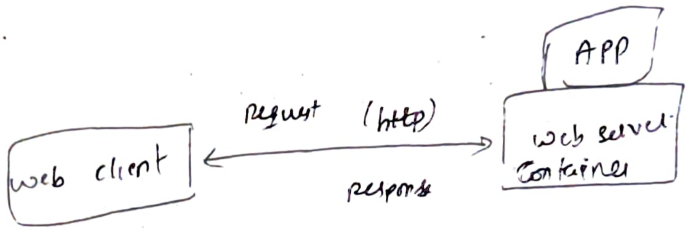
Java Web Development

Web Application

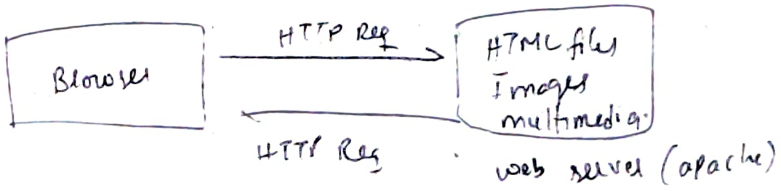
Definition:- Any application whose services than can be accessed through web.



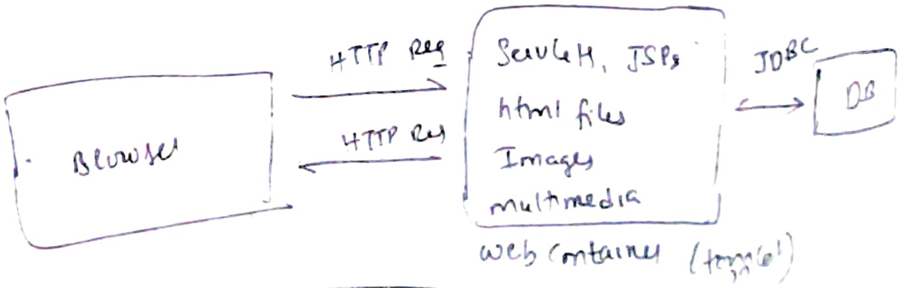
web apps	web Servers
google	Apache (not php extensions)
facebook	IIS.
	(not nodejs)

Static vs Dynamic web Application

Static



Dynamic.



→ dynamic ~~&~~ web Application runs on web Container not web server

→ Since web container also knows how to run JSP, servlets etc., along with sending static files

→ i guess here, web server + web container + ~~Servlets + Jsp~~ + much more (EJB) = Application server.

→ i guess tomcat has limited set of features (web server + specific cont ainer) to call it a full fledged Application server (in java world)

Container :- runs servlets inside it.

Servlet

lifecycle

init() → service() → destroy().

Container
- → Instantiation (Instantiates servlet)
- → Initialization (runs only once, setup DB Connections etc., etc., init().)
- → Servicing (calls service(); n times for n requests from client)
- → Destruction (calls destroy(); opposite of init();)

webApp standard structure by oracle that every java EE application should follow to be deployed in web container | Application server

```
WEBAPP
 |- home.html          ( servable files).
 |- login.jsp
 |- :
 |
 L WEB-INF
     |— web.xml        (Deployment Descriptor- for Configuration)
     |— classes        ( put all business logic)
     |    |- com. abc . ABC. class
     |    L :
     |— lib            ( 3rd party libraries) (will be in build path
     |    |— mysql.jar                          (added by tomcat)
     |    |— spring.jar
     |    L hibernate.jar
     L—— JSP (?)
```

Servlets:-

→ used for Dynamic web Application Creation.

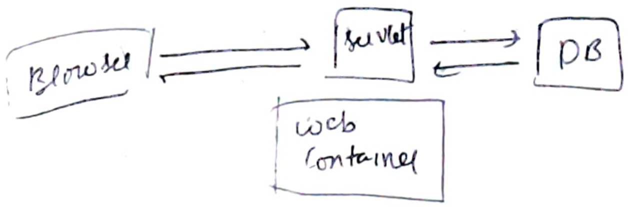
→ Technology in EE standard

→ Specifications (Specs for web Container developers)

→ API (interfaces, classes for making web apps)

Servlet

→ A java program running in web container.
(class)



→ takes requests, does logic, dB stuff, gives html or other responses

Servlet Annotations

→ Introduced in 3.0 version of Servlets
→ Configures various ~~web~~ servlet components instead of using web.xml

→ either Annotations ~~and~~ or web.xml ~~both~~ can be used, but web.xml will override annotations - if something is configured in both

* javax EE — jakarta EE

[implement Servlet (or)
extends GenericServlet]

```
class HelloWorldServlet extends GenericServlet {

@Overide

public void service( ServletRequest req , ServletResponse res)
        throws ServletException, IOException {

        res.setContentType("text/html");
PrintWriter   res.getWriter();
    out =

        out.println("<html>");
        out.println("hi");
        out.println("</html>");
```

① Create Servlet Class

② Create Deployment Descriptor

```
</web-app>

 <servlet>
   <servlet-name>
   <servlet-class>

  <servlet-mapping>
    <servlet-name>
     <url-pattern>
```

request.getParameter("name");

→

JDBC Architecture

JDBC client our application code using JDBC

JDBC api : java.sql.*

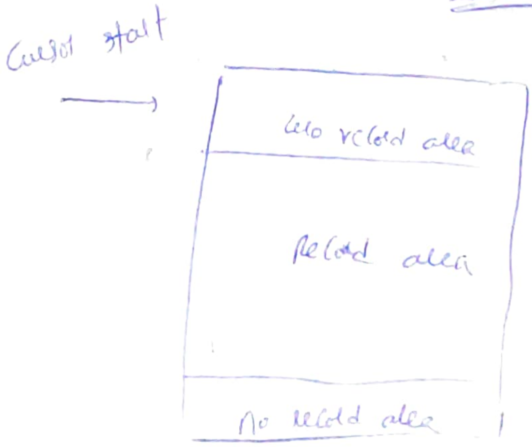
JDBC Driver :- Interface b/w DB and client (via Api)
(Can be from dB vendor or 3rd party)

JDBC DriverManager :- helper to setup JDBC driver
(runs only once, gives connection object)

→ ~~JDBC Driver also converts DB output to what java can understand (?)~~

`jdbc:mysql://localhost:port/dbname` url for Connection

ResultSet



→ next(); moves cursor one record at a time

→ next() also returns boolean if record is present at cursor

→ getXXX("colname");
(XXX: String, Integer)

→ upcasting is available with getxxx(" "),
(int can be obtained by .getDouble(" ");)

⇒ try (
 // anything created here autocloses if
 implements autocloseable interface (java7)
) {

}
// no need finally.

⇒ DriverManager() uses service provider mechanism for
find driver

Get vs Post

→ get is default. | → post is explicity
→ no body portion | → has body portion
→ Data belong to query string | → Data is part of body / payload.

→ has request and header fields

→ Restriction on length of query string | → no restriction

→ Get is idempotent

(doesn't change data in dB, if you do, dont!)

(only use for fetching)

→ Not idempotent

```
Class.forName("fqpath");  // used to load class dynamically
                          // when className is not known
                          until runtime, we can use this

                          // same as cpp dynamic DLL loading
```

Note:- .class files are enough to resolve symbols when compiling .javafiles that uses those symbols

so in essence, when using libraries and you import a class, having that library jar (without .java files but only .class files) in classpath is enough

---

<u>Init params</u>

```
<servlet>
  . . .
  . . .
  <init-param>
    <param-name>
    <param-value>
```

⇒ Init params can be passed to servlet using
web.xml.

→ helps in reducing hardcoded data

```
init( ServerConfig config) {

String value = config.getInitParameter(" pname ");

}.
```

<u>Annotations</u>

```
@WebServlet (" /url")

@WebServlet (urlPatterns = "/addServlet",
        initParams = {
                @WebInitParam (name=" ", value=" ")
        }
)
```

<u>ServletContext Interface</u>

→ Container Implements this and is available to all servlets

→ One servlet Context for Application

→ Shared by all Containers

→ Containers inject ServletContext while initializing servlet, once injected, we can retrieve it in several ways

```
init() {
    getServletContext();
}
init(ServletConfig config) {
    config.getServletContext();
}
service() {
    getServletContext();
}
```

<u>uses</u>

1) Store and manipulate data among Servlets

    setAttribute();
    getAttribute();
    removeAttribute();
    getAttributeNames();

2) Deal with context params

3) Create RequestDispatcher object for Inter servlet communication

4) To store information into server log files using log() method (rarely used)

Context Parameters

```
<web-app>
    <context-param>
        <param-name>
        <param-value>
```

Same as init params but ~~not~~ associated with ServletContext and not with each Servlet (basically global to all servlets)

String value = ~~serv~~ context . getInitParameter("name");

=> attributes cant be injected via web xml. Context, init params can be.

working directry	staging area	repo

git config --global -e
git config --global -list

root commit = first commit

~/.gitconfig.

$ . . .
. . .
. .

to https:// - - - - - -
5c05047..ba5b0f0 master → master

local comit

remote commit

→ $ git commit -am "commit" (add & commit in one go, only works with tracked file)

→ $ git ls-files (shows files in staging area)

→ git commit clones staging area into repo

→ git reset HEAD file.txt (~~unstages~~ ~~changes~~ restore staged file to as its in HEAD)

Prepared Stament

→ precompiles query statements for perfomance

→ child interface of Statement

```
PreparedStatement s = conn.preparedStatement("  ");
```

```
insert into employees values (?, ?, ?)
```

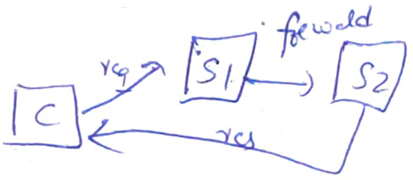
```
s.setXXX();        s.setInt(1, 100);
                   s.setString(2, "hello");

s.executeQuery();    // no parsing of Query
s.executeUpdate();
```

Interservlet Communication (with Request Dispatching)

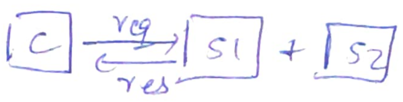
```
rd = req.getRequestDispatcher(uri);
```

(of servlet or static resoure)

```
rd.forward(req, res);
```



```
rd.include(req, res)
```



```
req.setAttribute();
req.getAttribute();
```

} to communicate

→ if Dispatched to static resource, the include/forwarded (req, res) are useless

Servlet Initialization

1) Lazy :- inits on first request [init()]
2) Pre :- inits on load

⇒ web.xml   servlet
          servlet-name
          load-on-startup : 1 (priority)
                            2 (~~high~~ low)

⇒ Pre is used by Spring MVC for example..

⇒ @WebServlet(urlPatterns="/abc", loadOnStratup= 0)

---

service(); takes priority over dopost and doGet

Listeners

→ event handling can be done

→ implements HttpSessionListener (for session)

ServletRequestListener (for req)

⋮

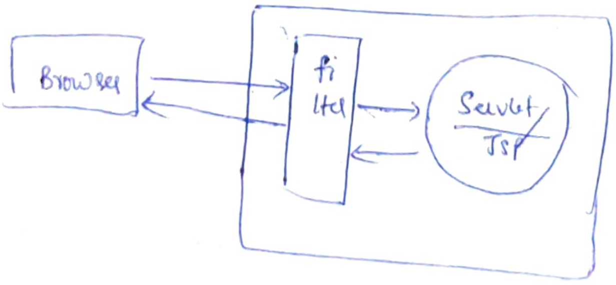
→ @WebListener

→ web.xml

```
<listener>
    <listener-class> MyListener
```

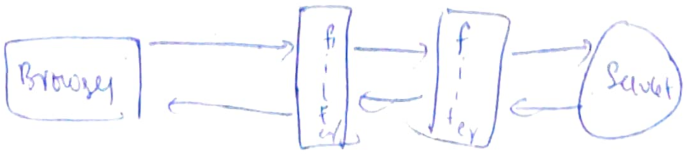
```
void requestInitialised(
        ServletRequestEvent e) {

}
void requestDestroyed(
        ServletRequestEvent e) {

}
```

Filter



→ filters can be ~~changed~~ chained

→ filters can be applied on ~~most~~ req, res



⇒ @~~Setter~~

⇒ implements Filter

```
init(FilterConfig conf) {
                 ^—— //similar to ServletConfig
}

doFilter(ServletRequest req, ServletResponse res, FilterChain chain) {

        // Chain has info about next location (filter or servlet)
        chain.doFilter(req, res);
}

destroy();
```

⇒ All filters are pre initialized

⇒ @WebFilter("/url")

Sessions

→ http is stateless for performance and scalability reasons

```
HttpSession s = req.getSession();
→ s. ~~get~~ set Attribute(String, String);
⇒ (String) s.getAttribute(String);
```

// Creates a session. if no sessionid is present in req, and returns the sessionid, res also now includes sessionid

[sessionids are stored in cookies]

// if req has session id then it will return that session

Session Lifetime

→ login to logout
website ~~[illegible]~~ open to close
tab etc.,

Session tracking

1) Identify user
2) State maintenance

Session Expiry

→ Explicit logout [use invalidate()]

→ no interaction [timeout, webcontainer destroys after]

ookie: JSESSIONID = XXXX (session entry)

Cookies

Cookies[] cs = req.get Cookies()

cs[0]. get Name()
cs[0] .get Value();

res. add Cookie (new Cookie ("name", "value"));

(tomcat can do url rewriting, if cookies are disabled, so that sessionid can be passed via url)