[1강. JSP와 Servlet 이해 및 환경구축]

- Native App: 우리가 흔히 말하는 어플리케이션. Data 중심. json, xml만 주고받는 형식. 보통 미들웨어라고 함.

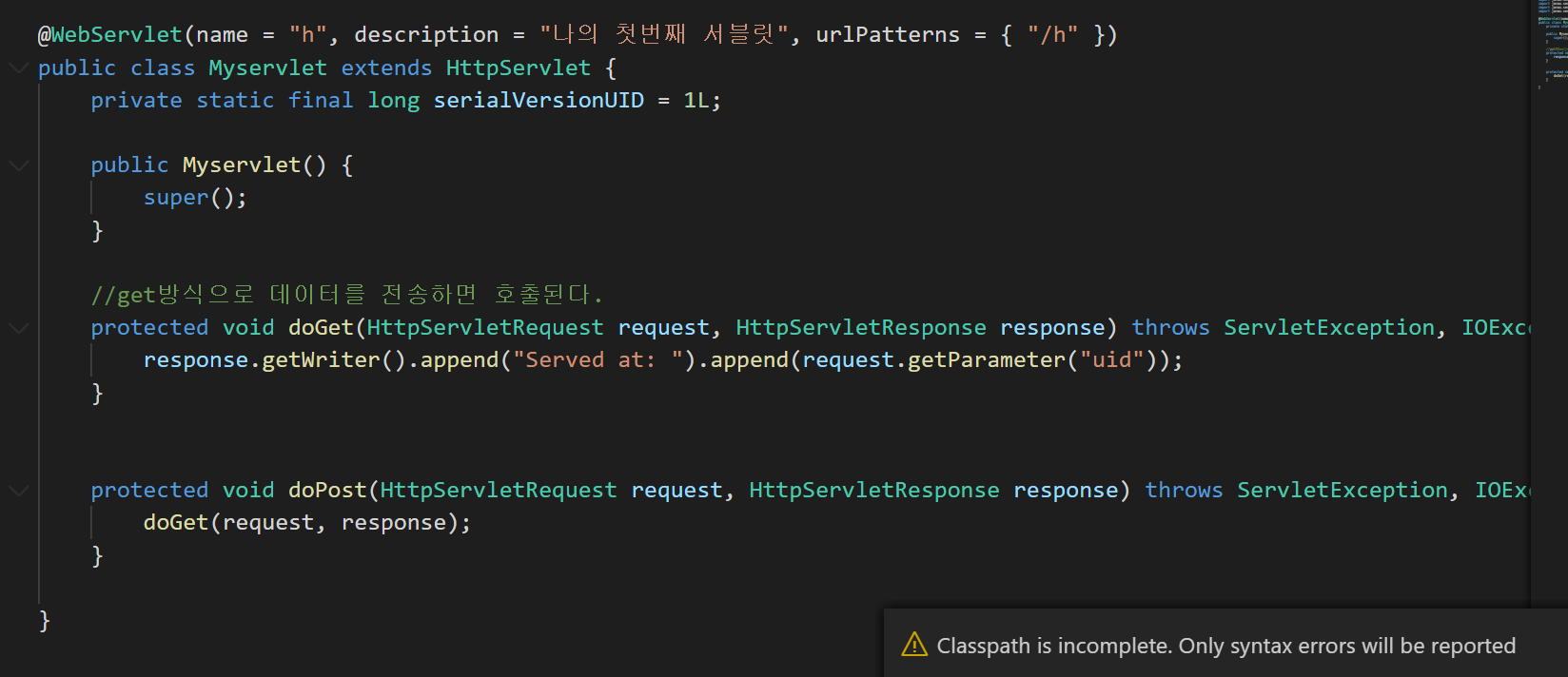
[2강. Servlet 기본 구성]

- Data 전송방식: json, xml



왼쪽이 json, 오른쪽이 xml. xml보다 json이 경량이어서 json을 쓰는 추세.

- 서블릿이란? 자바를 사용하여 web page를 동적으로 생성하는 서버측 프로그램. 웹 서버의 성능을 향상시키기 위해 사용되는 자바 클래스. 아래의 사진은 서블릿의 기본 모습.



- doGet(): get 방식으로 요청한 것을 처리하는 method

- doPost(): post 방식으로 요청한 것을 처리하는 method

- 서블릿 생성 or 수정하면 반드시 재가동해주기 🡪 run on server

(근데 수정했을 때 굳이 재가동 안해도 실행은 되었던 거 같음)

- 내가 서블릿을 고치면 알아서 리로드(다시 로드(데이터나 sw를 다시 램으로 불러들인다.))됨. 다 될 때까지 기다리기.

<주요 응답코드>

200: 서버가 정상적으로 응답함.

404: 요청했으나 파일이 없다.

403: 읽을 권한이 없다.

405: 허용할 수 없다.

500: 서버의 내부 오류

<JSON>

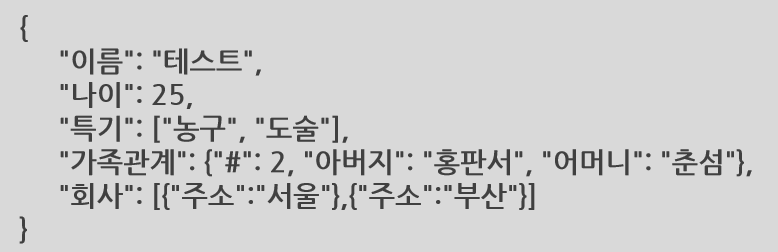
- 키와 값 쌍으로 이뤄짐. 인간, 컴퓨터, 언어 모두가 이해할 수 있는 Data Format. 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법.

- 언어, 플랫폼에 독립적.

- json의 공식적인 mime 타입(클라이언트에게 전송된 문서의 다양성을 알려주기 위한 메커니즘. 각 문서와 올바른 mime type을 전송하도록 서버가 정확히 설정하는 것이 중요.)

application/json(or text/plain도 사용 가능) 🡪 Json file 만들때 mime에 이렇게 등록 되어있어야 됨

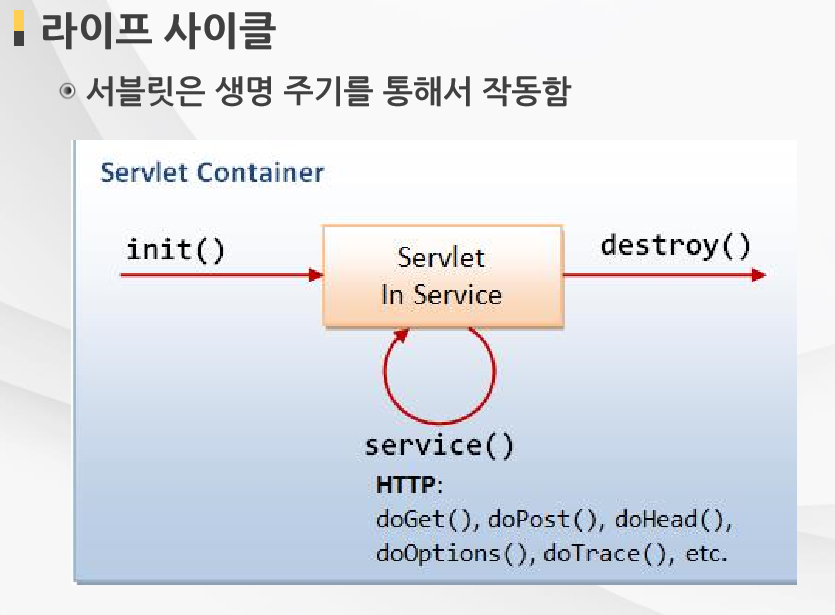
json 확장자: .json(jsp or servlet에서 동적으로 생성돼서 내려옴. )



값으로는 문자열, 수치, Boolean, 배열, 또 다른 json, json 배열, 널, 객체가 들어갈 수 있음.

- 파싱이란?? Data를 추출해내는 거. 파싱을 위해선 json이 문법적으로 오류가 없이 유효해야 함.

[3강. 서블릿 라이프 사이클]



- 요청 🡪 초기화 이뤄짐 🡪 구동(응답) 🡪 해제(더 이상 안쓰면)

-Servlet Container(= Web Container): 서블릿 관리, 실행함.

- init(): 로드 단계. 최초구동.

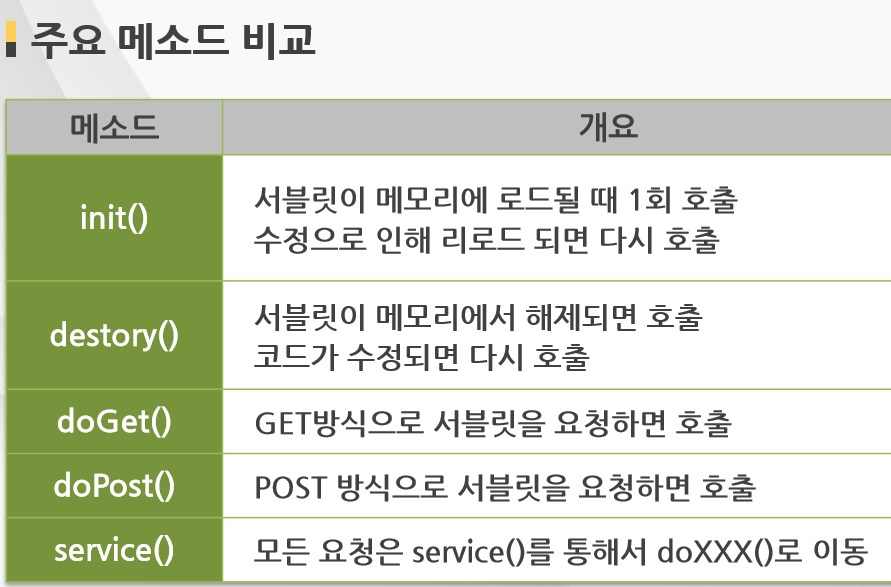
- servlet in service(): 요청이 들어오면 여기서 돌면서 응답을 함.

- destroy(): 코드 변경 or 시스템 에러 or 자원 부족 등으로 자원해지 해야 되면 이게 호출이 돼서 servlet이 밑으로 내려오게 됨. 요청하면 다시 올라가게 되고…

<Container&Servlet 동작 방식>

- 메모리 로드 🡪 client url 요청 🡪 웹서버 request, response 객체 생성 🡪 Servlet 인스턴스(메모리 하나가 메모리에 생성이 되는 거)와 Thread(service() call) 생성.

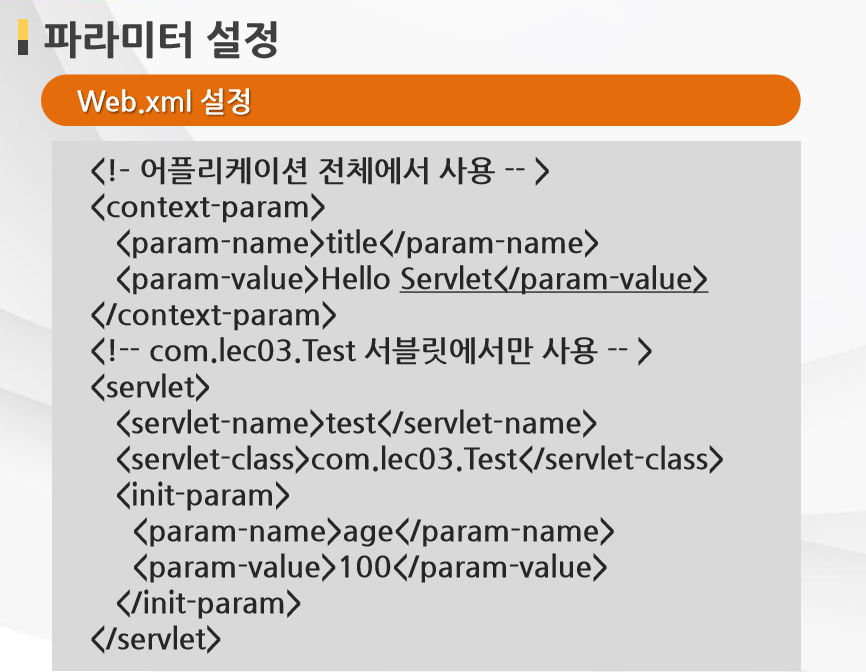
- service()는 각각 get or post 방식이냐에 따라 response, request를 인자로 해서 호출. 그럼 doGet, doPost 안에서 응답을 해서 웹서버에 전송.



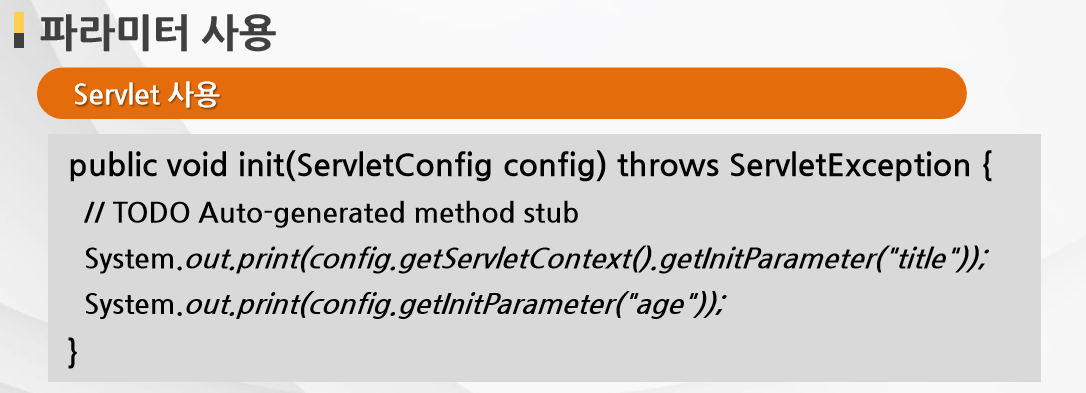
<Servlet 초기화 파라미터>

- 초기화 파라미터는 init에서 사용함. 상수 같은 역할을 함. 서블릿 구동 혹은 웹 전체적으로 구동될 때 특정한 데이터를 내려주는게 초기화 파라미터.

- 서블릿 내 초기화 파라미터 설정



위의 방법 말고도 서블릿 생성할 때도 설정할 수 있다.

 🡪 상수처럼 사용가능.

전체적인 거 사용: config.getServletContext().getInitParameter(“title”);

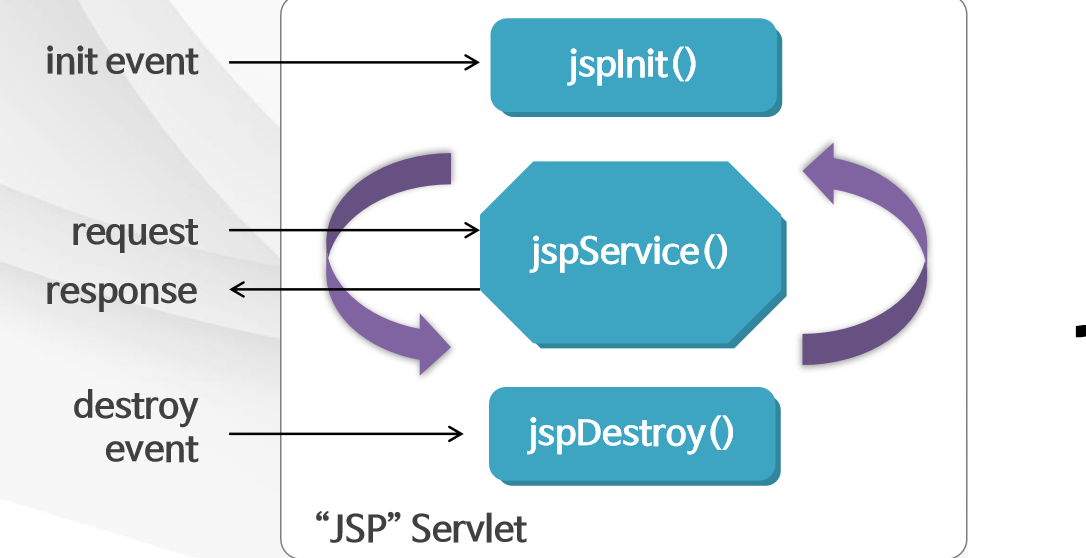
서블릿 자체적인 파라미터 사용: config.getInitParameter(“title”);

- web.xml 수정하고 나면 프로젝트 자체 재가동 하는게 맞음

[5강. jsp Life Cycle] 🡪 73p~

- 요청 🡪 로드 🡪 응답(처리)

< jsp Life Cycle]> 🡪 서블릿과 동일한 흐름.



- 초기화 이벤트가 들어오면 init실행.

- jspService(): 각종 요청, 응답 담당. 실제 우리가 구현하지는 않음. doGet, doPost처럼 별다른 메소드 없이 우리가 작성한 코드가 여기에 대응 된다.

- destroy event 🡪 jspDestroy: 리로드 될 경우

<주요 메소드>

- jspInit(): 시작(초기화)

- jspDestroy: 종료(리로드)

- jspService(): 요청 들어오면 호출됨. 별도 구현은 하지 않음.

<jsp 내 전역 변수 = 멤버변수>

- 멤버변수는 class의 멤버이고, 이 class는 servlet이 됨. 이 서블릿은 우리가 작성한 jsp가 변환한 부분. 멤버변수는 선언부에서 정의함.

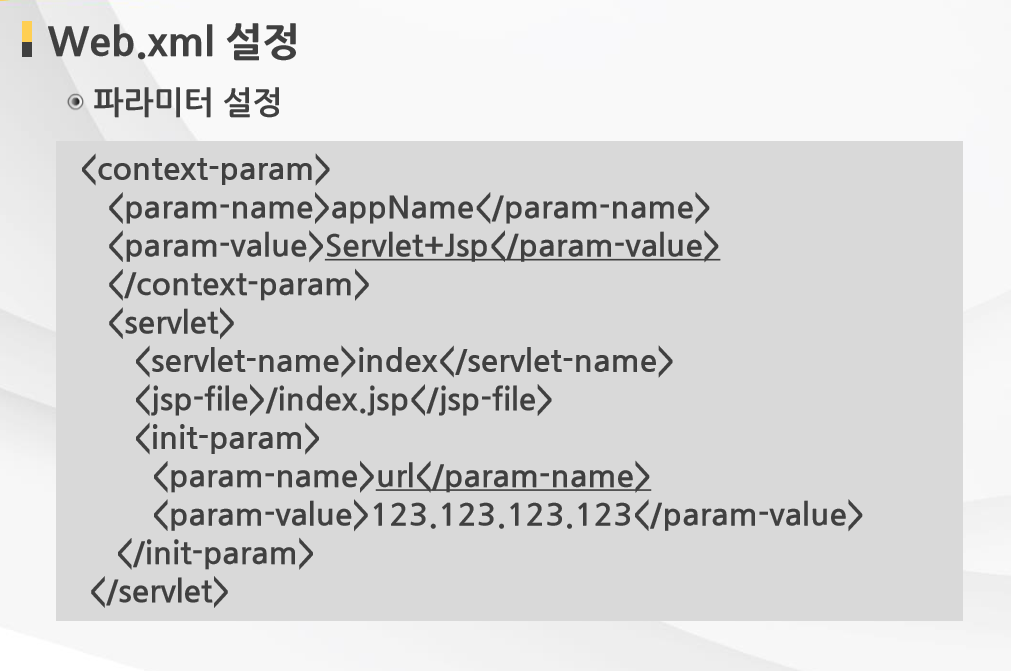
- 스크립트릿에서 작성하는 변수는 doGet(), doPost() 내의 지역변수가 됨.

- 요청이 오면 doGet(), doPost()가 호출이 되고, 이 지역변수가 doGet(), doPost() 안으로 들어가는 코드여서 항상 요청이 오면 지역변수는 초기화됨.

- 전역변수는 한번 초기화되고 변함이 없음. 리로드 되기 전까지 그 값이 계속 이어짐.

코드 변경 🡪 서블릿 리로드 🡪 재컴파일 🡪 초기화발생

jsp 내 초기화 파라미터 설정>

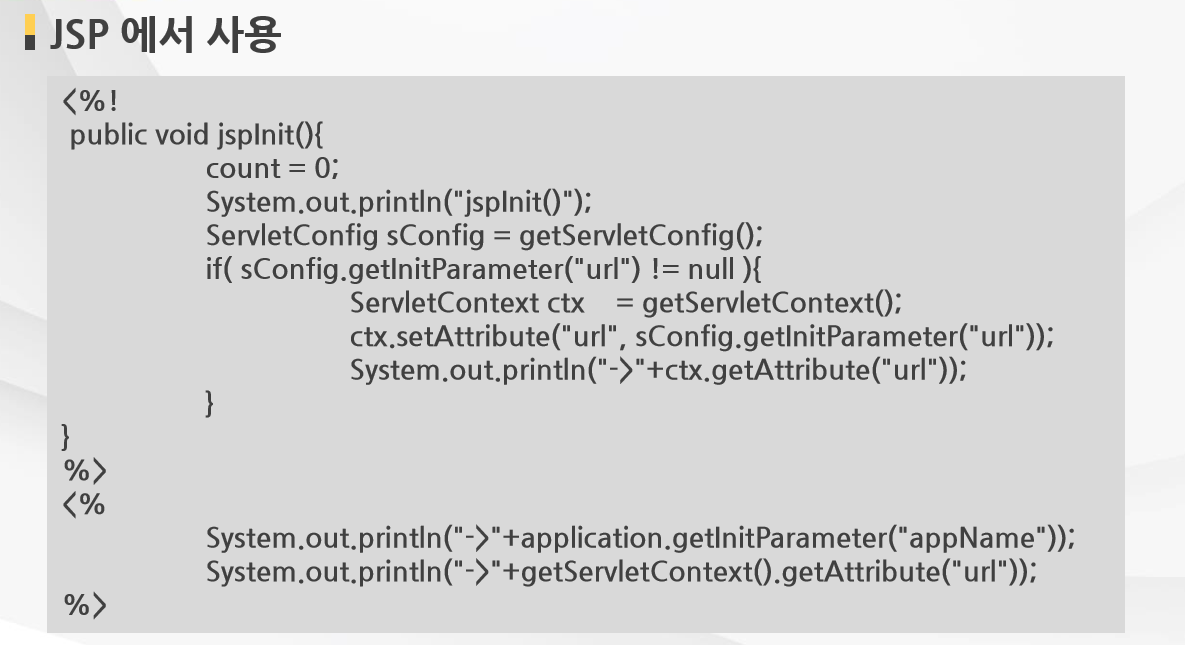


웹 전체에서 사용: web.xml의 <context-param> </context-param> 사이

서블릿에만 사용: <servlet></servlet>사이

- 서블릿처럼 최초에 설정하는 방식은 없다.

<jsp에서 초기화 파라미터 사용>



- 초기화 파라미터는 초기화(init)에서 처리됨. 초기화(init)에서 config 객체를 구해야 됨. 서블릿에선 매개변수로 왔지만 여기서는 getServletCofig();통해 config객체를 구하기

- getServletContext()로 값을 구해서 setAttribute()를 통해서 값을 저장 해놓고 필요한 페이지에서 사용.

- 어플 전체에 사용하는 방법은 application.getInitParameter();

- 각각 page에 쓰는 부분은 getServletContext()로 저장을 해야됨. 🡪 뭔말인지 모르겠음…

\*\*참고

Servlet 클래스는 톰캣 컨테이너 실행 시 각 context(web application)마다 한 개의

ServletContext 객체를 생성. 톰캣이 종료하면 이 객체도 소멸한다. ServletContext 객체는 웹 애플리케이션이 실행되면서 어플리케이션 전체의 공통자원이나 정보를 바인딩(프로그램에 사용된 구성요소의 실제 값 혹은 프라퍼티를 결정짓는 행위)해서 서블릿들이 공유하여 사용함.

//servletContext와 servletConfig

    //https://blog.naver.com/PostView.nhn?blogId=youngchanmm&logNo=221227744788&parentCategoryNo=&categoryNo=45&viewDate=&isShowPopularPosts=false&from=postView

    //전체 페이지에서 적용되는 파라미터 --> 서블릿에서는 servletContext로 가져오고 jsp파일에서는 application으로 가져옴. https://sumin172.tistory.com/148

[6강. Cookie 처리 기술] 🡪 교재 88~

- 클라이언트 브라우저에 뭔가를 저장하는 게 쿠키 기술이다. 저장하는 것이다 보니까 누구나 다 볼 수 있다.

- 쿠키가 특정 도메인, 특정 경로만 가게끔 할 수 있다.

- 쿠키는 웹 컴포넌트 간의 데이터 전달을 위해서 만든 기술이다. 웹브라우저에 데이터를 저장하는 기술. 특정 페이지가 뜰 때 쿠키를 읽어와서 뭔가 처리해주는 등의 역할을 한다.

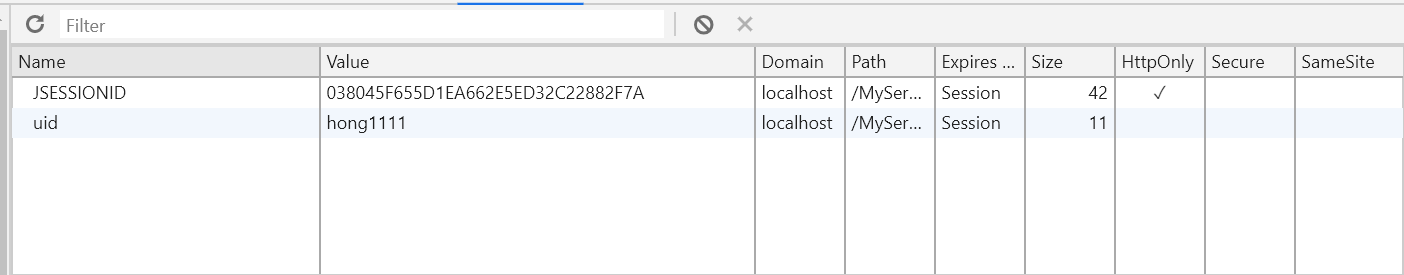
- 쿠키는 객체 하나로 표현 됨.

Cookie cook = new Cookie(“쿠키이름”, “쿠키값”); //쿠키 생성

response.addCookie(cook); // 쿠키 저장

- 쿠키 데이터 읽기: 읽어오는 쪽(Server)에서 갖고 와야 하니까 request.

request에서 쿠키를 다 모아오는데 배열로 옴. 하나씩 끄집어 내서 내가 원하는 쿠키가 발견되면 그 값을 뽑으면 됨.



- 도메인이 어떤 url을 통해서 온건지. path는 /MyServlet 밑으로 되어 있음. Expire는 쿠키를 언제까지 사용하는 가.

- 쿠키 수정: response.addCookie(new Cookie(“name”, “tomas”));

🡪 키 값이 동일하면 수정이 됨.

- 삭제 내가 삭제하고자 하는 쿠키의 이름을 찾아서 삭제

c.setMaxAge(0): 쿠키가 바로 삭제하도록 처리

c.setMaxAge(-1): 웹 브라우저가 끝날 때 쿠키가 삭제되도록 처리.

- 쿠키의 전송처리 이해: 쿠키를 보낼 때 특정 url을 통해서만 보낼 수 있게 할 수 있음. 혹은 특정 웹서버를 통해서만 보내게 할 수 있음 🡪 예외적인 케이스

\*\* 특정 경로명에서만 쿠키가 전송되게 제한.

Cookie cook = new Cookie(“쿠키이름”, “쿠키값”); //쿠키 생성

cookie.setPath(“/in/main”); //쿠키가 적용되는 url 경로를 지정해주고 있다. /in/main 밑에서 접근을 했을 때 쿠키가 생긴다는 거. 이게 없다면 모든 페이지에서 이 쿠키가 들어감.

response.addCookie(cook); // 쿠키 저장

\*\* 웹서버로 전송.

Cookie cook = new Cookie(“쿠키이름”, “쿠키값”); //쿠키 생성

cookie.setDomain(“.co.kr”); // .co.kr로 끝나는 모든 곳에서 전송이 가능하다.

response.addCookie(cook); // 쿠키 저장

- 수정을 하든, 삭제를 하든, 전송을 하든 뭔가 변화가 오면 addCookie를 하기. 그래야지 항상 브라우저에 반영이 됨.

[7강. 세션처리 기술] 🡪 교재 100~

- session: 데이터를 저장하는데 있어서 쿠키는 클라이언트한테 저장을 해놓는데 서버측에 데이터를 저장해 놓는 것이 세션이다.

- 세션 정의: 여러 컴포넌트(웹 컨테이너 상에 존재하는 웹페이지(jsp, servlet 등))들이 협력작업을 시작해서 데이터가 연속적으로 유지되는 기간을 의미.

- 세션은 서블릿, jsp 두 곳에서 모두 만들 수 있음.

- 서블릿에서 세션사용 <https://juns0201.tistory.com/115>

\*\* 세션 생성

HttpSession session = request.getSession();// request.getSession()을 통해 세션을 구함.

session.setAttribute(“name”, “홍길동”);

\*\* 세션 사용

HttpSession session = request.getSession();

String name = (String)session.getAttribute(“name”);

- jsp에서 세션 사용

\*\* 세션 생성 🡪 105페이지

//session이 바로 내장객체에 해당됨. 그래서 jsp가 서블릿으로 변환될 때 그때 내장객체는 자동으로 들어오게 되어있어서 바로 사용하면 됨.

session.setAttribute(“name”, “홍길동”)

\*\* 세션 사용

String name = (String)session.getAttribute(“name”);

- 서블릿 만들고 나면 서블릿 인식하게 하기 위해 서버를 재가동 해줘야됨.

- 서버측의 세션이라는 객체에 name라는 이름으로 저장이 되는데 이거는 개별적으로

브라우저별로 설정이 되는 부분이니까 여러사람이 쓴다고 해서 값은 값이 저장되지는 않음.

- 세션 주요 함수

\*\* 세션 삭제: session.removeAttribute(“키”); 🡪 딱 집어서 한 개를 삭제하는 거.

\*\* 세션 종료(세션 전체를 없애고 싶을 때): session.invalidate();

\*\* 세션의 최대 비활성 시간:

int timeout = session.getMaxInactiveInterval();

\*\* 세션의 최대 비활성 기간 설정: 세션의 기간 설정. 1000이 1초가 됨.

int timeout = session.setMaxInactiveInterval(500);

1번에서 값을 넣고 전송해서 2번으로 감.

- 서블릿의 세션 기술

\*\* 여러 웹 컴포넌트들이 데이터를 공유하는 방법. 데이터들을 연속적으로 저장하는 방법이 필요. 연속적으로 페이지를 이동하면서 데이터를 유지할 때 유용. 특정 페이지 접근을 제어(어떤 것을 눌렀을 때 로그인 창이 뜨는 경우(세션이 없기 때문에 뜨는 건) 등)할 때도 유용.

\*\* 세션은 서버 측의 자원이라서 많이 쓰면 상당히 서버의 성능을 저하시키는 원인이 되기 때문에 삭제를 잘 해야됨.

[8강. Error, Exception 처리 기술] 🡪 110~

- web component(jsp, servlet)상 에러: 직접 개발자가 처리

- web.xml에서 에러페이지를 등록하는데 미리 예상을 해서 이런 에러가 나면 여기로 가고 그래라를 설정하는 거.

- web.xml에서 상태코드(200, 404 등)에 따라서 에러 페이지 처리. 미리 등록해 놓아서 가게 한다.

<web component 상 발생하는 예외처리 🡪 웹에서 직접처리>

- 자바의 예외 처리인 try\_catch를 통해 예외처리하는 부분 <https://code4.tistory.com/5>

<https://valuefactory.tistory.com/194>

RequestDispatcher rd = request.getRequestDispatcher(“error.jsp”);

rd.forward(req, res);

// a라는 페이지에서 b라는 페이지로 직접적으로 보내버리는 게 RequestDispatcher에서 forward라는 메소드를 통해 처리가 될 수 있다.

// q.jsp에서 a를 요청했는데 a는 전체처리 중 10%만 표현하고 나머지는 b너가 표현해줘라는 식으로 돌려버리는 게 Dispatcher가 하는 역할

- 간결한 처리 방식

🡪 jsp 페이지 지시자에 에러페이지 등록: <%@ page errorPage=”error.jsp” %>

🡪 에러 상세 내용 출력(error.jsp):

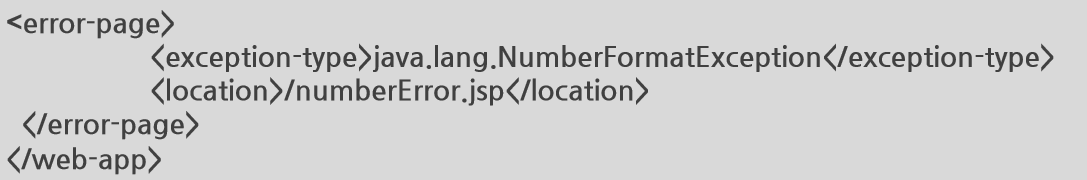
<%@ page isErrorPage=”true” %>

<%= exception.getMessage() %>

<web.xml 파일에 에러 페이지 등록 처리> 117p

- 익셉션 타입 (예외 상황)별로

\*\* web.xml 추가 🡪 웹전체에 등록하는 거. 위의 방법은 페이지 하나별로 등록을 하는 거.



NumberFormatException이 발생하면 numberError.jsp 페이지로 가라.

\*\* 에러 상세 내용 출력 numberError.jsp

<%@ page isErrorPage=”true” %>

<% response.setStatus(200) %> //응답 200을 표현해주고 있음.

- web.xml에 수정 후에는 재가동 시켜주기

<http 상태 코드(100~500) 별 에러 페이지 등록처리> 🡪 웹전체에 등록하는 거.



404는 page not found. 이 오류가 났을 때는 fileNotFound.jsp 페이지를 뜨게 하라.

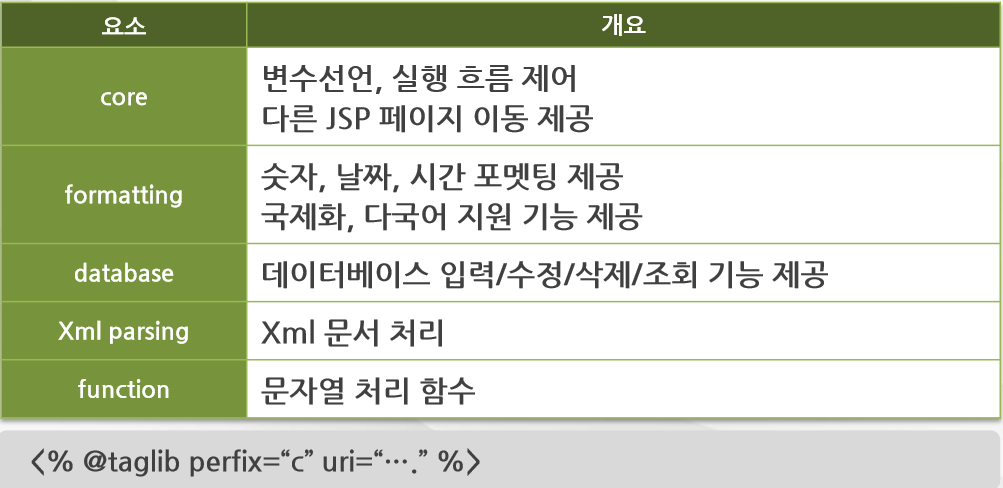
404는 코드적 오류가 아니기 때문에(페이지가 없는 거여서, 코드적 오류사항은 500에러 밖에 없음) exception.getMessage가 필요가 없다. 넣으면 널 포인트 익셉션 오류가 남.

[13강. jstl 정의 및 라이브러리] 🡪 교재164~ \*\*파싱: 어떤 정보를 뽑아낸다.

- jstl은 jsp를 view로 표현하는데 가장 최적화된 형태

- 환경구축: Dynamic Web project를 Maven이라는 형태로 변환을 할 것.

- 우리가 사용할 라이브러리: core, formatting, function



- 표준태그 <jsp:include> 같은 거는 그냥 사용해도 되는 반면 커스텀태그는 <% @taglib> 도 넣어줘야 하고 라이브러리를 설치해야됨. 태그라이브러리 지시어를 넣어줘야 인식이 됨.

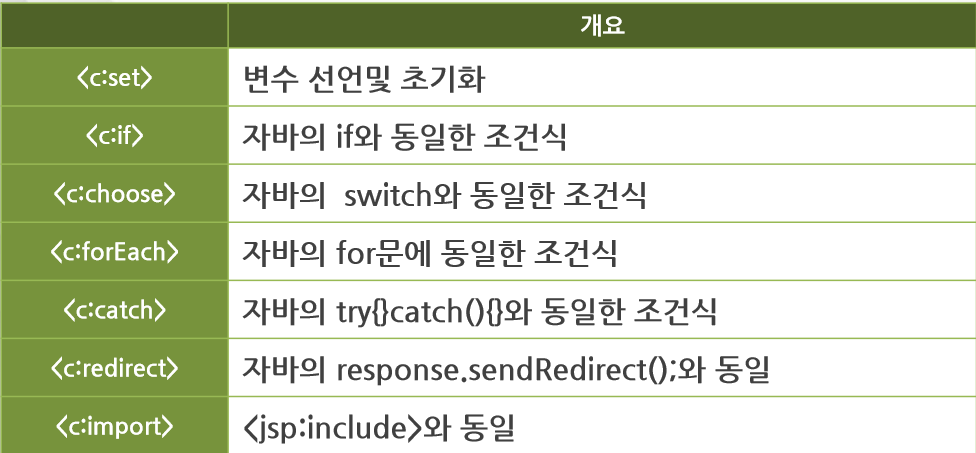
- Dynamic Web project 🡪 Maven 변환

Maven은 라이브러리들을 일일이 관리해주기 어려우니 메이븐이라는 저장소를 제공해줘서 xml로 표현해주면 알아서 repository에 들어오는 형태가 됨. 즉 프로젝트의 라이브러리들을 손쉽게 관리할 수 있게 됨.

- war? 우리가 마지막에 이 프로젝트를 만들어서 어떤 고유 아이피를 가진 곳에 배포를 하게 되는데 그때 최종파일이 war 파일이 됨. 그래서 메이븐 변환할 때 최종 패키지 파일이 war라고 설정이 되어 있는 거.

- Maven은 라이브러리를 관리하는 방식 중 하나. 나중에 모바일 쪽 연동할 때는 안드로이드를 깔게 되는데 그때는 Gradle을 쓰게 됨.

<core 라이브러리의 대표적인 거>



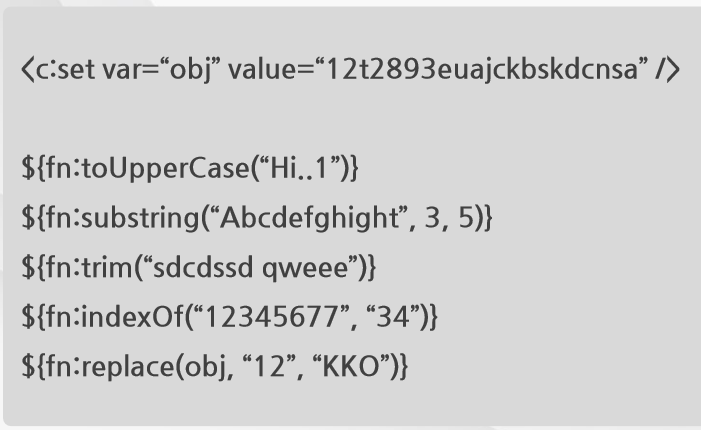
forward는 a->b->c에서 b가 다 처리 안하고 c로 간다 였는데, redirect는 그냥 request가 가는 게 아니라 화면을 던져버리는 거. 리퀘스트가 간다기 보다도 새로운 화면으로 넘어간다는 의미.

[14강. jstl 라이브러리2 🡪 formatting, function] 🡪 교재176~

- 포멧팅: 통화, 숫자, 날짜 등을 처리하는 부분



- 함수: 문자열에 대한 처리. EL식 안에서 사용할 수 있는 EL함수들의 라이브러리



toUpperCase: 대문자로

substring: 글자를 자른다.

trim: 공백을 제거한다.

indexOf: 해당 문자(두번째 매개변수)가 첫번째 매개변수에 존재하는 가

replace: 문자를 대체

\*\* function은 <>로 쓰지 않고 el식으로 쓴다.

[17강. Fliter & Listener] 🡪 교재204p

- 필터는 데이터를 가공하는 것. 웹이 여러 개가 있는데 약 100개가 있다고 한다. 이 모든 웹은 다 응답과 요청이 가능하게 만들어져 있는데 어떤 기능을 이 100개의 웹에 다 넣으라고 할 때 일괄적으로 다 넣으려고 하면 상당한 작업량이 된다. 이럴 때 필터를 쓰는 거.

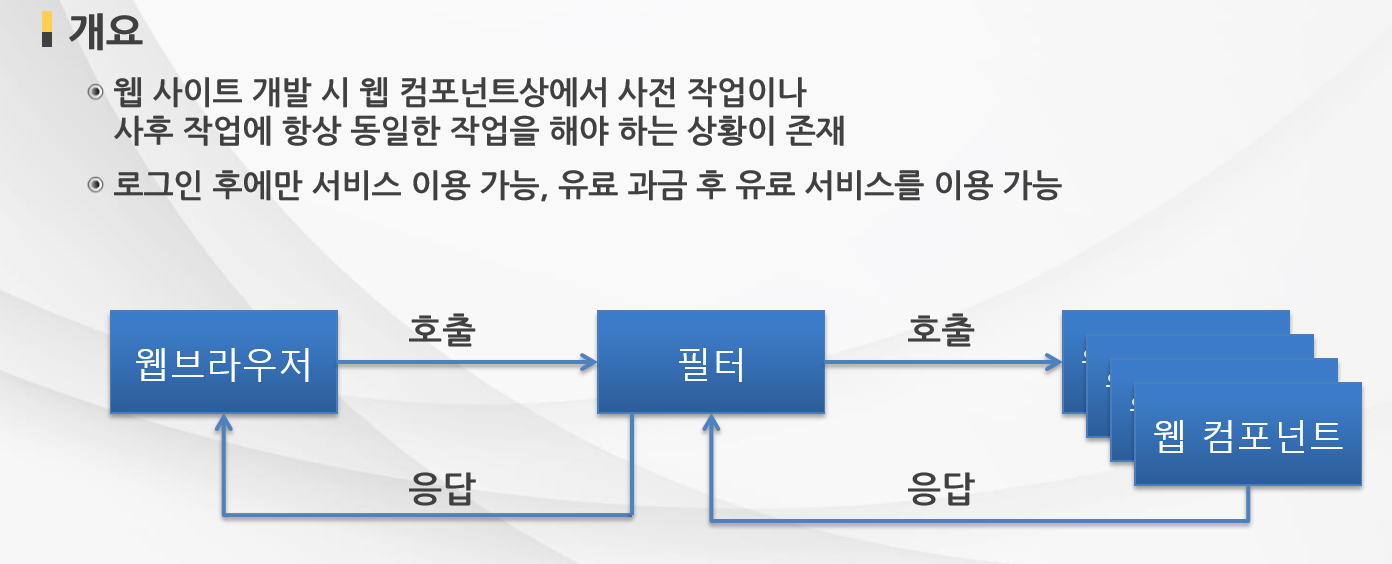
필터를 웹 앞에 넣어서 요청이 들어오면 필터에서 이 작업을 거쳐서 넘어가고 거쳐서 응답할 수있음. 이렇게 하면 1개 작업을 해서 100개 적용해서 생산성을 높일 수 있다.

사전, 사후작업에 대해서 어떤 조치를 취할 수 있는게 필터가 된다.

- 서버가 재가동이 된다든지, 요청이 들어온다든지 세션 이런것들에 대해서 컨테이너 입장에서는 다 이벤트라고 하는데 이 이벤트를 감지해서 뭔가 조치를 취해야 한다면 리스너를 구현해야함.

이런것들을 servlet/jsp에서는 제공을 해주고 있다.

1. 필터 이해 및 구성 요소 이해



- 사전작업: 요청 전, 사후작업: 응답 전

- 요청을 받기 직전, 응답을 하기 직전에 처리해야 할 것들. 그런 상황일 때 필터를 쓴다.

- 특정페이지에서 유료체크를 매번하면 상당히 코드가 복잡해지니까 필터에서 체크를 하는 거.

- 웹브라우저가 요청을 하면 필터가 먼저 체크하고 이상이 있으면 다시 돌려보내거나 다른 페이지로 보내서 응답을 하게 됨. 이상이 없으면 공통작업(처리)를 하고 응답을 함. 필터는 호출요청과 응답에 대해서 같이 처리해준다.



- implements Filter로 필터를 구현해줌. 구현해주면 기본적으로 init(생성), destroy(소멸)가 생김. 이것은 서블릿이 리로드될 때 한번씩 한번씩 구동이 됨.

- doFilter: 이 메소드에 리퀘스트와 리스판스 다 옴. 여기서 사전 작업과 사후작업을 진행하면 됨.

- 필터를 만들기만 해서는 처리가 안되고 등록을 해야됨.

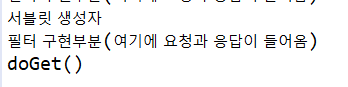


- 설정은 다 web.xml에서 함.

- 필터이름과, 필터클래스(패키지까지 다 포함해서 쓰기.)

- 필터를 매핑함. 특정 url하고. /\*는 /로 시작하는 모든 url에 이 필터를 적용하겠다는 것.

\* 필터 생성자, 초기화는 로드 단계에서 나오는 거. 필터가 먼저 호출된 이유는 jsp가 돌았기 때문.

\* 

필터가 먼저 돌고 doGet이 돌았음을 알 수 있음. 필터(doFilter)가 먼저 작동을 하고 그 다음에 서블릿으로 가서 응답(doGet)을 했다고 판단할 수 있음.

//세션이 없으면 MyServlet으로 갈 수 없다.

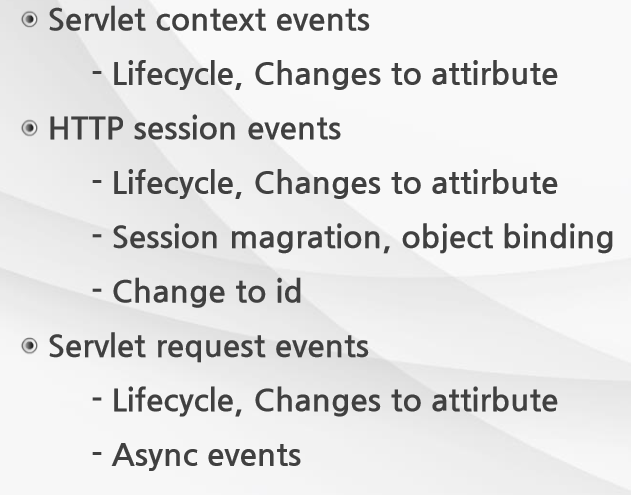
//특정 값이 포함된 요청이 아니면 해당 페이지로 갈 수 없다.

이런식으로 개발을 해서 필터에서 페이지까지 가는게 아니라 필터에서 걸러서 되돌려 버리는 거.

2. 리스너 처리

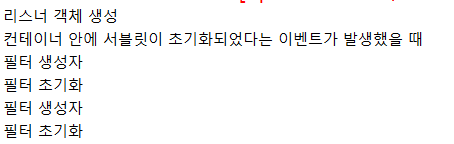
- (이벤트)리스너는 웹 컨테이너 안에서 event가 발생했을 때 자동으로 호출되는 코드.

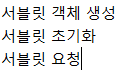
<event의 종류들>



- servlet context의 대표적인게 라이프 사이클. 라이프 사이클을 감지하거나 어트리뷰트(page, request, session, application)를 바뀌는 거.

- 세션 값의 변화. 요청의 변화, 라이프 사이클 등도 다 이벤트의 대상이 됨. 이런 변화가 있을 때 우리는 무엇을 할 것인가만 정해놓고 하면 됨. 필요없으면 안쓰면 됨.

\*\* <전체 프로젝트를 돌렸을때>

<서블릿 하나 돌렸을 때>

위 두 이미지를 봤을 때 리스너는 개별 서블릿에 대한 리스너가 아니라 Context 전체에 대해서 감지하고 있음을 알 수 있음. 서버가 재가동하면 해야할 것들을 리스너에 작업해놓으면 개별 서블릿을 편하게 쓸 수 있음. 이벤트들을 강제하는 것이니까 리스너들을 잘만 구성을 하면 서버 구성을 하는데 손쉽게 할 수 있다. 매번 뭐 할 때 마다 어떤 작업을 해야 하는데 올라오는 타이밍을 못잡겠다고 하면 리스너를 써서 잡으면 된다.

[18강. Wrapper] 🡪 215~

- Wrapping: 웹 브라우저와 웹 컴포넌트 사이에 데이터를 가공하는 것

- Wrapping하는 게 Wrapper

- 요청, 응답에 대한 래퍼 가능.

- 브라우저 – 필터 – 컨테이너로 되어 있는데 필터에 래퍼를 등록시켜서 요청, 응답을 가공시키는 거. 생산성을 높이고 개발을 용이하게 하는데 목적이 있다.

<래퍼 클래스의 이해>

- 클라이언트가 요청을 하면 웹서버가 받음. 웹 서버 안에 컨테이너가 있는데 이 컨테이너가 받아서 응답을 하는 구조를 가짐. 이 요청과 응답 사이에는 uid=kkk, pw=111이런 데이터가 왔다갔다 하는데 이런 데이터를 가공하는 게 래퍼의 목적. 이걸 하기 위해서는 필터가 필요. 필터가 받아서 가공해서 던짐.

- 요청을 가공: HttpServletRequestWrapper 클래스를 구현하면 됨.

- 응답을 가공: HttpServletResponseWrapper 클래스를 구현.

<요청 래퍼 구성>

- extends HttpServletRequestWrapper

- public String getParameter(String name): 파라미터가 결국은 데이터가 들어오는 위치.

- 생성자: 리퀘스트를 전역변수로 받아줌. 따라서 파라미터에서 바로 전역변수로 받은 request를 통해서 값을 뽑고 값을 무조건 대문자로 바꿔줌.   
- 절차: 요청한다(index.jsp) 🡪 그러면 필터가 잡음(근데 래퍼로 싸임) 🡪 그 다음 래퍼로 감. 래핑해서 대문자로 바꿈. 🡪 서블릿을 사용하면 바로 대문자로 출력됨.

< HttpServletResponseWrapper >

- extends HttpServletResponseWrapper

- 생성자에서 response 객체 받음.

- 응답 쪽의 대표 주자인 쿠키를 조작하는 부분. 쿠키값을 소문자로 넣는 것을 살펴볼 것.

[19강. Mysql을 이용한 Database JDBC 연결] 🡪 226~

- 라이브러리 설정: 자바 같은 언어쪽에서 mysql과 연결 위해서는 jdbc라는 인터페이스가 필요함. 이 라이브러리를 mysql에서 지원해줌.

<jsp query test> 🡪 life cycle에 최적화 시킴

- 페이지가 올라올 때 한번만 올라오는 멤버변수/메소드를 썼던 게 선언부. 여기에 커넥션 연결을 하면 페이지가 매번 연결하고 해제하는 부분을 그나마 해소 가능할 것.

<servlet query test> 🡪 life cycle에 최적화 시킴

- database 연결 및 해제는 시간이 많이 소요되는 작업이므로, 초기화 및 해제 코드에 배치. 기본 연결을 서블릿에서 했으니까 간략하게 수정만 하면 됨. 🡪 init(), destroy()를 만들면 됨. 그러면 이것을 이용한 디비를 연결하는 서블릿의 기본 폼을 살펴볼 것.

[20강. DBpool] 🡪 239~

- 기존의 연결방식: 요청이 들어오면 웹은 jdbc를 통해서 연결을 하고 쿼리를 치고 응답을 해서 연결을 종료함. 느려질 수 밖에 없음. 그래서 매번 연결, 종료를 요청할 때마다 하지 말고 이 부분을 따로 풀이라고 하는 부분에다가 연결 객체를 보관해서 요청이 오면 하나 빌려주고 응답해주기 전에 반납받는다.

- 커넥션 풀을 프로젝트에 설치해줘야 함.

<커넥션 풀>

- 커넥션을 미리 만들어서 풀에다가 커넥션 객체들을 집어넣는다. 필요할 때 풀에서 가져다 쓰고 다시 풀에 반납함. 그래서 커넥션을 생성하는데 시간이 소비되지 않음. 최초, 서버가 가동될 때 풀을 가득 채워놓음.

- 모듈설치: dbcp(풀링 라이브러리) 사용. 🡪 3가지 필요.

- META-INF에 context.xml 생성.

- maxActive = 최대활성화 수는 100개의 커넥션을 풀이 보관한다는 것.

- maxIdle = 평소엔 커넥션을 10개정도 유지하고 있는다. 10개 정도 유지하고 있다가 요청이 몰리면 커넥션을 만들 수밖에 없음. 처리해줘야 하니까. 이것의 최대치가 100개이다.

<!-- 팩토리가 커넥션을 찍어낸다는 의미 -->

- context.xml에서 정의한 리소스를 web.xml에 등록해줘야 함.

<커넥션 풀을 이용한 디비 연동>

- 설정이 끝났으니 쿼리를 쳐봐야함. 그렇기 위해서는 커넥션 객체를 획득해야됨. 풀에서 가져와야 됨.

//갖고오기 위해서 하는 코드

Context initCtx = new InitialContext(); //컨텍스트 초기화

Context envCtx = (Context)initCtx.lookup(“java:comp/env”) //env는 환경변수. 환경변수에 대한 위치를 가져온다.

DataSource ds = (DataSource)envCtx.lookup(“jdbc/java”) //그 중에서 jdbc/java이름을 얘를 가지고 와서

Connection conn = ds.getConnection(); // 거기서 커넥션을 구해온다

or

Context initCtx = new InitialContext();

DataSource ds = (DataSource)initCtx.lookup(“java:comp/env/jdbc/java”)

Connection conn = ds.getConnection();

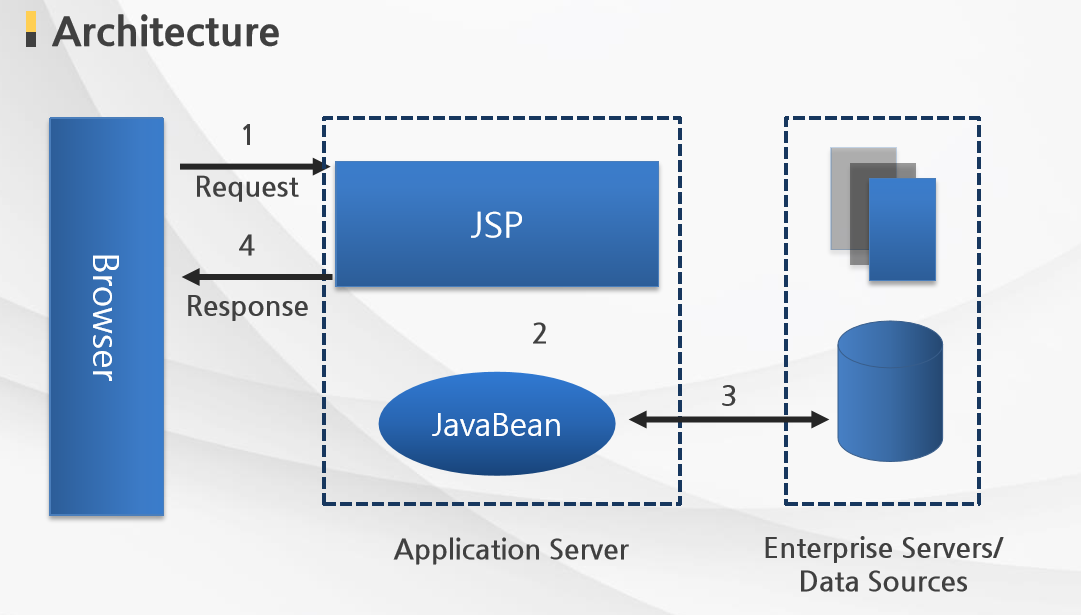
conn.close();

- 세트로 움직임. 커넥션 구해서 쿼리 치고 다 썼으면 반납하면 됨.

- init은 한번만 호출됨. 근데 매번 요청을 하면 doGet이 작동을 함. 즉 커넥션 풀의 위치는 doGet이어야 함. 그렇게 하면 매번 요청할 때마다 풀에서 커넥션 객체를 구해옴.

[21강. Model1 구현] 🡪 253~

<모델1의 서비스 흐름도 이해>



- 가운데는 application server. 여태껏 만든 jsp, servlet도 다 여기에 있음. 우리는 보통 이것을 컨테이너라고 얘기함.

- 요청이 들어오면 jsp가 빈으로 받아서, 요청온 데이터를 빈으로 담아서 디비 쿼리를 치고 다시 돌아와서 응답하는 구조 🡪 model1의 기본적인 서비스 흐름도

- 이것의 결론은 jsp가 혼자 다함. 받아서 빈에 넣고 쿼리치고, 받아서 응답데이터 꾸며서 보내는 것을 혼자 다함.

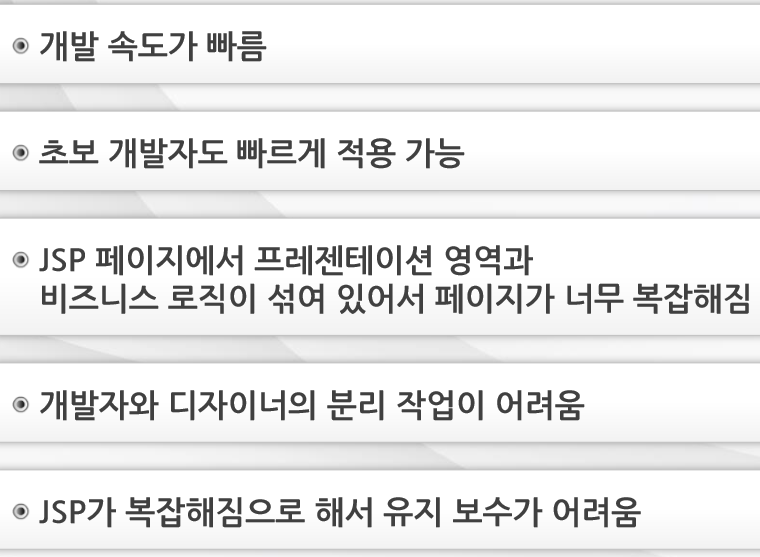
- 개발케이스:



- 빈은 데이터를 어떤 클래스라는 공간에 담아 놓는 거. 이런 것들을 담아서 덩어리로 보내고 받는 스타일.

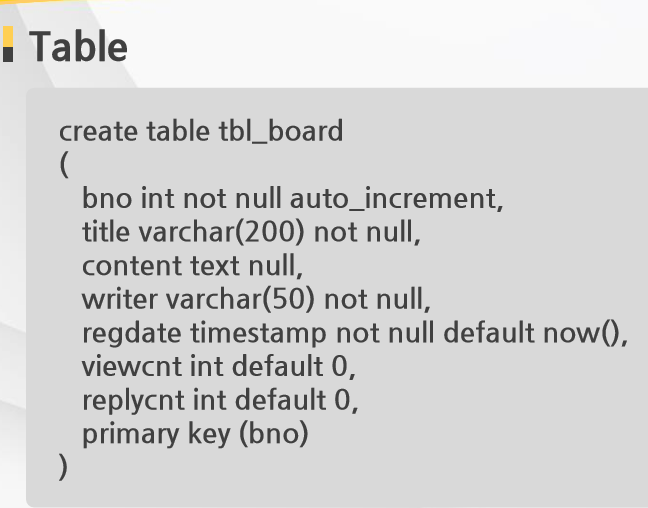
- jsp+custom tag는 model1에서 흔한 케이스는 아니지만 jstl이용해서 처리하는 부분. model2에 더 어울리는 모델.

- 장단점



개발 속도 빠름 🡪 jsp안에서 받고 디비가고 응답하는 거까지 다 해결하기 때문.

<게시판 테이블 연동>



- bno: 글번호 🡪 글을 쓰면 번호가 자동증가 됨.

- title: 글 제목

- content: 내용

- writer: 작성자

- regdate: 등록일

- viewcnt: 이 게시물을 본 카운트

- replycnt: 답변 횟수

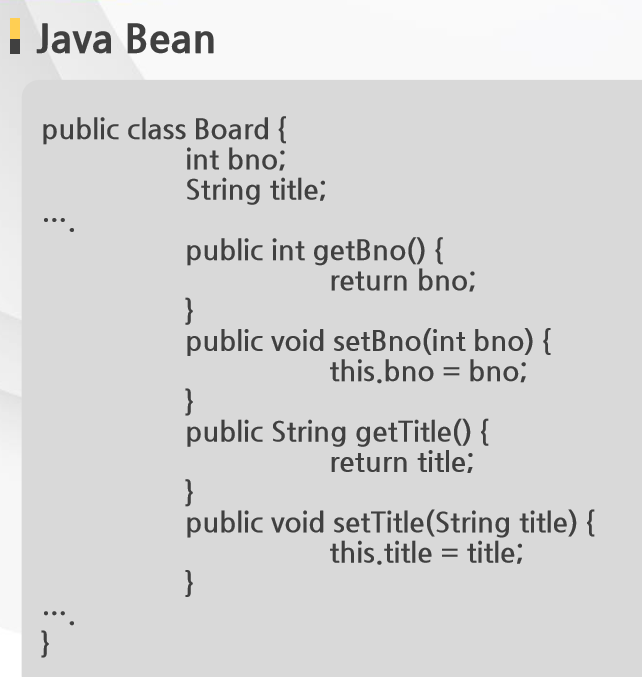
<실습방식>

- guest or kkk로 로그인하고 나서 로그인이 정상적으로 되면 게시물을 읽어와서 화면에 뿌려주는 방식으로 할 것.

- 로그인 같은 경우는

select \* from tbl\_user where uid=’guest’ and upw=’1234’를 해서 존재하면 결과가 나옴. 이런 식으로 로그인을 처리할 것.

- 전체 게시물 가져오는 것은 select \* from tbl\_board order by bno desc; // 최신순 🡪 내림차순

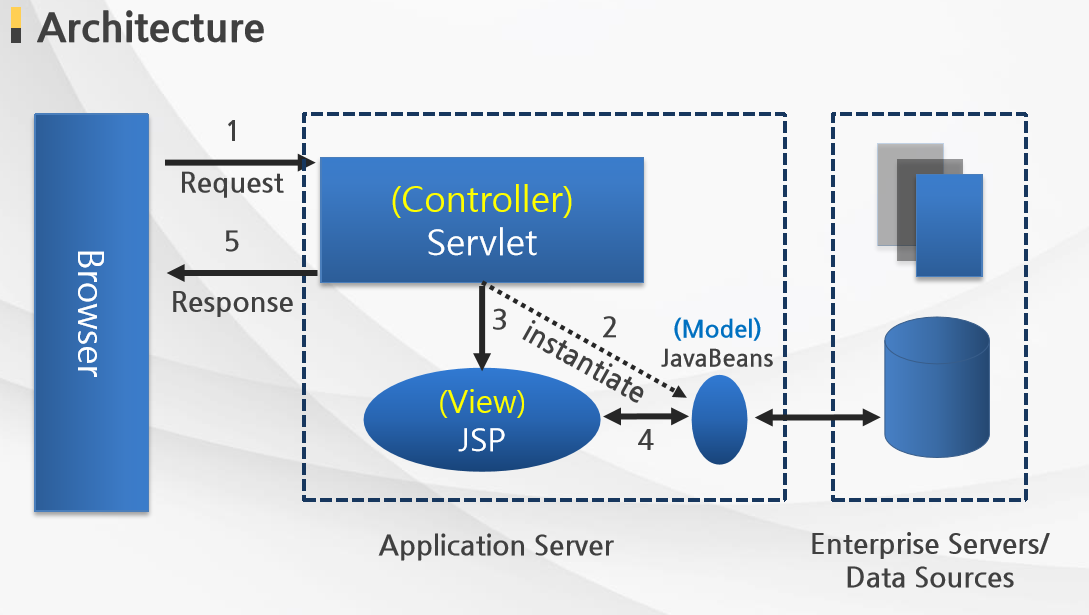


- 테이블에 있던 컬럼명을 고대로 멤버변수로 가져오고, 이거에 대해서 getter, setter를 만듦. 그래서 디비에서 쿼리가 수행이 되면 그대로 여기에 담아주는 거. 자동화된 형식은 아직 jsp 자체에는 없음. 나중에 여러가지 모델을 쓰면 자동 셋팅이 되긴 함. 지금은 수동이지만 셋팅을 해서 씀.

- 멤버변수에 컬럼명 그대로 넣어서 getter, setter 준비해서 디비 쿼리의 결과를 받을 준비를 함.

[22강. model2 구현

<model2의 서비스 흐름도 이해>

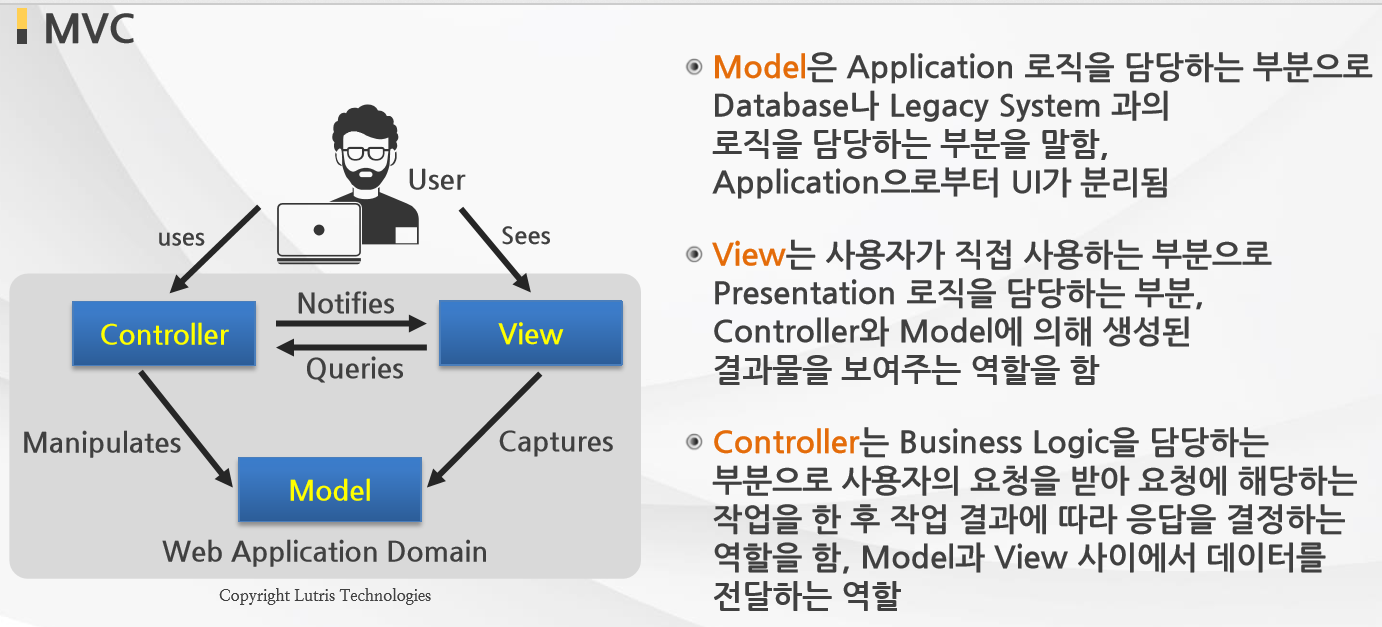


- 요청이 들어오면 컨트롤러가 받음. 얘는 브레인 같은 역할. 판단을 함. 받아서 디비로 쿼리를 보내서 빈으로 받아서 혹은 그냥 받아서 다시 어떤 연산을 함. 즉 비즈니스로직을 처리함.

- 쿼리를 통해서 빈으로 데이터를 담음. 이것을 우리는 model이라고 함.

- 컨트롤러는 연산을 한 결과물을 jsp로 던짐. 그러면 결과를 보여줌. 이것을 view라고 함.

- 그래서 model2는 MVC모델임.



-유저는 컨트롤러한테 요청을 하고 뷰를 통해서 결과를 받음. 그 사이에 모델이라는 게 존재하고 모델은 디비와 어떤 연관성을 가지고 있음.

- 컨트롤러는 어떤 연산을 담당함. 데이터 전달도 컨트롤러를 통해 전달됨.

- model은 application logic(=sql) 담당. 쿼리가 수행이 돼서 DB o legacy에서 데이터를 받아서 일종의 담는 그릇 역할을 함. 쿼리의 결과에 따라서 정렬된? 혹은 변형된 형태로 담기게 됨.

- 데이터는 모델, 결과물은 뷰, 브레인 역할은 컨트롤러.



<model1 서비스를 model2로 마이그레이션(이주하다)>

- 1으로 만든 모델을 2로 만든 모델로 변경을 시켜서 모델2에 대한 형태를 익혀볼 것.

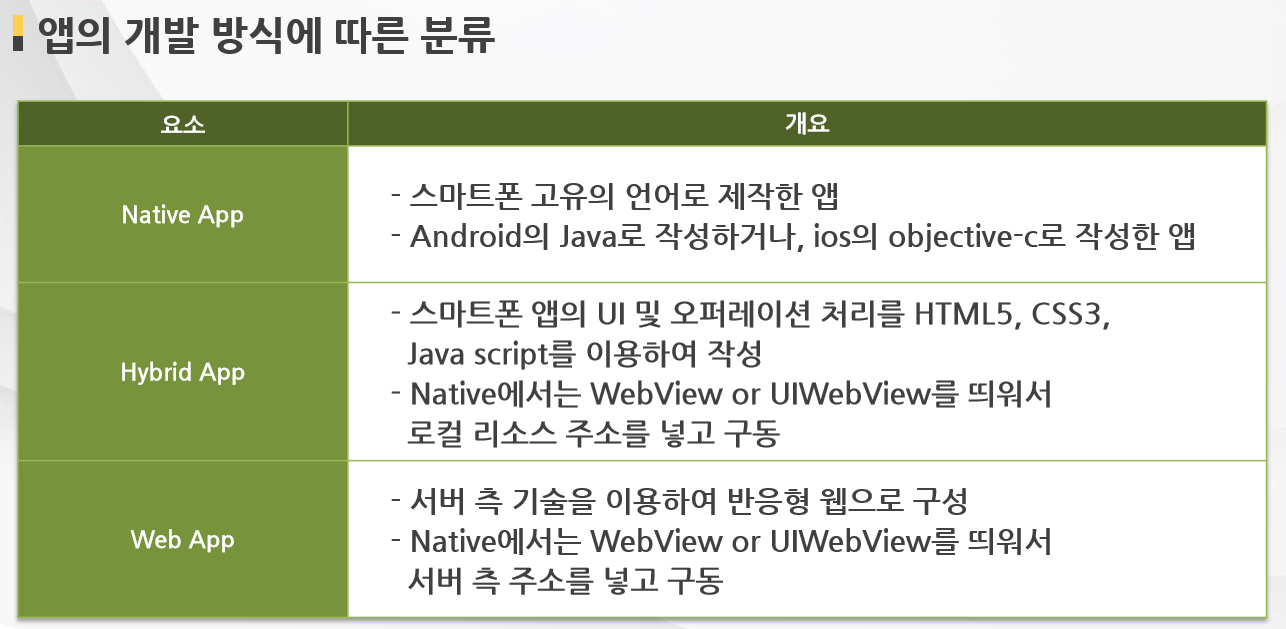
- 모델2에서 컨트롤러는 서블릿으로 함.

- 서블릿에서 컨트롤러 처리를 하고 결과를 다 실어서 뷰에다가 던짐.

[23강. 모바일 앱의 종류 및 서버측 처리 기술] 🡪 276~

- 실질적으로 모바일과 연동을 해볼 것.

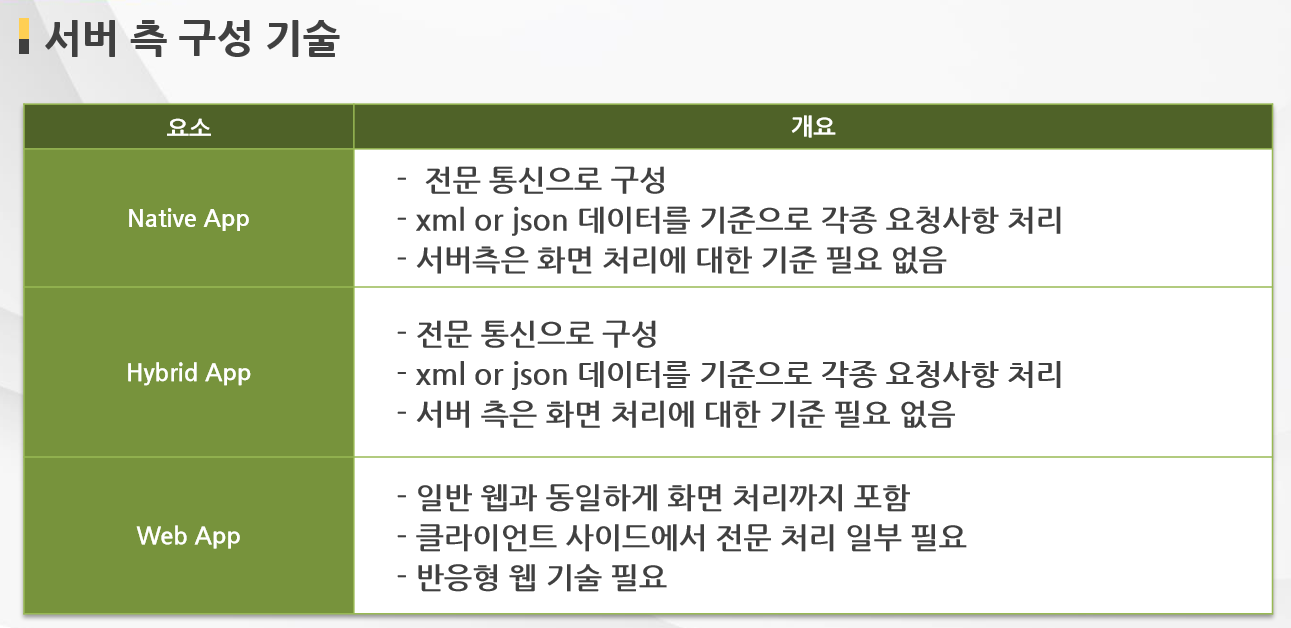
<앱의 개발 방식에 따른 분류>



- Native App:

- Hybrid App:

- Web APP: 클라이언트 쪽이 아무것도 없음. 클라이언트 쪽에는 webview밖에 없고 이것을 띄우는 url만 서버쪽에서 준비를 해서 주소만 넣고 띄우는 방식. 대표적인 게 네이버 웹, 항공사 예약앱등. 장점은 실시간 수정이 가능하다.

<앱의 종류에 따른 서버 구성 이해>  


- 네이티브와 하이브리드는 스타일이 똑같음. 전문 통신으로 구성. 전문 통신이기 때문에 view를 신경쓰지 않는다. 오직 data에만 집중을 함. 데이터를 표방하는 플랫폼과 언어에 독립적인 xml, json로만 데이터를 구성해서 보내줌.

- 하이브리드 앱은 자바스크립트의 ajax를 이용해서 통신을 하고, 네이티브는 자체 통신 라이브러리로 하는데 안드로이드는 volley, okhttp와 같은 라이브러리를 통해 통신을 함.

- 웹앱은 주고 받을 내용이 없다. 무조건 서버에서 다 완성시켜서 보내야함. 그래서 일반적으로 웹을 만드는데 모바일이니까 반응형을 많이 만듦. 그래서 반응형 웹을 만들어서 쫙 작성해서 일반적으로 똑같이 처리하면 되는데 브라우저가 모바일이라는 점만 기억해서 작업하면 됨.

<학습정리>

- 네이티 브: 스마트폰 자체 랭귀지로 개발. 안드로이드-자바, ios-object-c, swift

- 하이브리드: html, css, js로 만든 화면이 서버와 ajax으로 통신하는 스타일

- 웹앱: 반응형. 아예 서버에서 다 만들어 놓고 화면만 일종의 송출하는 기능. 우리가 앞에서 많이 했던 웹방식을 생각하면 됨.

- 네이티브와 하이브리드는 서버를 전문방식으로만 구성. 즉 xml, json만 데이터를 주고받으면 됨.

- 웹앱은 그냥 웹을 만들어서 주고 받으면 서버쪽 입장에서는 개발이 다 됨.

[24강. HybridApp에 따른 서버 개발]

- 하이브리드 앱을 위한 핵심은 웹서버에 있는 jsp or servlet을 구성해보는 거. 방식은 model2.

- 실습을 위해 로그인 처리, 회원가입 하는 부분 진행할 것.

- model2 방식으로 서버를 구성. 클라이언트 쪽은 html5, css3, js로 구성할 것. 얘랑 서버랑 ajax로 통신하는 형태. 통신데이터 결과는 json으로 도착하는 방식으로 처리.

- cordova는 3번까지 진행한 후 바로 구동시켰으나 우리는 구동하지 않고 안드로이드 스튜디오에서 이 프로젝트를 열어서 진행할 것.

 - 아까 index.html 작성했던 부분이 바로 이 내용이 되는 거.

지금 안드로이드는 html로 띄운게 됨. 이거는 바로 하이브리드앱이 되는 거.

- 웹에서 하이브리드 띄울 수 있게 준비를 해놓고 여기다가 파일을 집어넣어서 연동만 시키면 됨.

<회원 테이블 구성 및 sql 구성>

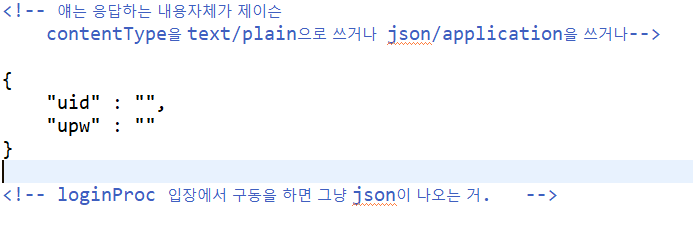
- 우리가 서버측에서 처리할 거는 회원가입과 로그인 2개.

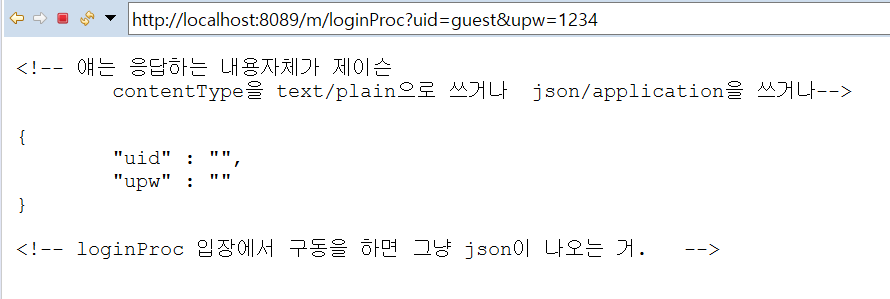
- 회원가입: insert into tbl\_user(uid, upw) values(‘uid’, ‘upw’);

- 로그인: select \* from tbl\_user where uid=’uid’ and upw=’upw’

- 응답은 코드1은 성공, 0은 실패로 간주. 로그인 성공하면 id, pw를 제이슨 형식으로 클라이언트로 보내주는 식으식 해볼 것. 사실 pw까지는 필요 없음… 응답 전문은 json으로 함.

json을 jstl로 만들어 볼 것. 나중에는 json을 라이브러리화 시켜서 zisun이라는 라이브러리을 활용할 것. 일단은 jstl로 json을 구성하여 응답한다. 이런 구조로 만들어 볼 것.



- 이렇게 받아야 하는데 이 자체가 화면을 가진 게 아님. 그래서 요청을 했는데 화면이 보내진 게 아니라 데이터만 보낸다. 이게 바로 전문 통신이 되는 거. 그렇다면 login.jsp입장에서는 결과를 받아서 뭔가 어디로 제어를 해야됨. 그래서 여기서부터는 주로 뭘쓰냐면, 하이브리드앱은 자바스크립트를 중심으로 쓰게 됨. 자바스크립트 중 보통 많이 쓰는 게 jquery.

- jquery를 직접 프로젝트에 넣을 수도 있고, cdn으로 url을 걸어서 쓸 수도 있음. 우리는 other cdn 중에 구글에서 지원해주는 cdn을 쓸 것.

- 제이쿼리는 기본적인 객체들을 $로 표현함. 🡪 $()

- document: html 문서 객체. html자체를 문서라고 표현함.

- <form>에서 submit을 누르면 바로 감. 이 이벤트를 인터셉트해서 ajax로 통신한다.

- 서버를 갖다와서 이렇게 데이터 오는게 확인이 됨.

- 하이브리드로 할 때는 다 html로 해야 됨.

- login.jsp를 html로 바꾸면 하이브리드의 실질적인게 됨.

- 하이브리드를 할 거니까 모두 다 html로 바꿈. 서블릿 같은 거는 말고…

- 회원가입 버튼 누르면 joinProc에 가고 여기서 일련의 과정을 쭉 거친뒤에 디비에 삽입이 되고 정상적이면 1, 아니면 0이되서 joinOk로 포워딩이 됨. joinOk에서는 코드값 찍어서 결과를 돌려줄것이고 이것을 join.html이 받아서 1이면 성공, 아니면 실패로 떨어지게 될 것.

login.html 🡪 join.html 🡪 joinProc 🡪 joinOk 🡪 join.html

- 이제 로그인, 회원가입하는 이 부분들을 하이브리드 앱쪽으로 옴겨야 함. 38:20~

[25강. NativeApp에 따른 서버 개발]

- json 처리 방식으로 gsun 파서를 이용. 🡪 구글이 만든 파서. 자바랑 유용하게 사용 가능.

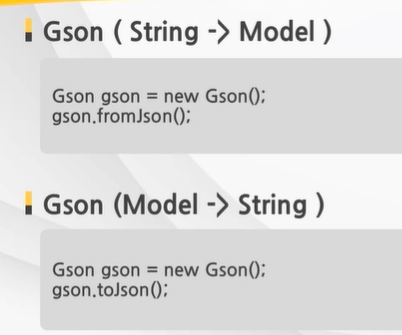
- 서블릿에서 바로 gsun을 이용해서 json을 응답하는 방식 🡪 스프링에서 하는 방식과 유사.

- volley: 통신 쪽 라이브러리

- Gson: json을 파싱하는 라이브러리.

- 서블릿할 때는 maven으로 안드로이드 할때는 gradle로   
<전문 통신을 통한 데이터 파싱 처리> 🡪 서버 쪽에서 처리하는 부분 살펴보기

- 데이터 처리할 때 gson을 이용해서 자바 레벨에서 처리하겠다.



- 스트링으로 온 걸 모델로 담는 게 fromJson이고, model로 된 거를 string으로 뽑는 게 toJson. 함수 매개변수로는 model class를 넣어서 json데이터를 한번에 받아내거나 클래스에 담겨있는 내용을 json 문자열로 바로 뽑아내는 역할을 하는게 gson.

- dispatcher 부분을 gson을 가지고 제이슨 스트링을 만들어서 응답하는 방식으로 바꿀 것.

- 안드로이드와 서버 연동 주의점

1. 서버 포트와 안드로이드 쪽 에서 로컬호스트로 접속하기 위한 10.0.2.2:port에서 포트 번호는 같아야 한다. 즉 서버포트가 8089면 10.0.2.2의 포트번호도 8089여야 한다.

2. 서버가 돌아가고 있지 않으면 당연히 연동도 오류가 난다.

3. 1,2번을 잘 지켰는데도 연동이 잘 되지 않는다면 AndroidManifest.xml의 <application> 부분에 **android:usesCleartextTraffic="true”를 넣어준다.**

<**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"  
 android:usesCleartextTraffic="true"**>  
 <**activity android:name=".MainActivity"**> 🡪 이런 형식으로

위의 것을 집어넣기 전에는 오류: java.io.IOException: Cleartext HTTP traffic to 10.0.2.2 not permitted 오류가 떴는데 이것을 집어넣자 아주 잘 돌아간다.

- 네이티브는 이렇게 서블릿,jsp와 json으로 통신을 해서 결과데이터를 가지고 화면을 구성하는 방식이 되는 거. 하이브리드는 html로 잡고, 네이티브는 자바로 잡는 다는 점이 다르고 서버 자체는 동일함.

- 서버측에서는 gson 라이브러리를 메이븐을 통해 설치하고 jstl사용 안하고 서블릿에서 바로 gson을 이용해서 json 데이터를 만들어서 클라이언트로 보내줌.

클라이언트와 서버 모두 gson라이브러리를 쓰고 똑 같은 모델을 사용함.

[26강. WebApp에 따른 서버 개발] 🡪 pass….