제21회 한국정보처리학회 춘계학술발표대회 논문집 제11권 제1호 (2004. 5)

성능 향상을 위한 J2EE 아키텍처 패턴

김정덕, 홍선주, 최성운
명지대학교 컴퓨터 공학과
e-mail : jdk2gu@mju.ac.kr

J2EE Architecture Pattern For The Performance

Jeong-Deok Kim, Sun-Joo Hong, Sung-Woon Choi
Dept. of Computer Engineering, Myong-Ji University

요 약

컴포넌트 기반 개발은 소프트웨어의 생산성을 높이며 유연성, 확장성, 호환성, 상호운영성 높은 정보시스템을 구축한다. 하지만, 이러한 특성이 성능을 저하시키는 요인이 되기도 한다. J2EE 환경에서 3-tier 컴포넌트 조립방식의 웹 어플리케이션 개발 시 성능 향상을 가져오는 아키텍처 패턴을 제시한다. 또한 그 제시한 아키텍처 패턴을 기반으로 한 구현을 통한 응답시간을 측정해 성능 향상 결과를 제시한다.

1. 서론

CBD(Component Based Development)는 컴포넌트 단위의 개발, 조립, 유지보수를 통해 정보시스템 구축 시 소프트웨어의 재사용을 극대화시키며 소프트웨어 개발의 생산성을 높이는 등의 장점을 제공한다. 이러한 CBD 장점은 유연성, 확장성, 호환성, 상호운용성 등의 컴포넌트 특성들로부터 기인하는데, 이것은 때로 시스템의 성능을 저하시키는 요인이 되기도 한다. 시스템의 재사용성을 높이고, 유연성과 확장성등을 보장하기 위해 시스템을 컴포넌트화하고 조립하는 과정에서 복잡해진 컴포넌트간의 통신문제는 시스템의 반응시간에 영향을 미칠 수 있다. 뿐만 아니라 호환성과 상호운용성을 위해 컴포넌트를 일반화하는 것은 오히려 특정 시스템 개발시 성능문제를 야기할 수 있다.

일반적으로 성능면에서는 기존의 클라이언트/서버 시스템이 3-tier 컴포넌트 조립방식의 웹 애플리케이션에 비해 우위를 차지한다. 컴포넌트 기반 시스템 개발에서 성능향상에 대한 연구는 꾸준히 계속되고 있지만 모든 시스템에 일관되게 적용할 수 있는 일반적인 대안은 현재까지 나와있지 않다.

본 논문에서는 J2EE(Java 2 Enterprise Edition)환경에서 성능향상을 위한 아키텍처 패턴을 제시한다. 또한 제시한 아키텍처 패턴을 기반으로 하는 시스템 구현을 통해 응답시간을 측정하고 이를 기존의 시스템과 비교 분석하여 성능향상 결과를 제시한다.

2. 컴포넌트 기반 시스템의 아키텍처

2.1 확장성 높은 3-Tier 혹은 n-Tier 시스템

클라이언트/서버 구조에서 서버 부하를 줄이고 기존의 레거시 시스템(legacy system)과의 통합을 원활하기 위해서 중간에 미들웨어(Middle-ware)를 두는 3-tier 혹은 n-tier 구조로 발전하였다. 이 구조를 가진 시스템은 미들 티어가 없는 클라이언트/서버 시스템보다 좀 더 확장성 있고 유연하다. 현재 이것은 일반적인 기업의 정보 시스템 아키텍처다.

J2EE 1.4 스펙에 나온 J2EE 아키텍처 다이어그램을 보면 Applet Container, Application Client Container, Web Container, EJB Container, Database 다섯 개의 요소와 그 관계를 보여주고 있다. Applet Container 와 Application Client Container 는 클라이언트에 해당된다. Web Container 는 웹 서버에 해당되고 클라이언트의 서비스 요청을 담당하는 부분이다. EJB Container 는 EJB 컨테이너 혹은 어플리케이션 서버가 해당되고 비즈니스 로직을 처리와 데이터베이스와의 상호운영을 담당한다. J2EE 는 3-Tier 혹은 n-Tier 시스템 구현을 위한 적절한 환경을 제공하고 있다.

2.2 J2EE 아키텍처와 성능

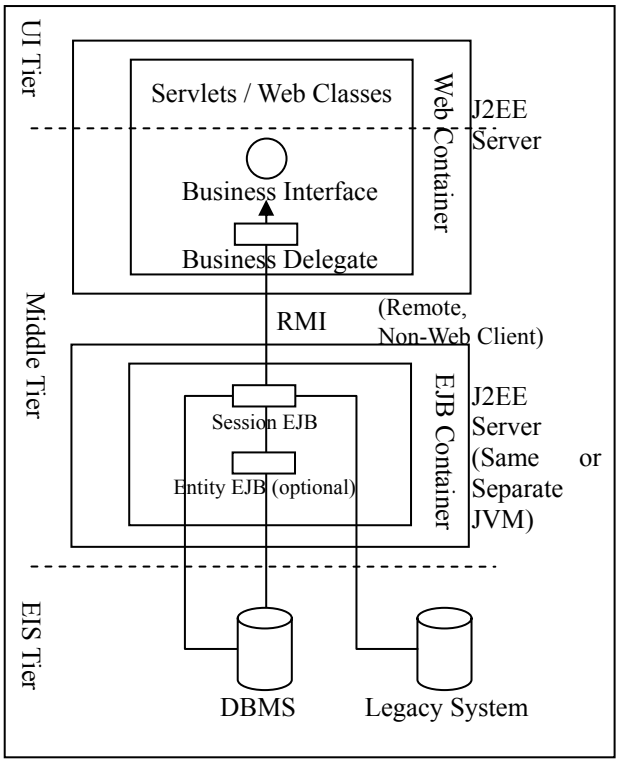
J2EE 아키텍처를 적용한 웹 어플리케이션은 네 가지로 구분할 수 있다[2].

1) 비 분산 아키텍처

- 비즈니스 컴포넌트 인터페이스를 사용하는 웹 어플리케이션
- 로컬 EJB 에 접근하는 웹 어플리케이션

2) 분산 아키텍처
- 원격 EJB 를 사용하는 분산 어플리케이션
- 웹 서비스 인터페이스를 공개하는 웹 어플리케이션

다음은 '원격 EJB 를 사용하는 분산 어플리케이션'의 아키텍처를 보여준다.



이 아키텍처에서 Web Container 의 객체와 EJB Container 의 객체는 서로 원격 호출(RMI : Remote Method Invocation)을 하게 된다. 원격 호출은 객체를 네트워크로 전송하기 위해 마샬링/언마샬링을 해야 한다. 따라서, 원격 호출은 로컬 호출에 비해서 매우 느리다[3].

성능 향상을 위해서 원격 호출을 최대한 줄이기 위해 엔터프라이즈 빈은 큰 단위의(Coarse-grained) 서비스를 제공하거나 Web Container 의 객체에서 원격 호출 참조를 저장(Cashing)함으로써 원격 호출을 줄일 수 있다. 이와 같은 성능 향상을 위한 전략을 패턴에서 찾을 수 있었다.

3. 성능 향상을 위한 J2EE 아키텍처 패턴

3.1 J2EE 패턴

패턴은 특정 상황(Context)의 반복되는 설계 문제에 대한 검증된 해결 방법이다[4]. 디자인 패턴은 성공적이고 증명된 설계와 아키텍처의 쉬운 재사용을 가능하게 하고, 새로운 시스템을 개발할 때 개발자들이 보다 쉽게 접근할 수 있게 한다.

현재 많은 디자인 패턴들이 나와 있는데, J2EE 디자인 패턴에는 두 개의 큰 카테고리가 있다. TheServerSide.com 패턴[5]과 Sun Java Center 의 J2EE 패턴[6]이다. 이 패턴 카테고리에 나온 패턴들 중에서 성능 개선을 위한 패턴은 다음과 같다[3].

	J2EE 패턴	TheServerSide.com 패턴
1	Service Locator	Service Locator
2	Session Façade	Session Façade
3	Transfer Object	Data Transfer Object

1) Session Façade
네트워크 호출을 감소 시키기 위해서 큰 단위의(Coarse-grained) 인터페이스를 구현한다.
2) Service Locator
반복적인 JNDI 룩업(lookup)을 피하기 위해 InitialContext 객체의 참조를 저장하는 캐싱 메커니즘을 제공한다.
3) Transfer Object
모든 데이터를 직렬화 가능한 하나의 객체로 패키징(Packaging)함으로써 원격 호출의 수를 줄여준다.

3.2 성능향상을 위한 아키텍처 패턴

J2EE 패턴의 Trasfer Object 와 TheServerSide.com 패턴의 DataTransfer Object 는 유사한 패턴이므로 J2EE 패턴의 명칭을 사용한다. 따라서, 위 3 가지 패턴의 조합으로 엔터프라이즈 어플리케이션을 만들 수 있는 두 가지의 아키텍처를 만들었다. 이는 패턴 간의 관계를 고려했다[3][6].

- 첫 번째 모델에 적용한 패턴

패턴명	종류
Session Façade	Façade
Transfer Object	POJO

- 두 번째 모델에 적용한 패턴

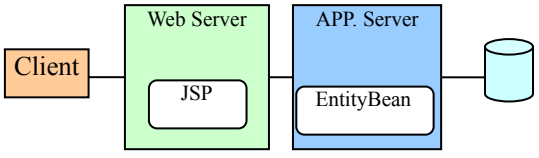
패턴명	종류
Session Façade	Façade
Service Locator	Singleton
Transfer Object	POJO

4. 성능 측정 비교 실험

4.1 구현 모델

응답 시간을 측정하기 위해 구현한 웹 어플리케이션은 총 4 가지다.

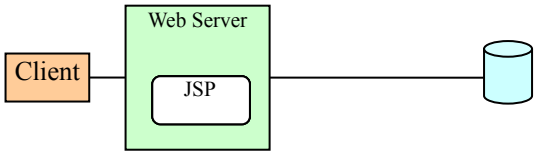
1) 모델 1



비즈니스 로직을 엔티티 빈에 포함시킨 아키텍처로 클라이언트와 엔티티빈사이의 네트워크 호출이 많아

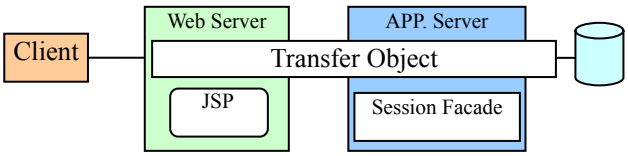
져서 네트워크 부하를 증가시키고 리모트 인터페이스의 엔티티빈에 원격 접속해 성능 저하의 원인이 된다.

2) 모델 2



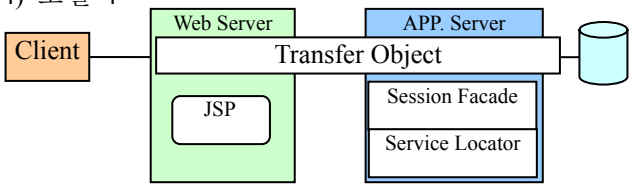
웹 어플리케이션을 개발하기 위한 가장 일반적인 방법으로 JSP 로 작성된 페이지 안에 비즈니스 로직을 포함하게 된다. 이는 성능은 좋으나 코드의 재사용은 불가능해 유지/보수가 어렵고 코드의 중복이 불가피하다.

3) 모델 3



Session Façade 패턴은 큰 단위(Coarse-grained)의 서비스를 제공해 원격 호출을 최소화한다. Transfer Object 는 UI 층과 미들웨어 층 사이의 데이터 전송을 위한 객체로서 한번의 호출로 필요한 데이터를 얻을 수 있어 잦은 원격 호출이 필요치 않게 된다.

4) 모델 4



Session Façade 와 Transfer Object 의 사용으로 인한 성능향상은 모델 3 과 동일하다. Service Locator 패턴은 반복적인 JNDI lookup 을 줄이기 위해 IntialContext 객체 참조를 저장하는 캐싱 매커니즘을 제공해 원격 호출의 수를 감소 시킨다.

4.2 응답 속도 측정을 위한 환경 및 도구

1) 테스트 환경

웹/어플리케이션서버 :
WebLogic 8.1

컴퓨터 사양 :
OS : Window
PentiumIV 1.6
RAM : 768M
LAN : 100M

데이터베이스 서버 :
오라클 8i

컴퓨터 사양 :
OS : Window
PentiumIII 550 Dual
RAM : 512M
LAN : 100M

2) 데이터베이스 구조

Oracle 에 기본적으로 생성되어있는 Scott 계정의 테이블 스키마를 사용했다. 사원(EMP)과 부서(DEPT) 테이블이 있고 사원 테이블은 ename, job, mgr, hiredate, sal, comm., deptno 컬럼을 가지고, 부서 테이블은 deptno, dname, loc 컬럼을 가진다. 한명의 사원은 하나의 부서에 소속되어 있기 때문에, 부서와 사원 관계는 일대다 관계를 가진다. 그래서, 사원 테이블의 deptno 는 포린 키로서 부서의 프라이머리 키 deptno 를 참조한다.

3) 엔티티 빈(CMP:Container Managed-Persistent) 구현

JSP 에서 직접 엔티티 빈(EntityBean)을 테스트를 위해 원격 인터페이스를 가지는 엔티티 빈을 각각의 테이블에 대해서 만들었다. 그리고, 패턴을 적용한 모델을 테스트 하기 위해서 로컬 인터페이스를 가지는 엔티티 빈을 각각의 테이블에 대해서 만들었다.

인스턴스이름	인터페이스 종류
EnDeptR	리모트 인터페이스
EnEmpR	리모트 인터페이스
EnDeptL	로컬 인터페이스)
EnEmpL	로컬 인터페이스

그리고, EJB-QL 문을 다음과 같이 작성해서 해당 테이블의 정보를 모두 가져오게 했다.

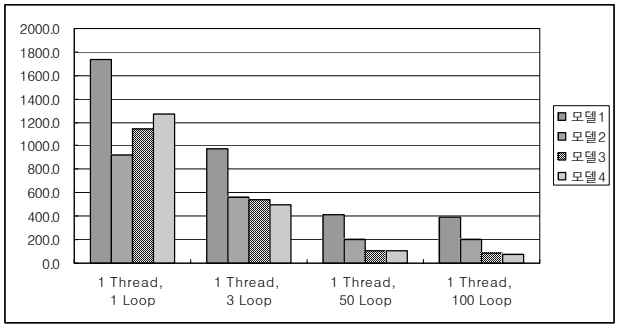
select o(a) from '엔티티 빈 이름' as a

4) 측정 도구 : JMeter Version 1.9.1

Apache JMeter 는 자카르타 프로젝트 중 하나며, 테스트 기능과 성능(Performance)를 측정하는 기능을 가진 순수 자바 어플리케이션이다. HTTP, FTP 서버에 부하를 주어 테스트할 수 있고, 완벽한 멀티 쓰레딩 프레임워크를 지원해 다수의 쓰레드가 동시에 테스트 데이터를 추출할 수 있게 했다. 또한 테스트한 성능의 분석을 그래픽하게 보여준다

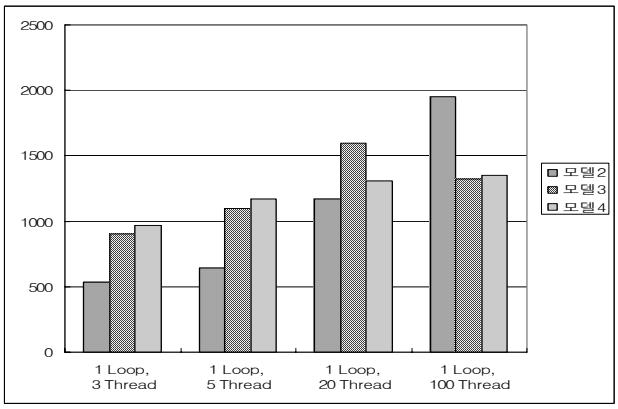
4.3 실험 결과

그림에서 Thread 는 한 명의 클라이언트이고 Loop 는 클라이언트 요청의 반복 횟수이다.



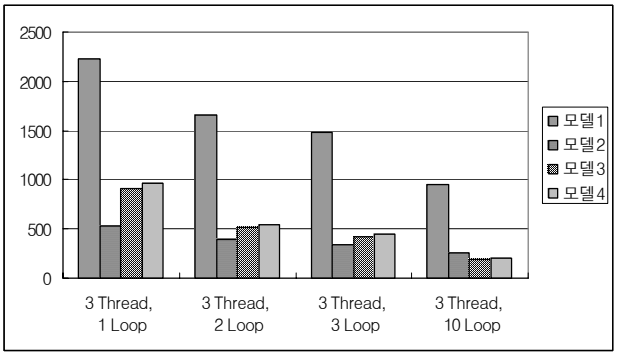
클라이언트가 한 명일 경우, 첫 번째 요청의 응답 속도는 모델 1 이 가장 좋은 성능을 보여주고 있다. 하지만, 계속적인 반복이 필요한 경우 Session Façade 와 Service Locator 를 적용한 모델 2 와 3 이 더 좋은 성능을 보여주고 있다.

사용자의 변화의 따른 성능을 알아보기 위해서 Thread 를 변화시키고 반복횟수(Loop)를 한번으로 고정했다.



대체적으로 모델 1 이 가장 좋은 성능을 보여주고 있다. 하지만, 사용자가 늘어나면서 Session Façade 나 Service Locator 패턴을 적용한 모델 2 와 3 이 더 좋은 성능을 보여주고 있다.

클라이언트 다수일 때 성능의 변화를 알아보기 위해서 3 명의 클라이언트(3 Thread)로 고정하고 반복 횟수(Loop)를 증가시켰을 때도 동일한 결과가 나왔다.



전체적으로 JSP 만을 이용한 모델 2 가 좋은 성능을 보이고 있다. 하지만, 클라이언트 늘어나고 서비스의 요청횟수가 많아질수록 모델 3 과 모델 4 의 성능이 좋아지고 있다. 클라이언트 수가 일정하지 않고 짧은 시간에 집중될 수 있는 웹 어플리케이션에서 좋은 성능을 발휘할 수 있음을 알 수 있다.

5. 결론

유연성, 확장성, 호환성, 상호운용성등의 보장을 고려하는 컴포넌트 기반 개발은 시스템 자체 성능 저하의 요인이 되기도 한다. 본 논문에서는 성능 문제를 해결하기 위한 방안으로 J2EE 환경에 적합한 패턴을 적용한 모델을 제시하였다. 또한 해당 모델을 구현하여 성능 측정을 해본 결과, 반복되는 요청이 많을수록 응답 속도가 향상됨을 알수 있었다.

추후 규모 및 복잡도 면에서 보다 다양한 시스템에 해당 아키텍처 패턴을 적용해 봄으로써 각 시스템의 특성에 따라 최적의 성능 개선 효과를 거둘수 있는 방안을 마련해야 할 것이다.

참고문헌

[1] 한국소프트웨어컴포넌트컨소시엄, “컴포넌트란 무엇인가? 알기 쉬운 소프트웨어 컴포넌트”

[2] Rod Johnson, “expert one-on-one J2EE Design and Development”, Wrox

[3] Craig A.Berry, John Camell, Matjaz B. Juric, Meeraj Moidoo Kunnumpurath, Nadia Nashi, Sasha Romanosky, “J2ee Design Patterns Applied : Real World Development with Pattern Frameworks”, Wrox

[4] Eric Gamman, Richard Helm, Ralph Johnson, John Vlissides, “Design Pattenrs : Elements of Reusable Object-Oriented Software”, Addison Wesley

[5] Floyd Marinescu, “EJB Design Patterns : Advanced Patterns, Processes, and Idioms”, Willey

[6] Deepak Alur, John Crupi, Dan Malks, “Core J2EE Patterns : Best Practices and Design Strategies Second Edition”, Sun