

- 1회 2004.8 | 오픈소스를 이용한 시스템 통합, 기술거운 도전의 시작
- 2회 2004.9 | 첫 번째 도전! 스트럿츠와 벨로시티의 통합
- 3회 2004.10 | 결합도가 악한 아키텍처를 위한 대안 기술, 스프링
- 4회 | 엔티티 빈의 대안, 하이퍼네이트

편자는 현재 EIP 형태의 C3D 기반 KMS 개발을 책임지고 있으며, EJB3.2.0 컨테이너용 디플로이/모니터링 도구, 통합물류 시스템, CCS 환경을 EJB3 기술로 자동변환하기 위한 Reflect 등 다수의 프로젝트 개발에 참여했다. 오픈소스의 가능성에 큰 기대를 걸고 VSSH 프로젝트를 기획, 시도하고 있다.

현재 EJB3 와 오픈소스 기반 기술의 SI 프로젝트를 완성화하고 웹 기반의 기술에 심취해 즐겨운(?) 프로젝트를 하고 있다. 현재 <http://eclipsenew21.org>의 홈페이지를 운영하고 있으며 컴퓨터 문명에서 벗어나 자연을 사랑하는 개발자이고 싶다고 한다.

## 시스템 통합을 위한 오픈소스 프레임워크 VSSH 3 결합도가 악한 아키텍처를 위한 대안 기술 스프링

지난 호에서는 오픈소스 프레임워크를 통합하기 위한 첫 번째 시도로 벨로시티와 스트럿츠가 어떻게 연동될 수 있는지를 살펴보았다. 이번 호에서는 VSSH 구성을 위한 핵심 기술인 스프링 프레임워크에 대해 살펴볼 것이다. 스프링을 통해 최근 자바 커뮤니티의 관심이 집중되고 있는 IoC 컨테이너와 AOP에 대한 개념을 접해보도록 하자. 그리고 스프링을 스트럿츠와 연동하기 위해 필요한 절차에 대해서도 알아보자.

자바를 이용해 기업용 비즈니스 시스템을 구축하는 것은 보통 일이 아니다. 개발자들은 복잡도를 낮추기 위해 MVC 패턴이 녹아있는 n-계층 C/S 환경을 구성하기 시작했으며, 점차 시간이 흘러가면서 대규모 웹 애플리케이션은 다음의 다섯 가지 계층으로 일반화되어 적용되고 있다.

- ◆ 프리젠테이션 계층(Presentation Layer)
- ◆ 제어 계층(Control Layer)
- ◆ 비즈니스 로직 계층(Business Logic Layer)
- ◆ 퍼시스턴스 계층(Persistence Layer)
- ◆ 도메인 모델 계층(Domain Model Layer)

다양한 프로젝트를 경험하면서 자바 커뮤니티는 최상의 실천 사례들을 디자인 패턴의 힘을 빌어 J2EE/EJB 패턴으로 구체화시켜 나갔다. 그 중 가장 널리 알려진 것은 코어 J2EE 패턴(참고자료 ②)과 EJB 패턴(참고자료 ③)이다. 개발자들은 J2EE 기술의 장점을 최대한 살리기 위해서는 패턴에 기반한 아키텍처의 구성이 무엇보다 중요하다는 사실을 깨닫기 시작했으며, 그 아키텍처의 구현인 WAF(Web Application Framework)들이 수없이 많이 쏟아져 나오기 시작했다. WAF들의 춘추전국 시대는 스트럿츠와 웹워크(WebWork)에 의해 일단락 된 듯하다. 우리나라의 경우 스트럿츠는 실질적인 웹 애플리케이션 프레임워크의 표준으로 자리잡았다고 볼 수 있다. 많은 개발자들은 스트럿츠를 사용함으로써 좋은 프레임워크가 어떤 코드가 어디에 위치되어야 하는지를 알려주는 가이드라인을 형성해 주며, 자연스럽게 좋은 설계를 이끌어내는 장점이 있음을 알게 되었다. 또한 좋은 설계를 위해 필수적인 기본 패턴들이 미리 구현되어 있어 훨씬 더 중요한 애플리케이션 로직 자체에 집중할 수 있는 여유를 준다는 점도 깨달았다.

최근 들어 자바 커뮤니티는 엔터프라이즈 영역에서 주류를 이루던 J2EE 기술에 대한 대안적 기술들의 등장에 많은 관심을 쏟고 있다. 그 대부분은 특정 업체의 주도가 아닌 개발자들의 경험을 바탕으로 하는 창조적인 아이디어에 기반한 것이며 오픈소스 형태로 진행 중이다. 뿐만 아니라 J2EE 기술 자체도 JSP 2.0과 JSF(Java Server Faces), 그리고 EJB 3.0, 타이거(J2SE 5.0)라는 큰 변화를 앞두고 있는 형편이다. 이처럼 다양한 대안적 기술들이 존재하는 상황에서 우리는 각 계층을 구현하기 위해 어떤 기술을 선택하고, 또 각각을 어떻게 하나의 아키텍처로 통합해야 하는 것일까?

이러한 궁금증을 가져본 독자들이라면 마크 이글의 글(참고자료 ④)을 한번쯤 읽어볼 필요가 있다. 그는 아키텍처를 구성하는 각각의 계층들이 애플리케이션 내에서 명확히 구별되는 기능을 가지고 있으

며, 서로 다른 계층을 침범하거나 그 기능에 있어 중복되는 점이 없어야 한다고 주장한다. 그의 글을 토대로 각 계층에서 제공해야 하는 기능은 무엇인지, 또한 제공하지 말아야 할 기능은 무엇인지를 간단히 살펴보고자 하자.

## 프리젠테이션 계층

- ◆ **역할** : 프리젠테이션 계층은 말 그대로 사용자 인터페이스에 불과하다. 식당을 예로 들면 손님이 접하게 되는 메뉴판과 전달될 음식을 차려놓는 식탁에 해당한다.
- ◆ **기능** : 사용자가 선택할 수 있는 기능이 표시되어 있어야 하고, 요청에 필요한 부가적인 정보 전달을 위한 입력 양식이 있어야 한다. 또한 전달된 자료를 효과적으로 보여주기 위한 프리젠테이션 로직이 포함된다. 하지만 비즈니스 로직이나 퍼시스턴스 계층에서 처리하는 일을 직접 수행하거나(스크립트 릿 사용), 각 계층의 컴포넌트와 직접적인 통신이 있어선 안된다. 모든 요청은 제어 계층을 통해 처리되어야 한다는 뜻이다. 고급 레스토랑(엔터프라이즈 시스템)에서는 웨이터(지배인)를 통해서만 요구를 전달하고, 그 결과를 전해들어야 한다. 직접 주방장에게 주문을 하거나 자기가 직접 요리를 하는 것은 자기 집(프로토타입)이나 동네 자장면 가게(소규모 웹 애플리케이션)에서나 가능한 일이다.
- ◆ **대안 기술** : 현재 가장 주류를 이루는 기술은 JSP 1.2와 JSTL과 같은 태그 라이브러리를 결합하는 방식이다. 과도기적인 형태로 벨로시티와 타일즈 태그 라이브러리가 결합된 형태도 현재 주목을 받고 있다. 하지만 점차적으로 JSF에 기반한 JSP 2.0에 주류 기술로 옮겨갈 가능성이 크고, 프리젠테이션 계층 개발에 있어서도 JSF를 지원하는 IDE를 채택하는 경우가 늘어날 것이다.

- ◆ **주요 패턴** : Composite View 패턴

## 제어 계층

- ◆ **역할** : 제어 계층은 프리젠테이션 계층과 비즈니스 로직 계층을 분리하기 위한 컨트롤러를 제공한다. 식당으로 치면 지배인의 역할과 종업원의 역할을 병행하는 것이라고 볼 수 있다.
- ◆ **기능** : 전체 시스템의 설정 상태를 유지해야 하며, 그를 통해 어떤 요청이 들어왔을 때 어떤 로직이 처리해야 하는지를 결정한다. 사용자 요청을 검증하고 로직에 요청을 전달하는 일과 로직에서 전달된 응답을 적절한 뷰에 연결짓는 것 역시 제어 계층의 몫이다. 손님이 바다가재 요리를 요청했을 때 종업원은 그 요리가 서비스 가능한 것인지 또한 누구에게 시키면 되는지를 알고 있어야 한다는 뜻이다. 요청을 전달받은 요리사가 바다가재 요리를 주면 그것을 식탁까지 운반해 주는 것 역시 종업원의 몫이다. UI 검증, 요청 및 응답 전달, 로직에서 던져진 예외 처리, 도메인 모델을 뷰와 연결하기 등의 고유 기능 외에는 어떤 기능도 포함하지 않는다.
- ◆ **대안 기술** : 현재 WAF들은 대부분 제어 계층의 핵심 기능을 포함한다. 터빈이나 에스프레소 등이 선전하고 있지만, 스트럿츠와 웹워크가 대세라고 생각된다. 당분간 별다른 대안 기술이 등장할 가능성은 적다. 오히려 스트럿츠를 확장시켜 자사 고유의 프레임워크로 최적화 시키는 작업이 활발히 진행될 것이다.

- ◆ **주요 패턴** : Front Controller 패턴, Service to Worker 패턴(또는 Command 패턴), Intercepting Filter 패턴, Application Controller & Context Object 패턴

## 비즈니스 로직 계층

- ◆ **역할** : 비즈니스 로직은 말 그대로 핵심 업무를 어떻게 처리하는지에 대한 방법을 기술하는 곳이다. 식당에서 종업원이 고객의 요구를 전달해 주면, 재료를 이용해 요리를 만드는 요리사라고나 할까? 비즈니스 로직 계층은 애플리케이션에서 가장 재사용될 확률이 높은 요소이기 때문에 신경써서 설계해야 한다.
- ◆ **기능** : 비즈니스 로직에는 핵심 업무 로직의 구현과 그에 관련된 데이터의 적합성 검증 외에도 다양한 부가적인 구현이 추가된다. 트랜잭션 처리라든가, 다른 계층들과 통신하기 위한 인터페이스를 제공한다거나, 해당 계층의 객체들간의 관계를 관리하는 것 등이 그것이다. 비즈니스 로직 계층에 있어야 할 코드들이 프리젠테이션 계층이나 퍼시스턴스 계층에 여기저기 흩어져 있는 애플리케이션을 찾아보기란 그리 어려운 일이 아니다. 이런 구조는 각각의 계층을 모호하게 만들어 유지보수시 많은 시간을 필요로 하게 만든다. 가장 신경 써서 개발해야 할 비즈니스 로직에 그동안 신경을 쓰지 못했다는 것. 프리젠테이션 계층과 퍼시스턴스 계층 사이의 다리 역할을 충실히 하도록 함으로써 애플리케이션에 유연성을 더하는 것. 그것이 스프링이 탄생하게 된 배경이라고 할 수 있다.
- ◆ **대안 기술** : 지금까지 비즈니스 로직의 구현은 크게 EJB를 사용하는 것과 일반 자바 객체(POJO)를 사용하는 것으로 나눌 수 있었다. EJB를 사용하는 경우 개발자들의 많은 불만이 EJB 3.0을 사용함으로써 해결되리라 예상된다. 하지만 해외를 중심으로 해서 EJB를 사용하든, 사용하지 않든 비즈니스 로직들을 체계적으로 관리하는 IoC 컨테이너에 대한 관심이 증가하고 있는 추세이다. IoC 컨테이너에 대해서는 뒤에서 다시 자세히 살펴보도록 하겠다.
- ◆ **주요 패턴** : Business Delegate 패턴, Session Facade 패턴, Service Locator 패턴, Application Service 패턴, EJB Home Factory 패턴

## 퍼시스턴스 계층

- ◆ **역할** : 퍼시스턴스 계층은 데이터 처리를 담당하는 계층이다. 주로 데이터의 생성·수정·삭제·선택(검색)과 같은 CRUD 연산을 수행하게 된다. 식당으로 보자면 주방장이 사용할 재료를 담당하는 재료 담당자라고나 할까? 이 데이터는 주로 데이터베이스에서 처리되는 경우가 많아, 영속성을 의미하는 퍼시스턴스 계층이란 용어를 사용했다. 하지만 데이터가 처리되는 다른 업무 시스템이나, 웹 서비스, XML, 파일 시스템 등을 모두 고려한다면 레거시 개념을 갖는 EIS 계층이란 표현이 더 적합할 것이다.
- ◆ **기능** : 이 계층에서 수행하는 일은 관계형 정보를 저장하고, 수정·삭제하는 것과, 그러한 일을 수행하는 데 필요한 질의문을 관리하는 것, 그리고 가져온 관계형 정보를 객체화시키는 일이다.
- ◆ **대안 기술** : EJB 사용에 있어서 개발자들이 가장 불만스러워 하는 것은 CMP 방식의 엔티티 빈일 것이다. 그러한 불만은 객체 관계 매핑(ORM)을 이용한 JDO라

는 대안기술을 탄생시켰고, 또한 하이버네이트라는 또 하나의 오픈소스 기술을 실무로 끌어들었다. JDBC를 이용한 DAO 객체를 구성하는 방법도 소규모 애플리케이션에서는 여전히 인기를 끌고 있다. EJB 3.0과 JDO/하이버네이트, 그리고 JDBC를 이용한 POJO 방식 중 어떤 것이 개발자들에게 낙점될 지는 아직 미지수다.

- ◆ **주요 패턴** : Data Access Object 패턴, Domain Store 패턴, Sequence Blocks

## 도메인 모델 계층

- ◆ **역할** : 도메인 모델은 각 계층 사이에 전달되는 실질적인 비즈니스 객체라고 할 수 있다. 식당을 예로 든다면, 음식이 담긴 그릇이라고 비유할 수 있겠다.
- ◆ **기능** : 도메인 모델 계층은 흔히 데이터 전송 객체(DTO) 형태로 개발자가 직접 제작해서, 리퀘스트나 세션과 같은 컨텍스트에 담아 넘기게 된다. 하지만 데이터 베이스의 모든 정보를 일일이 객체로 만드는 것은 귀찮을 뿐 아니라, 계층간의 통신 과정에서 데이터가 유실될 위험도 있기 때문에 최근에는 도메인 모델을 서비스로 제공하여 자동화하는 경우가 많다.
- ◆ **주요 패턴** : Data Transfer Object 패턴, Value List Handler 패턴

지금까지 엔터프라이즈 시스템을 구축하기 위해 일반적으로 사용되는 다섯 계층에 대해 간단히 정리해 보았다. 뻔한 이야기들을 길게 늘어 쓴 이유는 2가지 사실을 강조하기 위해서이다. 첫째, 각각의 계층은 저마다의 분명한 역할이 존재하며, 그 역할을 충실히 수행할 수 많은 대안기술(Alternative)들 사이에서 개발자는 무엇을 선택할 지 결정을 내려야 한다. 둘째, 각각의 기술들은 독립적으로도 충분한 가치를 지니고 있지만, 가장 장점을 발휘하는 제 위치에서 서로 연계되어 사용될 때 그 시너지 효과가 더욱 크다.

## 악한 결합도를 가진 아키텍처 구성

이 글을 읽는 상당수의 독자들은 애플리케이션을 개발할 때 스트럿츠를 사용해 본 경험이 있을 것이다. 프리젠테이션 계층에는 태그 라이브러리가 접목된 JSP를 이용했을 것이며, 업무 로직과 영속성 처리는 그 규모에 따라 POJO나 EJB를 적절히 섞어 사용했을 것이다. 하지만 정말 스트럿츠만으로 충분했었는지 묻고 싶다.

〈그림 1〉은 앞에서 정리한 각 계층을 도식화하고, 각 계층별 대안 기술을 요약해서 표기한 것이다. 언뜻 보기에 전체 구조는 흠잡을 곳이 없어 보이기도 한다. 그렇다면 다음의 3가지 질문에 대답해 보자.

- ❶ 제어 계층에서 비즈니스 로직 계층에 구현된 기능을 이용하기 위해서는 구체적인 설정 정보를 알아야 한다. 그런데 만일 이용하고자 하는 기능에 대한 설정이 변경된다면 어떻게 할 것인가? 컴포넌트의 설정과 그 사용을 분리할 수 있는 방법은 과연 무엇일까(힌트 : 리팩토링의 저자이기도 한 마틴 파울러는 그의 홈페이지에



서 이에 대한 해결책으로 제어 역행화(Inversion of Control) 패턴이란 것을 소개하고 있다)?

**2** 비즈니스 로직 계층에는 순수한 업무 로직의 구현 외에도 보안, 인증, 로그와 같은 시스템 전반에 걸친 기능들이 공존한다. 이러한 기능은 업무 로직과 뒤섞여 코드를 관리하기 어렵게 하는 주범이 되기도 한다. 이에 대한 해결책은 없는가(힌트 : AOP는 바로 그러한 문제를 다루기 위한 새로운 패러다임이다)?

**3** 지금 엔터프라이즈 자바 영역은 표준화된 주류 기술과 창의적인 다양한 아이디어로 무장한 오픈소스 기술들로 인해 수없이 많은 선택을 개발자들에게 강요하고 있다. 지금 현재는 대안들 중 하나를 선택했다라도 시간이 지남에 따라 그 선택을 바꾸게 될 가능성도 얼마든지 존재하는 것이다. 스프링을 웹워크로 교체하더라도 또는 EJB 구현을 하이버네이트로 교체하더라도 그러한 변경의 영향이 전체에 파급되지 않고, 해당 계층에 국한되도록 할 수 있는 보다 포괄적인 프레임워크는 없을까(힌트 : 이러한 기능을 수행하는 프레임워크를 경량급 컨테이너(Lightweight Container)라고 부른다. 경량급 컨테이너의 핵심 원리가 바로 제어 역행화 패턴이다. 스프링 프레임워크는 경량급 컨테이너임과 동시에 AOP를 지원한다. 이것이 우리가 스프링(Spring)에 관심을 가지는 이유이다)?

각 계층별 결합도를 떨어뜨리는 것(loosely-coupled)은 좋은 아키텍처를 구성하기 위해 필수적인 요건이라고 할 수 있으며, 요즘처럼 대안 기술이 많은 경우에는 특히 중요하다고 하겠다. 약한 결합도를 가진 아키텍처를 구성하기 위한 핵심 기술이 바로 스프링 프레임워크이다.

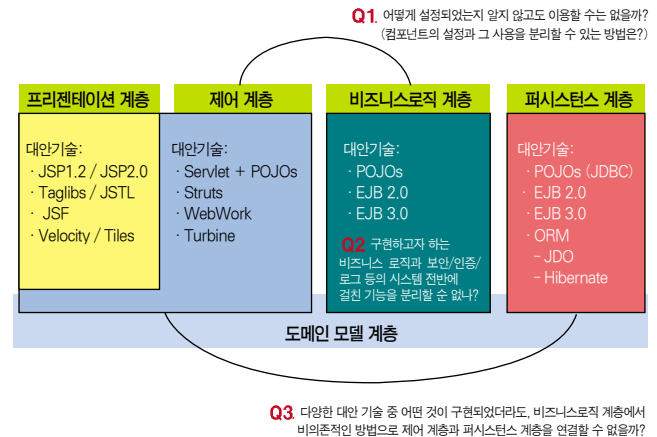
## 스프링 프레임워크 개요

스프링은 그 이름 자체로도 많은 의미를 내포하고 있다. 봄! 이 얼마나 설레는 단어인가? 봄이라는 이름만으로도 무거운 J2EE의 사용으로 지친 개발자들에게 이제 겨울이 끝나고 새로운 계절이 돌아오고 있음을 함축적으로 표현해내고 있다. 스프링은 로드 존슨이 쓴 『Expert one-on-one J2EE Design and Development』란 책에서 소개된 소스코드를 기반으로 2003년 2월 오픈소스로 시작된 프로젝트이다. 스프링이 추구하는 바는 크게 두 가지이다.

- 1** 복잡하고 무거운 J2EE 기술의 사용을 쉽고 가볍게 만들어주고, 자연스럽게 검증된 최상의 실천 사례들을 구현하도록 함으로써 좋은 프로그램이 작성될 수 있도록 유도한다.
- 2** 기존의 잘 알려진 기술들을 프레임워크 내에서 일관된 방법으로 쉽게 사용할 수 있도록 돕는다.

이를 위해 스프링은 다른 프레임워크와는 차별화된 다음과 같은 특징을 가진다.

<그림 1> 아키텍처를 구성하는 다섯 계층과 대안 기술



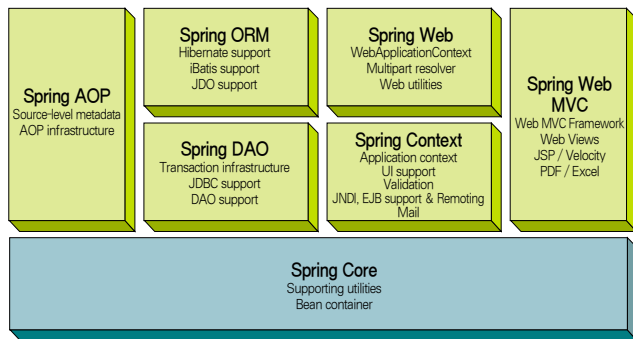
- ◆ 스프링은 EJB를 사용하지 않는 관계없이 비즈니스 객체들을 효과적으로 구성하고, 관리하는 방법을 제공하는 데 초점을 맞춘다.
- ◆ 스프링은 계층화된 아키텍처를 갖고 있으며, 그 중 어떤 부분도 독립적으로 사용될 수 있도록 모듈화되어 있다. 뿐만 아니라 각각의 모듈은 일관된 방법으로 사용할 수 있기 때문에 한번 익숙해지고 나면 사용이 무척 쉽다.
- ◆ 스프링은 전체 프로젝트의 설정을 관리할 수 있는 일관된 방법을 제공함으로써, 개발자들이 각종 프로퍼티 파일을 작성하지 않도록 유도한다. 이것은 IoC라는 스프링의 특징 때문인데, 객체들간의 의존성이 따로 관리됨으로써 비즈니스 로직이 EJB로 개발되었든 일반 자바 객체로 개발되었든 동일한 방법으로 해당 로직을 이용할 수 있는 이점도 추가된다.
- ◆ 스프링 기반으로 작성된 애플리케이션은 스프링의 API에 의존하지 않는다. 이것은 어떤 애플리케이션 서버와도 쉽게 연동되도록 하며, 심지어 스프링을 사용하지 않았을 때조차도 비즈니스 로직의 재사용이 가능해지는 요인이 된다.
- ◆ 스프링은 AOP 지원을 통해 주요 비즈니스 로직과 시스템 전반에 걸친 기능 모듈을 완벽히 분리해내도록 도와준다.
- ◆ 스프링은 작성된 코드에 대한 유닛 테스트를 쉽게 할 수 있도록 도와준다.

## 스프링의 기능과 사용 시나리오

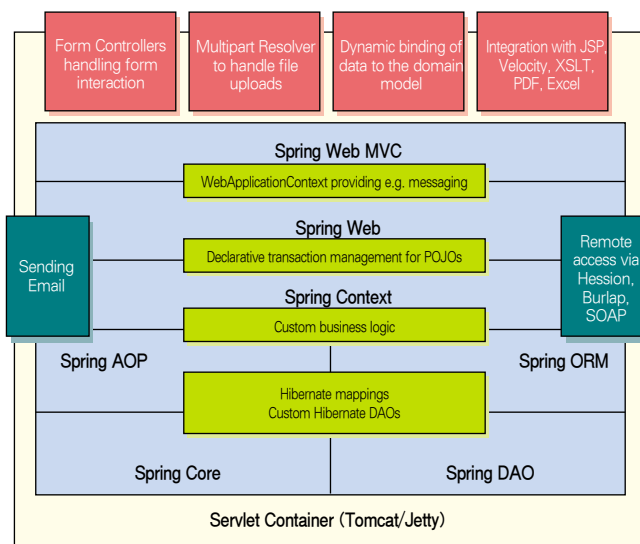
현재 스프링은 1.0 버전이 출시된 상태이다. 스프링 홈페이지(참고자료 ⑤)에서 spring-framework-1.0.2-with-dependencies.zip 파일을 다운받기 바란다. 이 파일은 의존성 있는 관련 라이브러리가 모두 포함된 버전이다. 스프링의 전체 기능은 크게 7개의 모듈로 구성된다(<표 1>).

스프링 배포 파일의 압축을 풀면 dist란 디렉토리가 나타난다. 그 안에 있는 spring.jar가 앞에서 언급한 스프링의 모든 기능을 포함하는 파일이다. 각각의 기능 중 필요한 부분을 따로 사용할 경우를 위해

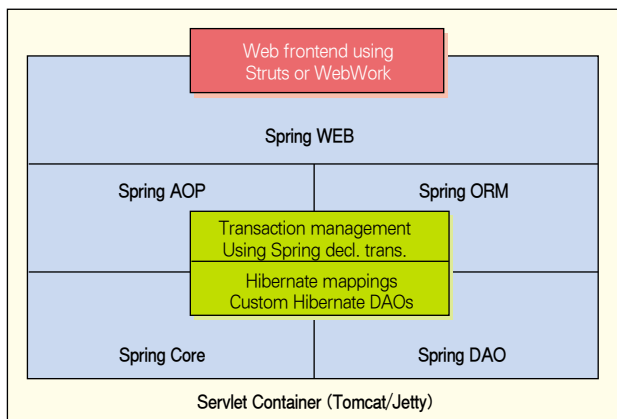
〈그림 2〉스프링의 기능 요소



〈그림 3〉시나리오 1. 완전한 형태의 스프링 웹 애플리케이션



〈그림 4〉시나리오 2. 서드파티 WAF와 ORM을 연계한 웹 애플리케이션



패키지별로 묶은 별도의 JAR 파일이 함께 제공된다.

스프링을 이용한 일반적인 형태의 웹 애플리케이션은 〈그림 3〉과 같은 구조를 가진다. 톰캣, 웹로직, 웹스피어, JBoss를 포함한 어떤 웹 애플리케이션 서버에서도 동작된다. IoC 컨테이너의 핵심인 Core 패키지와 AOP 지원을 위한 AOP 패키지, 기능을 구현한 빈 객체에 대한 접근을 제공하는 Context 패키지는 반드시 포함되어야 한다. 대부분 데이터베이스 처리를 수행하기 때문에 DAO 패키지도 일반적으로 포함된다. 스프링 애플리케이션은 일반적으로 ORM 솔루션을 채택하고, 특히 하이버네이트와 궁합이 잘 맞기 때문에 하이버네이트를 설치하고 ORM 패키지를 이용하는 경우가 일반적이다. 그 위에서 Web 패키지를 기반으로 Web MVC 패키지를 이용해서 애플리케이션을 개발한다.

만일 스프링의 Web MVC 패키지를 이용하지 않고, 스트럿츠나 웹워크를 제어 계층에 사용하고 싶다면 Web MVC를 스트럿츠가 대체하는 〈그림 4〉와 같은 구조로 웹 애플리케이션이 개발될 것이다.

이 밖에도 EJB를 사용하는 경우 AbstractEnterpriseBean이라는 POJO를 이용해서 EJB를 스프링에서 관리하도록 하고, SlsbInvoker를 이용해서 EJB에 대한 접근 경로를 제공하는 EJB 사용 유형이 있다. 또한 웹 서비스를 포함한 다른 애플리케이션과 연동하는 경우를 위해 Remote 패키지가 제공되기도 한다.

스프링은 그 자체로도 한 권의 책을 쓸 수 있을 만큼 방대한 기술이다. 그 모두를 짧은 연재를 통해 소개한다는 것은 불가능하다. 그러한 이유로 스프링에 대한 소개는 이쯤에서 마무리하고, 스프링을 이해하기 위해 반드시 알아야 하는 IoC(Inversion of Control) 컨테이너의 개념과 AOP(Aspect Oriented Programming)를 간단히 설명하고,

〈표 1〉스프링의 기능 요소

패키지 명	기능
Core	스프링의 핵심으로 기능과 그 설정을 분리하기 위한 IoC 기능이 구현된 BeanFactory를 제공한다.
Context	Core 패키지와 마찬가지로 스프링의 기본 기능이다. 이것은 JNDI가 EJB를 비롯한 리소스에 대한 접근 경로를 제공하는 것처럼 스프링 기반에서 구현된 기능 객체(Bean)들에 대한 접근 방법을 제공한다.
DAO	DAO 패키지는 JDBC에 대한 추상화 계층으로 지루한 JDBC 코딩이나 예외 처리를 없애준다. 또한 트랜잭션 관리 기능도 제공한다.
ORM	객체/관계 맵핑을 위한 JDO, 하이버네이트, iBatis 등과의 통합을 위한 패키지이다. ORM 제품들을 스프링의 기능과 조합해서 사용할 수 있게 한다.
AOP	AOP 어레이언스와 호환되는 AOP 구현 패키지이다.
Web	일반적인 웹 애플리케이션 개발에 필요한 기본 기능을 제공하고, 웹워크나 스트럿츠와의 통합을 위해 사용되는 패키지이다.
WebMVC	스트럿츠와 같은 일반적인 WAF의 기능을 스프링 버전으로 구현하고 있는 패키지이다. 기존에 사용하고 있던 WAF가 없다면 사용을 고려해 볼만 하다.

2개의 작은 예제를 소개하는 것으로 이번 호를 마무리하겠다. 부족한 설명은 필자들이 운영하는 VSSH 포럼(참고자료 ❶)이나, 스프링 관련 기사 및 자료들을 정리한 리소스 맵(참고자료 ❷)을 통해 여러분 스스로 익혀나가기 바란다.

## IoC 컨테이너와 AOP

스프링은 다른 프로젝트에서 개발된 컴포넌트를 조립해서 응집력 있는 애플리케이션의 개발이 가능하도록 도와주는 IoC 컨테이너이며, 다른 말로 경량급 컨테이너(Lightweight Container)라고도 한다. IoC 컨테이너의 또 다른 종류로는 PicoContainer와 아파치의 아발론, 그리고 HiveMind 등이 있다. 그 중에서 현재 가장 널리 사용되고 있는 것은 스프링과 PicoContainer이다. IoC 컨테이너는 다른 컨테이너와 달리 애플리케이션 코드와 컨테이너간의 의존성을 최소화하는 것이 특징이다.

IoC 컨테이너가 컨테이너에 대한 의존성을 최소화하면서 컴포넌트를 묶어주는 일을 수행하는 밑바탕에는 제어 역행화(Inversion of Control)라는 개념이 깔려 있다. 제어 역행화는 리팩토링의 저자이기도 한 마틴 파울러의 홈페이지(참고자료 ❸)에 잘 정의되어 있다. 제어 역행화라는 용어는 직관적이지 못하기 때문에, 또 다른 말로 연관성 삽입(Dependency Injection)이라고도 불려진다. 연관성 삽입 패턴은 컴포넌트의 설정을 그것의 사용에서 분리해야 한다는 원칙(The principle of separating configuration from use)에서 출발한다. 그러한 원칙을 위한 또 다른 사례는 J2EE 패턴 중 서비스 로케이터(Service Locator) 패턴이다.

의존성 삽입 패턴을 이해하기 위해 간단한 예를 하나 살펴해보도록 하자. 어느 특정 감독이 만든 영화를 검색해서 그 결과를 전달해주는 컴포넌트를 사용하는 것이다. 코드에서 보여지는 findAll()이라는 메소드를 가지는 finder 객체가 필요하단 사실을 알 수 있다.

```
class MovieLister...
    public Movie[] moviesDirectedBy(String arg) {
        List allMovies = finder.findAll();
        for (Iterator it = allMovies.iterator(); it.hasNext(); ) {
            Movie movie = (Movie) it.next();
            if (!movie.getDirector().equals(arg)) it.remove();
        }
        return (Movie[]) allMovies.toArray(new Movie[allMovies.size()]);
    }
}
```

일반적으로 이럴 때 기능 확장을 위해 MovieFinder와 같은 인터페이스를 작성하게 된다.

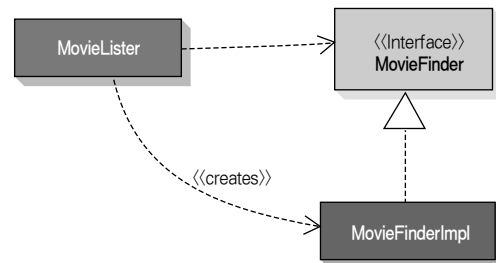
```
public interface MovieFinder {
    List findAll();
}
```

영화 정보가 콜론으로 구분된 CSV 파일에 기록되어 있다면, MovieFinder 인터페이스를 구현한 ColonDelimitedMovieFinder 클래스가 필요할 것이다. 또한 그 정보는 MovieLister에 생성자를 이용해서 초기화될 것이다.

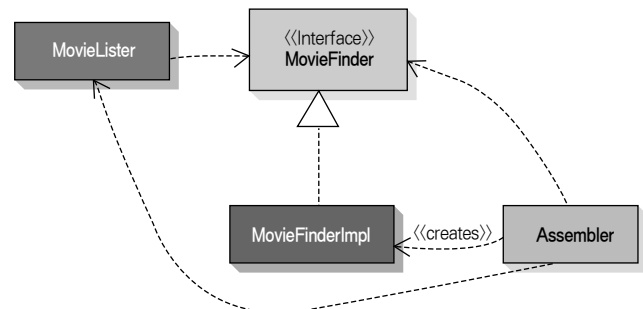
```
class MovieLister...
    private MovieFinder finder;
    public MovieLister() {
        finder = new ColonDelimitedMovieFinder("movies1.txt");
    }
```

전체적인 시스템 구조는 <그림 5>와 같은 형태이다. MovieLister 클래스는 MovieFinder 인터페이스 정보만을 이용해서 기능을 구현할 수 있지만, 해당 기능을 사용하기 위해 MovieFinderImpl 클래스 중에서 ColonDelimitedMovieFinder를 이용한다는 구체적인 설정 정보가 클래스의 코드에 포함되어 있다. 이것은 데이터가 DB나

<그림 5> 일반적인 제어 흐름을 통한 의존성 표현



<그림 6> 제어 역행화 패턴을 통한 의존성 삽입



XML로 변경되어 RDBMovieFinder나 XMLMovieFinder로 교체 될 경우 코드를 수정해서 다시 빌드해야 한다는 사실을 의미한다.

사실 MovieLister는 정보가 CSV 파일에 기록되어 있든, DB에 기록되든, XML에 기록되든 영향을 받지 않아야 정상이라고 할 수 있다. 어떻게 하면 이처럼 불필요한 의존 관계를 없앨 수 있는 것일까?

그 해답은 설정 정보에 따라 어떤 구현 객체를 사용할 것인지를 결정하는 어셈블러를 이용해서 <그림 6>에서 보여지는 구조로 애플리케이션을 개발하는 것이다. MovieLister 클래스에서 선언된 MovieFinder 타입의 finder를 초기화하는 방법은 크게 2가지가 있다. 첫 번째는 생성자를 통해 속성을 초기화하는 것이고, 두 번째는 setMovieFinder()와 같은 Setter 메소드를 이용하는 것이다. 이러한 설정을 자동화하는 어셈블러를 구현해 놓은 것이 바로 IoC 컨테이너이다. 연관성 삽입은 크게 Constructor Injection, Setter Injection, Interface Injection의 3가지 유형을 가진다. Constructor Injection이 생성자를 이용해서 의존성을 설정해주는 방법이고, Setter Injection이 Setter 메소드를 이용해서 의존성을 설정해주는 방법이다. Pico Container는 Constructor Injection을 주로 사용하고, 스프링은 자바 빈 규칙을 이용한 Setter Injection을 주로 사용한다. 스프링의 IoC적인 특징은 AOP를 구현하는 핵심적인 원리가 되기도 한다. AOP는 아직 국내에는 생소한 분야이다. 김대곤님이 작성한 간단한 소개글(참고자료 ⑧)을 통해 AOP의 개념을 잡아보도록 하자.

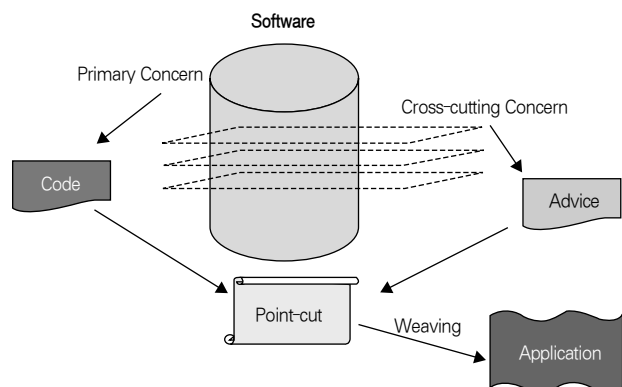
자금 이체를 하는 프로그램을 작성한다고 가정해 보자. 출금 계좌와 입금 계좌 그리고 이체 금액을 입력받아 SQL 문장 또는 함수 한번 돌리는 것으로 모든 프로그래밍이 끝나는가? 그렇지 않다. 해킹을 방지하기 위해 사용자가 적절한 보안 프로그램을 설치했는지 점검하는 코드도 있어야 하고, 사용자가 인증되었는지 점검하는 코드도 써야 하고, 상대방 은행에서 적절하게 처리되었는지도 점검해야 하고,

혹시 사용자가 이체 버튼을 두 번 누른 것은 아닌가 체크해야 하고, 시스템 로그도 남겨야 한다. 즉, 구현하려고 하는 기능 뿐 아니라 보안, 인증, 로그, 성능과 같은 다른 기능들도 녹아 있어야 한다는 뜻이다. 어쩌면 이체를 위한 코드보다 잡다한 다른 측면의 문제들을 다루는 코드가 더 길어질 수 있다. 이런 코드들은 입금이나 출금 같은 다른 곳에서도 공통적으로 사용되는 것이다.

구현하려고 하는 비즈니스 기능들을 AOP에서는 Primary(Core) Concern이라는 용어로 표현한다. 보안, 로그, 인증과 같이 시스템 전반적으로 산재된 기능들은 Cross-cutting concern이라고 부른다. AOP는 Cross-cutting concern을 어떻게 다룰 것인가에 대한 새로운 패러다임이라고 할 수 있다. 그럼 AOP가 등장하기 이전에 우리는 어떻게 Cross-cutting Concern을 처리해왔을까? 매우 간단하다. Primary Concern를 구현한 프로그램에 함께 포함시켰다. Primary concern, Cross-cutting concern이 하나의 프로그램 안에 들어가게 되면, 프로그램을 이해하기가 힘들고, Cross-cutting concern 코드가 여기저기에 산재되어 수정하기 힘들게 된다. 당연히 생산성이 떨어지고, 품질이 떨어지고, 유지보수 비용은 많이 들게 된다.

그럼 AOP는 Cross-cutting concern를 어떻게 처리하는가? AOP에서는 Primary Concern 구현하는 코드 따로, Cross-cutting concern 구현하는 코드도 따로 작성한다. 나중에 2개를 조합한 완벽한 애플리케이션이 만들어지는 것이다. AOP에서는 Cross-cutting concern 구현한 코드를 Advice라고 하며, Primary concern 구현한 코드를 Code라고 부른다. Code와 Advice를 연결해주는 설정 정보를 Point-cut이라고 하며, 둘을 조합해서 애플리케이션으로 완성하는 과정을 Weaving(조합)이라고 부른다. 기술적 용어로서의 'Aspect'는 Advice와 Point-cut을 함께 지칭하는 단어이다. Point-cut은 어떤 Advice를 Code 어느 위치에 둘 것인가 하는 것이다. 스프링의 AOP 패키지는 이러한 AOP 개념을 구현한 것으로, 그 기반에는 설정을 이용으로부터 분리하는 의존성 삽입 패턴이 녹아 있다.

<그림 7> AOP의 개념



## 예제 1. 스프링의 WebMVC를 이해하자

이제 스프링을 이해하기 위해 2가지 예제를 살펴볼 것이다. 첫 번째 예제는 스프링 공식 홈페이지에 소개된 것으로 순수하게 스프링이 제공하는 기능만을 이용해서 개발된 예제(참고자료 ⑨)이다. 국내에는 스프링과 관련된 자료가 전혀 없는 관계로 초보자들을 위해 이 예제를 번역하고 몇 가지 설명을 추가해 총 4부 8개의 강좌로 재구성해서 VSSH 포럼에 등록(참고자료 ⑩)해 두었다. 자세한 설명을 원하는 독자는 관련 자료를 참고하기 바란다.

여기서 소개하는 예제는 앞에서 소개한 강좌 중 2부까지 구현된 샘플이다. 예제를 실행해보기 위해서는 톰캣과 Ant, JDK 등이 설치되

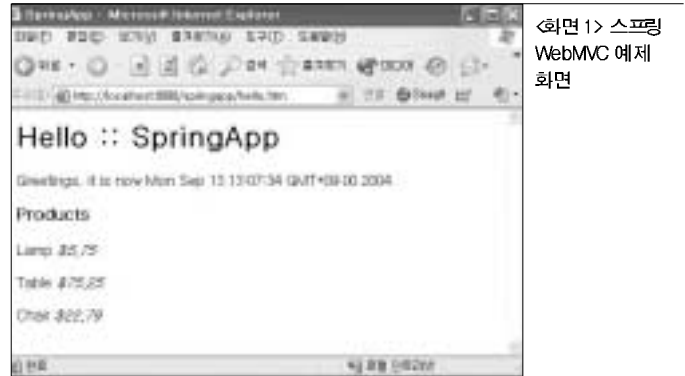
어 있어야 한다. 예제(springapp.zip)를 다운로드한 다음, 작업 디렉토리에서 압축을 풀면 build.properties 파일이 보일 것이다. 해당 파일을 열어 배포될 경로와 톰캣 홈, 그리고 톰캣 관리자의 URL/아이디/패스워드 정보를 자신의 환경에 맞게 변경한다. 그런 다음 build와 deploy 타겟을 ant를 이용해서 순서대로 실행한다. 웹 브라우저를 이용해서 http://localhost:8080/springapp로 접속하면 다음과 같은 결과 화면을 볼 수 있을 것이다.

우선 예제의 web.xml 설정부터 살펴해보도록 하자. DispatcherServlet이 springapp란 이름으로 등록되어 있고, htm으로 끝나는 모든 URL 패턴이 해당 서블릿으로 맵핑되어 있다. DispatcherServlet은 스트럿츠의 ActionServlet의 기능과 유사한 일종의 FrontController 서블릿이다.

```
<web-app>
  <servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>springapp</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
</web-app>
```

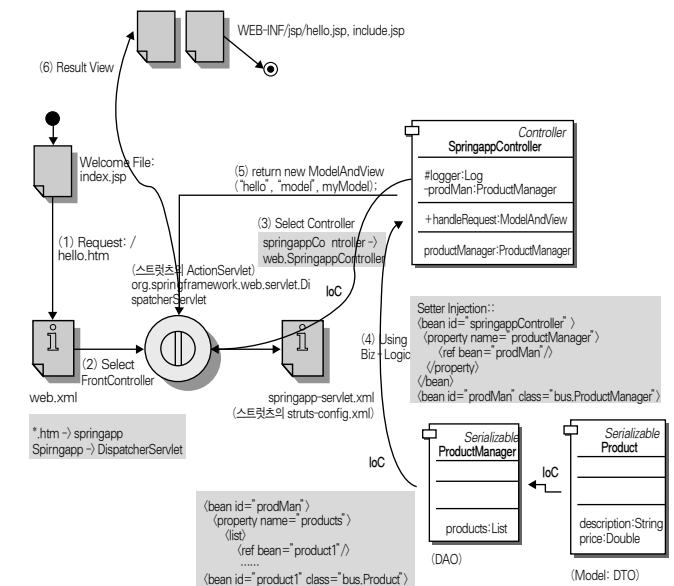
그렇다면 DispatcherServlet이 처리하는 제어 행위를 위한 struts-config.xml과 같은 설정 파일이 필요할 것이다. 스프링 MVC 패키지에서는 그 파일을 해당 서블릿 이름에 “-servlet.xml”을 붙여 작성하도록 권장하고 있다. 앞의 경우 서블릿의 이름을 springapp로 주었기 때문에 springapp-servlet.xml이 설정 파일이 되는 것이다. 설정 파일을 보면, <beans>라는 루트 엘리먼트 밑에 <bean>이라는 설정이 반복되어 적용되고 있다. 이것이 스프링에서 BeanFactory를 이용해서 제공하고 있는 Setter Injection을 통해 설정과 그 사용을 분리하는 방법이다.

```
<beans>
  <bean id="springappController" class="web.SpringappController">
    <property name="productManager">
      <ref bean="prodMan"/>
    </property>
  </bean>
  <bean id="prodMan" class="bus.ProductManager">
    <property name="products">
      <list>
```



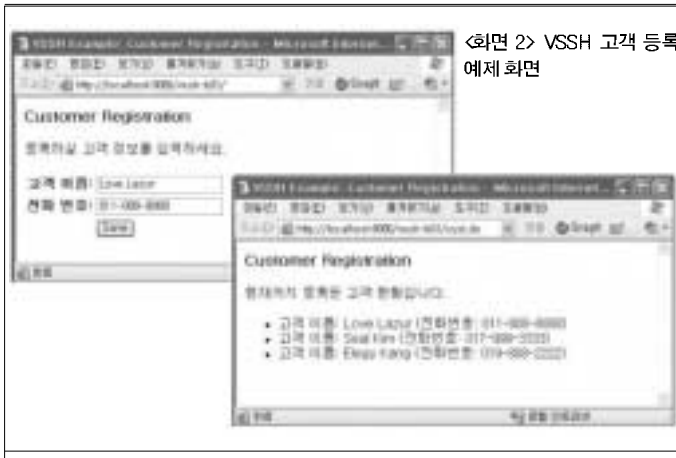
<화면 1> 스프링 WebMVC 예제 화면

<그림 8> 스프링 WebMVC 예제 동작원리



```
<ref bean="product1"/>
<ref bean="product2"/>
<ref bean="product3"/>
</list>
</property>
</bean>
<bean id="urlMapping" class="
org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="/hello.htm">springappController</prop>
    </props>
  </property>
</bean>
... (생략) ...
</beans>
```





〈화면 2〉 VSSH 고객 등록  
예제 화면

전체 애플리케이션의 동작 과정을 한번 살펴보자. 웰컴 파일인 index.jsp에는 hello.htm에 대한 링크가 걸려있다. hello.htm에 대한 요청은 web.xml의 설정에 의해 springapp로 이름지어진 스프링web 패키지의 DispatcherServlet으로 전달된다. DispatcherServlet은 스프링의 core 패키지에 있는 BeanFactory를 이용해 IoC 컨테이너의 기본 원리에 따라 동작된다. 설정 파일인 springapp-servlet.xml을 통해 모든 의존성이 결정되어 제어가 반전되는 현상을 볼 수 있다. 우선 <prop key="/hello.htm">springapp Controller</prop>에 의해 hello.htm 요청을 SpringappController 클래스가 처리하게 되는데 그 사실을 DispatcherServlet은 전혀 모른다. SpringappController는 요청을 처리하는 과정에서 Product Manager의 구현을 필요로 하는데, 그게 어떤 클래스의 인스턴스인지를 본인은 모른다. 설정에 의해 자동적으로 bus.Product Manager가 결정되고, 해당 인스턴스가 거꾸로 SpringappController에 찾아와 연결된다. 이러한 제어의 반전이 스프링을 IoC(Inversion of Control) 컨테이너라고 부르게 하는 이유다.

다음 코드는 스트럿츠의 액션과 유사한 역할을 수행하는 Controller 구현 샘플이다. ProductManager를 사용하고 있는데 신기하게도 setProductManager() 메소드만 있을 뿐, 인스턴스를 초기화하는 호출이 이루어지지 않는다. 스프링이 빈 설정을 참고해서 자동화하기 때문이다. handleRequest()의 결과로 ModelAndView가 리턴되어 넘어가는데, 이것은 스트럿츠의 ActionForward와 컨텍스트 정보를 합친 것으로 이해하면 되겠다.

```
public class SpringappController implements Controller {
    protected final Log logger = LogFactory.getLog(getClass());
    private ProductManager prodMan;
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response)
```

```
        throws ServletException, IOException {
        String now = (new java.util.Date()).toString();
        logger.info("returning hello view with " + now);
        Map myModel = new HashMap();
        myModel.put("now", now);
        myModel.put("products", getProductManager().getProducts());
        return new ModelAndView("hello", "model", myModel);
    }

    public void setProductManager(ProductManager pm) {
        prodMan = pm;
    }

    public ProductManager getProductManager() {
        return prodMan;
    }
}
```

## 예제 2. 스프링과 스트럿츠, 하이버네이트를 통합한 예제

그럼 스프링을 기존의 스트럿츠 환경과 어떻게 연동할 수 있을까? 대표적인 ORM인 하이버네이트와는 어떻게 연동할 수 있을까? 이를 소개하기 위해 간단한 회원 관리 샘플을 개발했다. 이 예제는 벨로시티와 스트럿츠, 스프링, 하이버네이트를 통합하고자 하는 VSSH의 최종적인 모습을 보여주며, DBMS는 HSQLDB를 사용했다. 예제 파일(vssh.zip)을 다운받아 자신의 작업 디렉토리에 압축을 풀고, was.properties 파일을 열어 자신의 운영 환경에 맞게 경로를 수정한다. 그런 다음, ant를 이용해서 makeDist, war-deploy 태스크를 순서대로 실행하면 빌드 및 배포 과정이 끝난다. 웹 브라우저를 이용해서 http://localhost:8080/vssh-b01에 접속하면 〈화면 2〉와 같은 결과 화면을 볼 수 있을 것이다. 이 예제는 다음 호에서 다시 자세히 설명할 예정인데, 미리 상세한 내용을 알고 싶은 분은 LoveLazur의 홈페이지(참고자료 ⑪)를 참고하기 바란다. 이번 예제부터는 복잡하기 때문에 이클립스와 같은 오픈소스 IDE를 이용하는 것이 좋다.

스트럿츠와 스프링의 연동 방법은 각각을 이해하고 있다면 아주 간단히 처리된다. WEB-INF/lib 디렉토리에 각각의 라이브러리 JAR 파일을 추가한 다음, 스트럿츠 설정 파일인 struts-config.xml에 플러그인 설정만 하나 추가하면 되는 것이다. 플러그인으로 설정한 ContextLoader는 전달된 설정 파일을 이용해서 전체 애플리케이션 구성에 사용될 ApplicationContext를 구성하는 역할을 수행한다.

```
<struts-config>
<form-beans> ... </form-beans>
<action-mappings> ... </action-mappings>
<message-resources parameter="messages" />
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property="contextConfigLocation"
```

```
value="/WEB-INF/applicationContext.xml, /WEB-INF/action-servlet.xml"/>
</plug-in>
</struts-config>
```

applicationContext.xml에는 애플리케이션 로직과 관련된 객체들을 앞서 설명한 빈 설정 방식으로 연동해서, 제어 역행화가 이루어지도록 하면 된다.

```
<beans>
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName"><value>org.hsqldb.jdbcDriver</value></property>
<property name="url"><value>jdbc:hsqldb:data/vssbdb</value></property>
<property name="username"><value>sa</value></property>
<property name="password"><value></value></property>
</bean>
<!-- Hibernate SessionFactory -->
<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
<property name="dataSource"><ref local="dataSource"/></property>
<property name="mappingResources">
<list><value>model/Customr.hbm.xml</value></list>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">net.sf.hibernate.dialect.HSQLDialect</prop>
<prop key="hibernate.hbm2ddl.auto">create</prop>
</props>
</property>
</bean>
<!-- Transaction manager , can replace class Attribute ex.Transaction-->
<bean id="transactionManager" class="org.springframework.orm.hibernate.HibernateTransactionManager">
<property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>
<bean id="customerDAO" class="dao.CustomerDAOImpl">
<property name="sessionFactory"><ref local="sessionFactory"/></property>
</bean>
</beans>
```

주의할 점은 모든 클래스에서 사용되는 객체 참조에서 항상 자바 빈 규칙에 따르는 setter()를 만들고, 직접 인스턴스를 초기화하지 않고 객체들의 의존성을 설정 파일에 적어줌으로써 스프링이 자동으로 의존성 관계를 삽입하도록 해야 한다는 것이다. 다음의 Customer Action을 살펴보면 ICustomerBizManager 타입의 cmgr이라는 인스턴스를 갖고 있는데, setCustomermanager()란 메소드만 있을 뿐 어디에도 초기화하는 루틴이 포함되어 있지 않다. 그런데도

cmgr.createCustomer()를 사용하고 있는 것을 볼 수 있다.

```
public class CustomerAction extends BaseAction {
private ICustomerBizManager cmgr = null;
public void setCustomerManager(ICustomerBizManager cmgr) { //def.
this.cmgr = cmgr;
}
public ActionForward list(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response)
throws Exception {
request.setAttribute("custlist", cmgr.getCustomers());
return mapping.findForward("list");
}
public ActionForward create(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response)
throws Exception {
DynaActionForm custForm = (DynaActionForm) form;
cmgr.createCustomer((Customer) custForm.get("cust"));
ActionMessages messages = new ActionMessages();
messages.add(ActionMessages.GLOBAL_MESSAGE, new ActionMessage(
"cust.saved"));
return list(mapping, form, request, response);
}
}
```


이것은 action-servlet.xml과 applicationContext.xml에서 다음과 같은 설정이 되어 있으므로 그 정보에 따라 적절한 customer Manager가 결정되기 때문에 가능해진다.

```
<beans>
<bean name="/cust" class="control.CustomerAction" singleton="false">
<property name="customerManager">
<ref bean="customerBizManager"/>
</property>
</bean>
<bean id="customerBizManager"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionManager"><ref local="transactionManager"/></property>
<property name="target"><ref local="customerBizManagerTarget"/></property>
<property name="transactionAttributes">
<props>
<prop key="create*">PROPAGATION_REQUIRED</prop>
<prop key="update*">PROPAGATION_REQUIRED</prop>
<prop key="delete*">PROPAGATION_REQUIRED</prop>
<prop key="*">PROPAGATION_REQUIRED,readOnly</prop>
</props>
</property>
</bean>
</beans>
```

## 대안 기술에 대해 관심을 갖자

지금까지 스프링 프레임워크의 필요성을 설명하고, 기본 개념이라 할 수 있는 연관성 삽입 패턴과 AOP에 대해 간단히 살펴보았다. 또한 스프링의 특징과 스프링을 활용한 예제를 소개했다. 스프링이 아직 국내에 소개된 적이 없었으니 외국의 문서와 예제, 서적들을 뒤적거리면서 관련 자료를 정리하는데 한달이 넘는 시간이 소요되었다. 하지만 아쉽게도 그동안 정리한 내용을 모두 소개하기에는 할당된 지면이 너무 부족해 꼭 필요한 부분만을 알려주는데 그친 듯하다. 설명이 미흡한 부분은 필자들의 VSSH 포럼을 활용하고, 궁금한 점이 있으면 언제든 질문해 주기 바란다.

솔직히 말해 이 글을 쓰고 있는 필자도 여러분보다 몇 달 먼저 스프링을 공부한 정도의 수준밖에 되지 않는다. 미국, 일본, 중국, 독일 어디서나 스프링 관련 자료를 쉽게 구할 수 있었지만, 안타깝게도 국내에는 전혀 자료가 없었다. 해외에서는 크게 주목을 받고 있는 스프링과 IoC 컨테이너에 대한 내용들이 왜 국내에서는 전혀 다루어지지 않고 있는지를 곰곰이 생각해본다. 우리네 개발자들은 촉박한 개발 일정에 쫓겨 신기술이나 대안 기술들을 공부할 만큼 다들 여유가 없는 것일까? 아니면 그러한 기술을 이미 익히고 있는 개발자들이 자신의 머리 속에 그 지식을 꼭꼭 숨겨두고 공개하지 않는 것일까? 지금 자바 커뮤니티는 주류 기술의 버전 향상과 신기술의 출현 그리고 오픈소스 기반의 다양한 대안 기술의 등장으로 어느 때보다 급격한 기술변화를 눈앞에 두고 있다. 아직 그러한 변화를 느끼지 못하고 있는 개

발자라면 지금이 바로 그러한 변화에 대한 대비를 시작해야 할 때임을 깨달아야 한다. 또한 그 과정에서 얻어진 노하우를 공유함으로써 더 큰 이익이 자신에게 돌아온다는 사실도 잊지 말기 바란다. 

정리 | 강경수 | elegy@korea.net.com



이 + 달 + 의 + 디 + 스 + 켓  
spring.zip <http://www.imaso.co.kr>

## 참 + 고 + 자 + 료

- 1 VSSH 포럼, <http://java.techedu.net/phpBB2>
- 2 코어 J2EE 패턴 (2판), 김중호 역, 피어슨 에듀케이션 코리아
- 3 EJB 디자인 패턴, 이용원, 함태연 역, 인사이트
- 4 오픈소스 자바 프로젝트를 응용한 웹 애플리케이션 개발, 마크 이글,  
[http://cafe.naver.com/deve.cafe?iframe\\_url=/BoardRead.do%3Farticleid=253](http://cafe.naver.com/deve.cafe?iframe_url=/BoardRead.do%3Farticleid=253)
- 5 스프링 프레임워크 홈페이지, <http://www.springframework.org>
- 6 스프링 관련 리소스 모음, <http://java.techedu.net/phpBB2/viewtopic.php?t=117>
- 7 Inversion of Control Containers and the Dependency Injection pattern, Martin Fowler,  
<http://martinfowler.com/articles/injection.html>
- 8 Aspect Oriented Programming(AOP), 김대곤,  
[http://network.hanbitbook.co.kr/view.php?bi\\_id=968](http://network.hanbitbook.co.kr/view.php?bi_id=968)
- 9 Developing a Spring Framework MVC application step-by-step, Thomas Risberg,  
<http://www.springframework.org/docs/MVC-step-by-step/Spring-MVC-step-by-step.html>
- 10 봄? 봄! Spring을 이용한 MVC 애플리케이션 개발, 김승권,  
<http://java.techedu.net/phpBB2/viewtopic.php?t=173>
- 11 Hibernate with Spring, 이종하,  
<http://eclipse.new21.org/phpBB2/viewtopic.php?t=401>



**www.imaso.co.kr**

“매일 마소를 만날 수 있습니다”

통권 250권을 넘은 마소의 감동이 이마소(imaso.co.kr)에서 계속됩니다. 지난 해 오존한 이마소가 마소의 고급 정보를 물론 매일 선선한 뉴스와 개발자들의 소통 등 자잘한 일상까지 담아 매일 업데이트되며 새로워지고 있습니다. 이제, 이마소에서 만나요.