595	172955	167237
평균 응답시간	1745	1729	1672

- 시간 단위는 millisecond

표 5. 시나리오 2 의 응답시간 결과(캐시 적용)

	JDBC	iBatis	Hibernate
수행 횟수	100	100	100
총 응답시간	178595	6168	26275
평균 응답시간	1785	61	262

- 시간 단위는 millisecond

3. 결 론

지금까지 데이터 영속성을 관리하기 위한 3가지 기술들에 대해 살펴 보았다. 실제 복잡한 어플리케이션에서의 테스트는 아니었지만, 구별되는 2가지 시나리오를 통해 각 기술들에 대한 특징을 확인 할 수 있었다. 이를 통해 각 기술들의 적용 전략 및 장단점을 도출 할 수 있다.

JDBC

- 기존에 JDBC 개발에 익숙하며, SQL의 튜닝에 대한 지식이 있는 경우 적합하다.
- 여러 어플리케이션에서 일반적으로 고려할 수 있는 기술이다.
- 분산 환경이나 트랜잭션에 민감한 경우에는 적합하지 않다.
- 반복적인 개발로 인해 일반적으로 생산성은 다른 기술보다 떨어진다.
- 개발자의 능력에 따라 개발 생산성, 성능, 신뢰성이 크게 좌우된다.
- 개발 SQL 문장이 DB에 종속될 수 있다.

iBatis

- 기존에 JDBC 개발에 대한 이해가 어느 정도 있으며, 객체 지향 분석/설계에는 익숙하지 않은 경우 적합하다
- 반복적인 개발 부분은 프레임워크에서 제공하여 개발 생산성 및 신뢰성이 크게 향상된다.
- JDBC와 유사하거나 더 높은 성능을 기대할 수 있다.
- 개발 SQL 문장이 DB에 종속될 수 있다.
- 분산 및 트랜잭션 처리에 비교적 용이하다.
- 성능 향상을 위해 결과 값 및 SQL문의 캐시 등을 제공한다.

Hibernate

- 객체 지향 분석/설계 및 개발에 익숙한 경우 적합하다.
- 객체 지향으로 설계된 모듈들을 그대로 매핑할 수 있어 개발 생산성 및 유지보수성과 추적성이 뛰어나다.
- 개발 IDE의 지원으로 개발 생산성을 높일 수 있다.
- 반복적인 개발 부분은 프레임워크에서 제공하여 개발 생산성 및 신뢰성이 향상된다.
- 직접 SQL문을 작성하지 않으며, 프레임워크에서 SQL문을 최적화 하여 생성해 주므로 개발 생산성 및 성능이 향상된다.
- 분산 및 트랜잭션 처리에 비교적 용이하다.
- 상세한 튜닝 지정이 가능하다.

지금까지 각 기술들에 대한 메커니즘과 2가지 시나리오 테스트를 통한 자료를 통해 각각의 장단점과 활용 방안을 찾아보았다. 테스트 결과에서 알 수 있듯이 Hibernate와 같은 ORM 기술이 매우 매력적인 측정치를 보여주었으며, 상세한 튜닝이나 캐시 등 성능을 개선시키기 위한 효과적인 메커니즘도 제공하고 있다. 위의 결과에 기초하여 볼 때, 개발 생산성, 신뢰성, 성능 등의 요소에 있어서 ORM 기술이 기존의 JDBC를 적용할 때 보다 더욱 뛰어남을 확인 할 수 있었다.

향후에는 시나리오 자체가 매우 복잡한 경우 얼마나 효율적으로 기술들이 적용될 수 있는지를 검증할 것이다. 본 논문에서 기초 데이터로 사용하지는 않았지만, 실제로 50000건의 조회 테스트를 수행했을 때, Hibernate의 경우 캐싱 지원에 있어서 메모리에 큰 부하를 주는 것을 확인 할 수 있었다. 이러한 부분에서의 튜닝 등이 중요 이슈가 될 수 있다. 그리고 SQL의 최적화에 있어서 복잡한 쿼리의 경우 얼마나 충분히 최적화 할 수 있는지의 여부도 향후 충분한 테스트를 통해 검증할 것이다.

참고문헌

[1] Mark L. Fussell, " Foundations of Object Relational Mapping" , http://www.chimu.com/publications/index.html, 1997.

[2] Clinton Begin, "SQL Maps for Java, Developer Guide", April 22, 2004.

[3] Clinton Begin, "iBATIS Data Access Objects Developer Guide", April 22, 2004.

[4] Gavin King, Hibernate Reference Documentation, Version: 3.0, 2005

[5] Christian Bauer, Gavin King, " Hibernate in Action" , MANNING, 2005.