JAVA 서블릿 / JSP

- **컨테이너**
  - 통신(커뮤니케이션) 지원 : 서블릿과 웹 서버가 서로 통신할 수 있게 함
  - 생명주기(라이프사이클) 관리 : 서블릿의 탄생/죽음을 인스턴스화 → 초기화 메소드를 호출 → 요청이 들어오면 서블릿 메소드를 호출해줌 ex) 가비지 컬렉션
  - 멀티스레딩 지원 : 요청이 들어올 때마다 새로운 자바 스레드를 하나씩 만들어줌
  - 선언적인 보안 관리 : 보안관리는 xml 배포 서술자에다 기록하면 됨(직접 자바 코드 수정 X)
  - JSP 지원 : JSP를 설계해줌

- **MVC(Model-View-Controller)** : 애플리케이션을 3가지 역할로 구분하는 개발 방법론으로 사용자가 Controller를 조작하면 Controller는 Model을 통해 데이터를 가져오고 그 데이터를 바탕으로 View를 통해 시각적 표현을 제어해 사용자에게 전달함

  <MVC>

  - Model : 데이터와 관련된 부분
  - View : 사용자한테 보여지는 부분
  - Controller : Model과 View를 이어주는 부분

  <MVL를 지키면서 코딩하는 법>

  1) 모델은 컨트롤러와 뷰에 의존하지 않아야 함

     ⇒ 모델 내부에 컨트롤러와 뷰에 관련된 코드가 존재해선 안 됨

  2) 뷰는 모델에만 의존해야 하고, 컨트롤러에는 의존하면 안 됨

     ⇒ 뷰 내부에 모델의 관련된 코드는 존재해도 되며 컨트롤러와 관련된 코드는 존재해선 안 됨

3) 뷰가 모델로부터 데이터를 받을 때, 사용자마다 다르게 보여줘야 하는 데이터에 대해서만 받아야 함

⇒ 뷰에는 공통적인 부분을 모델은 개인적인 데이터를 관리

4) 컨트롤러는 모델과 뷰에 의존해도 됨

⇒ 중개자 역할이기 때문

5) 뷰가 모델로부터 데이터를 받을 때, 반드시 컨트롤러에서 받아야 함

⇒ 컨트롤러가 중간다리 역할이기 때문 → 각자에 역할에 충실

<JSP환경에서 MVC>

- 컨트롤러

  - Request객체에서 사용자가 입력한 정보를 뽑아내 모델에 대해 어떤 작업을 해야 하는지 알아냄

  - 모델 정보를 수정하거나 뷰(JSP)에게 넘겨줄 새로운 모델을 만드는 등과 같은 작업을 수행

- 뷰

  - 뷰는 컨트롤러로부터 모델 정보를 읽어옴

  - 사용자가 입력한 정보를 컨트롤러에게 넘겨줌

- 모델

  - 비즈니스 로직으로 데델 정보를 읽거나(getter) 수정함(setter)

  - MVC 패턴에서 모델은 DB와 통신하는 유일한 곳

<WEB에서 MVC>

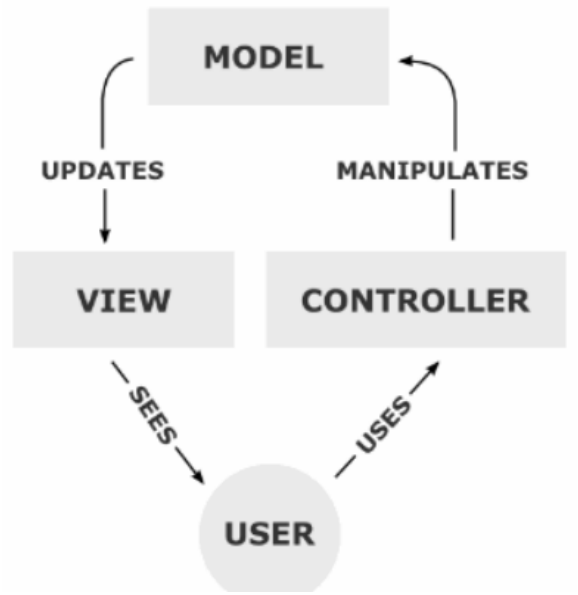
1) 사용자가 웹사이트 접속(Users)

2) Controller는 사용자가 요청한 웹페이지를 서비스하기 위해서 모델을 호출(Manipulates)

3) Model은 DB나 파일과 같은 데이터 소스를 제어한 후 그 결과를 Return

4) Controller는 Model이 리턴한 결과를 View에 반영(Updates)

5) 데이터가 반영된 View는 사용자에게 보여짐(Sees)



- 배포 서술자(DD)
  - 이미 테스트된 소스 코드에 대한 수정을 최소화
  - 소스 코드가 없어도 애플리케이션을 목적에 맞게 수정할 수 있음
  - 코드 변경이나 컴파일을 다시 하지 않아도 서버(DB 연결 등)를 바꿀 수 있음
  - 접근 제어 목록(ACL), 보안 역할과 같은 보안에 관련된 업무도 쉽게 관리할 수 있음
  - 프로그래머가 아닌 사람이 웹 애플리케이션을 배포하고 설정을 수정할 수 있음

- 내부에서만 사용하는 이름과 완전한 클래스명을 서로 매핑함

  <servlet>

  <servlet-name>배포(DD)된 이름</servlet-name>

  <serlvet-class>진짜 서블릿 이름</serlvet-class>

  </servlet>
- 내부에서 사용하는 이름과 URL 이름을 서로 매핑함

  <servlet-mapping>

<servlet-name>배포(DD)된 이름</servlet-name>

<serlvet-class>가공된 이름(클라이언트가 사용할 이름)</serlvet-class>

<servlet-mapping>

- 웹서버
  - 클라이언트⇒서버 : 요청에는 요청한 것에 대한 이름과 주소(URL) 정보가 들어있음. URL이 바로 요청한 자원에 대한 주소임
  - 서버 : 클라이언트에게 보낼 컨텐츠가 있고 이 컨텐츠는 웹 페이지가 될 수도, 이미지일수도 있으며 또 다른 형식일 수 있음
  - 클라이언트 : 사람 또는 응용프로그램
  - 서버⇒클라이언트 : 응답에는 클라이언트가 요청한 내용이 들어있으며 만약 서버가 못 찾는 경우 오류 메세지가 대신 들어가있음
  - 혼자할 수 없는 일

    1) 동적인 컨텐츠 생성

    2) 서버 상에 데이터 저장
  - 포워딩 : 도메인으로 접근한 사용자를 다른 도메인으로 보내는 행위

    EX) ooo2.org로 사용자가 접근하면 opentutorials.org로 보낸다.

- HTTP : 웹 상에서 클라이언트와 서버가 서로 대화하기 위한 규약 → 응답/요청으로 이루어짐
  - 헤더 : 사용된 프로토콜 종류 / 요청 성공 여부
  - 몸체 : 컨텐츠 종류 ex) HTML 등
  - 요청

    - HTTP 메소드(실행할 액션)

    - 접근하고자 하는 페이지(URL)

    - 폼 파라미터(메소드의 매개변수와 비슷)
  - 응답

- 상태 코드(요청 성공 여부 확인)

- 컨텐츠 타입(텍스트, 그림, HTML 등)

- 컨텐츠(HTML 코드, 이미지 등)

- GET : 서버에게 자원 요청. 즉 서버로부터 뭔가를 돌려 받음

  <예시>

  GET/select/select.jsp?colojr=dark&taste=malty HTTP/1.1

  요청라인(HTTP메소드)/웹 서버 상 자원에 대한 경로?파라미터&개별파라미터 프로토콜버젼

  …(밑엔 요청 헤더 있음)

- POST : 서버에게 자원을 요철할 때 필요한 정보를 함께 넘겨줌

  <예시>

  POST/advisor/selectBeerTaste.do HTTP/1.1

  요청라인(HTTP메소드)/웹 서버 상 자원에 대한 경로 프로토콜버젼

  …(밑엔 요청 헤더 있음)

  …(헤더 밑엔 파라미터)

  <사용>

  - GET은 글자수 제한이 있음

  - GET의 데이터 전송방식은 URL 뒤에 붙임

- 순서

  사용자가 주소창에 URL을 입력 ⇒ 브라우저는 해당 정보로 HTTP GET 요청을 만듦

  ⇒ HTTP GET을 서버로 날림 ⇒ 서버는 요철한 페이지를 서버에서 찾고 / HTTP 응답을 작성

  ⇒ HTTP 응답을 클라이언트로 내려보냄 ⇒ 브라우저는 HTML을 화면에 띄움

- URL

  <예시>

  http://www.naver.com:80/beeradvice/select/beer.html

  프로토콜 : 서버와 대화하기 위해 사용

  서버 : IP 주소에 매핑

포트 : 포트는 URL 옵션임. 구별하기 위한 식별자

서버에서 자원의 위치

자원 : 요청된 컨텐츠 이름

- 서블릿 : 자바를 사용해 웹페이지를 동적으로 생성하는 서버측 프로그램 또는 그 사양을 말하며 웹 서버 성능을 향상하기 위해 사용되는 자바 클래스의 일종임 → 자바 코드 안에 HTML을 포함

  ⇒ mine() 메소드 존재(X)

- JSP : HTML코드에 JAVA코드를 넣어 동적 웹페이지를 생성하는 웹어플리케이션 도구 → HTML 문서 안에 자바 코드

⇒ JSP가 실행되면 자바 서블릿으로 변환되며 웹 어플리케이션 서버에서 동작

- 톰캣 : 웹 서버와 연동하여 실행할 수 있게 자바 환경을 제공하고 JSP(자바 서버 페이지)와 자바 서블릿이 실행할 수 있게 환경을 제공하고 있음

  ⇒ 오라클과 포트 충돌이 일어나므로 톰캣 포트를 바꿈(8081)

- 톰캣 사용

  → D:\JAVA Workspace\SERVLET_JSP\apache-tomcat-9.0.65\bin 에서 startup.bat 실행 후 웹 페이지에 localhost:8081 접속

  → 이후 파일 작업은

  D:\JAVA Workspace\SERVLET_JSP\apache-tomcat-9.0.65\webapps\ROOT 안에 저장해 사용함

- Context (가상 경로) ⇒ 현재 지양하는 존재(서버를 껐다 켜야하기 때문)
  - root 폴더 안에 폴더 안에 …. 가다보면 URL이 길어짐. 이때 하위 디렉터 리를 별개의 사이트로 취급할 수 있게 따로 빼놈(서비스)

대신 문맥은 유지할 수 있음 ⇒ 물리적(ex.하드링크)으론 2개, 논리적(ex.심볼릭링크)으론 1개

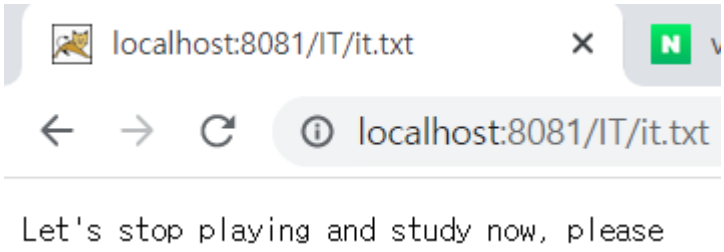
- conf 파일 안에 server.xml 파일에서 코드 변경

```
....
<Host name="localhost" appBase="ROOT디렉터리의 부모 디렉터리">
    <Context path="가상 디렉터리 이름" docBase="실제 디렉터리 경로" privileged="tru
e"/>
</Host>
....
```

<예시>

```
....
<Host name="localhost"  appBase="webapps"
unpackWARs="true" autoDeploy="true">
    <Context path="it"
    docBase="D:\JAVA Workspace\SERVLET_JSP\apache-tomcat-9.0.65\webapps\IT"
    privileged="true"/>
....
```

<결과>



⇒ 현재 localhost:8081/IT/it.txt인 경로인데 실제 IT 디렉터리는 ROOT 디렉터리 안에 있지 않고 ROOT 디렉터리 부모인 webapps 디렉터리 안에 있음. 즉 ROOT 디렉터리와 같은 위치에 존재함

-