**Servlet 이란?**

웹프로그래밍에서 클라이언트의 요청을 처리하고 그 결과를 다시 클라이언트에게 전송하는 Servlet 클래스의

구현 규칙을 지킨 자바 프로그래밍 기술

예를 들어 사용자가 로그인을 하려고 할 때, 사용자는 아이디와 비밀번호를 입력하고 로그인 버튼을 누른다.

이 때 서버는 클라이언트의 아이디와 비밀번호를 확인하고 다음 페이지를 띄워주어야 하는데 이러한 역할을

수행하는 것이 바로 Servlet이다. 그래서 서블릿은 자바로 구현 된 CGI라고 흔히 말한다.

* 특징

1. 클라이언트의 요청에 대해 동적으로 작동하는 웹 어플리케이션 컴포넌트
2. Html을 사용하여 요청에 응답한다
3. Java Thread를 이용하여 동작한다
4. MVC 패턴에서 Controller로 이용된다.
5. HTTP 프로토콜 서비스를 지원하는 javax.servlet.http.HttpServlet 클래스를 상속받는다.
6. HTML 변경 시 Servlet을 재컴파일 해야 하는 단점이 있다.

일반적으로 웹서버는 정적인 페이지만을 제공한다. 그렇기에 동적인 페이지를 제공하기 위해서 웹서버는 다른

곳에 도움을 요청하여 동적인 페이지를 작성해야 한다. 동적인 페이지로는 임의의 이미지만을 보여주는 페이지와

같이 사용자가 요청한 시점에 페이지를 생성해서 전달해 주는 것을 의미한다. 여기서 웹서버가 동적인 페이지를

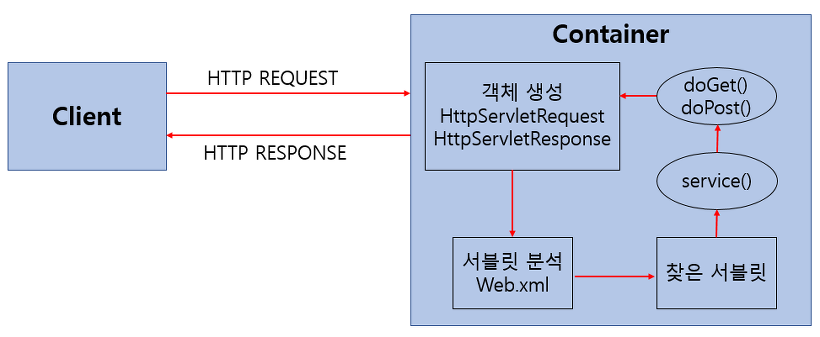
제공할 수 있도록 도와주는 어플리케이션이 서블릿이며, 동적인 페이지를 생성하는 어플리케이션이 CGI이다.

\* CGI(Common Gateway Interface) : 별도로 제작된 웹서버와 프로그램 간의 교환방식이다. 별도로 만들어 놓은

프로그램에 HTML의 Get or Post 방법으로 클라이언트의 데이터를 환경변수로 전달하고, 프로그램의 표준 출력

결과를 클라이언트에게 전송하는 것이다. 즉, 웹 브라우저용 출력 화면을 만드는 방법이다.

**Servlet 동작 방식**



1. 클라이언트가 URL을 클릭하면 HTTP Request를 Servlet Container로 전송한다.
2. HTTP Request를 전송받은 Servlet Container는 HttpServletRequest, HttpServletResponse 객체를 생성한다.
3. web.xml은 사용자가 요청한 URL을 분석하여 어느 서블릿에 대한 요청을 한 것인지 찾는다.
4. 해당 서블릿에서 service 메소드를 호출한 후 클라이언트에 따라 doGet() 또는 doPost()를 호출한다.
5. doGet() 또는 doPost() 메소드는 동적 페이지를 생성한 후 HttpServletResponse 객체에 응답을 보낸다.
6. 응답이 끝나면 HttpServletRequest, HttpServletResponse 객체를 소멸시킨다.

**Servlet Container**

서블릿을 만들었다고 스스로 작동하는 것이 아니고 서블릿을 관리해주는 것이 필요한데 그러한 역할을 하는

것이 바로 서블릿 컨테이너이다. 서블릿 컨테이너는 클라이언트의 요청을 받아주고 응답할 수 있게 웹서버와

소켓을 만들어 통신하며 대표적인 예로 톰캣이 있다. 톰캣은 실제로 웹서버와 통신하여 JSP와 Sevlet이 작동하는

환경을 제공해준다.

* 역할

1. 웹서버와의 통신 지원

서블릿 컨테이너는 서블릿과 웹서버가 손쉽게 통신할 수 있게 한다. 일반적으로 우리는 소켓을 만들고 listen,

accept 등을 해야하지만 서블릿 컨테이너는 이러한 기능을 API로 제공하여 복잡한 과정을 생략할 수 있게

한다. 그래서 개발자가 서블릿에 구현해야 할 비즈니스 로직에 대해서만 초점을 두게 도와준다.

1. 서블릿 생명주기(Life Cycle) 관리

서블릿 컨테이너는 서블릿의 탄생과 죽음을 관리한다. 서블릿 클래스를 로딩하여 인스턴스화하고, 초기화

메소드를 호출하고, 요청이 들어오면 적절한 서블릿 메소드를 호출한다. 또한 서블릿이 생명을 다 한 순간에

적절하게 Garbage Collection을 진행하여 편의를 제공한다.

\* Garbage Collection : 자동 메모리 관리 기능

1. 멀티쓰레드 지원 및 관리

서블릿 컨테이너는 요청이 올때마다 새로운 자바 쓰레드를 하나 생성하는데, HTTp 서비스 메소드를 실행하고

나면, 쓰레드는 자동으로 죽게된다. 원래는 쓰레드를 관리해야 하지만 서버가 다중 쓰레드를 생성 및 운영을

해주고 있어 쓰레드의 안정성에 대해 걱정하지 않아도 된다.

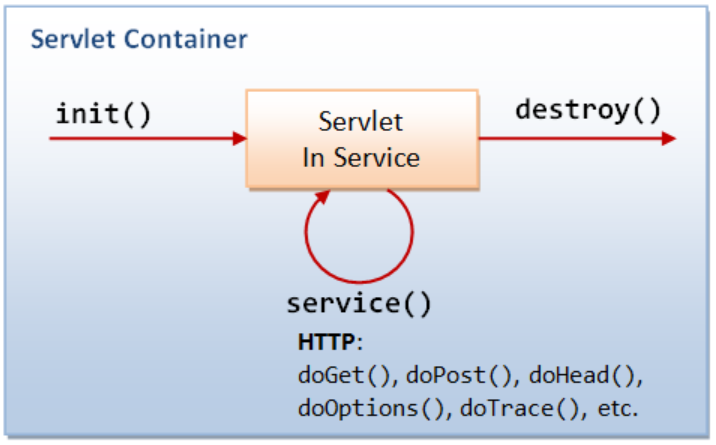
1. 선언적인 보안 관리

서블릿 컨테이너를 사용하면 개발자는 보안에 관련된 내용을 서블릿 또는 자바 클래스에 구현해 놓지

않아도 된다. 일반적으로 보안관리는 XML 배포 서술자에 기록하므로 보안에 대해 수정할 일이 생겨도

자바 소스 코드를 수정하여 다시 컴파일 하지 않아도 보안관리가 가능하다.

**Servlet Container 생명주기**



1. 클라이언트의 요청이 들어오면 컨테이너는 해당 서블릿이 메모리에 있는지 확인하고 없을 경우

Init() 메소드를 호출하여 적재한다. Init() 메소드는 처음 한번만 실행되기 때문에 서블릿의 쓰레드에서

공통적으로 사용해야하는 것이 있다면 오버라이딩하여 구현하면 쉽다. 실핼 중 서블릿이 변경될 경우

기존 서블릿을 파괴하고 init()을 통해 새로운 내용을 다시 메모리에 적재한다.

1. Init()이 호출된 후 클라이언트의 요청에 따라서 service() 메소드를 통해 요청에 대한 응답이 doGet()가

doPost()로 분기된다. 이때 서블릿 컨테이너가 클라이언트의 요청이 오면 가장 먼저 처리하는 과정으로

생성된 HttpServletRequest, HttpServletResponse에 의해 request와 response 객체가 제공된다.

1. 컨테이너가 서블릿에 종료 요청을 하면 destroy() 메소드가 호출되는데 마찬가지로 한번만 실행되며

종료시에 처리해야하는 작업들은 destroy() 메소드를 오버라이딩하여 구현하면 된다.

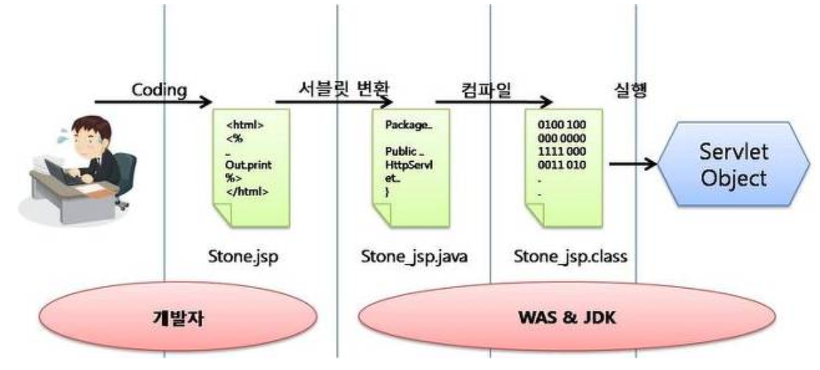
**JSP**

서블릿은 자바 소스코드 속에 HTML 코드가 들어가는 형태인데, JSP는 이와 반대로 HTML 소스코드 속에 자바

소스코드가 들어가는 구조를 갖는 웹어플리케이션 프로그래밍 기술이다. 서블릿 규칙은 꽤나 복잡하기 때문에

JSP가 나오게 되었는데 JSP는 WAS에 의하여 서블릿 클래스로 변환하여 사용되어진다.

**JSP 동작 구조**



1. 웹 서버가 사용자로부터 서블릿에 대한 요청을 받으면 서블릿 컨테이너에 그 요청을 넘긴다.
2. 요청을 받은 컨테이너는 HTTP Request와 HTTP Response 객체를 만들어 이들을 통해 doGet() 또는

doPost() 메소드 중 하나를 호출한다.

1. 만약 서블릿만 사용하여 사용자가 요청한 웹 페이지를 보여주려면 out 객체의 println 메소드를 사용해

HTML 문서를 작성해야 하는데 이는 추가/수정을 어렵게 하고, 가독성 또한 떨어지기 때문에 JSP를

사용하여 비즈니스 로직과 프레젠테이션 로직을 분리한다.

여기서 서블릿은 데이터의 입력, 수정 등에 대한 제어를 JSP에게 넘겨 프레젠테이션 로직을 수행한 후

컨테이너에게 Response를 전달한다.

1. 이렇게 만들어진 결과물은 사용자가 해당 페이지를 요청하면 컴파일이 되어 자바파일을 통해

.class 파일로 만들어지고 두 로직이 결합되어 클래스화 된다.

즉, out 객체의 println 메소드를 사용해서 구현해야 하는 번거로움을 JSP가 대신 수행해준다.

**Transaction**

트랜잭션은 하나의 작업을 수행하기 위해 필요한 데이터베이스의 연산들을 모아 놓은 것으로, 데이터베이스에서

논리적인 작업의 단위가 된다. 트랜잭션은 장애가 발생했을 때 데이터를 복구하는 작업의 단위도 된다.

일반적으로 데이터베이스 연산은 SQL 문으로 표현되므로 트랜잭션 작업 수행에 필요한 SQL 문들의 모임으로

이해하여도 좋다. 하나의 트랜잭션은 Commit되거나 Rollback된다.

\* Commit : Commit 연산은 트랜잭션에 대한 작업이 성공적으로 끝났고 데이터베이스가 다시 일관된 상태에

있을 때, 이 트랜잭션이 행한 갱신 연산이 완료된 것을 트랜잭션 관리자에게 알려주는 연산이다.

\* Rollback : Rollback 연산은 하나의 트랜잭션 처리가 비정상적으로 종료되어 데이터베이스의 일관성을 깨뜨렸을

때, 이 트랜잭션의 일부가 정상적으로 처리되었더라도 트랜잭션의 원자성을 구현하기 위해 이 트랜잭션이 행한

모든 연산을 취소(Undo)하는 연산이다. Rollback시에는 해당 트랜잭션을 재시작하거나 폐기한다.

* 특성

1. Atomicity(원자성)

* 트랜잭션의 연산은 데이터베이스에 모두 반영되거나 전혀 반영되지 않아야 한다.
* 트랜잭션 내의 모든 명령은 반드시 완벽히 수행되어야 하며, 모두가 완벽히 수행되지 않고 하나라도

오류가 발생하면 트랜잭션 전부가 취소되어야 한다.

1. Consistency(일관성)

* 트랜잭션이 그 실행을 성공적으로 완료하면 언제나 일관성 있는 데이터베이스 상태로 변환한다.
* 시스템이 가지고 있는 고정요소는 트랜잭션 수행 전과 트랜잭션 수행 완료 후의 상태가 같아야 한다.

1. Isolation(독립성)

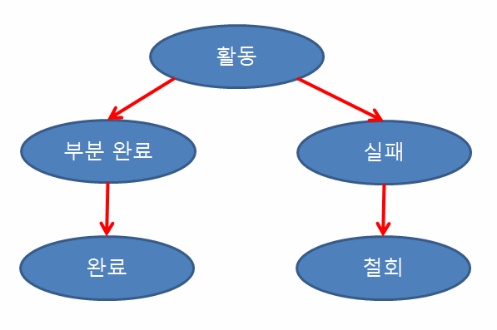
* 둘 이상의 트랜잭션이 동시에 병행 실행되는 경우 어느 하나의 트랜잭션 실행 중 다른 트랜잭션의

연산이 끼어들 수 없다

* 수행중인 트랜잭션은 완전히 완료될 때까지 다른 트랜잭션에서 수행 결과를 참조할 수 없다.

1. Durability(지속성)

* 성공적으로 완료된 트랜잭션의 결과는 시스템이 고장나더라도 영구적으로 반영되어야 한다.
* 트랜잭션의 상태



활동(Active) : 트랜잭션이 실행중인 상태

실패(Failed) : 트랜잭션 실행에 오류가 발생하여 중단된 상태

철회(Aborted) : 트랜잭션이 비정상적으로 종료되어 Rollback 연산을 수행한 상태

부분 완료(Partially Committed) : 트랜잭션의 마지막 연산까지 실행했지만, Commit 연산이 실행되기 직전의 상태

완료(Committed) : 트랜잭션이 성공적으로 종료되어 Commit 연산을 실행한 후의 상태

**Static Pages / Dynamic Pages**

1. Static Pages

* Web Server는 파일 경로 이름을 받아 경로와 일치하는 file contents를 반환한다.
* 항상 동일한 페이지를 반환한다.
* ex) html, css, javascript 와 같이 컴퓨터에 저장되어 있는 파일들

1. Dynamic Pages

* 인자의 내용에 맞게 동적인 contents를 반환한다.
* 웹 서버에 의해서 실행되는 프로그램을 통해서 만들어진 결과물을 반환한다.
* 개발자는 Servlet에 doGet()을 구현한다.

\* doGet() : GET방식으로 정보를 전달했을 시에 이를 처리하는 메소드

**Web Server**

* 개념

: 소프트웨어와 하드웨어로 구분된다.

1. 하드웨어 : Web 서버가 설치되어 있는 컴퓨터
2. 소프트웨어 : 웹 브라우저 클라이언트로부터 HTTP 요청을 받아 정적인 컨텐츠(.html .css .jpeg 등)를

제공하는 컴퓨터 프로그램

* 기능

: HTTP 프로토콜을 기반으로 하여 클라이언트(웹 브라우저 또는 웹 크롤러)의 요청을 서비스하는

기능을 담당한다. 요청에 따라 두 가지의 기능 중 적절하게 선택하여 수행한다.

1. 정적인 컨텐츠 제공한다. WAS를 거치지 않고 바로 제공한다.
2. 동적인 컨텐츠 제공을 위한 요청 전달한다.

클라이언트의 요청(Request)을 WAS에 보내고 WAS가 처리한 결과를 클라이언트에게 전달(Response)한다.

ex) Apache Server, Nginx, IIS(Window 전용 Web Server) 등

**Web Server 종류**

1. Apache

아파치 소프트웨어 재단에서 만드는 웹 서버 프로그램이다. NCSA HTTPd를 기반으로 만들어졌는데,

NCSA HTTPd가 유닉스 기반으로 만들어졌기에 NCSA HTTPd를 아파치는 리눅스에서도 돌리는 것을 목표로

만들어진 프로그램이다. 그 이후 리눅스와 함께 퍼져나갔고 리눅스가 서버 OS의 최다 점유율을 차지하자

아파치도 자연스럽게 최다 점유율을 차지하게 되었다. 현재 거의 모든 리눅스가 이 아파치를 지원한다.

아파치는 확장성이 상당히 좋다. 모듈이라는 개념으로 수많은 기능을 덧붙일 수 있다. 이 모듈을 통해

다른 프로그램과의 연동도 가능하다. 이 때문에 여러가지 서버 사이드 프로그래밍 언어나 DBMS와도 궁합이

잘 맞는다. 그것으로 대부분 PHP를 이용한다. 그리고 오픈소스 DBMS인 MySQL이 나오자

“Apache+PHP+MySQL” 을 통틀어 “APM”이라고 통칭하면서 웹 서버를 위한 기본 3종 세트처럼 되었다.

WAS로 tomcat을 연동하는 경우라면 mkd\_jk, mod\_proxy, mod\_proxy\_ajp 방식을 Apache 자체적으로

지원해주기 때문에 다양하고 효율적으로 tomcat을 연동할 수 있다. (Apache와 Tomcat을 같이 사용하는 이유)

1. Nginx

러시아 프로그래머 이고르 시쇼브가 Apache를 코딩하다 C10K 문제를 보고 이를 극복하기 위해 개발한

오픈소스 웹 서버 프로그램이다. 목표는 가벼우면서도 강력한 프로그램이다.

대부분의 사람들이 Apache를 이용할 때 적은 기능들만을 사용하는데 그러한 기능들은 Nginx에도 그대로

존재하고 속도는 보다 빠르므로 Nginx를 이용하는 사람이 많아지는 추세이다.

Nginx를 사용할 때 환경 설정을 바꾸고 나서 서버 데몬을 재시작할 필요 없이 reload 시그널을 보내면 된다.

즉, 프로세스를 재시작할 필요가 없다는 장점이 있다.

\* C10K 문제 : 10000개 이상의 소켓을 열게 되면 하드웨어 성능이 충분함에도 불구하고 I/O 처리 방식의

문제 때문에 프로세스가 제대로 처리 못하는 문제점

1. IIS(internet Information Services)

마이크로소프트 윈도우에서 사용 가능한 웹 서버 소프트웨어이다. 윈도우 서버에 기본으로 내장되어 있고,

제작사인 마이크로소프트에서 직접 개발하여 제공하는 프로그램으로 윈도우 서버와의 호환성은 최강이다.

덕분에 윈도우에서는 다른 서버 프로그램을 쓰기보다는 이걸 쓰는 경우가 많다.

리눅스에서 Apache라면, 윈도우에서는 IIS가 웹 서버 프로그램을 대표한다.

**WAS(Web Application Server)**

* 개념

: DB 조회나 다양한 로직 처리를 요구하는 동적인 컨텐츠를 제공하기 위해 만들어진 Application Server

HTTP를 통해 컴퓨터나 장치에 애플리케이션을 수행해주는 미들웨어이다.

\* 미들웨어 : 분산 컴퓨팅 환경에서 서로 다른 기종의 하드웨어나 프로토콜, 통신환경 등을 연결하여

응용프로그램과 그 프로그램이 운영되는 환경 간에 원만한 통신이 이루어질 수 있게 하는 소프트웨어이다.

웹 컨테이너(Web Container) 혹은 서블릿 컨테이너(Servlet Container) 라고도 불린다.

\* Container : JSP, Servlet을 실행시킬 수 있는 소프트웨어

* 역할 : WAS = Web Server + Web Container

Web Server 기능들을 구조적으로 분리하여 처리하고자 하는 목적으로 제시되었다.

1. 분산 트랜잭션, 보안, 메시징, 쓰레드 처리 등의 기능을 처리하는 분산 환경에서 사용된다.
2. 주로 DB서버와 같이 수행된다.

현재는 WAS가 가지고 있는 Web Server도 정적인 컨텐츠를 처리하는 데 있어서 성능상 큰 차이가 없다.

* 기능

1. 프로그램 실행 환경과 DB 접속 기능 제공
2. 여러 개의 트랜잭션(논리적인 작업 단위) 관리 가능
3. 업무를 처리하는 비즈니스 로직 수행

ex) Tomcat, JBoss, JEUS 등

**WAS 종류**

1. Tomcat

아파치 소프트웨어 재단에서 개발하는 Java 기반의 서블릿 컨테이너이자 웹 서버이다.

JSP/Spring으로 웹사이트를 구축한다면 톰캣은 거의 반드시 사용된다고 볼 수 있다.

톰캣에는 오로지 서블릿/JSP 및 HTTP 처리 엔진만 들어있다. 톰캣의 존재로 인해 JSP 사용자가 크게 늘었다.

웹서버의 기능이 추가되어 현재는 톰캣만 설치해도 어느정도의 웹서버 역할을 하지만 아직도 JSP를

사용하는 많은 웹 프로그래머들은 아파치+톰캣 혹은 Ngnix+톰캣을 병행하여 사용한다.

톰캣의 정적 데이터 처리 성능도 좋아지고는 있지만 웹서버의 정적 데이터 처리 성능이 더 좋기 때문이다.

1. JBoss

자바를 기반으로 하는 오픈 소스 미들웨어의 총칭이다. 대표적으로 JAVA EE 스펙을 지원한다.

JBoss와 Tomcat은 모두 Java Servlet Application 서버이지만 JBoss의 경우 훨씬 기능이 더 많다.

EJB 및 엔터프라이즈 Java 응용 프로그램에서 작업하는 개발자에게 유용한 기타 기술을 포함하여 완전한

JEE 스택을 제공한다(Tomcat 보다 무겁다). JEE가 제공하는 모든 기능을 필요로 할 때 JBoss를 선택한다.

JEE 스택을 필요로 하지 않는다면 보다 가벼운 Tomcat을 사용하는 것이 좋다.

1. JEUS

제우스는 티맥스소프트사에서 제작한 한국산 웹 애플리케이션 서버이다. 현재 기업에서 사용하는

웹 시스템에 사용되고 있으며, 웹 서버인 웹투비와 같이 사용되고 있다. 국내 시장에서 1위를 차지한다.

제우스의 가장 큰 특징은 기업 업무에서 탁월한 안정성을 보장할 뿐만 아니라 웹 환경에서의 대용량 처리가

가능하며, 복잡한 시스템의 효과적인 구현을 위한 ‘클러스터링(Clustering)’과 ‘부하분산(load balancing)’

기능을 제공한다는 것이다.

**Web Server와 WAS를 구분하는 이유**

1. Web Server : 정적 컨텐츠만 처리하도록 기능을 분배하여 서버의 부담을 줄일 수 있다.
2. WAS : Web Server 만을 이용한다면 정적이기 때문에 사용자가 원하는 요청에 대한 결과값을 모두 미리

만들어 놓고 서비스를 해야 한다. 이렇게 수행하기에는 자원이 절대적으로 부족하다.

따라서 WAS를 통해 요청에 맞는 데이터를 DB에서 가져와 동적으로 결과를 만들어 제공함으로써 자원을

효율적으로 사용할 수 있다.

* WAS가 Web Server의 기능까지 모두 수행하지 않는 이유

1. 기능을 분리하여 서버 부하 방지

* WAS는 기본적으로 동적 컨텐츠를 제공하기 위해 존재하는 서버이다.
* 정적 컨텐츠 요청까지 WAS가 처리한다면 정적 데이터 처리로 인해 부하가 커지게 되고,

동적 컨텐츠의 처리가 지연됨에 따라 수행 속도가 느려진다.

즉, 이로 인해 페이지 노출 시간이 늘어나게 될 것이다.

1. 물리적으로 분리하여 보안 강화

* SSL에 대한 암복호화 처리에 Web Server를 사용

1. 여러 대의 WAS를 연결 가능

* Load Balancing을 위해서 Web Server를 사용
* Fail over(장애 극복), fail back 처리에 유리
* 특히 대용량 웹 어플리케이션의 경우 Web Server와 WAS를 분리하여 무중단운영을 위한

장애 극복에 쉽게 대응할 수 있다.

예를 들어, 앞 단의 Web Server에서 오류가 발생한 WAS를 이용하지 못하도록 한 후 WAS를

재시작함으로써 사용자는 오류를 느끼지 못하고 이용할 수 있다.

1. 여러 웹 어플리케이션 서비스 가능

* 하나의 서버에서 PHP Application과 Java Application을 함께 사용하는 경우

이외에도 접근 허용 IP 관리, 2대 이상의 서버에서 세션 관리 등도 Web Server에서 처리하면 효율적이다.

즉, 자원 이용의 효율성 및 장애 극복, 배포 및 유지보수의 편의성을 위해 WEB과 WAS를 분리한다.

**SSL(Secure Sockets Layer)**

월드 와이드 웹 브라우저와 웹 서버 간에 데이터를 안전하게 주고받기 위한 업계 표준 프로토콜이다.

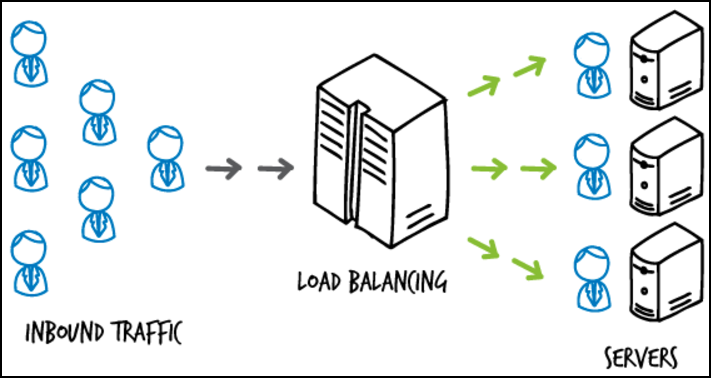
SSL은 웹 제품뿐만 아니라 파일 전송 규약(FTP) 등 다른 TCP/IP 애플리케이션에 적용할 수 있으며, 인증 암호화

기능이 있다. 인증은 웹 브라우저와 웹 서버 간에 서로 상대의 신원을 확인하는 기능이다.

* 작동 원리



**Load Balancing(부하분산)**



컴퓨터 네트워크 기술의 일종으로 둘 혹은 셋 이상의 중앙처리장치 혹은 저장장치와 같은 컴퓨터 자원들에게

작업을 나누는 것을 의미한다. 이로써 가용성 및 응답시간을 최적화 시킬 수 있다.

부하분산을 위한 대부분의 응용 프로그램은 다수의 서버를 가지고 한 가지 종류의 인터넷 서비스를 지원하는

방식이다. 보통 부하 분산은 트래픽이 많은 웹 사이트, IRC 네트워크, FTP 사이트, NNTP 서버 그리고 DNS 서버에

적용되고 있다.

인터넷 서비스를 위해서는 소프트웨어를 이용한 부하분산이 적용되며, 이 소프트웨어는 중간에 위치해 실제

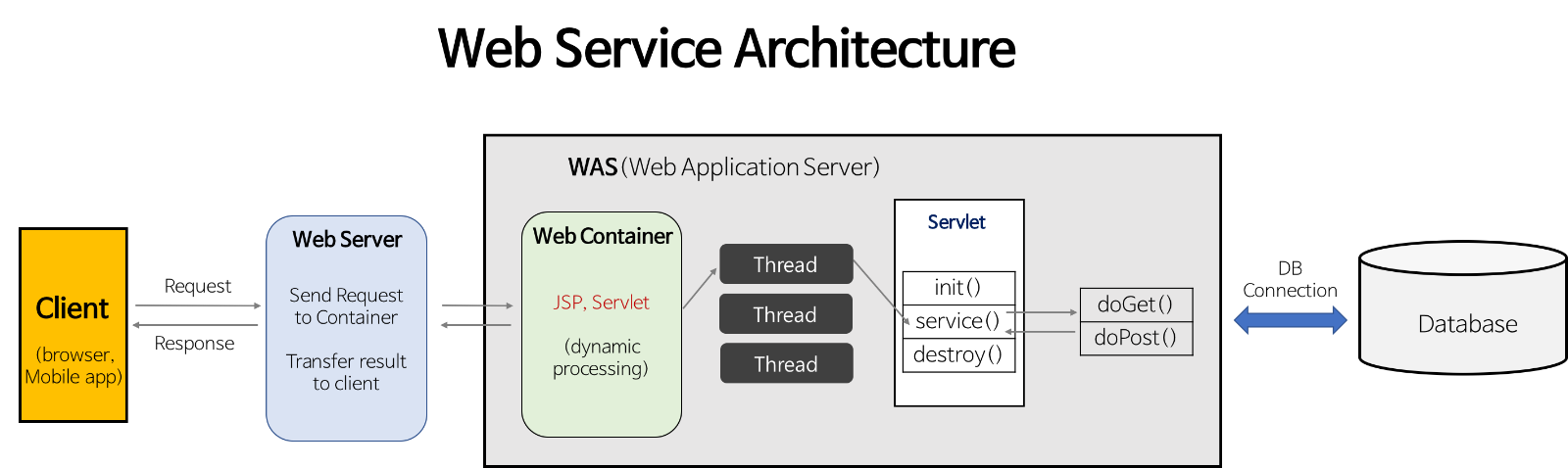
서비스하는 서버와 클라이언트를 포트를 이용해 중개하고 있다. 그러나 사용자는 이를 알아차리지 못하고 이러한

것을 투명성이라고 한다. 또한 보안이라는 측면에서 내부 네트워크 구조를 숨김으로써 크래킹을 막을 수 있고

웹서버의 부하를 나눠 갖음으로써 DDoS와 같은 공격으로부터 보다 안전하다.

로드밸런서에도 부하가 많이 물리면 병목현상(Bottle Neck)이 생길 수 있으므로 이중화 하는 것이 좋다.

\* 병목현상 : 시스템의 가용 자원 중 부하가 많이 걸려 전체 시스템 효율의 저하를 초래하는 현상



1. Web Server는 웹 브라우저 클라이언트로부터 HTTP 요청을 받는다.
2. Web Server는 클라이언트의 요청(Request)을 WAS에 보낸다.
3. WAS는 관련된 Servlet을 메모리에 올린다.
4. WAS는 web.xml을 참조하여 해당 Servlet에 대한 Thread를 생성한다. (Thread Pool 이용)

\* Thread Pool : 작업 처리에 사용되는 스레드를 제한된 개수만큼 정해 놓고 작업 큐에 들어오는

작업들을 하나씩 스레드가 맡아 처리하는 것을 말한다.

1. HttpdServletRequest와 HttpdServiceResponse 객체를 생성하여 Servlet에 전달한다.

* Thread는 Servlet의 service() 메소드를 호출한다.
* Service() 메소드는 요청에 맞게 doGet() 또는 doPost() 메소드를 호출한다.

1. doGet() 또는 doPost() 메소드는 인자에 맞게 생성된 적절한 동적 페이지를 Response 객체에 담아

WAS에 전달한다.

1. Was는 Response 객체를 HttpResponse 형태로 바꾸어 Web Server에 전달한다.
2. 생성된 Thread를 종료하고 HttpServletRequest와 HttpServletResponse 객체를 제거한다.

**Spring을 사용하는 이유(구성요소)** \* Spring : JAVA / JSP 기반의 오픈소스 웹 프레임워크

: JSP, Servlet, PHP 등등 그냥 웹을 개발할 수 있는데 Spring이라는 프레임워크를 사용해야하는 이유가 무엇일까?

시스템 개발 시 비즈니스 로직 외에도 고려할 사항들(타시스템과의 연계, 분산 트랜잭션 지원, 보안 등)이 많다.

또한 개발이 진행됨에 따라 비즈니스 로직이 점점 복잡해지고 잦은 변경이 요구되는데 스프링은 이와 같은

복잡함을 해결하기 위한 다양한 특징을 가지고 있다.

\* 비즈니스 로직 : 업무에 필요한 데이터 처리를 수행하는 응용 프로그램의 일부를 말하며 데이터 입력, 수정,

조회 및 보고서 처리 등을 수행하는 루틴. 즉, 보이는 것의 뒤에서 일어나는 각종 처리를 의미한다.

* 특징

1. 경량 Container

• Java 객체를 담고 있으며, 생성 ~ 소멸과 같은 라이프 사이클을 관리하고 Spring Container로부터 필요한

객체를 가져와 사용한다.

• 스프링에서는 BeanFactory와 이를 상속한 ApplicationContext 두 가지 유형의 컨테이너를 제공한다

BeanFactory : 스프링 설정 파일(applicationContext.xml)에 등록된 <bean> 객체를 생성하고 관리하는

가장 기본적인 컨테이너 기능만을 제공한다. 컨테이너가 구동될 때 객체를 생성하는 것이 아니라

클라이언트 요청에 의해서만 객체를 생성한다. (lazy loading)

ApplicationContext : BeanFactory를 확장한 컨테이너로 트랜잭션 관리나 메시지 기반의 다국어 처리 등

다양한 기능을 지원한다. 또한 컨테이너가 구동되는 시점에 <bean>에 등록되어 있는 클래스들을

객체화하는 즉시 로딩 방식으로 동작한다. (pre-loading)

1. DI(Dependency Injection)

• 프로그래밍에서 의존성은 객체와 객체 간의 의존을 의미한다.

• 기본 방식으로 메소드를 다른 클래스에서 같이 사용할 때 직접적으로 연결되어 있어 메소드의 이름이

변경된다면 다른 클래스에서 메소드를 찾을 수 없기 때문에 똑같이 변경해주어야 한다.

• 그래서 Spring Framework에서 제공하는 DI를 이용하게 된다.

DI의 역할은 기본 방식의 의존성을 제거하고, 별도로 3자가 만들어주는 의존객체를 각 클래스에

뿌려주는 기능으로 변경의 유연성을 제공한다.

• DI로 프로그램을 설계하면 얻는 이점으로는 Unit Test가 용이해진다, 코드의 재활용성을 높여준다,

객체 간의 의존성(종속성)을 줄이거나 없앨 수 있다, 객체 간의 결합도를 낮추면서 유연한 코드를

작성할 수 있다.

1. AOP(Aspect Oriented Programming)

• 한 어플리케이션 내의 다양한 모듈에서 공통적으로 이용되는 기능을 분리시켜 사용하는 것이다.

• 모듈의 핵심 기능 외의 기능을 해당 모듈에 응집되지 않도록 하기 위한 기술이다.

• Java는 하나의 클래스에 대한 수직적인 흐름만 제어할 수 있지만 Spring을 이용하면 특정 클래스들에

대한 수평적인 제어가 가능해진다.

1. POJO(Plain Old Java Object)

• 특정 기술에 종속되어 동작하는 것이 아닌 순수한 자바의 객체지향적 특징을 살리는 것을 말한다.

• 코드의 간결함 (비즈니스 로직과 특정 환경 / low레벨의 종속적인 코드를 분리하기 때문)

자동화 테스트에 유리함, 객체지향적 설계의 자유로운 사용이 가능함

즉, 특정 규약과 환경에 종속되지 않고 객체지향적 특징에 충실하다.

• Spring에서 POJO를 ‘Beans’ 라고 부른다.

• Beans는 애플리케이션의 핵심을 이루는 객체이며, Spring IoC(Inversion of Control) 컨테이너에 의해

인스턴스화, 관리, 생성된다. (자주 사용하는 객체를 singleton으로 만들고 어디서든 불러 쓸 수 있다.)

Beans는 우리가 컨테이너에 공급하는 설정 메타 데이터(XML 파일)에 의해 생성된다.

(컨테이너는 이 메타 데이터를 통해 Bean의 생성, Bean Life Cycle, Bean Dependency 등을 알 수 있다.)

애플리케이션의 객체가 지정되면 해당 객체는 getBean() 메서드를 통해 가져올 수 있다.

1. PSA(Portable Service Abstraction)

• 직역하면 일관성 있는 서비스 추상화이다.

JDBC처럼 같은 일을 하는 다수의 기술을 공통의 인터페이스로 제어할 수 있게 한 것을 서비스 추상화

라고 한다.

스프링 프레임워크에서는 서비스 추상화를 위해 다양한 어댑터를 제공한다.

OXM, ORM, 캐시, 트랜잭션 등 다양한 기술에 대한 PSA 즉, API를 제공한다.

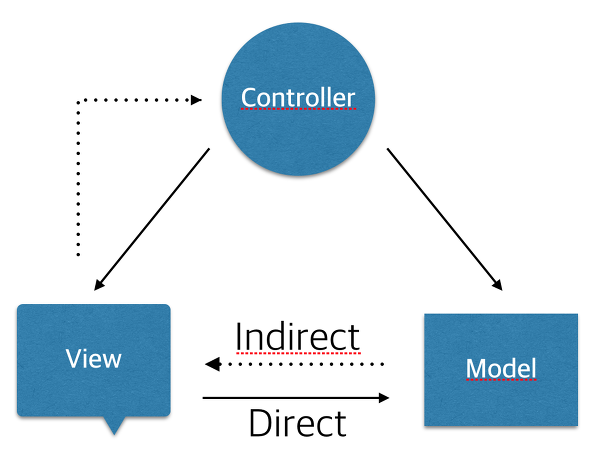
**MVC(Moder-View-Controller)**

개발할 때, 3가지 형태로 역할을 나누어 개발하는 방법론이다.

비즈니스 처리 로직과 사용자 인터페이스 요소를 분리시켜 서로 영향없이 개발하기 수월하다는 장점이 있다.

다시 말해 디자이너와 개발자의 영역이 분리될 수 있다.

그에 반해 View와 Model 사이의 의존성이 높다는 단점이 있다.



* Model : 핵심 데이터와 기능을 캡슐화. 데이터를 가지고 있는 객체 또는 JAVA POJO이다.

상태의 변화가 있을 때 컨트롤러와 뷰에 이를 통보함, 통보를 통해 뷰는 최신의 결과를 출력,

컨트롤러는 모델의 변화에 따른 적용 가능한 명령을 추가, 제거, 수정할 수 있다.

* View : 사용자에게 정보를 Display, 각 뷰마다 컴포넌트 하나씩 연결된다. 사용자는 오직 컨트롤러를

통해서만 시스템과 상호작용한다. 모델로부터 결과물을 생성한다.

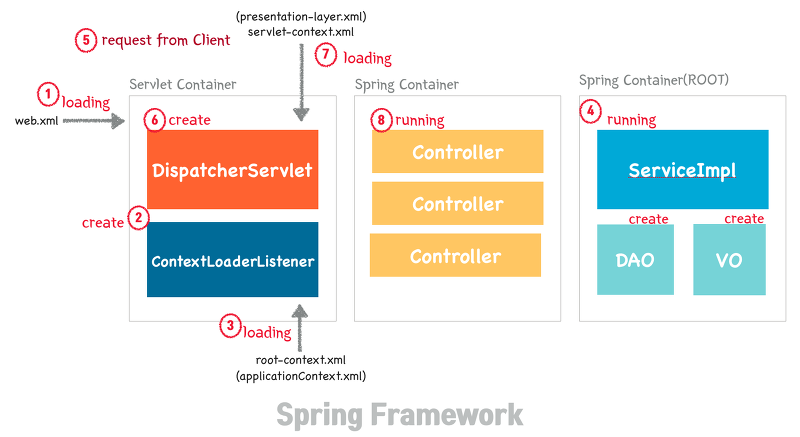
ex) 웹 어플리케이션의 HTML, 안드로이드의 xml

* Controller : 데이터와 비즈니스 로직 사이의 상호동작을 관리한다. 모델 객체로의 데이터 흐름을

제어하고 데이터가 update 되었을 때 뷰를 갱신한다. 컨트롤러는 뷰와 모델의 역할을 분리한다.

모델에 명령을 보내 모델의 상태를 변경할 수 있다

**Spring 실행순서**



1. 웹 어플리케이션이 실행되면 Tomcat(WAS)에 의해 web.xml이 loading된다.
2. web.xml에 등록되어 있는 ContextLoaderListener(Java Class)가 생성된다.

ContextLoaderListener 클래스는 ServletContextListener 인터페이스를 구현하고 있으며,

ApplicationContext를 생성하는 역할을 수행한다. 또한 Servlet의 생명주기를 관리해준다.

1. 생성된 ContextLoaderListener는 root-context.xml을 loading한다.
2. root-context.xml에 등록되어 있는 Spring Container가 구동된다.

이 때 개발자가 작성한 비즈니스 로직에 대한 부분과 DAO, VO 객체들이 생성된다.

1. 클라이언트로부터 웹 애플리케이션 요청이 온다.
2. DispatcherServlet이 생성된다.
3. DispatchServlet은 servlet-context.xml을 loading한다.
4. 두번째 Spring Container가 구동되며 응답에 맞는 PageController들이 동작한다.

이 때, 첫번째 Spring Container가 구동되면서 생성된 DAO, VO, Servicelmpl 클래스들과 협업하여 알맞은

작업을 처리하게 된다.

ContextLoaderListener : web.xml에서 모두 load되도록 등록할 때 사용된다. 서블릿 이전에 서블릿 초기화하는

용도로 쓰이며, contextConfigLocation 파라미터를 쓰면 ContextLoader가 load 할 수 있는 설정파일을 쓴다.

web.xml에 이 문장들이 빠지면 default로 /WEB-INF/applicationContext.xml을 설정파일로 사용한다.

DispatcherServlet : Servlet Container에서 HTTP프로토콜을 통해 들어오는 모든 요청을 프레젠테이션 계층의

제일 앞에 둬서 중앙집중식으로 처리해주는 프론트 컨트롤러(Front Controller)이다.

클라이언트로부터 어떠한 요청이 오면 Tomcat과 같은 서블릿컨테이너가 요청을 받는데, 이때 제일 앞에서

서버로 들어오는 모든 요청을 처리하는 프론트 컨트롤러를 Spring에서 정의하였고 이를 DispatchServlet이라고

한다. 그래서 공통처리 작업을 Dispatch 서블릿이 처리한 후 적절한 세부 컨트롤러로 작업을 위임해준다.

ContextLoaderListener와 DispatcherServlet은 각각 WebApplicationContext 인스턴스를 생성하게 되는데,

ContextLoaderListener가 생성한 컨텍스트가 root 컨텍스트가 되고, DispatcherServlet이 생성한 인스턴스는

Root 컨텍스트를 부모로 사용하는 자식 컨텍스트가 된다. 이 때 자식 컨텍스트들은 root 컨텍스트가 제공하는

빈을 사용할 수 있기 때문에 ContextLoaderListener를 용하여 공통빈을 설정하는 것이다.

ServiceImpl : 비즈니스로직을 수행하는 역할을 한다. 여기서 비즈니스 로직은 기능을 구현하는 구현부이다.

예를 들어 글 작성, 글 수정, 글 삭제, 글 조회 등 기능을 비즈니스 로직이라고 하며, 비즈니스 로직을

수행하기 위해 Service는 인터페이스(Interface) 파일로, ServiceImpl은 클래스(Class) 파일로 작성한다.

DAO(Data Access Object) : Database의 data에 접근을 위한 객체이다.

웹서버는 DB와 연결하기 위해 매번 커넥션 객체를 생성하는데 이것을 해결하기 위해 ConnectionPool을

이용한다. ConnectionPool이란 커넥션 객체를 미리 만들어 놓고 가져다 쓰는 것이다. 커넥션풀은

커넥션을 또 만드는 오버헤드를 효율적으로 하기 위해 DB에 접속하는 객체를 전용으로 하나만을 만들고

모든 페이지에서 그 객체를 호출해 사용하면 커넥션 하나만으로 모든 DB와의 연결을 하는 것이 DAO 객체이다.

DTO(Data Transfer Object) / VO(Value Object) : 계층 간 데이터 교환을 위한 객체(자바빈즈)를 말한다.

DB에서 데이터를 얻어 Service나 Controller 등으로 보낼 때 사용하는 객체이다.

즉, 데이터가 Presentation Logic Tier로 넘어오게 될 때는 DTO의 모습으로 바뀌어 오고가는 것이다.

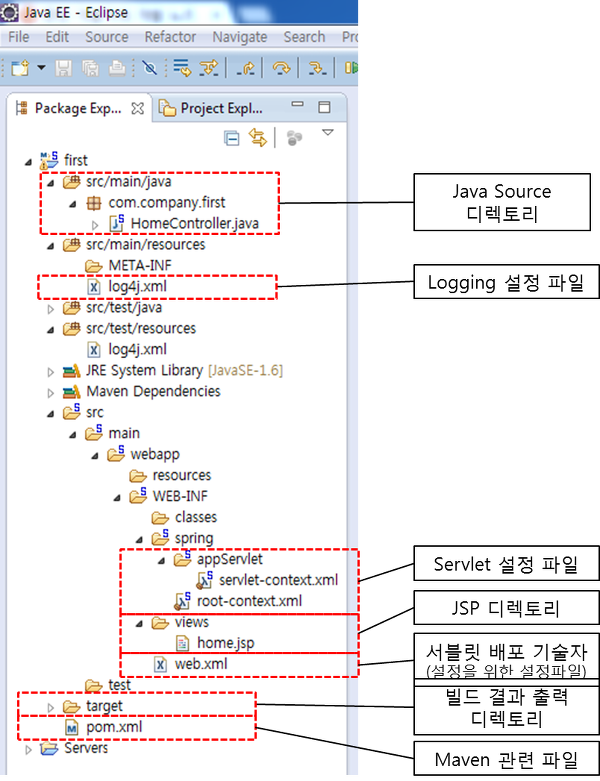
로직을 갖고 있지 않는 순수한 데이터 객체이며, getter/setter 메소드만을 갖는다.

하지만 DB에서 꺼낸 값을 임의로 변경할 필요가 없기 때문에 DTO클래스에는 setter가 없다.

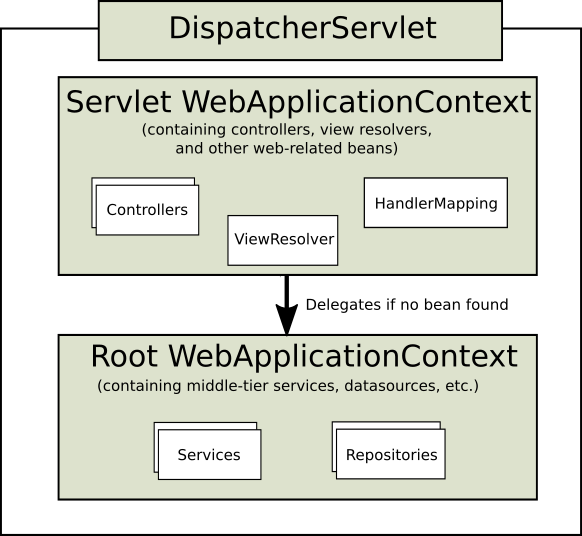
VO는 DTO와 동일한 개념이지만 read only 속성을 갖는다.

VO는 특정한 비즈니스 값을 담는 객체이고, DTO는 Layer간의 통신 용도로 오고가는 객체를 말한다.

**Spring 폴더구조**



1. src/main/java : java 파일이 모여 있는 디렉토리이다.
2. src/main/resources : java 코드에서 사용하기 위한 리소스(mapper, ms 등) 파일을 모아서 관리한다.
3. src/main/webapp : 메이븐의 기본 폴더로 webapp 폴더 밑에 모든 jsp 및 js 등의 파일이 포함된다.
4. servlet-context.xml, root-context.xml : 서블릿 관련 설정파일이다.
5. pom.xml : maven에서 참조하는 설정파일이다.



Xml 파일은 모두 객체(Bean)를 정의한다.

web.xml : 웹프로젝트의 배치 기술서(환경설정파일)로 스프링 프로젝트가 실행되면 가장 먼저 web.xml을 읽어

들이게 되고 위에서부터 차례로 태그를 해석하기 시작한다.

root-context.xml : 스프링의 환경설정 파일로 별다른 내용을 작성하지 않으면 web.xml에서는 root-context.xml을

건너 뛰게 된다. 공통빈을 설정하는 곳으로 주로 View 지원을 제외한 bean을 설정한다.

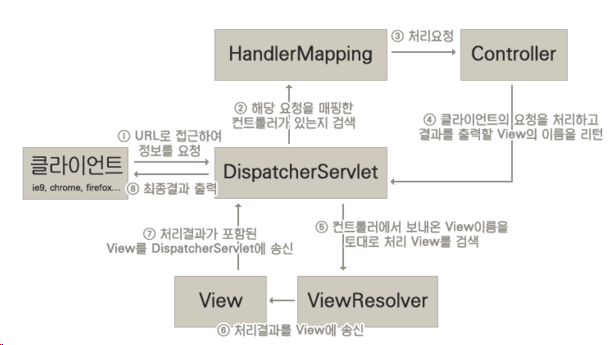
ex) Service, Repository(DAO), DB 등등

servlet-context.xml : web.xml에서 DispatcherServlet로 이동하게 되고 이 파일을 참조하게 된다.

요청과 관련 객체를 정의한다. 주로 View 지원 bean을 설정한다. DispatcherServlet과 관련된 설정을 한다.

ex) Controller, ViewResolver, HandlerMapping, Interceptor, MultipartResolver 등등

위 폴더에서의 동작 원리



1. 클라이언트 요청
2. web.xml에서 Dispatcher Servlet 클라이언트 요청을 핸들링
3. servlet-context.xml에서 해당 클래스의 웹요청을 처리하는 컨트롤러를 사용

(HandlerMapping으로 Controller 검색)

1. 해당 Controller가 요청 처리 후 Home(View)로 리턴. 뷰의 이름은 /WEB-INF/view/home.jsp로 바뀐다.
2. View에 출력

**Maven**

자바용 프로젝트 관리도구로 Apache License로 배포되는 오픈 소스 소프트웨어이다.

필요한 라이브러리를 특정 문서(pom.xml)에 정의해 놓으면 네트워크를 통해 라이브러리들을 자동으로

다운로드받아준다. 그러므로 개발자들이 jar 파일을 공유하고 카톡으로 보내는 등의 쓸모없는 짓을 할 필요 없이

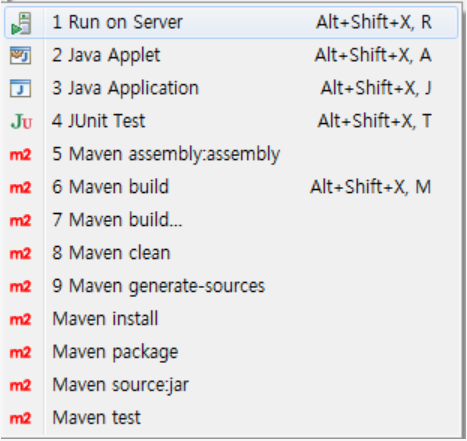
pom.xml에 필요한 것을 명시해놓으면 네트워크를 통해 라이브러리를 매우 손쉽게 다운받을 수 있다.

* LifeCycle

메이븐은 프레임워크이기 때문에 동작 방식이 정해져있는데, 일련의 단계(Phase)에 연계된 Goal을 실행하는

과정을 Build라고 하며 미리 정의되어 있는 Build들의 순서를 라이프사이클(LifeCycle)이라 한다.

즉, 미리 정의된 빌드순서를 라이프사이클이라 하고, 각 빌드 단계를 Phase라고 한다.



Maven build : 메이븐 빌드를 실행한다.

Maven build… : 새로운 사용자 빌드를 만든다.

Maven clean : target에 지정된 모든 소스를 삭제한다.

Maven generate-sources : 컴파일 과정에 포함될 소스를 생성한다.

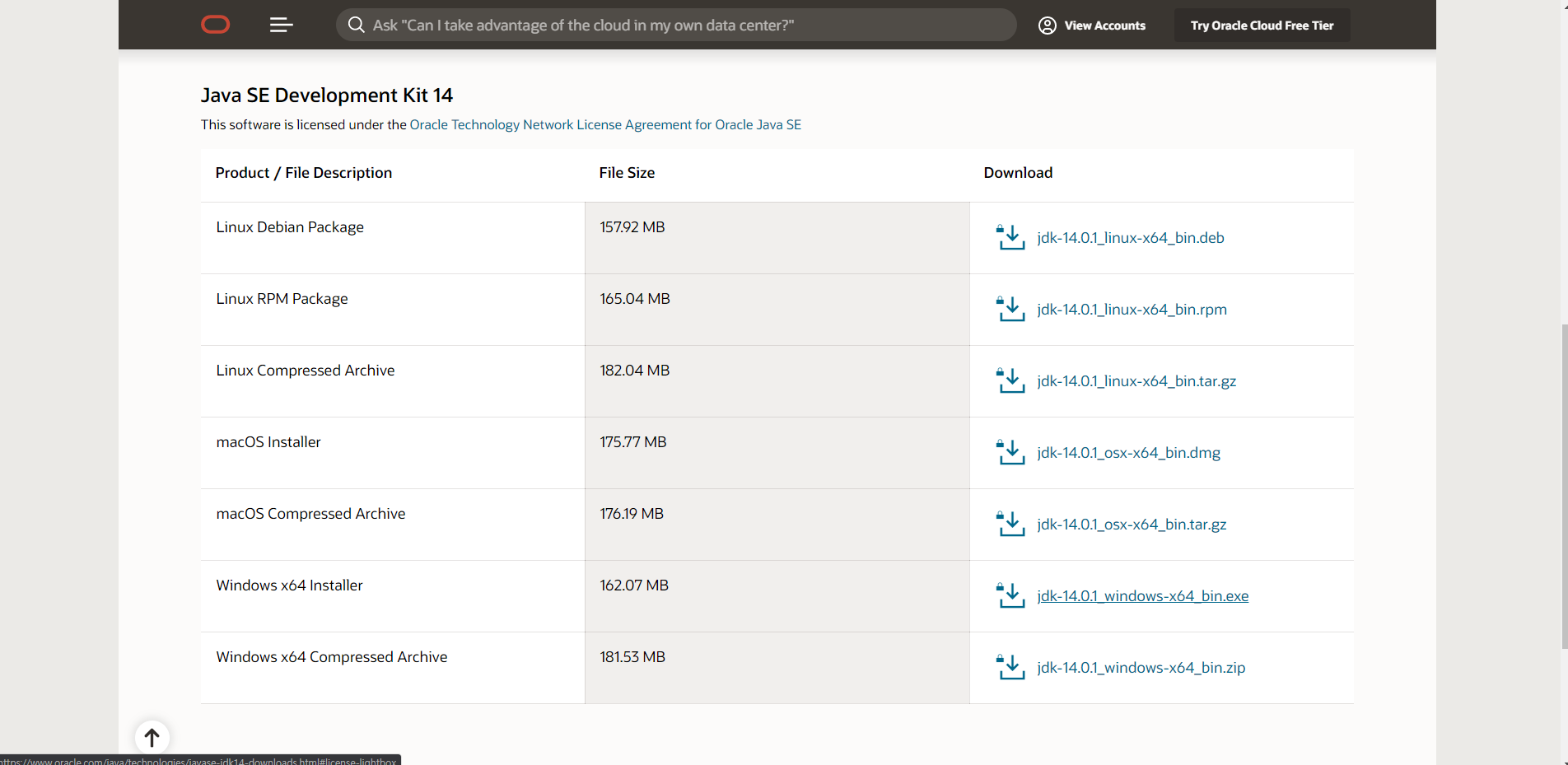
Maven install : Local Repository에 패키지를 복사한다.

**Spring 개발 환경 구성**

1. C드라이브에서 dev폴더 생성(C:\dev)
2. JDK 설치

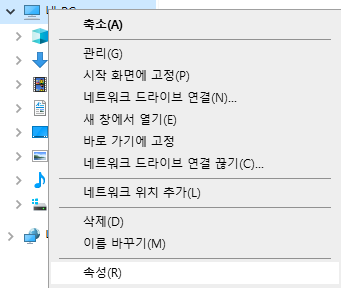
C:\dev 에서 jdk-14.0.1 폴더 생성

<https://www.oracle.com/java/technologies/javase-jdk14-downloads.html>

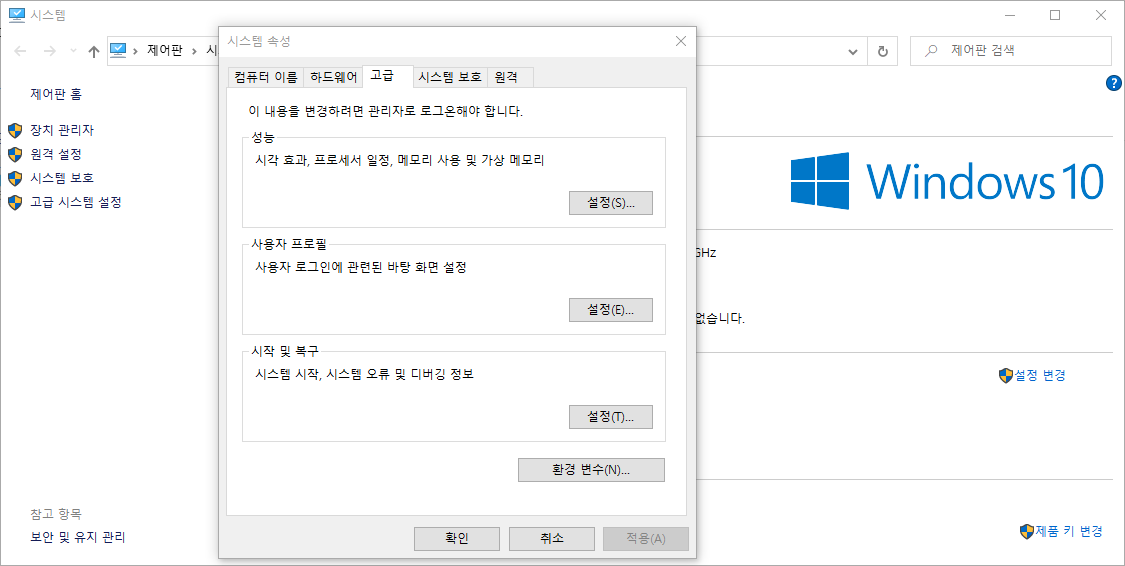


Windows x64 Installer 다운로드 후 실행 (설치 중 경로를 C:\dev\jdk-14.0.1로 설정)

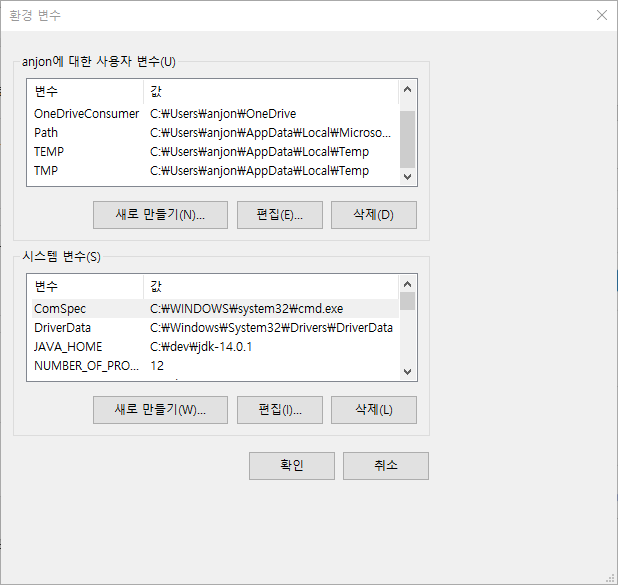
* 환경변수 설정



내 PC 우클릭 -> 속성을 누른다.

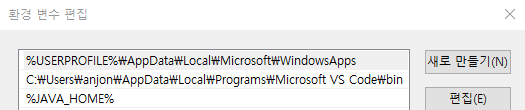


고급 시스템 설정을 누른다 -> 환경 변수를 누른다.



시스템 변수에서 새로 만들기를 누르고 변수 이름은 JAVA\_HOME, 변수 값은 C:\dev\jdk-14.0.1로 설정해준다.

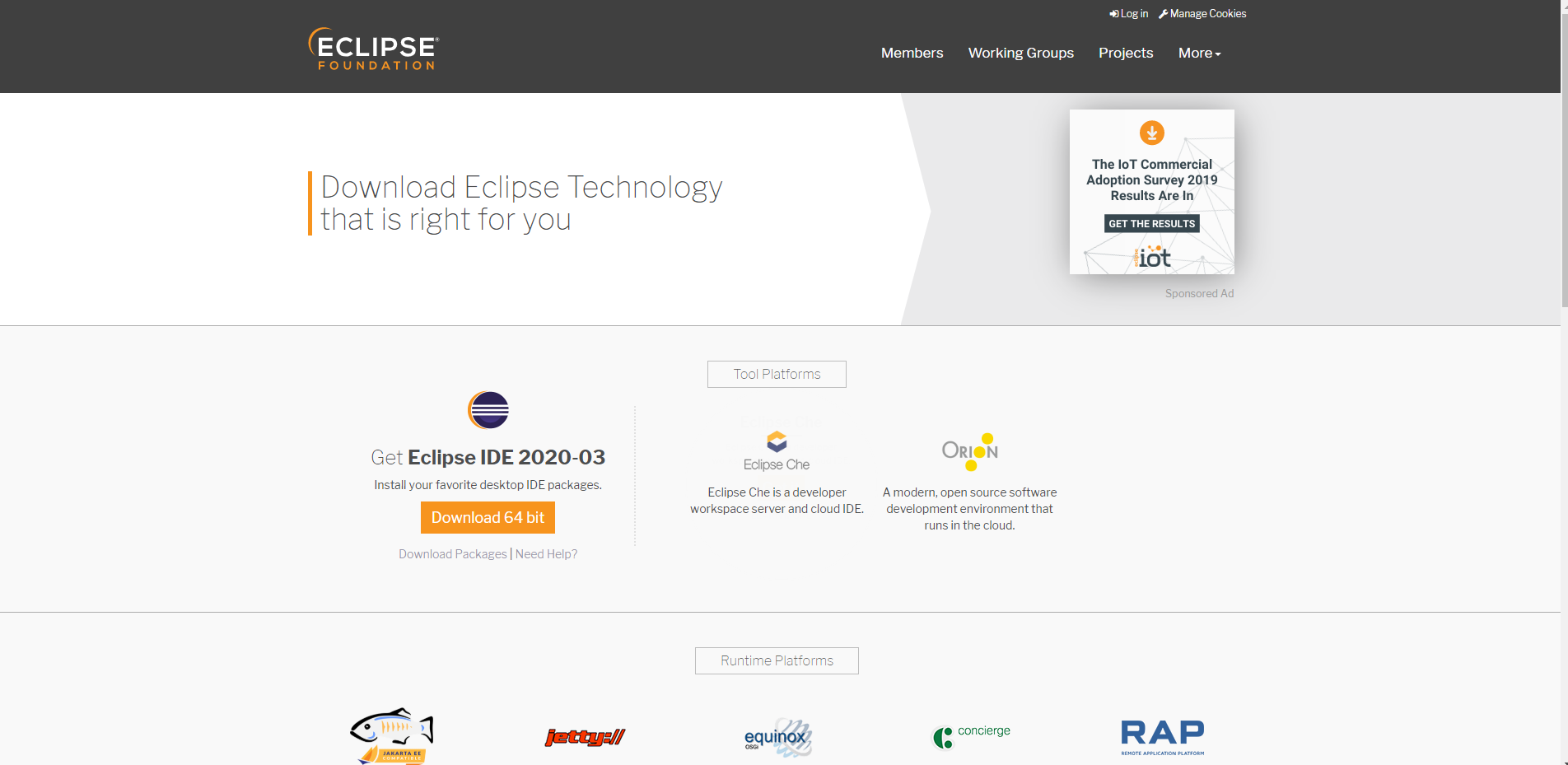
그리고 사용자 변수에서 Path를 눌러 편집한다. (새로 만들기를 눌러 %JAVA\_HOME%을 추가)



1. Eclipse 설치

C:\dev에서 Eclipse에서 사용할 workspace를 만들어준다. (C:\dev\workspace)

<https://www.eclipse.org/downloads/>



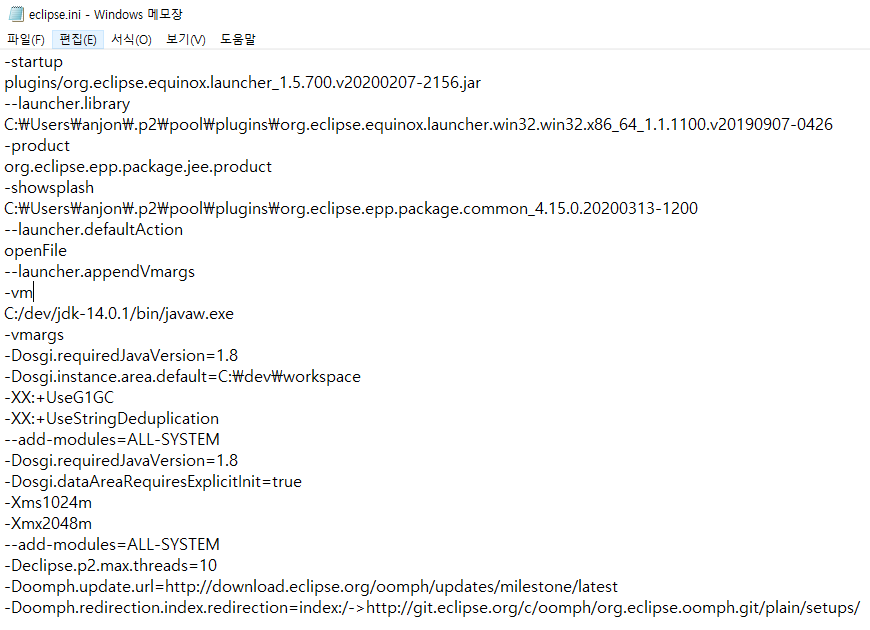
Download 64 bit 다운로드 후 실행



설치 중 선택지에서는 Eclipse IDE for Enterprise Java Developers를 선택, 경로는 C:\dev로 설정한다.

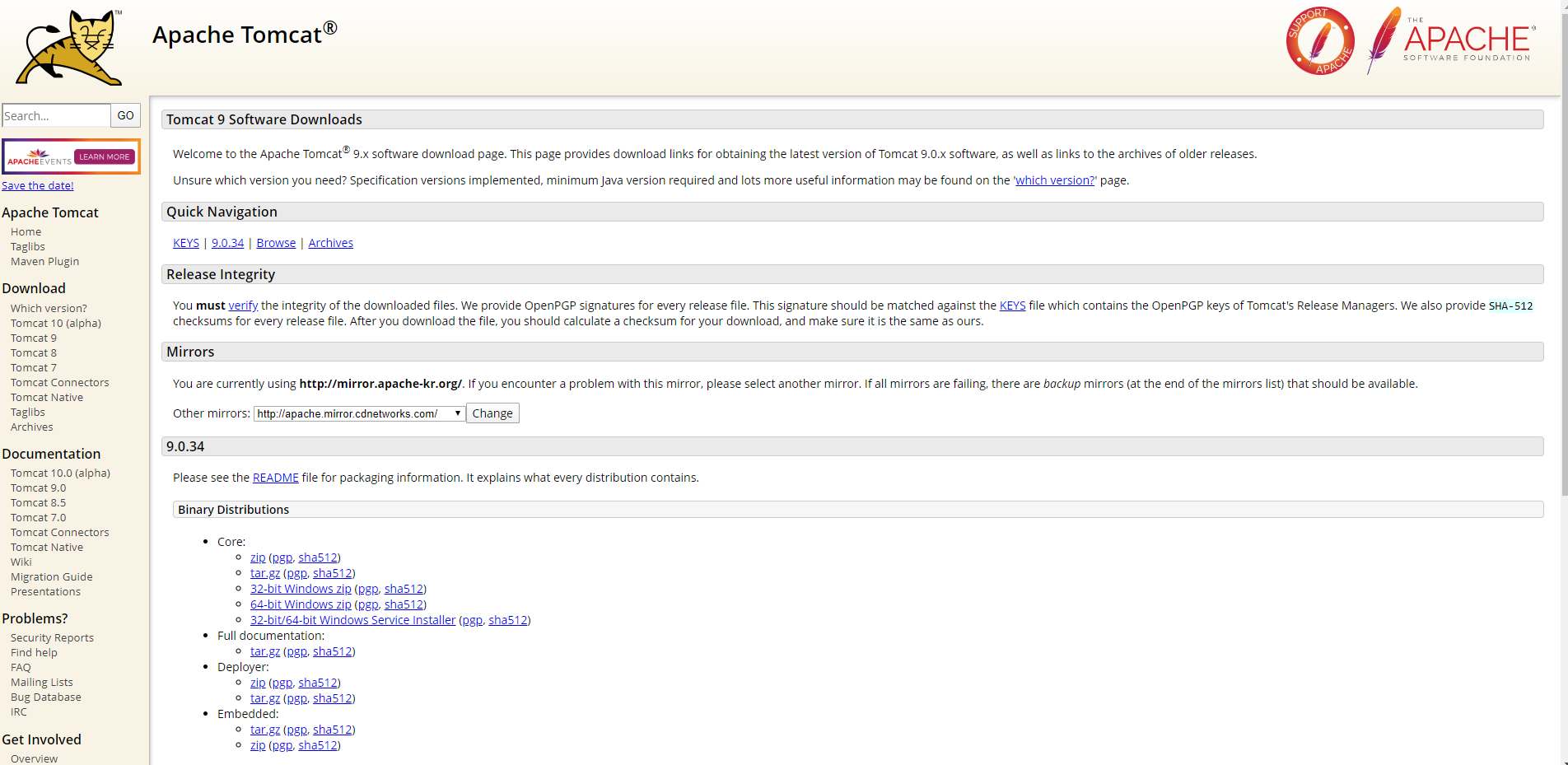
* 경로 설정

C:\dev\eclipse의 eclipse.ini 파일을 열고 아래의 사진과 같이 -vm C:\dev\jdk-14.0.1\bin\javaw.exe를 추가한다.



1. Tomcat 설치

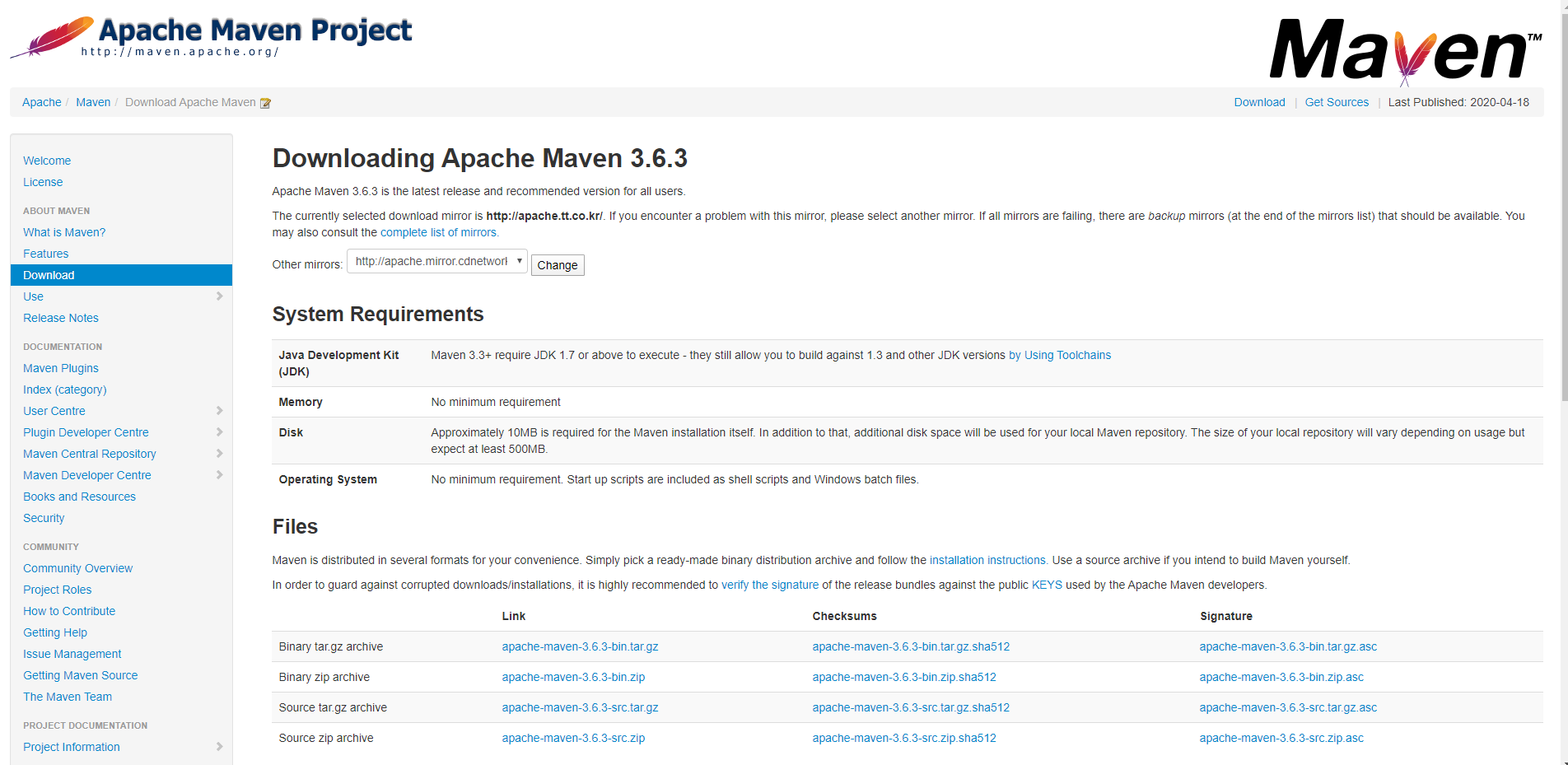
<https://tomcat.apache.org/download-90.cgi>



64-bit Windows zip을 다운로드하여 압축 해제 후 C:\dev 폴더로 옮겨준다.

1. Maven 설치

<http://maven.apache.org/download.cgi>

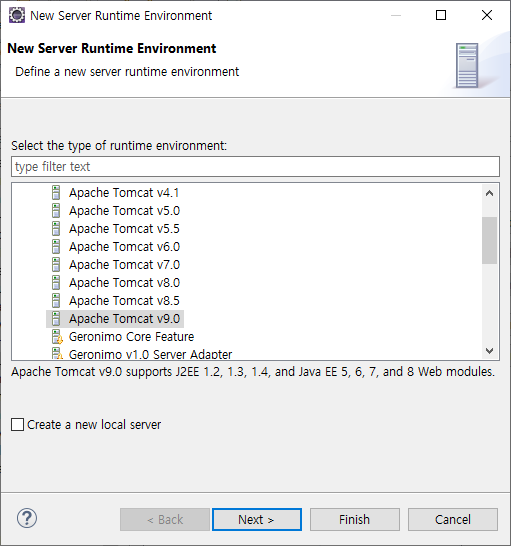
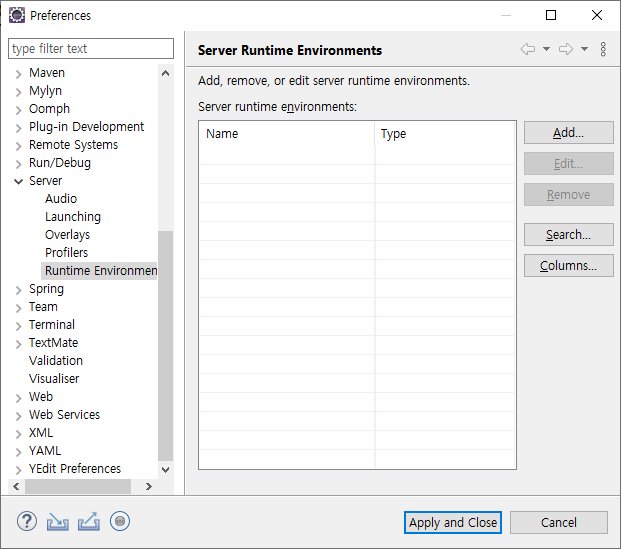


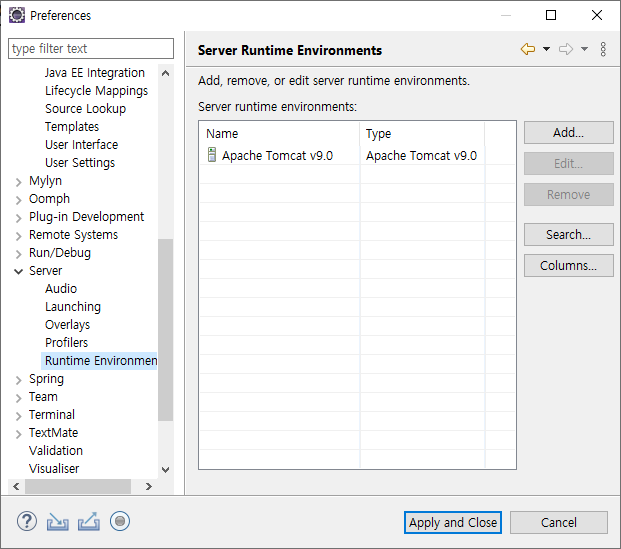
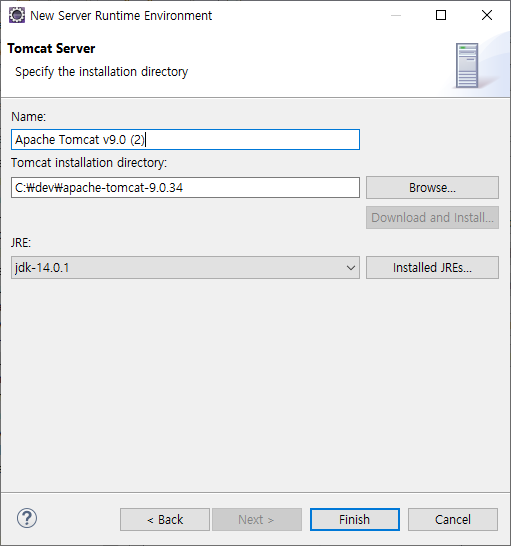
Binary zip archive(apache-maven-3.6.3-bin.zip)을 다운로드하여 압축 해제 후 C:\dev 폴더로 옮겨준다.

1. Eclipse 내부 설정
2. Tomcat

Eclipse 상단 메뉴 중 Window -> Preferences 를 선택

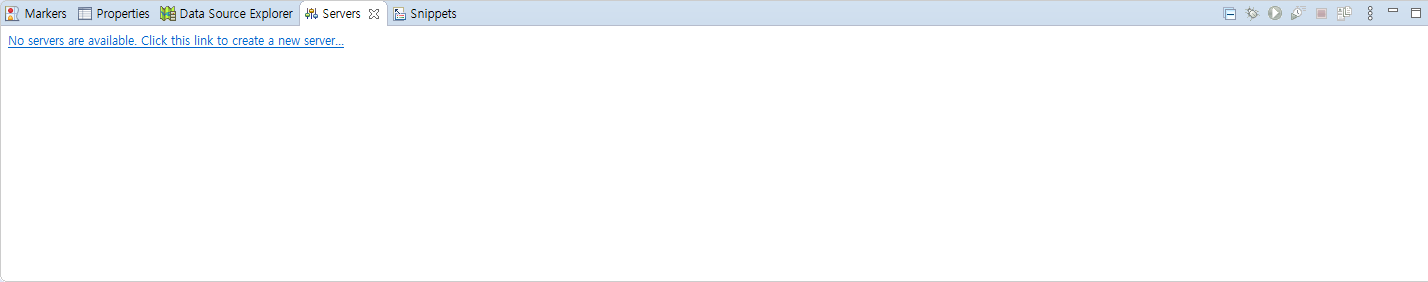
Server -> Runtime Environment 선택 후 Add를 눌러 본인에게 맞는 Tomcat 버전 선택, Next >

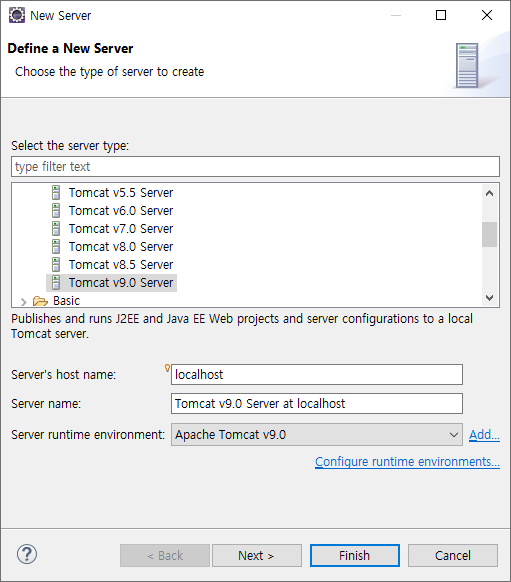




사진과 같이 올바른 경로 설정과 JRE 선택 후 Finish하면 위 사진과 같이 서버가 생긴 것을 볼 수 있다.

Eclipse 하단의 Servers를 선택 후 No servers are available. Click this link to create a new server… 를 누른다.





이와 같은 창이 뜨면 Finish를 누른다.

No servers are available. Click this link to create a new server… 링크가 있던 곳에

 이렇게 서버가 생겨 있다.

와 같은 버튼을 누르거나 Ctrl + Alt + R을 눌러 서버를 시작시킨다.



콘솔 창에 이러한 문구가 나온다면 Tomcat 서버가 정상적으로 켜진 것이다.

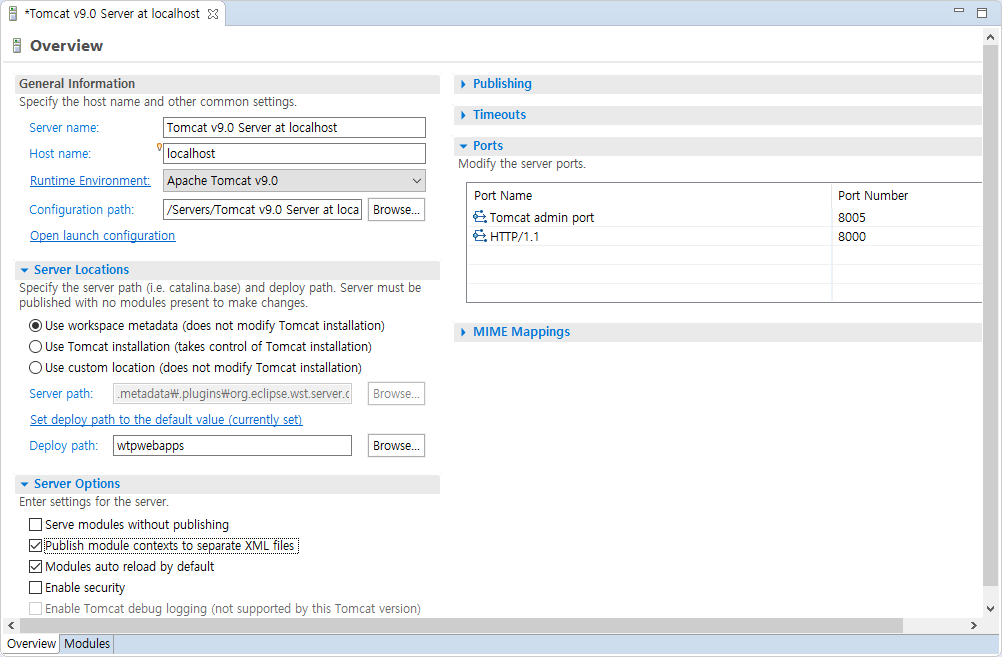
기본적으로는 8080포트이나 만약 다른 서버로 8080포트가 사용중이어서 서버가 켜지지 않는다면

서버의 포트를 변경시켜주어야 한다.

* 포트 변경 (2가지 방법)

1. Eclipse 내부에서 설정

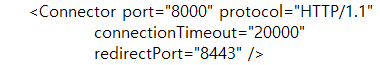
 을 더블클릭하면



이와 같은 화면이 된다. 오른쪽의 HTTP/1.1에 8080포트로 되어 있는 것을 원하는 포트로 변경시키면 된다.

1. 설정 파일 변경

C:\dev\apache-tomcat-9.0.34\conf 의 server.xml 파일을 열어준다.



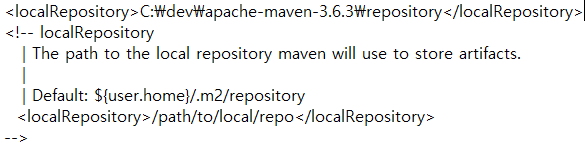
파일 내부에서 위 사진과 같은 부분의 8080을 원하는 포트로 변경시키면 된다.

1. Maven

* Repository 경로 설정 (라이브러리가 올바른 위치에 다운로드 되도록 하기 위함)

C:\dev\apache-maven-3.6.3 에서 repository 폴더를 만든다.

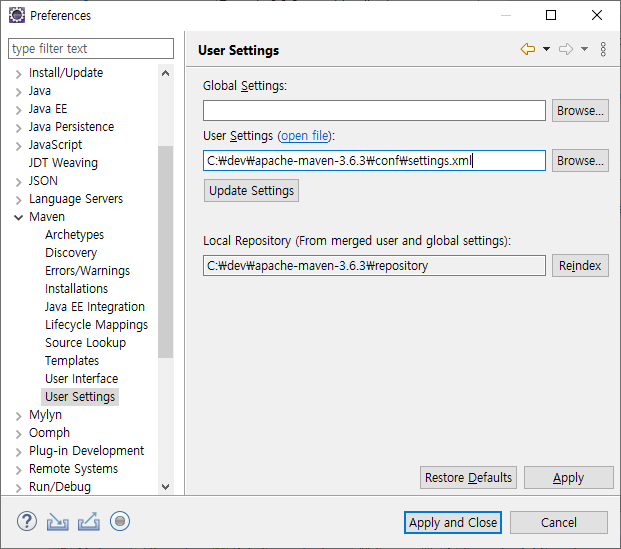
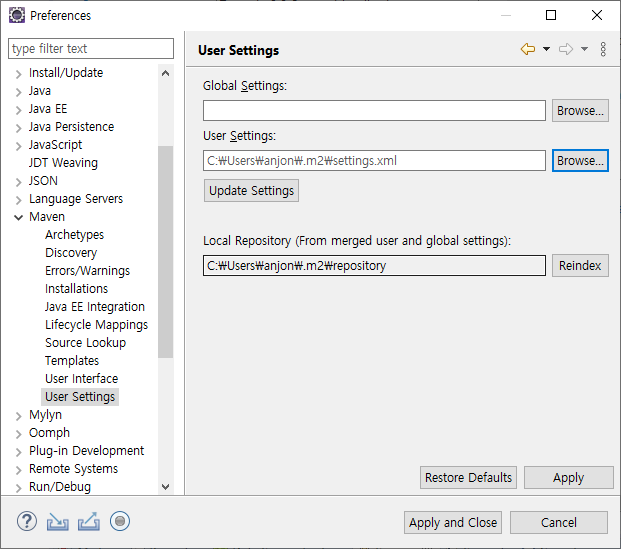
C:\dev\apache-maven-3.6.3\conf 에서 settings.xml 파일을 연다.



이처럼 주석되어 있는 localRepository 상단에 경로를 설정하여 준다.

Eclipse 상단 메뉴 중 Window -> Preferences 를 선택

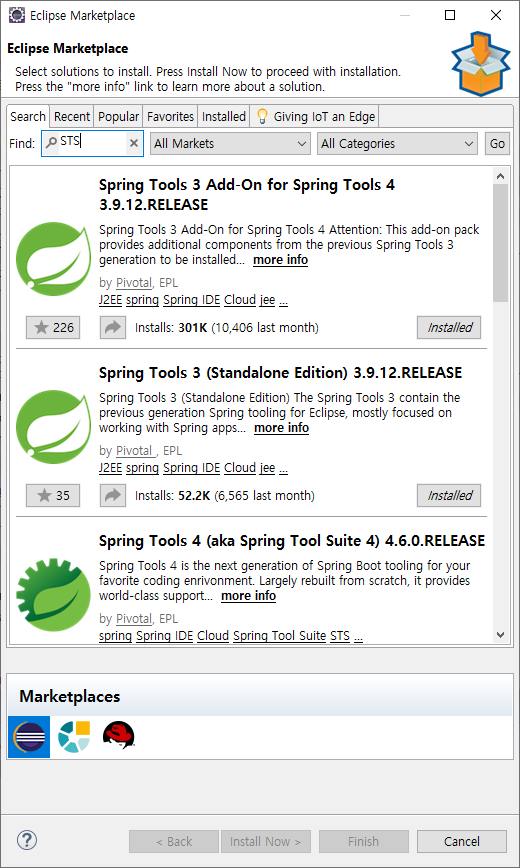
Maven -> User Settings 선택 후 User Settings 오른쪽의 Browse… 를 누른다.



Browse… 를 눌러 settings.xml를 선택하여 주고 적용한다.

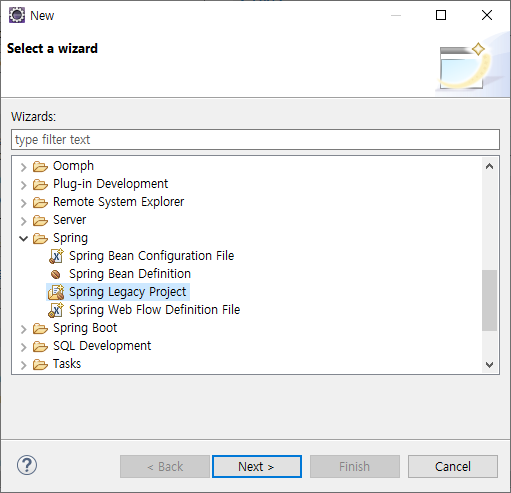
1. Eclipse 플러그인 설치

Eclipse 상단 메뉴 중 Help -> Eclipse Marketplace 를 선택한다.

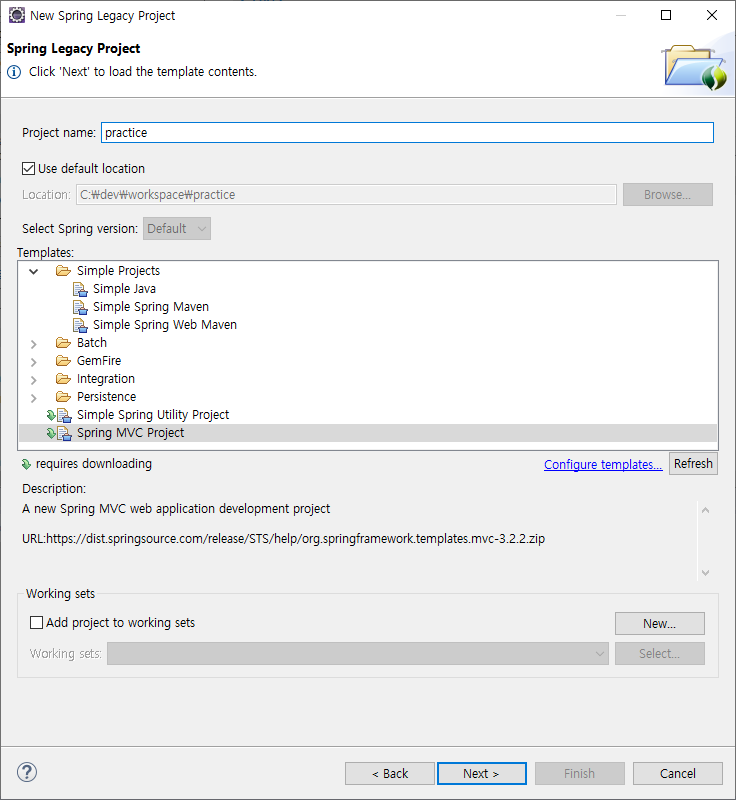
 STS를 검색 후 설치한다.

1. 프로젝트 생성

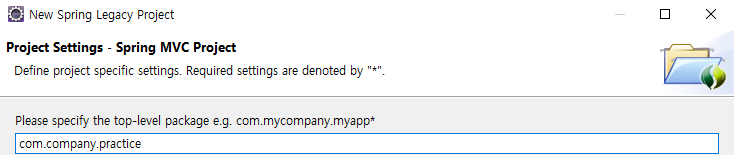
Eclipse 상단 메뉴 중 File -> New -> Other… 를 선택한다.



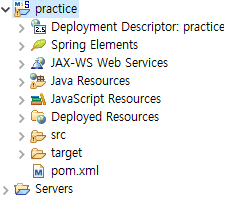
Spring -> Spring Legacy Project 선택 후 Next >



Project 이름을 정하고 Spring MVC Project 를 선택한 뒤 Next >

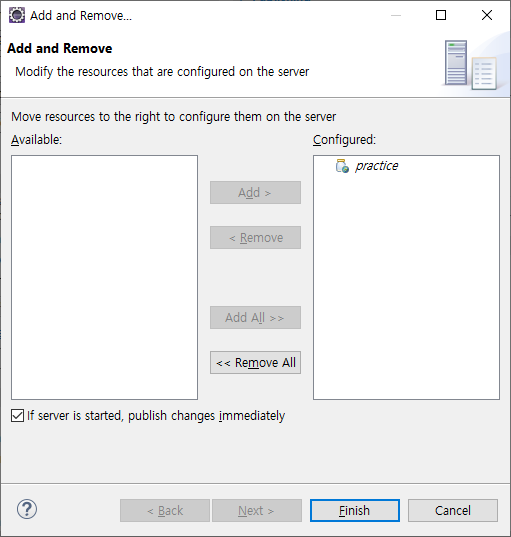


Package 설정 후 Finish

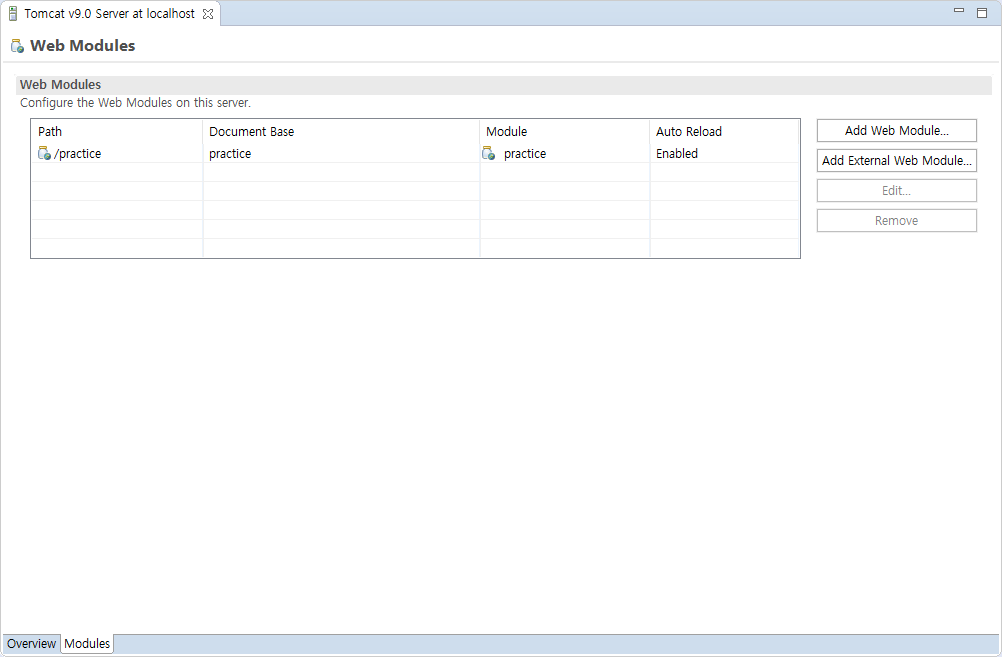


완료 된 화면이다.

 Server에서 우클릭 -> Add and Remove 를 선택한다.



프로젝트를 선택 후 Add > 를 눌러 추가하고 Finish를 누른다.



 더블 클릭 후 Modules를 눌러 Path를 확인할 수 있다.

인터넷에서 localhost:설정한포트/Path(ex: http://localhost:8000/practice 에 접속하여 아래 사진과 같은 결과가

나온다면 성공적이다.

