**Java For Web Application**

**Application** 🡪 Là chuyển đổi toàn bộ qui trình mua hàng trong siêu thị 🡪App mua hàng trên online.

**+ Have:** Class Student{} 🡪 **How to use class**.

**+Use Class:**

**Student student = new Student();** 🡪 **Khởi tạo Object** 🡪 **Phân phác bộ nhớ**.

**+Size of an object in JAVA:**

1.Unlike C/C++ ( has function size of () to get size’s memory of an object)

2.Java has no direct method.

3.Using instrumentation package.

4.Any object has 2 components for memory:

+ 16 bit for memory ( depend on system).

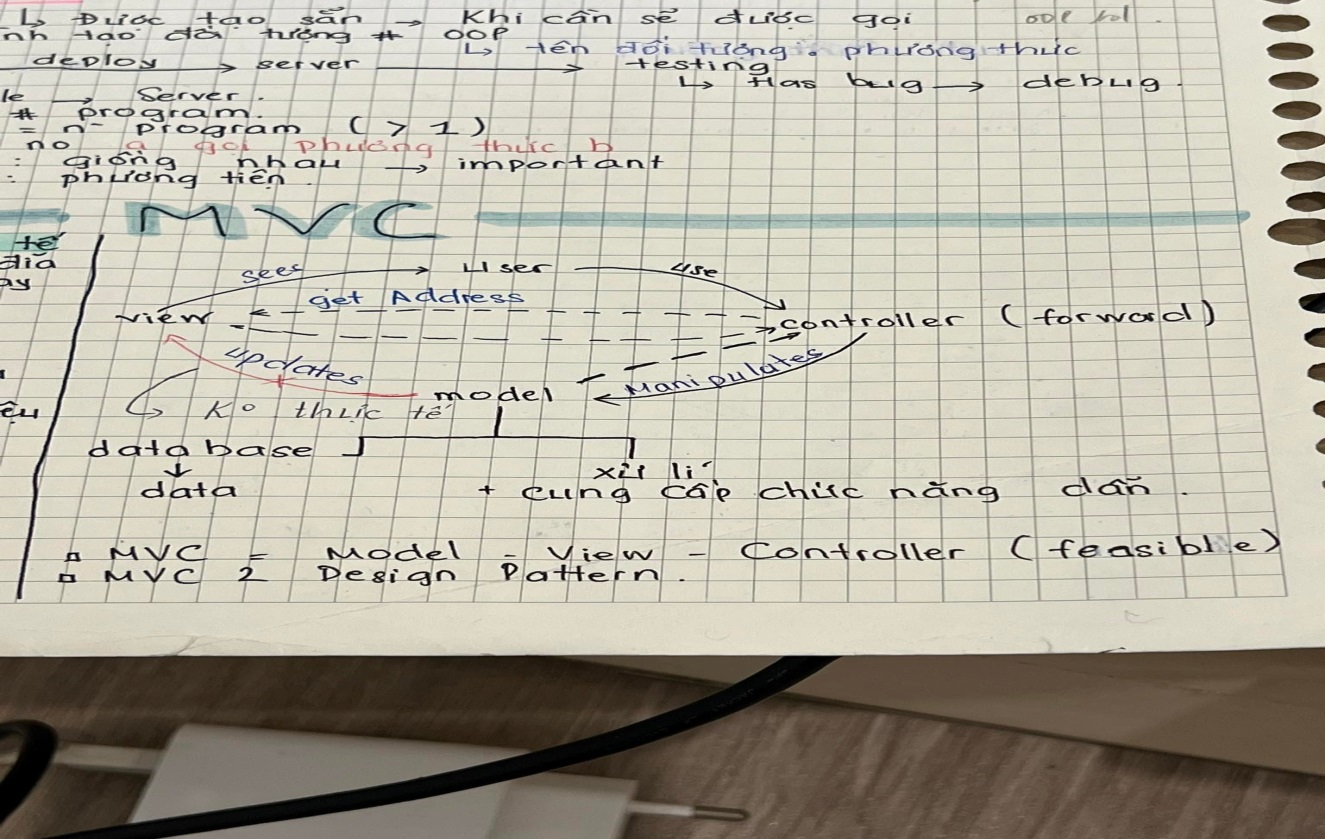
+ Memory for data initialized(data initialized: dữ liệu khởi tạo🡪 int(32), float(32), String(number of character or character size: 8)

**+ Syn of int/float:** **memory + size( int/float).**

**+Syn of thing**: **memory + char size \* number of char**

**+Create new object()**: **Size = 16** // 16 bits just storing address of object.

**MVC**

+ 

**+Tình huống thực tế:**

1.Xem nội dung các đĩa trong đầu máy

2.Chiếu lên TV.

3.Sử dụng Remote.

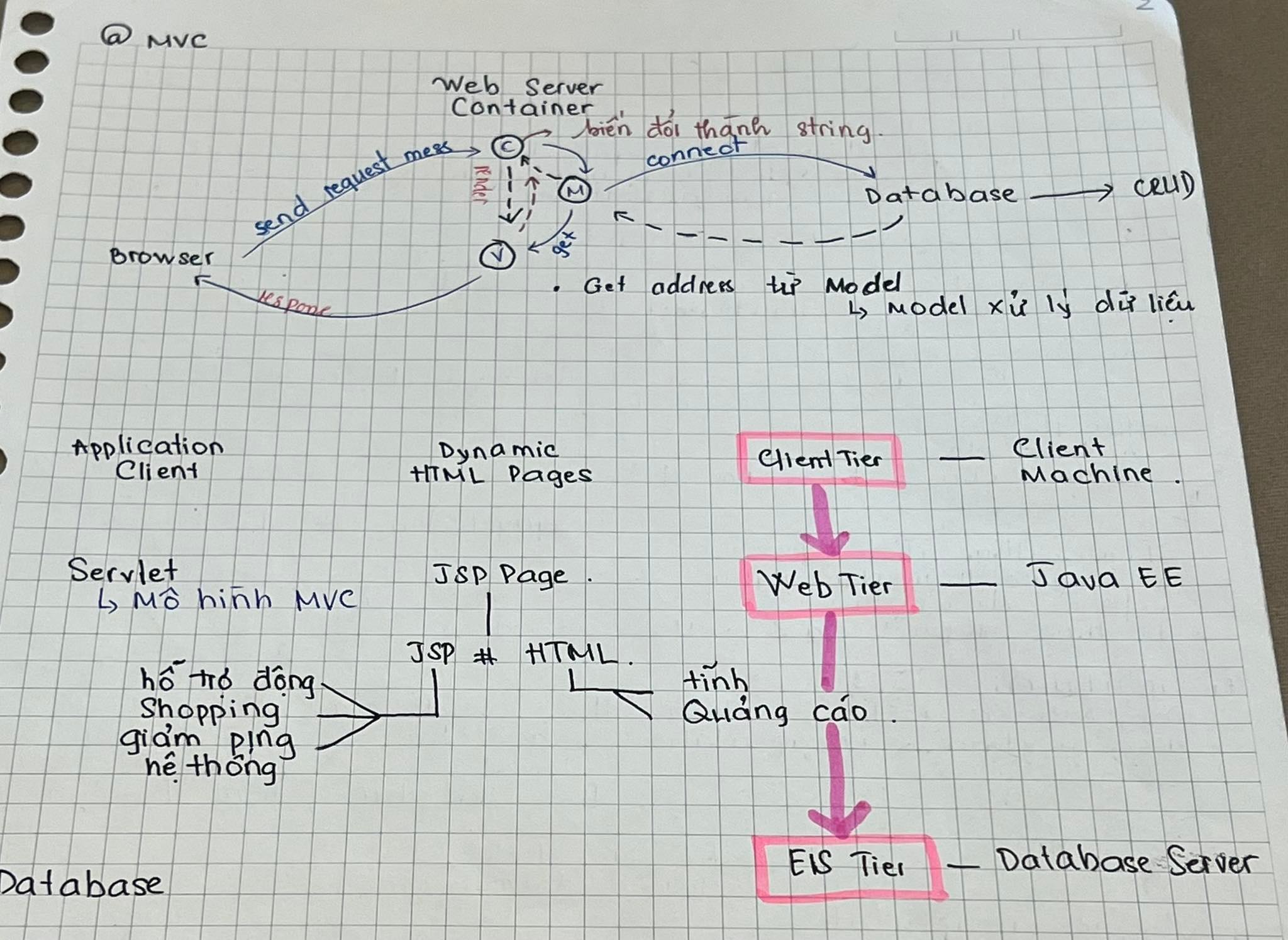
**+Remote:**

1.Phát tín hiệu.

2.Mạch 🡪 Phát tín hiệu hồng ngoại.

3.Không có: CPU(***Central Processing Unit)***, Bộ nhớ, Vi xử lí

4.Không xử lí.



**Component**

+JDBC = Java Database Connect.

+ PBL \_Web: 1.Login Servlet JDBC

2.Search Break down.

**JAVA Classes**

+**Abstract class**: **lớp trừu tượng** 🡪 **Ẩn cài đặt chi tiết + chỉ hiển thị tính năng đến người dùng**.

+**Abstract class:**

**CLASS CHA – Khai báo phương thức( phương thức quyết định bởi class con)**

**+Phương thức Abstract:**

1. Không có thân phương thức

2.Không có định nghĩa

**+Kế thừa Abstract class 🡪 Use extends.**

**+Encapsulation: tính đóng gói**

- Đóng gói các thuộc tính + phương thức inside class.

-**Access Modifier** 🡪 **Encapsulation**:

+ Ngăn chặn sự truy cập của các lớp sau.

+Thay đổi phương thức một thuộc tính của class 🡪 **Che giấu dữ liệu** **( Data hiding).**

**- Why use encapsulation ? ( tại sao sử dụng tính đóng gói)**

+ Đóng gói các phương thức + thuộc tính liên quan vào 1 class.

+ Thay đổi giá trị thuộc tính thông qua phương thức

+ Rõ ràng + dễ đọc.

**- Access Modifier ( public, protected, default, private).**

**+Inheritance**: **Tính kế thừa**

- Phát triển lớp mới dựa trên các lớp đã có.

- Xây dựng lớp mới có sẵn các đặc tính của lớp cũ.

- Lớp kế thừa = lớp con

-Có thể khai báo thuộc tính + phương thức phù hợp.

- Tái sử dụng mã nguồn thông qua lớp.

**The Servlet Model**

**- Nội dung chính:**

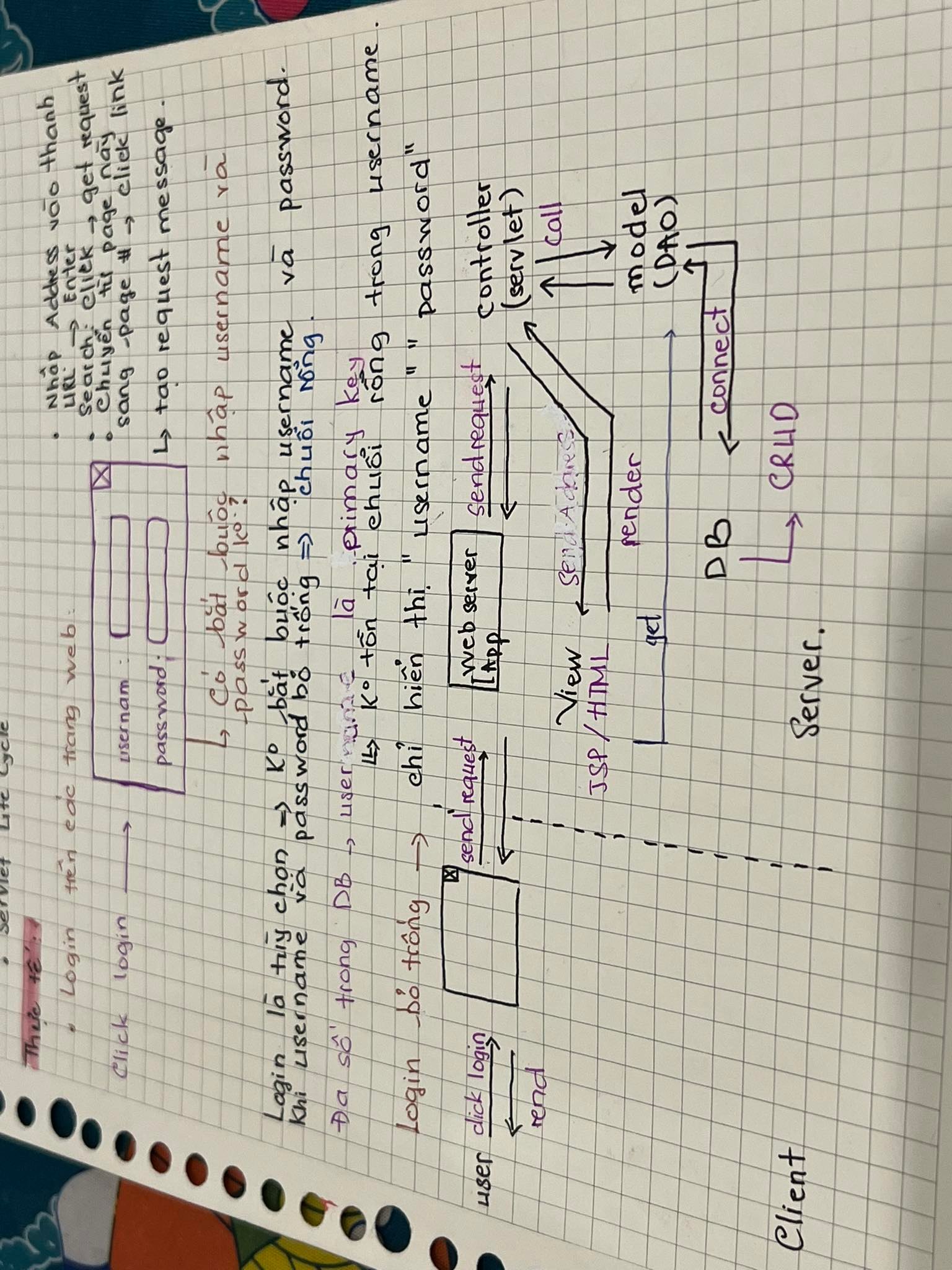
+ HTTP Methods

+ Form Parameters

+ Servlet Life Cycle

+ Requests

+ Responses.



**DAO**

- DAO = Model

**- Một thành phần DAO = 1 table trong DB**.

- Data Access Object.

- Java Class

-**Cung Cấp phương thức Access trên DB**.

- Đặt tên gợi nhớ:

**+ Tên table dưới dạng DB + chữ cái đầu viết hoa + DAO**

+ VD: table\_name: tbl Student

🡪DAO\_name: TblStudentDAO

- **Nằm trong Package tên tbl + chữ cái đầu viết thường.**

+VD: tblStudent.

**Model**

-Sản phẩm của Java Class.

-Cung cấp phương thức để get/thao tác trong DB.

- Maps 1-1: dễ kiểm soát /Controller.

-Là Keyword để giao tiếp với list java.

🡪**Phân tích object ban đầu thành** : **bit string, byte string 🡪 để giao tiếp 🡪 gọi là cơ chế zenalizable**.

**Make Application.**

**1)Có Web Server: Open NetBean: Quan sát + control Server:**

**Window**🡪**Services**🡪**Server**🡪**TomCat**🡪**Right click**🡪 **Start**🡪**Open Output**🡪**All lines have org.( Không đỏ)** 🡪 **End Start** 🡪 **Find line: Server Start up….**

**2) Kiểm tra sự sống của server:**

Line: Server Startup… 🡪 **Nhìn lên 2 dòng** 🡪 **HTTP + số** (**8084**) 🡪**Open Web**🡪 **gõ vào URL** 🡪**Enter**🡪 **Có TomCat** 🡪 **Success**.

**3)Tạo Project:**

File 🡪 **New Project** 🡪 **Java Web** 🡪 **Web Application** 🡪**Nhập Projectname** 🡪**Project location** 🡪**Next** 🡪**Server** 🡪**Java EE Verson** 🡪 **Context Path** 🡪 **Next** 🡪 **Finish**.

**1.Project location:**

-Không lưu ổ C.

-Tên không khoảng trống/ link dẫn không có khoảng trống.

**2.Server:**

-TomCat.

**3.Java EE Verson:**

-7

**4.Context Path:**

-Tên Project ( ứng dụng)

- Chứa toàn bộ App trong Server

- Tên folder trong serve.

🡪 **Tự tạo tên folder chứa App.**

**5.Project**: có hình quả địa cầu.

**6.Web page**: ( từ META – INF and WEB – INF)

- Tầm vực public 🡪 **User có thể sử dụng Browser để Access trực tiếp all resource**.

**7.META – INF folder:**

- File: context.xml

deploy

🡪**Context.path + value**

🡪**Sử dụng thay đổi , sửa chữa** 🡪 **Customer**.

🡪**Có thể thay đổi tùy ý.**

**8.WEB – INF:**

+Các package folder còn lại là private.

**4) Bỏ chọn Deploy on Save:**

- **Reason**: 🡪 **Nếu code chưa complete** 🡪 **Bug**.

- **Way**: **Right click project** 🡪 **Properties** 🡪**Run** 🡪**Deploy on Save** 🡪 **Bỏ chọn.**

- **Explain**: **Khi Ctrl + S** 🡪 **Tất cả code được đóng gói** 🡪**Server**.

- **Finish bỏ chọn** : **Hình quả địa cầu (project) có thêm cây búa**.

**5) Clean and Build project:**

- Right click project 🡪 **Clean and build**.

-Không giống clean and build của java class.

-Process of clean and build.

1.Netbean đi vào bên trong webserver/tomcat server duyệt để kiểm tra context path của app có tồn tại hay không. **Nếu có : Xóa toàn bộ ra khỏi server: undeploy**.

2.Xóa toàn bộ code đã được build.

3.Build lại toàn bộ code mới và đóng vào cấu trúc gói tên gọi là war.(zip)

**6) Deploy:**

- Right click project 🡪 **Deploy**.

**7) Process:**

a)Finding xem context path có trong server chưa ( có -> xóa)

b) Copy war 🡪 Server ( paste)

c)Contextor find context path. ( nằm trong context.xml)

d) Tự động tạo folder cho context path.

**BUG**

- Undeploy: không xóa App khỏi server được.

- Code: bị truy cập, không thể xóa.

- Sai cú pháp, không build được ( code sai).

**8)Deploy complete:**

- Context path. Xml đã finish.

**9) Mở server:**

<http://localhost.8084>

**10) Sau khi complete all:**

- Undeploy project.

- **Stop Tomcat Server**: **Right tomcat** 🡪**Stop**.

-Nếu không undeploy trực tiếp được.

Disk C 🡪 **User** 🡪 **ASUS** 🡪 **AppData**( **folder ẩn->Find by search to**) 🡪 **Roaming** 🡪**Netbeans** 🡪 **8.2** 🡪**Apache-Tomcat** 🡪**Work** 🡪 **Catalia** 🡪**localhost** 🡪**delete folder**.

- Tắt trình duyệt để chạy mới.

**10) Summary after all:**

-Cách vận hành và điều khiển context path.

- Cấu trúc của một thư mục web.

🡪 Nếu đặt view trong source package, META – INF, WEB – INF 🡪 Bug 404, 505 ( lỗi chính tả ).

🡪 Giao diện: Page Resource ( Sai tên, Sai vị trí ).

**HTML**

- Web tĩnh

- Mark up tags.

- Text language.

- Web form:

🡪Mở tag phải đóng tag.

🡪 Có tag mở phải có tag đóng.

🡪Nếu thiếu: ( browser trắng chỉ chạy 1 nữa ).

**\* Cách khắc phục:**

- Right click web.

🡪View page 🡪Mở code để đóng = tag còn thiếu.

- Chạy không cần Server.

-Doctype = import.

- 2 phần chính: + **Header: hiển thị trên url**.

+ **Body: hiện thị ra browser**.

-Chạy đường dẫn , không chạy server.

- Lấy dữ liệu của user -> server.

- Action: xác định address URL( controller làm) của resource phía server đón nhận form để xử lí.

- Method: cách thức truyền data từ client 🡪Server.

-Server được support 2 phương thức : GET Và POST.

Username  
Password

Username  
Password

login

Enter

User -----------> -----------> Value of 🡪 Server( giá trị truyền về server: parameter)

**+ Parameter:**

- Cặp giá trị truyền từ client về server thông qua request.

- name, value.

- Format: name = value.

- nhiều hơn một parameter truyền từ client ->server: phân tách bằng context path và address truyền.

-Nếu truy cập tên bị sai / không tồn tại: Không truyền 🡪 Object không tồn tại 🡪 giá trị = null.

|  |  |
| --- | --- |
| **GET** | **POST** |
| -Thấy được trên đường truyền ở url  -Truy cập Resource ở phía Server dưới dạng read only.  -Mất tách Data.  -Thao tác  -Mặc định. | - denote đặt trong checkbox  - Bảo mật vì giấu được thông tin trên url  -Có thể thêm xóa ở phía server.  -Số lượng parameter > data. |

**Nút Lệnh**

-Submit: toàn bộ controller 🡪 parameter 🡪đẩy về phía server.

-Reset: Xóa tất cả giá trị.

-Buttom: Tìm lại Javascript.

-Select: + Value: trong nút lệnh = level của nút.

+ Combo box: Sổ ra để lựa chọn option.

-Text area: Có thể nhập kí tự Enter.

**Parameter ( tham số)**

- Text box < input type = “text ” name = “txtText” value = “” >

-Parameter name: txtText.

-Parameter value: + rỗng khi không có giá trị nhập vào text box.

+ Gía trị nhập vào Textbox.

-Parameter value của hidden luôn là rỗng vì hidden được ẩn đi.

- Checked # check = unchecked 🡪 Null.

**🡪Phải được checked gửi về parameter**.

-Name của status phải giống nhau 🡪**Không bao giờ bằng null vì luôn có 1 giá trị được chọn một trong các option phải có checked**.

-Nếu value không tồn tại lấy thân thẻ làm value.

**EX:** TextArea < textarea name= “txtArea” rows = “4” cols = “20” >

This is a form parameter demo!!!! 🡪**Parameter value**.

**\* Đặc tính:**

- Cho phép đặt trùng tên ở phía Client.

**Ex:**

- textList = Servlet.

- textList = EJB.

- Không có value, không có thân 🡪 **Lấy tên trình duyệt ( type).**

-Parameter của nút lệnh = level nút lệnh.

-Reset không là parameter 🡪 **không truyền từ client về phía server**.

- Button: tìm code của javascript 🡪 **không là parameter**.

**Request – Request Message – Request Object**

-**Parameter**: cặp name – value được truyền từ phía client về phía server thông qua request.

-**Khái niệm truyền**:

1) Khi tất cả sẵn sàng ( form,….)

2)Liên quan đến HTTP Protocol.

**\*HTTP Protocol**:

- Là 1 Web Server.

-Mạng là sự kết nối các máy tính với nhau.

-Kết nối để truyền thông tin

-Hệ thống mạng: kết nối và truyền thông tin.

-số định danh: duy nhất 🡪 **giúp xác định máy tính là duy nhất**.

- Một máy tính chạy rất nhiều phần mềm 🡪 Mỗi phần mềm trên máy tính nằm trong server 🡪 Xác định 1 phần mềm chạy trên 1 máy tính của server thông qua port.

-Trên máy tính mỗi port chỉ dành cho 1 phần mềm.

-IP: port 🡪 Xác định duy nhất 1 ứng dụng được chạy.

-**DNS**: **Domain name server** 🡪**Phân giải tên miền**.

**Chương trình cơ bản**

- Các view để vào folder web page ngoài META/WEB – INF 🡪**User có thể truy cập các resource thông qua browser.**

- Netbean có thể lưu trữ vị trí của lần trước đó 🡪 **Sai vị trí: lỗi 404**.

**\* Tạo HTML:**

-Right click webpage 🡪 **New** 🡪**other** 🡪 **web** 🡪**html**🡪 **name** 🡪 **finish**.

**\* Bộ Control:**

-Window 🡪 **IDE Tools** 🡪**Papette( hiện bên tay phải)**🡪 **Form**🡪 **Kéo thả vào body tag**.

- **Gõ form + Ctrl space** 🡪**Form HTML** 🡪 **Enter**.

🡪 Action: xác định địa chỉ URL của resource ở phía server để xử lí.

-**Nếu là trang login** 🡪 chọn **Post.( vì Username + password phải được ẩn đi**)

-**Gõ tất cả trong form** 🡪 **Chỉ có form hỗ trợ đẩy về server**.

**\* Để login là page chạy đầu tiên:**

-Copy action trong form( loginServlet)

-**Right click source page** 🡪 **New** 🡪 **Other**🡪 **Web** 🡪**Servlet**.

+ Name: Paste Action (loginServlet)

+Package: Tên Controller.

-**Next**: 🡪 **Add information to deployment descriptor ( web.xml)** : **hỗ trợ phát sinh tập tin web trong html**.

-Trong WEB- INF: 🡪 chọn web.xml.

Sau < /Session – config >

<Welcome – file – list >

<Welcome – file > login.html </Welcome – file>

</ welcome – file – list >

**Sơ đồ:**

1)**Nhập URL 🡪 Enter** : Phân giải tên miền( browser/HĐH)

- IP: Port 🡪**Gửi request về phía server** 🡪**xin kết nối** 🡪**khi có tín hiệu từ server gửi về “seen”🡪 kết nối được thiết lập.**

2)Connect được server thiết lập với client thì gửi request về phía server.

🡪 **Lấy toàn bộ ở phía client biến thành text message và gửi đi**.

🡪 **Server nhận để xử lí** 🡪 **text package** 🡪**browser**.

**\* Đặc tính của HTTP protocol (Hypertext Transfer Protocol):**

-Bao gồm cặp request – respone: + Request: gửi từ client 🡪 phía server.

+ Repone: gửi từ server 🡪 phía client.

- HTTP Protocol = State Less Protocol: + Respone trả về client và client trình bày xong.

+ Server tự động quên

+Server sau khi thực hiện request 🡪 Server quên client.

+Tất cả các bước chạy lại từ đầu.

+Client luôn nhớ Server nhưng server không nhớ client.

+Server không lưu trữ bất kì thông tin nào từ client sau khi hoàn tất request ( sau khi respone).

-**Request message**: là một trong những thứ mà khi thao tác các bước 1, 2, 3:

1) Nhập URL trên thanh address bar 🡪 Enter.

2) Sau khi click complete: HĐH/Browser của máy user 🡪 thực hiện tạo ra request message.

**Cấu trúc Request Message**

-Request line.

-Header information.

- **Message body ( có thể không có )** : 🡪 **Chứa parameter** 🡪 **không có khi HTTP protocol là GET**.

**1) Request line:**

- HTTP Method là Value Method trong form.

A pointer to the resource request in the form of URL. 🡪 **Chỉ đến địa chỉ URL mà resource ở phía server để xử lí 🡪 Action.**

dispatch

Send request

- request Message ------------------------>Server ------------------> Container.

Xác định bởi IP: port ( chính).

- **Container:** ( Software)

🡪khác với các software bình thường: Có khả năng xử lí request và đưa lại respone.

🡪là 1 compiler.

**-Nếu form login sử dụng GET 🡪 Sẽ gắn toàn bộ parameter trên address bar phân tách bởi( ? , & )**

**-Nếu là POST 🡪 Tự động đem parameter 🡪message body**.

-Message body phân cách với header information = Enter.

- Ở phía client khi user tương tác với post sau đó click nút lệnh.

🡪Thành phần browser tạo ra request message gắn kèm trong HTTP Protocol request đẩy về phía server.

🡪Message: dạng text 🡪 đẩy về server🡪 convert thành object để sử dụng.



-Tạo ra Request message 🡪 web server ----------------🡪 Container.

IP: port 🡪 Là xác định duy nhất.

Dispatch: Đưa chính xác.

- Khi request message vừa chạm container 🡪 tạo request object.

🡪 **Convert tất cả thành phần liên quan đến name, value trong request message 🡪Properties của request objet.**

- Forward toàn bộ 🡪Controller 🡪truyền địa chỉ.

- Đứng tại controller 🡪Request object đã tồn tại.

-Lấy giá trị của một thuộc tính:

🡪 GET

**🡪 Lấy giá trị parameter : getparameter()**

**🡪 Lấy giá trị của String : getString()**

🡪Value Parameter là dữ liệu kiểu String.

🡪Request = new Request(); // tất cả giá trị ban đầu request.getmethod()

🡪 Có getparameter (\*)

🡪Không có setparameter (\*\*)

(\*)(\*\*) 🡪 Parameter lấy từ client.

**-Lấy parameter**: getparameter() của đối tượng request object 🡪 Chỉ lấy phần tử đầu tiên trong mảng.

2)Header Information: cặp name: value chứa toàn bộ thông tin máy của user.

3)Message body: Chứa parameter.

**Summary**

**1) Nhập URL trên thanh Address bar của browser 🡪 Enter**.

Click button / link.

**2)HĐH/Browser sẽ liên kết( kết nối) với Server. Sao khi connect thành công.**

**3)Tạo Request Message:**

a)Request line:

- HTTP Method

- Address URL trỏ tới resource ở phía server để xử lí.

-HTTP protocol format.

b)Header Information: Cặp name value chứa toàn bộ thông tin máy của user.

c) Message body: Chứa parameter.

**4)HĐH/ Browser gửi request message về phía server.**

**5)Server dispatch tới container**.

-Software ở phía server được xác định thông qua IP: port.

-Có khả năng xử lí request và đưa lại respone.

-Compiler.

**6)Sau khi request message vừa chạm container 🡪 create request object.**

**7)Container convert tất cả name, value vào properties của request object.**

**8)Forward thẳng đến controller**.

**Ôn Tập HTTP Protocol**

-**Khi Server đồng ý kết nối với client**: 🡪**Server truyền tín hiệu seen về client**.

- **Client thực hiện cơ chế phân giải tên miền IP**: **port**🡪 **Server nhận được tín hiệu yêu cầu kết nối** 🡪 **thực hiện một số tính toán/kiểm tra** 🡪 **khi hoàn thành** 🡪 **gửi tín hiệu seen** 🡪 **clientconnect**.

- **Khoảng thời gian chờ client connect vs server** 🡪 **cấu hình theo khoảng thời gian ( tùy thuộc vào hệ thống/server quyết định).**

-**Khi gửi tín hiệu** 🡪**Mang chậm** 🡪 **Browser download**.

-**Yêu cầu connect ít nhất tín hiệu phải được gửi lại 2 lần** 🡪**Acnolic signal** 🡪**không có tín hiệu trả về**.

**\*Container:**

- Là 1 phần mềm.

- Được xác định bởi port.

**\* Bản chất HTTP protocol:**

-Gửi resquest và trả về respone.

-Phân biệt send/call: 🡪 **Lấy động từ thể hiện by reactional**.

**\* Active and Passive:**

- **Request message**: **phía client** 🡪 **passive** 🡪 **text**.

- **Request object** : **phía server** 🡪 **active** 🡪 **Nạp vào để xử lí**.

-Passive: phía client.

**\* Sơ đồ:**

- Resource( dữ liệu đã được xử lí): Mô hình tùy thuộc vào project sẽ có mô hình khác nhau. EX: MVC,MVC2,…..

-Sau khi hoàn tất:

**a) Connect resource + DB 🡪Trả về resource.**

**b) Connect resource 🡪 Call model( hướng về resource) 🡪đưa cho view 🡪 view thực hiện cơ chế render 🡪 trả về controller.**

-Tất cả bằng gì muốn truyền từ server 🡪Client đặt trong đối tượng respone của HTTP Protocol.

**Respone – Respone Message – Respone Object**

-Giống request – request message – request object.

-Respone object được tạo ra bởi container.

-Một khi request object xuất hiện 🡪 Respone object phải được tạo.

EX: khi được yêu cầu làm một phương thức nào đó 🡪 khi phương thức được thực hiện xong 🡪 kết quả phải được đặt trong một vùng nhớ và trả về địa chỉ bộ nhớ.

🡪Khi tạo ra phương thức 🡪 Đưa sẵn vị trí để lưu kết quả .

- **Request object thực sự sống sau bước 7 ( convert request message thành properties của request object).**

🡪 **Bước 8: tạo respone object** 🡪 **Container tạo** 🡪 **(9) Forward** 🡪 **(10) Call Model**🡪 **(11)Connect DB** 🡪...

-Khi forward đến controller : không chỉ request object forward mà cả request object và respone object cũng forward.

-Login Servlet đã tạo trước đó : bên trong có 2 phương thức : do GET và do POST.

- Hai phương thức được kích hoạt trong request message: request line: Nơi đề cập đến phương thức ( Http method) .

- Nếu phương thức sai 🡪 Code sai.

- Nếu trong request line gọi **GET** 🡪 Container gọi do **GET**.

- Nếu trong request line gọi **POST** 🡪 Container gọi do **POST**.

|  |  |
| --- | --- |
| **Invoke** | **Call** |
| Container tự viết code 🡪 tự gọi | Biết cốt lỗi 🡪 chủ động |

- GG Search : không có cơ chế gọi 🡪 xác định đường truyền đi là GET/POST.

- Thành phần GG Search đưa đến thành phần GG Search sẽ gọi do GET/ do POST ( container tự gọi)

- Gọi dựa trên HTTP method trong request line của request message.

- Phương thức doGET/doPOST tự truyền vào request object/respone object.

- Kiểu dữ liệu :

a) Request object: HTTP Servlet Request . ( Servlet Request )

b) Respone object : HTTP Servlet Respone ( Servlet Respone)

**\* Cơ chế Web:**

- Ứng dụng không gọi tường minh, gọi thông qua message.

**\* Sơ đồ:**

- Controller sau khi xử lí xong 🡪 Set value vào trong respone.

- Client giao tiếp với Server 🡪 giao tiếp = message.

-Respone message 🡨 Convert respone object.

🡪 **Sau khi set value vào respone object** 🡪 **Container convert** 🡪**respone message**.

**Request Message And Respone Message**

- Request message : có 3 phần 🡪 Respone message có 3 phần

- Respone message:

1.Status line

2.Header Information

3.Message body.

**1) Status line:**

-Xác định version format.

-Chứa HHP version.

-Trả về mã code xử lí ( respone code)

-Trả về status: + Message rút gọn thể hiện kết quả xử lí.

+ Thể hiện mã xử lí.

-State line: thể hiện kết quả đạt/ không đạt.

EX: 200 OK ( đa số) 🡪 **Nếu khác: không tồn tại HOẶC không đọc được.**

**2) Header Information:**

- Request message: toàn bộ infor của máy user.

- Respone message:

+ **Toàn bộ infor server ( server)**

EX: Lập trình Front-End: Biết được đang dùng server loại gì ?

+ **Last modified date: xác định message rời khỏi server khi nào** 🡪**Kiểm tra xem có bị hack hay không**.

+ **Content-length: xác định kích thước message là bao nhiêu byte** 🡪**Cơ chế đọc file: luôn có kí tự kết thúc file.**

**+ Content- Type**: Message đang trả về và nội dung trong message body đang chứa theo định dạng nào ( text/html) 🡪 **Nếu không hỗ trợ all định dạng 🡪 Type = Download**.

- Header Information của server trả về cho client nhiều hơn rất nhiều so vs client truyền cho server.

**\* Lỗi :** Học các lỗi in đỏ.

404: Not Found

408: Code chạy vô hạn ở server.

500: Code quá nhiều lỗi.

**\*Sơ đồ:**

- **Khi thành phần của respone object** 🡪 **gán cho respone message(convert)**🡪**đưa về cho container** 🡪**trả về server** 🡪**thực hiện cơ chế send respone** 🡪**client** 🡪**stateless**🡪 **disconnect** 🡪 Hủy tất cả thông tin client ở phía server( request object/respone object).

-Request object sinh ra bởi container khi request message vừa chạm container.

- Respone object do container sinh ra khi request object được khởi tạo thành công 🡪 convert thành properties.

- Cả 2 đều chết khi respone message gửi về phía client.

- Đối tượng xuất hiện ở server: 🡪 Không có HTTP 🡪 không có request/Respone object.

- Toàn bộ: Request/Respone object/message 🡪 Do container quản lí.

**\*GET/POST:**

-Địa chỉ biến thể thành 7 loại:

**a) GET:**

+ HEAD: return header information.

+ OPTION: return HTTP method.

+ TRACE: chính xác là get gắn thêm cơ chế trace.

+ Đặc tính thứ 3: Read only 🡪 Tất cả biến thể của Get 🡪 Read Only.

**b) POST:**

+ PUT : lưu trữ / sữa chữa

+ DELETE: Xóa

+ 2 phương thức chính : GET/POST.

**CGI**

-Web tĩnh 🡪 Web động.

- Sử dụng C/C++ để viết

🡪 Rất khó để viết giao diện đồ họa.

🡪Không hỗ trợ web page 🡪Mỗi yêu cầu 🡪Tạo 1 request

🡪 Mỗi Request 🡪 Tạo 1 object 🡪 chết server 🡪404.

-CGI: Xử lí chức năng tại nguồn server.

**Servlet**

**- Bản chất** : Chia 2 nhánh : + Servlet Class

+ Servlet object

**- Servlet class**:

+ Java class.

+ Kế thừa mọi thuộc tính về java class.

+ Bộ trợ Web động.

**- Servlet object**:

+ Khởi tạo tại thời điểm ứng dụng deploye 🡪 container khởi tạo

+ Tồn tại đến khi ứng dụng undeploy /server bị tai nạn.

+ Tồn tại mãi mãi

🡪 Truy cập server nhanh 🡪Search nhanh.

+ Out put 🡪 HTML String.

🡪 **Servlet chỉ khởi tạo code duy nhất một lần ở lần đầu tiên 🡪 do container khởi tạo và nằm mãi mãi trong server**.

🡪 **Nạp container tại thời điểm ứng dụng deploy**

🡪 **Hỗ trợ protecting** 🡪**Tự động Auto Refresh**.

+ **Auto Refresh**: **Mỗi server khác nhau** 🡪 **Search keyword khác nhau** 🡪 **Không share thông tin = search độc lập.**

🡪 Phía bên server chỉ có 1 ( duy nhất) .

**Phân biệt DAO và Servlet**

- **Dựa vào Web deploy system 🡪 để khởi tạo servlet**.

EX: Khởi tạo LoginServlet

🡪 Phải có <servlet> </servlet>

🡪Bắt buộc thành phần của servlet class phải được khai báo trong web. Xml

- **Thẻ đầu tiên:** **<servlet > servlet name </servlet>**

🡪 Khi ứng dụng deploy 🡪 Server/container sau khi tìm được context path 🡪 tạo project sẽ quay lại tag < servlet >

- **HTML đọc servlet name/servlet class 🡪 tạo code**.

**- tag < servlet > khai báo + khởi tạo servlet object**

+ **Servlet name**: Tên biến

+ **Servlet class**: **kiễu data** 🡪 **nằm tại source package trong web project** 🡪 **không thể truy cập** 🡪**servlet có tồn tại nhưng không thấy** 🡪**muốn thấy**: **Phải có Servlet Package**: **Khai báo URL**: **truy cập servlet** 🡪 **truy cập thông qua đường dẫn**

+**Truy cập servlet** 🡪 **<servlet –mapping >:** **cung cấpURL để access servlet** 🡪 **sử dụng URL –Pattern**.

**\* Thứ tự:**

-**Truy cập**: **truy cập < url – pattern>** 🡪 **Gọi URL-Pattern để xác định thuộc về servlet nào** 🡪**Tìm context path tương ứng**.

🡪 **Khai báo** 🡪 **sẽ được container gọi theo chiều ngược lại**.

-Gõ: Http: //8084 / Context – path.

- **Thực hiện đầy đủ các bước tạo request message** 🡪**phân giải tên miền** 🡪 **tìm Context – path** 🡪 **tìm tệp xml** 🡪 **đọc theo chiều ngược** 🡪 **tìm context – path của project tương ứng**.

\* Code: trong Servlet do container gõ vs đăng ký cấu hình đúng.

-**Servlet** 🡪**Tạo Web động**.

-**Browser** 🡪 **kiểu data : html** 🡪**String trong java class**.

**\*Sơ đồ:**

- Mọi render hoàn tất 🡪 Container gán data cho request object.

- EX:

- Protected void service( Http Servlet Resquest, Http Servlet Respone, Respone) throw ServletException , IO Exception { System. Out.println(“Service”);

🡪**Create String.**

-**Lớp mặc định có sẵn** : **Server**.

🡪 **Liên quan đến controller** : **tạo java class** 🡪 **Kế thừa**

( Extends http servlet) (Http Servlet)

-Các thành phần liên quan đến parameter 🡪 có Http.

- Lớp chứa data thực thi thật sự là request/respone object 🡪 Kiểu Http Servlet…….

**OOP – Java**

- Khởi tạo mặc định các biến ban đầu trong properties trong object tại thời điểm khởi tạo object 🡪 Sử dụng default.

\* Phương thức trong vòng đời của servlet:

**1) Init(): thay thế default**: 🡪 default constructor của servlet object do container tạo thực thi 1 lần duy nhất tại request đầu tiên 🡪 Khởi tạo value ban đầu cho servlet.

**2) Service:**

-Luôn thực thi trong mỗi request.

- Format: do http method()

-Kích hoạt dựa trên http method.

- Bắt đầu = “**Do**”

**3)Detroy:**

- Chạy một lần duy nhất.

- Trước khi undeploy.

**Life Cycle Servlet**

**Uninstantiated:**

+ Ứng dụng chứa deploy

+Being to deploy (1)

**Intialization:**

+ init()

+ first request (4)

**Instantiation:**

+ deploying ( 2)

- 🡪 🡪

**Meet request:**

+Service()

+ meet multi request.

+first to n request ( 5)

**Destroy:**

+destroy()

+undeploy/server:crashed/unload servlet (6)

**Unload unavaliable**

**Failure:**  
+deployed (3)

\***Trong file cấu hình**: 🡪 **đứng ngoài context** 🡪**Sử dụng “/”** 🡪 **context path**.

\***Trong các thành phần khác**: 🡪 **Bản chất là context path** 🡪 **không có “/”.**

**Ôn tập**

**-** Modified tính năng nào thì upgrade tính năng đó 🡪 Không deploy toàn bộ.

-Life Cycle của Servlet gồm:

+ Sinh ra bởi container 🡪 khi ứng dụng deploy.

+ Chết khi undeploy/Server crash.

+ Hoạt động thông qua 3 phương thức:

**1) Unit:**

- Chạy duy nhất 1 lần ở request đầu tiên 🡪 thay thế cho **default constructor**

- **default constructor**: Khởi tạo **value ban đầu** cho object.

- **value ban đầu:** Biến ( field/properties của object).

**Connect Database**.

**\* How to connect DB:**

- EX: Use code to read file.

**1) File:**

- Nhẹ hơn DB.

- Thao tác trên file = thao tác trên cả file.

- Đọc file = load cả file lên bộ nhớ.

- Ngôn ngữ lập trình: 🡪 **Trỏ file/stream( java)**

🡪 **Trỏ vào bên trong memory** 🡪 **Load file + clock bên dướ**i.

- Line by line: Đọc từng dòng.

- File được lưu dưới dạng record.

- Một dòng = một cord.

- Lưu có khoảng trắng 🡪 **khái niệm limitor(;) : dựa trên limitor tách.**

- C/Java sử dụng hàm 🡪 **Các token.**

- Khi viết code 🡪 **bị gán chặt vs cấu trúc file**: 🡪 **Thay đổi** 🡪 **toàn bộ**.

**\*Service:**

-Luôn thực thi trong mỗi request.

-Format: do http method()

- Kích hoạt dựa trên http Method.

- Bắt đầu bằng “do”.

**\*Destroy:**

-Chạy một lần duy nhất.

-Trước khi undeploy.

**2) DB:**

- Cấu trúc datafile nặng hơn

- Thao tác trên tbl

-Lưu dưới dạng B – Tree ( Balance)🡪 **hỗ trợ cơ chế Search** .

- Tất cả tbl đọc trong memory 🡪 **Search nhanh**.

- Đọc theo dạng ngôn ngữ SQL 🡪 Truy suất mong muốn.

- Trên 1 dòng chỉ có 1 đối tượng

- Nhiều người thao tác cùng lúc.

- Tốc độ truy suất nhanh.

\* Code viết 1 lần 🡪 sử dụng mãi mãi: JDBC 🡪 Provid API.

- API: cung cấp code để connect DB.

- Phương thức/interface cho all: DBMS

- DataBase Manager System( Microsoft SQL/ My SQL/……) 🡪 Dù có thay đổi cấu trúc 🡪 duy nhất 1 dòng code.

-Với khái niệm của ngôn ngữ Java: API: kiểu dữ liệu của Java. ( String/double)

- Trong DBMS ( nvarchar/varchar/float/text/money,….)

**Convert kiểu dữ liệu**

- CREATE LIST ( do DBMS create) : 🡪 Kiểu dữ liệu ngôn ngữ lập trình <-> Kiểu DB( ngược lại).

-Mapping kiểu dữ liệu vs DB( ngược lại).

EX: String = nvarchar.

Double = float

- Name of list: Drive.

Driver thực hiện mapping dữ liệu giữa ngôn ngữ lập trình vs DB.( ngược lại)

**LOAD DRIVER**

-**DB là dữ liệu** 🡪 **đảm bảo cơ chế Security** 🡪 **DBMS tạo ra Driver**.

- **RDBMS: Relation/Relation Ship**. 🡪 **Hỗ trợ kết nối**.

- Mỗi dòng trong DB = rows = 1 object trong bộ nhớ ( properties).

-Mỗi dòng sẽ có modul/field.

- **DB muốn truy vấn dữ liệu => Mapping**.

🡪 **Không trực tiếp như file**.

-**JDBC = Java DataBase Connectivity**.

+ Support 2 library ( java.sql/ javaac.sql) 🡪 Import vào code

=> **Gọi thành phần API Support connect BD**.

- **Sử dụng API** 🡪 **Phải có Driver**.

- **Kết nối DB** 🡪 **Phải có object connection**.( con)

- **Kết nối DB được thiết lập** 🡪 **Có Driver** 🡪 **Xuất hiện Driver Manager**.

- **Driver chỉ là list** 🡪 **Passive**.

-**Khi quá trình connect diễn ra**:

+ **Dữ liệu được cop từ DB** 🡪 **đẩy lên bộ nhớ** 🡪 **object làm nhiệm vụ truy cập và convert dữ liệu** 🡪 **Sau khi complete** 🡪 **Đưa xuống DB** 🡪 **qua Driver Manager convert** 🡪 **dữ liệu**.

+ Sử dụng Drive To giao tiếp giữa DB với ứng dụng ( ngược lại) khi Convert dữ liệu từ DB phát triển lên ngôn ngữ lập trình ( ngược lại).

**Điều Kiện để có Connection**

**\* Driver Vs DriverManager:** Phương thức được gọi getConnection()

🡪 Sau khi có Connection 🡪 Điều kiện tiên quyết cần có : Câu lệnh sql 🡪 Nạp vào object Statement 🡪 Do Connect tạo ra 🡪Connect Create Statement 🡪 Thực thi câu lệnh ( Execute Query) 🡪 Trả về Result Set 🡪 Trỏ đến 1 mảng dữ liệu 🡪 Trỏ đến vị trí đầu tiên mặc định: **BOF = Begin of Found** 🡪 Mỗi câu lệnh nạp vào 🡪Vị trí cuối cùng chốt dữ liệu **EOF= End of Found**.

- ResultSet: Phương thức dịch chuyển từng vị trí một và lấy dữ liệu.

- **JDBC ⬄ dịch chuyển : Forward only** 🡪 **Không dịch chuyển lùi**.

**\* Result Set:**

- Không phải mảng.

- Bên trong là 1 con trỏ 🡪 **Trỏ đến mảng dữ liệu**.

- Vị trí trỏ ban đầu luôn là BOF.

-Dịch chuyển để lấy dữ liệu.

- **Dữ liệu lấy về không tồn tại khi BOF + EOF nằm cùng vị trí**.

- **Next** : trỏ ra ngoài vùng nhớ ( không có địa chỉ).

- Truy vấn có hay không khi Result Set luôn tồn tại.

- Không có giá trị null.

-**Thực thi không có** 🡪**Result Set vẫn tồn tại** 🡪**BOF/EOF nằm cùng vị trí** 🡪 **Con trỏ trỏ vào BOF**.

- Result Set luôn khác null: Con trỏ dịch chuyển dưới dạng có hay không có dữ liệu.

**Ôn tập**

- Mở connect: Driver + Driver Manage + username + password.

- Diver: Passive.

- DiverManager: Active.

=> Driver Manager load driver.

**\* Phân biệt các loại statement:**

**1)Statement:**

-Thực hiện câu lệnh không tham số.( do không có mệnh đề where)

- Không có mệnh đề Where.

-Là câu lệnh tĩnh.

**2)Prepare statement:**

- Thực hiện câu lệnh có tham số ( do có mệnh đề where)

- Không cần thực hiện lại 2 bước check cú pháp + load lên bộ nhớ khi thực hiện lại việc truyền tham số.

**3)Callable statement:**

- Áp dụng trong trường hợp gọi store Procedure.

**\* Statement:**

1) Check cú pháp.

2)Load cú pháp vào bộ nhớ

3)Thực thi:

**Create.preparestatement…..** 🡪**Truyền SQL vào trong statement**.

**ExecuteQuery()**: **Không truyền**.

**ExecuteQuery() + PreparedStatement**: **thực thi lệnh**.

**\*Preparedstatement/Callablestatement**: chỉ check và load cú pháp ở lần thực thi đầu tiên 🡪 Sau đó chỉ thay đổi giá trị parameter( tham số) truyền vào ( ?) 🡪 Từ trái sang phải + bắt đầu = 1.

**\*Store Procedure API:**

+ Giống viết function vào trong java class.

+ Hạn chế SQL Exception.

+ Gọi parameter 🡪truyền tham số.

**Web Container Model.**

- Các Server mà model support khi ứng dụng deploy.

- Các app chạy trên máy tính/ bộ nhớ bất kì 🡪phải được support.

- Chạy trên app trên HĐH được provide 4 segment:

1)Code segment

2) Data segment:Nơi lưu trữ toàn bộ biến toàn cục ( static).

3)Stack: Nơi hàm được nạp vào khi gọi.

4)Heap: chạy/tạo bộ nhớ động.

+ Bản chất object là một vùng nhớ động.

+ object vừa lưu trên heap/Stack.

+ object : biến/ properties in object được lưu trong Heap.

- Có 4 vùng nhớ được cấp phát cho HĐH : trong đó statement ( segment) là vùng nhớ.

**Cấu trúc gói War**

-Có 2 phần chính:

**1)Phần trong thư mục WEB-INF:**

- Chứa xml.

- Thư mục library trước đó nằm ngoài WEB-INF.

- **Khi đóng gói** 🡪 **tự động** 🡪 **copy những gì trong library và tạo thư mục library** 🡪 **trong WEB-INF** 🡪 **đưa gói jar vào trong thư mục vừa tạo ( sqljdbc4. Jar)**

**2) Phần ngoài WEB – INF:**

- Đặt tất cả các resource mà cho phép user ở máy tính của họ sử dụng browser để truy cập vào 🡪 là WEB Page nằm ngoài WEB/META – INF.

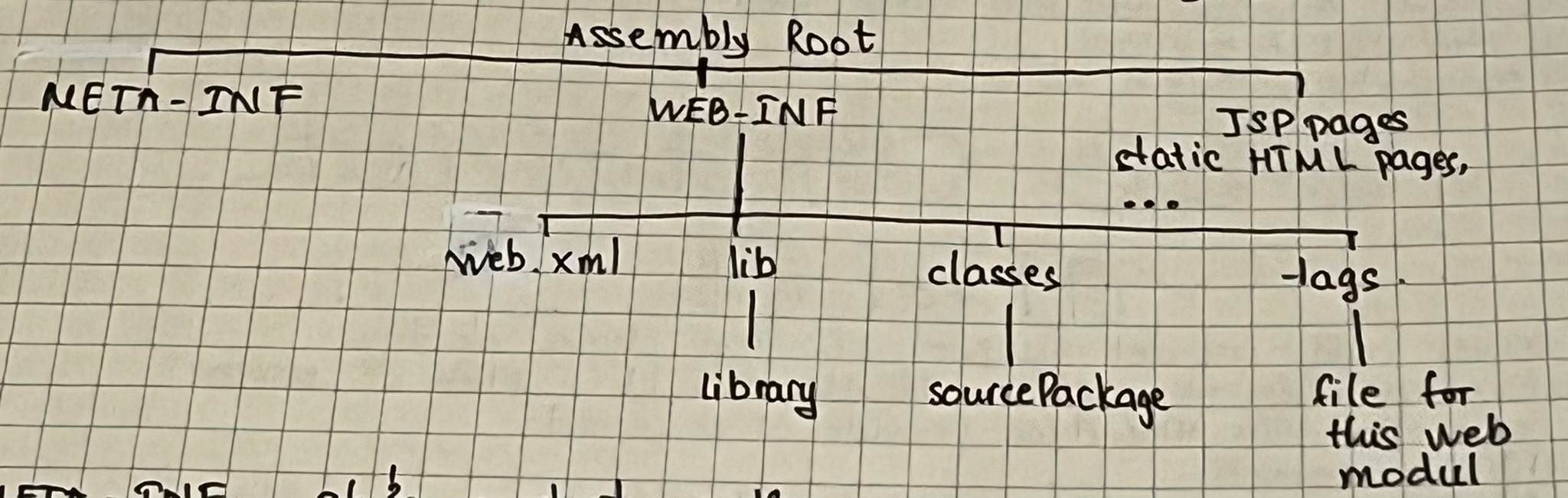
-Khi thực hiện cơ chế clean and build 🡪 tự động đưa toàn bộ những gì trước đó vào war.

- Cấu trúc project của netbean là cấu trúc dùng để thao tác.

**\* Toàn bộ Source Package**(bao gồm cả package) - ( nằm ngoài WEB-INF) : 🡪 Được dịch thành file class 🡪đưa vào file class.

- File class : là file được tạo ra bên trong thư mục WEB- INF khi đóng thành gói war.

**\*Tags sẽ xuất hiện trong trường hợp trên trang jsp có tạo các package mở rộng của thành phần trang web( nằm trong WEB – INF).**

****

- **META – INF: chứa context – path** 🡪 **Nằm trong thư mục contex.xml**

- Assembly root:Thư mục ngoài cùng 🡪 Đóng gói war ( đóng bên trong assembly root)

- Gói war = zip.

🡪 Có thể tạo thư mục sau đó copy thành phần gói war vào 🡪 Nén thành zip.

-Nếu rar 🡪 chuyển thành zip để đóng gói 🡪 zip thuần.

**\*Deploy:** Copy gói war vào trong thư mục web apps.

- Undeploy = delete gói war trong thư mục webapps.

**\* Copy:**

1) Container thực hiện giải nén.

2) Đọc tập tin context. Xml trong META – INF 🡪 Tìm Context path( tên Folder) 🡪Tạo Folder

3) Toàn bộ nội dung bung vào Folder.

**\*Giải nén khi tạo thư mục bởi vì:**

1) Chưa giải nén 🡪 không biết tên thư mục.

2) Chưa giải nén 🡪 không biết có thể giải nén hay không.

=> Không tạo thư mục trước khi giải nén 🡪 Lưu vào vùng nhớ tạm 🡪 chờ khi có thư mục được tạo ra với tên context path thì bung vào Folder.

**\* Tại sao không đóng gói ngoài thư mục:**

+ 2 đường dẫn: Khi đóng gói ngoài 🡪 Giaỉ nén 🡪 Lấy Context path 🡪 gọi contex – path(context path\ tên Prj\Tên đường dẫn) không đến được đường dẫn.

3) Tìm Controller: 🡪Controller được khai báo ở xml

4)Đọc xml 🡪 Load phần liên quan đến controller + servlet 🡪Đọc để gọi class đúng.

5) Load class: 🡪 Khi cần sử dụng 🡪Qua lib load thư viện tương ứng.

**\* Cơ chế của server/container luôn tiết kiệm bộ nhớ:**

- Dựa trên xml 🡪 load thành phần của thư mục class🡪 load thành phần serlet trong controller.

-Sau khi thực hiện xong 1 thành phần servelet 🡪 tiếp tục quay lại xml 🡪 Cho đến khi hết tập tin xml( welcome – file) 🡪 deploy thành công.

**Thực hiện**

**1) Mở Command Prompt:**

-Enter 3 lệnh: + javac -Nếu chạy cả 3 : cấu hình đúng

+ java -Còn lại: Sai

+ keytool

**2) Cấu hình lại:**

\***Environment variable** 🡪 **Environment variable** 🡪 **New.**

Variable name: JAVA –HOME

Variable value : Paste đường dẫn.

\* **Open File Exploer** 🡪 **Disk C** 🡪 **Program Files** 🡪 **Java** 🡪**Chọn jdk** 🡪 **copy đường dẫn**.

\* **Click thành phần trên System variable** 🡪 **P** 🡪 **Dẫn đến Path** 🡪**Edit** 🡪**Edit Text** 🡪**OK** 🡪**Variable Value** 🡪**End** 🡪**Ctrl + V ( paste đường dẫn jdk)** 🡪 **\bin** 🡪**OK Call**.

**3) Mở lại Command Prompt**: Thực thi 3 lệnh.

**4) Cấu hình ,Server:**

- Giải nén tomcat vào Disk ( D)

- **Open Environment variable** 🡪 **Environment variable** 🡪 **System variable** 🡪**New** 🡪**Variable name: CATALINA\_HOME** 🡪 **OK** 🡪 **OK**.

-Variable value: Đường dẫn đến tomcat vừa giải nén.

**5) Mở lại Command Prompt:**

- cd đường dẫn đến thư mục bin trong tomcat vừa giải nén 🡪 Vào thư mục tomcat 🡪 bin 🡪 copy đường dẫn 🡪 right click.

- Startup 🡪 Run + bung cửa sổ mới

- Shutdown 🡪 Tắt server.

**\*Servlet Context:**

- **Tại App deploy vào trong container** 🡪**container cấp phát 1 vùng nhớ** 🡪 **serlet context** 🡪**vùng nhớ thuộc về container do container support**.

- **Tại App deploy thành công** 🡪 **Container cấp phát cho user 1 vùng nhớ** 🡪 **tồn tại từ lúc App được deploy đến khi app undeploy** 🡪 **Servlet Context** 🡪**Memory segment at server(do container tạo ra)** 🡪 **Tại app deploy đến khi undeploy/server crash**.

🡪 **Vùng nhớ được cấp phát cho ứng dụng là vùng nhớ trung gian để các resource trong app giao tiếp vs nhau được cấp phát vùng nhớ ở trong app nằm trong phía server**.

**\* Deploy 2 app vào trong container**🡪 2 gói war vào thành phần bên trong webapp 🡪 tạo ra 2 vùng nhớ riêng biệt ( Servlet contex).

- Giữa 2 thành phần app không giao tiếp vs nhau 🡪 **Độc lập.**

- 2 vùng nhớ servlet context của 2 app riêng biệt các object trao đổi với nhau phải thông qua vùng nhớ trung gian.

- Được truy cập toàn bộ tài nguyên:🡪 tính chất: global: dùng chung cho app.

**\* Scope:**

- Một vùng nhớ ở phía server: + Có thời gian tồn tại nhất định.

+ Servlet Context là một vùng nhớ ở phía server và có thời gian tồn tại nhất định là scope = context Scope ( Application Scope)

-Type Context Scope: Servlet Context.

-Đối tượng ở phía server cũng là Scope: Request ob …….. 🡪 **Không hình thành khi không có res ob**.

-Request scope = **Request ob** 🡪 Type: **HttpServlet Res**

- **HttpServlet Res:**thành phần res message vừa phải có http truyền đi.

- **Request ob :** được cấp phát khi chạm container 🡪 Chết khi res message trả về phía client servlet là java class.

\* **Khởi tạo giá trị ban đầu cho servlet** 🡪 **sử dụng init()**

\* Khởi tạo giá trị ban đầu cho toàn bộ app:

**\* Initialize Parameter:**

- Context Scope là vùng nhớ được khởi tạo cho toàn bộ app 🡪 khởi tạo giá trị ban đầu cho toàn bộ app khi app run.

- Khai báo parameter:

+ Khai báo trong thành phần của tập tin web.xml

+ Cụ thể**: < contex – param >** luôn ở đầu html 🡪 **dùng chung cho toàn bộ thành phần bên dưới.**

+ Gồm: **- Name : < param – Name>………..</>**

**- Value : < param – Value > ………...</>**

**\* Container:** Cung cấp vùng nhớ cho những hoạt động ở phía server.

|  |  |  |
| --- | --- | --- |
| **Request Parameter** | **Context Pnitialize Parameter** | **Config Initialization Parameter** |
| 1.Name,value truyền từ client🡪 Server  2.Format: Name value  3.Read only  4.Type: String  5.Cố tình truy cập name không tồn tại: Null  6.Cho phép trùng tên ở phía client 🡪 tạo mảng ở server lấy toàn bộ mảng. | 1.Name , value tồn tại khi app deploy 🡪 Undeploy khởi tạo khi app deploy thành công.  2.  3.  4.  5.  6.Không có khái niệm trùng tên 🡪Nếu trùng  \* jdk >= 7: Lấy phần tử cuối cùng  \* jdk < 7 : Deploy time error. | 1.Name, value được khởi tạo giá trị sau khi servlet được khởi tạo  2.  3.  4.  5.  6.Không có khái niệm trùng tên 🡪 Nếu trùng  \* jdk >= 7: Lấy định nghĩa đầu  \* jdk < 7 : Lấy định nghĩa cuối. |

**\*For example:**

- Đếm số lần truy cập website.

- Lấy context hiện thành sử dụng:

**Servlet Context sc = getServlet Context();**

**String var = sc.getInitParameter(“parName”);**

- Từ ContextScope gọi hàm getParameter().

-Parameter giá trị khởi tạo = 5 .

**Với init()** 🡪 **Lấy hàm hiện hành.**

**Get giá trị xuống** 🡪 **Đếm số lần truy cập : count = Integer.parseInt()**.

- Đếm số lần truy cập 🡪 Tăng count lên 1 đơn vị 🡪 int ra màn hình 🡪 lần truy cập đầu tiên

Mặc định: 5

Run: 6

\* Tại thời điểm màn hình hiễn thị 6 🡪 tạo Init counter gọi initialize là số 5 + 1 = 6.

🡪 Tạo thêm 1 servlet tên second Counter 🡪 Code lại Init Counter 🡪 Khi chạy ra 6 🡪Vì : Read Only.

- Code : getParameter = gán vào count.

**\* Context Parameter: trùng nhau:**

- param – value = 5

-param – value = 500

🡪getLastitem.

\* Khi mảng trùng 🡪 getParameter 🡪 không dịch chuyển.

\* Khởi tạo giá trị ban đầu cho servlet 🡪 sử dụng init() của servlet.

\* Khởi tạo giá trị ban đầu cho thành phần app sử dụng Context Initializationparameter()

\* Khởi tạo giá trị ban đầu cho object:

1) Default Constructor: 🡪 Gán giá trị mặc định.

2)Thiết lập giá trị từ ngoài vào để thiết lập giá trị mặc định cho đối tượng khởi tạo thông qua constructor và truyền tham số.

\* Servlet ngoài khởi tạo các thành phần giá trị liên quan đến default 🡪 hỗ trợ đưa giá trị từ ngoài vào bên trong thành phần servlet thông qua servlet config 🡪Hỗ trợ đưa Infor khởi tạo giá trị mặc định ban đầu cho thành phần servlet.

1)getParameter

2)getServletName: lấy tên Servlet.

3)getServletValue: lấy giá trị Servlet.

\*Nội dung hỗ trợ khai báo 1 thành phần Initialization Parameter cho một Servlet 🡪 Config Initialization Parameter.

\* Thẻ <Servlet>: bổ sung thẻ khai báo thành phần của initialize parameter của servlet.

**<init- param> <param – name><paramvalue>** 🡪 **local**

🡪 Muốn bao nhiêu thành phần khởi tạo có sẵn 🡪 nhân bản bấy nhiêu.

\*Config Initialization Parameter: tầm vực local 🡪 Chỉ tồn tại và Servlet được khai báo mới có thể truy cập.

\* Lấy Config hiện hành của một servlet 🡪 Sử dụng:

ServletConfig sc = getServletConfig()

🡪Thay vì cấu hình trong < context param> 🡪 Cấu hình trong <servlet>

- <servlet> : Servlet Context

\* Lấy thành phần Config 🡪 Ép kiểu dạng int 🡪 giá trị tăng : 6

-Truy cập giá trị Initialize parameter bên trong Config 🡪 Tạo < servlet> thứ 2 🡪 Copy + Paste code 🡪 Tại màn hình thời điểm chưa kích hoạt servlet 🡪 Gía trị = null 🡪 Biến trả về 0 🡨 Ép kiểu 🡪 Không tồn tại.

\*Config Initializeparameter: thuộc về một servlet 🡪 Sử dụng servlet khác truy cập : Null.

\* Config parameter ở vị trí của servlet object ( phải khởi tạo hoàn tất)

- Scope ở vị trí container.

-Context Initialize Parameter – trong thành phần Context Scope.

**Search**

1) Nhập chuỗi bất kì.

2) Last name trong database có chứa chuỗi 🡪 Hiển thị dưới dạng **data grid**.

Ex:Khi nhập a: toàn bộ Infor mà lastname có chứa a được hiển thị sau.

+ Login: login = truy vấn dữ liệu dưới dạng data base \_Answer : Có Hoặc Không.

+ Search: dữ liệu cần đưa lên bộ nhớ để thực hiện duyệt và trình bày ra giao diện.

**\* Mô hình:**

Call

forward

Dispatch

- Khi user click 🡪 send req 🡪server ----------------->container ------------------>servlet ------> Model(DAO) --------------> Database

Connect Query

- Search trên giao diện:Không bắt buộc nhập giá trị vào Search.

🡪 Không nhập : + Không báo lỗi

+ Không làm gì cả 🡪 Quay về Search Page.

🡪 Dữ liệu trả về :Nhiều dòng ( nhiều hơn 1 dòng dưới table).

-Khi dòng dữ liệu trả về được đưa lên phía thành phần của app 🡪 Mapping vs 1 ResultSet 🡪 Chỉ là con trỏ.

-Một dòng dữ liệu được load lên DAO bộ nhớ biến thành DTO list.

-DTO: xuất hiện khi load toàn bộ dữ liệu lên DAO.

**DTO**

- Data transfer object.

-Java class.

- Đại diện cho dòng dữ liệu trong table. ( Map 1:1 dòng của table).

-Mapping trực tiếp bên trên table tương ứng là 1 field bên trong thành phần của 1 dòng tương ứng với 1 properties ở bên trong thành phần java class.

-Map /declared ( khai báo) mapping field of row to properties of object 🡪 Không nhất thiết phải map hết.

-Tất cả các thuộc tính properties được khai báo 🡪 Không khai báo public

+ Định nghĩa toàn bộ getter/setter method cho các properties.

+ Khai báo private/protected.

+ Truy cập thông qua behavior( mã set) 🡪Encap…..

- Implement Serializable ( ở phía server).

-Qui tắc đặt tên:

+ Class: tên bảng chữ cái đầu viết in + DTO

+ Package: tên bảng chữ cái đầu viết thường.

- Difine default constructor.

**\* DTO list = ? :**

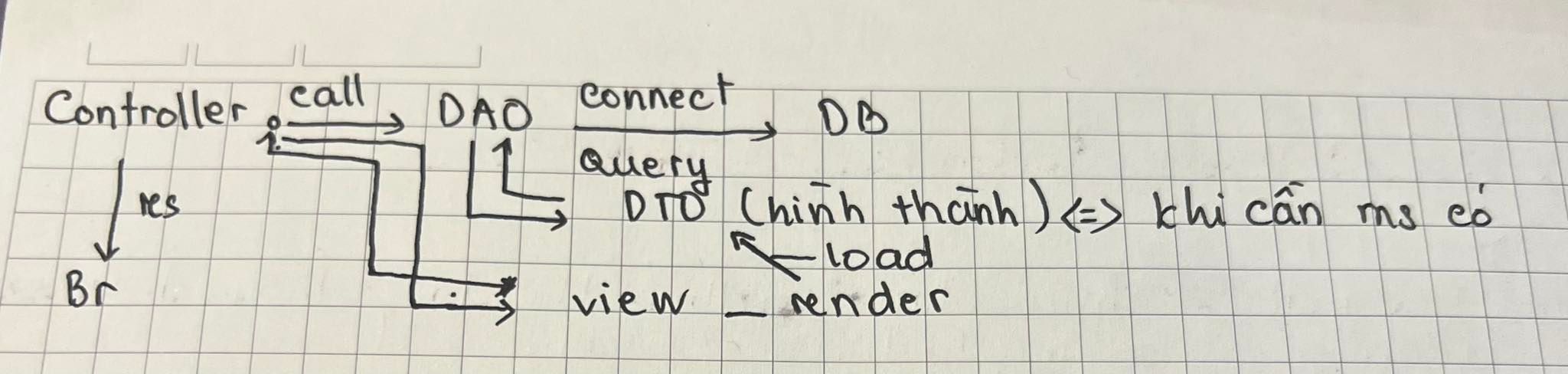
- Một thành phần của 1 dòng :

-Gỉa sử keyword tồn tại dưới thành phần database:

1 row = 1 DTO

Nhiều Row = DTO list.

**DTO** 🡪 **Servlet/Controller** 🡪 **duyệt dữ liệu + trình bày** 🡪 **Đưa Result** 🡪**Respone object**.

- 

**Store**

- **Http protocol = stateless Protocol khi resMess trả về Br** 🡪**Server: disconnect**⬄ **Hủy những đối tượng lưu trữ ở phía server ( req/res ob).**

EX: 1) Login vào một trang web 🡪Can Remember.

2) Shopping online 🡪 Giỏ hàng 🡪 Add Product 🡪 Product trước vẫn còn nhưng theo khái niệm 🡪 Mỗi lần chuyển, res trả về 🡪Dữ liệu mất.

=>**Khi ResMess trả về phía Br 🡪 Ở Server vẫn tồn tại một vùng nhớ nào đó 🡪Có thời gian tồn tại đến khi req mới trở lại 🡪 Không mất**.

=> **Scope 🡪 Dù ResMess trả về client 🡪ContextScope vẫn còn tồn tại 🡪 Services được container support. ( Http bình thường 🡪 Hủy All).**

- Lưu trữ thông tin bằng cách tận dụng vùng nhớ Scope để chứa dữ liệu 🡪 Name: Attribute

**\*Attribute:**

- Name, value đặt trong Scope.

-Nằm trên thành phần bộ nhớ, sau Scope 🡪 tận dụng Scope chứa dữ liệu 🡪 Create Attribute.

-Format: Name, Value.

-Type: Object. ( Muốn lưu trữ thông tin người dùng dù ResMess trả về 🡪 Biến thành thuộc tính chọn Scope cụ thể đặt vào ) khi ảnh hưởng thời gian tồn tại.

-Không tồn tại bằng Null.

- Thuộc tính nằm trong Scope ở server 🡪Thuộc tính Name trong 1 Scope: Unique khi Unique name không có khái niệm trùng.

- Thời gian tồn tại <= thời gian tồn tại ở Scope.

- Có thể remove thuộc tính kể cả khi Scope đang tồn tại gán null khác Hủy.

-Scope được Container support : Lưu trữ thông tin người dùng ở phía Container để đảm bảo rằng khi res trả về vẫn còn thông tin ở server.

**\* Session Scope:**

- **Tương ứng với một người dùng** 🡪 **Chỉ mở khi user muốn** 🡪 **server create session scope tương ứng user**.

EX: Login ( Remember user name + password)

**- Tồn tại đến khi có yêu cầu đóng Session**.

🡪 Người dùng cố tình đóng session. ( Ex: Logout)

🡪Đơn vị: 30p ( minutes)- trong xml.

🡪Session time out = lần cuối cùng tương tác vùng nhớ.

**- Không tương tác/ không mở lại/ tắt trình duyệt** 🡪 **Session out**.

🡪 Tương tác vùng nhớ phía server => Session/scope, có khái niệm last access time 🡪 Hỗ trợ API lấy.

**\*Thứ tự Scope:**

- Từ < -> > : Request scope < Session Scope < Context Scope.

- Chọn phù hợp:

**\* Context Scope:**

- Tại 1 thời điểm có nhiều người dùng tương tác với Context Scope ( Vùng nhớ dùng chung cho toàn App.

**App Web 🡪 tại 1 thời điểm có nhiều người dùng truy cập = Distributed Application.**

=> **Race Condition ( tranh chấp dữ liệu) => Không nên sử dụng Context Scope**.

- Để không xảy ra Race condition 🡪 Vùng nhớ có nhiều dùng chung nên dùng ở dạng Read Only 🡪 Hạn chế sử dụng 🡪 Khi sử dụng nên sử dụng các giá trị không bao giờ đổi vs người dùng 🡪 Ai truy cập cũng giống nhau 🡺 Context Initialize Parameter.

**\* Session Scope:**

- Tại một thời điểm có 1 người dùng truy cập 🡪 Session tạo ra do ý muốn của người dùng.

- Tại thành phần Session có nhiều request tương tác + có thời gian tồn tại dài.

-Được dùng khi cần đáp ứng nhiều hơn một request của người dùng.

**\* Request Scope:**

- Request Scope = Req object tạo ra bởi 1 Req.

- Tại một thời điểm có 1 req truy cập.

- Trình bày dữ liệu + Không lưu trữ.

**\* Ví dụ về chọn Scope:**

**1) List DTO trong search 🡪 Chọn Req Scope 🡪 Search rồi + không có nhu cầu search lại =>Không cần lưu trữ.**

**2)Trong app với mọi thao tác chức năng điền thông tin người dùng có nhiều hơn một lỗi xảy ra**

+ Khi có lỗi => Thông báo cho user + thông báo ngay.

+ Có lỗi => Sửa ngay 🡪 Tạo Erorr object lưu trên Context Scope ( không lưu lại lỗi).

3) Thông tin người dùng khi login lưu trong session Scope khi và chỉ khi trong khoảng thời gian cho phép những tương tác trên hệ thống 🡪dữ liệu không xóa để có thể tiếp tục tương tác.

**\* Cookie –client KHÁC Scope – server( Duy trì phiên làm việc phía server)**.

**\* Function:**

-Truy cập thuộc tính trong scope có 4 phương thức:

\* Tên Scope.function() 🡪 to use.

**1) setAttribute ( String name, object obj)**

- Khi container gặp setAttribute chỉ định trên 1 Scope.

**1) Vào Scope kiểm tra “Name” có tồn tại không** + Có : Lấy giá trị obj cập nhật vào Attribute đang có.

+ Không : buil mới attribute và gán giá trị obj vào.

**2)Hai chức năng : Update và Create**.

**2) getAttribute ( String Name)**

- Lấy thuộc tính trong Scope. ( theo tên )

- Container kiểm tra xem có tồn tại không + Có : trả ra obj

+ Không : Null.

- Obj trả ra là object thuần => Sử dụng cơ chế ép kiểu tương ứng + tường minh về kiểu cần dùng.

**3)Remove Attribute ( String Name):**

-Hủy Attribute mong muốn bên trong Scope khi Scope vẫn còn tồn tại.

- Container kiểm tra xem thuộc tính có tồn tại không + Có: Hủy

+ Không : Không làm gì cả ( không báo lỗi).

**4)getAttributeNames()**

-Lấy tất cả tên thuộc tính về theo dạng Enumeration khi 1 Scope có nhiều người dùng/Req tương tác =>Số lượng thuộc tính không nắm được => Sử dụng 3 function trên để tiếp tục thao tác.

**\*Cookie KHÁC Session:**

- Cookie: file lưu ở chính máy người dùng và chứa thông tin để khi có Req 🡪 đưa thông tin từ máy người dùng 🡪Server để thực hiện action. ( giá trị trong file cần mã hóa).

-Session: Vùng nhớ ở server chứa thông tin người dùng trong khoảng thời gian nhất định.

- 3 Scope:

a.Request:HttpServletRequest ( req object) – Tồn tại ngắn nhất.

b.Session: Http Session

c.Context: Servlet Context.

**\* Duy trì Request Scope:**

-Login: Respone.sendRedirect

🡪Khi thực hiện chuyễn từ Controller sang thành phần Res object để sử dụng cơ chế đưa vào trong Res object để render và đưa nghe lại client qua phương thức respone.SendRedirect()

-Bản chất Res trong servlet:  
 + Form: Truyền giá trị từ “txtName” vào một button vào MiddleServlet.

+ MiddleServlet: không xử lí 🡪 sendreq đến EndServlet + đưa thêm thông tin phụ trợ thông qua khái niệm thuộc tính.

+ EndServlet: Tổng kết thông tin/kiểm tra giá trị = null.

**\* Problem:**

- **respone = respone object** 🡪 **Containe Convert** 🡪 **resMess** 🡪**Send Client.**

🡪 **Đã respone 🡪 toàn bộ thông tin bị xóa ( req/res object).**

- **SendRedirect** 🡪 **Req mới được tạo ra khi và chỉ khi tạo ra req object new** 🡪**Send EndServlet**.

- Tạo ra req object new 🡪 Parameter attribute ( Req cũ) không còn tồn tại( null).

\* Khi SendRedirect 🡪 đã Res 🡪 hình thành một thành phần Request object mới và Send qua thành phần resource 2 🡪 Thực hiện và trả về khi res khác req/res object cũ => Không duy trì res ob.

**\*Nguyên nhân null**: Thành phần Req object bị hủy 🡪 Khi res Mess trả về.

=>Để req object không bị hủy khi res Mess không trả về.

-Req ob tồn tại ở server một khoãng thời gian nhất định 🡪 không để res trả về.

- Request Dispatcher : Cơ chế duy trì req ob thêm một khoảng thời gian bằng cách ngăn Res trả về.

-Thành phần đặc tính: flexibility + Resource xử lí: không biết.

+ Chỉ biết Resource đầu tiên.

**- Container đảm bảo thông tin người dùng đưa đến controller được giữ nguyên: Forward 🡪 Paste by reference khi đưa toàn bộ thành phần object vào thành phần controller đang cần** .

- Forward ( After): Container đưa cho controller.

- Forward ( Before): req 🡪 req khác.

- Thay vì dùng **Respone.SendRedirect** thì ta dùng **ResquestDispatcher** 🡪 chỉ định đối tượng cần đưa tới + forward toàn bộ Req/Res object giống container forward res/req đến controller.

=> Res không được trả về 🡪 Trả về: EndServlet => Result: Có giá trị khi req object vẫn còn đưa từ Resource này đến Resource khác 🡨 Đưa thêm thông tin 🡨 Hỗ trợ thành phần thuộc tính

- Forward: chuyển toàn bộ khi ngăn chặn res trả về khi resource cuối cùng = resource bên trên thành phần cuối cùng người dùng chỉ thấy Middle Servlet.

**\* Example: order fast food.**

1) Customer order fast food.

2) Staff order fast food.

3) Chief returns food to staff.

4)Staff returns to Customer.

- Customer ⬄client.

- Staff⬄ Controller.

-Chief ⬄ controller 🡪Send cho các model ( chief con)

-Order = send request.

**\* Request Dispatcher:**

-Luôn có thành phần điều phối🡪 đưa vào trong để xử lí 🡪 feasible.

\* Flexibility: Break down.

- Mô hình MVC2: + phải có 1 servlet làm điều phối.

+ Một servlet chĩ làm một chức năng và phải thông qua Dispatch \_ Dispatch Controller.

\* Security: trên đường truyền luôn là thành phần điều phối.

**\* Mã giải:**

If ( button is search) {

Code search

}

-Call login 🡪 login hàm 🡪 không Search.

-Call Search 🡪 Search hàm 🡪 không login.

- Controller không làm 🡪 điều phối để 1 thời điểm có nhiều req được nhận.

**\* Đặc tính hai của Request Dispatcher:**

-Load balancing ( chia tải office server) ⬄ nhận và truyền.

- Có khả năng đón nhận nhiều req cùng lúc chuyển cho từng thành phần riêng biệt để xử lí ⬄ không ảnh hưởng đến hệ thống.

- Chia nhỏ thành phần chức năng để làm việc ⬄ một servlet làm một chức năng kể cả điều phối.

**\* Request Dispatcher Summary**: hỗ trợ giao tiếp giữa các resource ở trong thành phần server

- Để giao tiếp: duy trì request ob 🡪 để resource cuối cùng rs:

- flexibility

- Security 3 Đặc tính của Request Dispatcher.

- Load Balancing

**\*Include trong Request Dispatcher:**

- Nhúng toàn bộ tài nguyên ( 2 đến n) vào resource đầu tiên

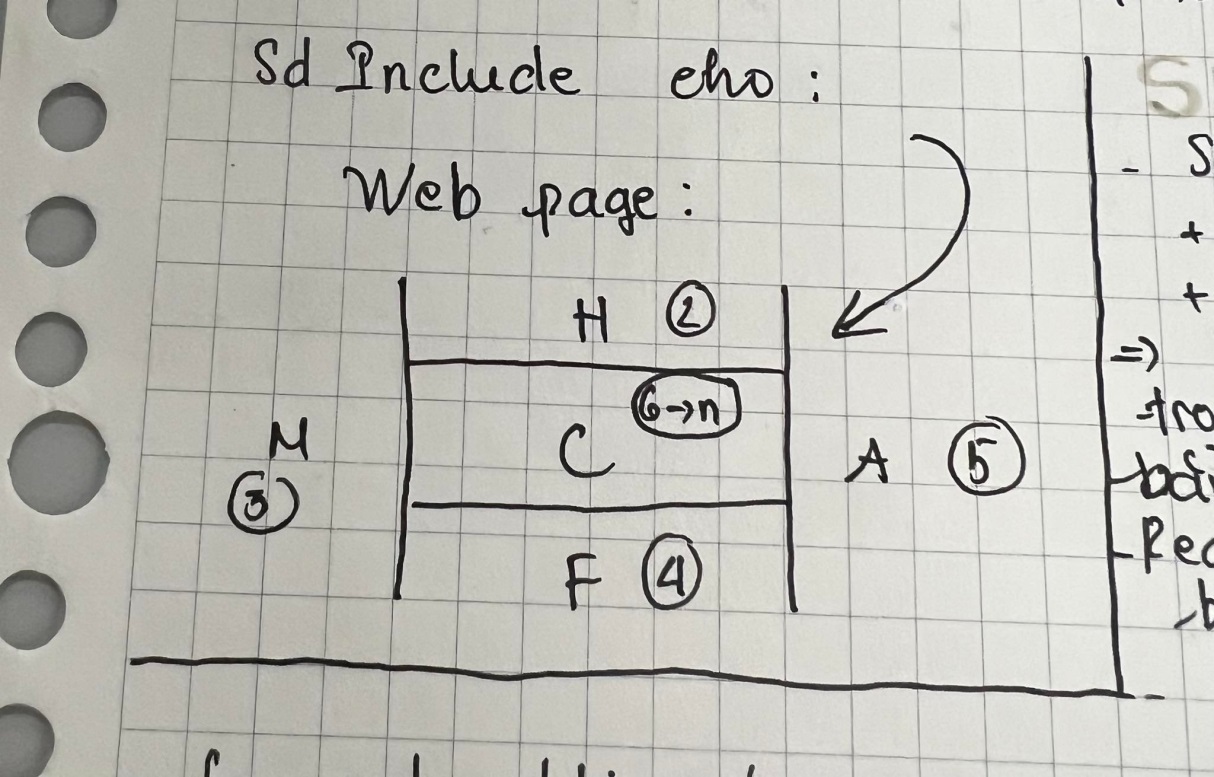
- Nhúng : Sao chép toàn bộ nội dung trong EndServlet dán vào vị trí include.

- Ex: File A include B( c)

🡪 Compile file A thành file duy nhất.

-Đem từ ( 2 -> n ) sao chép vào tài nguyên 1 🡪 tính năng container được kích hoạt 🡪 Compile lại và biến thành một thành phần tài nguyên duy nhất để đáp ứng Res/Req.

- Sử dụng Include cho:



**Summary**

- **Service do container support** : Scope

+ Session: Có thời gian dài

+ Context: mãi mãi.

=> Để giao tiếp giữa các đối tượng với nhau trong server thì kéo dài thời gian của request Scope bằng cách sử dụng Request Dispatcher.

-**Request Dispatcher**: khái niệm về mô hình break down được hình thành.

- **include**: sao chép nội dung servlet được include vào service được chỉ định.

- **Forward**: khi có một Req 🡪 thành phần của servlet đó, servlet được break down những set nhỏ 🡪 gom toàn bộ tài nguyên ( 2 -> n ) của các servlet khác nhét vào servlet duy nhất đó sau có compile trở thành một tài nguyên.

-Compile một lần duy nhất ở Req đầu tiên 🡪 Static: Sau khi đáp ứng Req 🡪Các tài nguyên con được làm mới lại 🡪 không lấy mới 🡪 chờ đến khi có Req tiếp theo 🡪 mới lấy 🡪 Compile lại.

- **Include** : chỉ làm một lần 🡪 không cập nhật ( tĩnh ).

|  |  |
| --- | --- |
| **FORWARD** | **INCLUDE** |
| - Qua một thành phần sau đó chuyển cho các thành phần khác xử lí.  -Thành phần cuối cùng xử lí là Res.  - Khi thực thi có nhiều thành phần được thực thi 🡪 Tài nguyên cuối cùng thực thi | -Qua một thành phần xử lí.  -Những thành phần khác không xử lí mà gom toàn bộ code của implement đưa vào thành phần xử lí và res ⬄ chia nhỏ công việc nhưng thực thi chỉ có một. |

\* Servlet object: Bên cạnh hỗ trợ cho thành phần bên ngoài vẫn có thể hỗ trợ và giải quyết tác vụ cho những Req khác.

\* Forward cho thiết kế button: từ Request Dispatcher => MVC Patter.

- Hệ thống chỉ có một servlet làm điều phối , các servlet còn lại đãm nhận một chức năng vì sự hỗ trợ của RequestDispatcher => giao tiếp giữa các Req object với nhau.

- Recompile = dán code vào ⬄ trước khi thực thi sao chép toàn bộ code của tài nguyên ( 2,3,..n).

- Khi nào có Req dựa trên câu lệnh chính yếu của servlet đầu tiên để thực thi code :

1.Deploy servlet đầu tiên.

2.Lưu Address 🡪 Servlet 2,3,…

3. Khi deploy lại🡪 Mới Recompile ( vì include = tĩnh ).

**Thuyết trình**

**1)Sự khác biệt giữa HTML và XML:**

+ html 🡪 Bản chất là những thành phần có cú pháp .Mỗi thẻ html đều có nghĩa 🡪 Đưa Css format html 🡪 phải tuân thủ luật cú pháp .

+ xml 🡪 Định nghĩa kiễu dữ liệu / thuộc tính mới ⬄ cho phép tạo thẻ mới 🡪 mô tả cái muốn thêm.

**2)JSP:**

- Java Server Page

- Text File ( ojsp) -> render: xml, html, js, css, java.

**3) Cơ chế xữ lí:**

-User gửi req 🡪 Server 🡪 Server tiếp nhận req 🡪 Server check đây là req đầu tiên 🡨 jsp được đưa tới jsp engine 🡨 req yêu cầu trang jsp gửi cho trang trang jsp 🡪 tại jsp engine file jsp được paste 🡪 tạo ra file java có tên **Name\_jsp. Java**

- Các nội dung java code trong jsp được convert thành code java được đặt trong jsp service() – phương thức.

-Sau khi convert toàn bộ vào trong trang jsp 🡪container compile thành file .class 🡪 tạo ra jsp servlet ob 🡪 thực thi yêu cầu req và trả về ResMess = html.

- Khi có Request đầu tiên 🡪Server 🡪Container định vị trang jsp se qua jsp engine 🡪 convert file text trở thành tập tin . java 🡪 Container compile thành file.class 🡪 load lên thành servlet ob 🡪 thực thi => Đưa kết quả về Br.

\* Request thứ 2 trở đi : Container kiểm tra file jsp đó.

+ Không : Load file class lên bộ nhớ. (\*\*)

+ Convert -> ReCompile.

(\*\*) => Tại sao không compile lại file class.

1.File java không đổi => ob tương ứng không đổi => chỉ cần thực thi ( có sẵn trong bộ nhớ )

2.jsp servlet ob là một servlet khác thường ( chỉ phục vụ cho một Req/Page)

3) Compile 1 file 2 lần => cùng kết quả.

\* Ở Req 2 :

A.Check xem file có bị thay đổi không.

+ Có : thực thi lại từ đầu

+ Không: Chỉ thực hiện load.

B. jsp servlet ob mang khái niệm khác servlet ob ( After) 🡪 Khởi tạo khi deploy app 🡪 Undeploy.

Scope ( service do container support).

**1.Container đang cung cấp cho servlet ob 3 Scope:**

- Context : ServletContext

- Session: Http Session

- Request: Http Servlet Request.

**2.Bên jsp servlet ob:**

- Context: Được thế hiện = Application.

- Session:

- Request :

- Page : Chỉ áp dụng cho jsp ( là resource ở server) + tồn tại = page time ( từ khi page dc load -> unload) , type : Page Context.

\* Trong thành phần bộ nhớ container: một Req có rất nhiều Page + Mỗi page xử lí những tác vụ khác nhau .Sau khi gọi một trang 🡪 page khác load 🡪 unload page cũ.

🡪 Lun có một page phía server ( các page dc chuyển qua liên tiếp vs nhau ) 🡪 Tiết kiệm bộ nhớ.

**4) Scripting Element:**

\*Scripting Element:

- Nhúng code java vào jsp 🡪 Biến , hàm, phương thức.

- Syntax: <% \_ %>

**a) Declaration:**

- Syntax: <%! ­\_ %>

- Dùng đễ khai báo biến và định nghĩa hàm.

- Biến ( Variable): tầm vực default.

- Phát sinh bên ngoài \_ jsp service() bên trong class ( inside class, outside\_ jspservice()).

- Chu kì sống của servlet ob:

1)init(): + Chạy một lần ở Req đầu tiên.

+ Thay thế default constructor.

2) service() \_ được trigger dựa trên nguyên nhân khi có Req đến 🡪 dựa vào Req line ( method của req line) – chạy nhiều lần.

3) destroy() : + Chạy 1 lần

+ Trước khi deploy.

- JSP \_ Cơ chế bản chất toàn bộ thành phần ở bên trong 3 phần:

- jsp init: có thể override ngoại trị (service) 🡪 hàm abtract => cấm override.

- jsp service: ( chạy 1 lần)

- jsp destroy: giống servlet object.

- Nếu cố tình override – jsp service.App sẽ phát sinh lỗi tại thời điểm: Request time.

+ Viết sai 🡪 deploy ( jsp) không đổi 🡪 app không sữ dụng tính năng đó 🡪 không có lỗi.

+ Bản chất jsp là text file => hoàn toàn không có lỗi.

+ Request time error xuất hiện khi quá trình xử lí đụng đến trang jsp 🡪 Có load jsp 🡪 jsp engine 🡪 diễn ra nhiều giai đoạn.

1) Convert từ jsp text file 🡪 java.

2) Convert complete 🡪 Compile error.

3) Convert + compile complete 🡪 thực thi phát sinh lỗi.

**b) Scriptlet:**

- Syntax: <% \_ %>

- Viết trên nhiều dòng + được khai báo biến – phát sinh trong – jsp service() 🡪 tầm vực local.

- Không được khai báo phương thức 🡪 Vì –jsp service() đã là phương thức 🡪 không thể truy cập phương thức trong phương thức.

-Trong phương thức -> Định nghĩa class 🡪 trong class có phương thức 🡪 vẫn có thể gọi phương thức ( bởi vì class là một ob instance trong phương thức)🡪 truy cập phương thức 🡪 thông qua ob instance gọi phương thức.

=> Không có khái niệm gọi phương thức thông qua phương thức .

- Code phát sinh của Scriptlet phát sinh ở trong - jsp service()

- Chức năng : Xử lí request.

**c) Expression:**

- Nhận giá trị của biến \_ Được phép gọi hàm \_ Có thể tính được biểu thức.

-Được Container Convert 🡪 Biến thành câu lệnh out.writter 🡪 Convert thành chuỗi đặt vào out.write

- Viết trên một dòng , không có **;**

🡪 Sử dụng để định trị biễu thức + gọi phương thức -> out.writeln 🡪 output vào trong html 🡪 String 🡪 Res ob.

-Syntax: <% =expression %>.

**5)Implicit ob:**

- Những ob có sẵn có thể sử dụng không cần tạo trên jsp.

- Dùng trong : scriptlet/expression.

- Khi sử dụng không cần khai báo 🡪 gọi ob phương thức.

- Bên server có sẵn khi có Req gửi đến

- Implicit ob:

+ Session:

+ Application:

+ Page:

+ Exception:

+Servlet Config = Config

+ Out = print writer

- Là những đối tượng có sẵn trong thành phần - jsp service()

- Được load khi có Req phía sever.

- Out 🡪 Có sẵn chỉ cần get.

- Từ Req : + Get Session

+ Get Context

- Liên hệ vs nhau dựa trên thành phần hệ thống thông qua API 🡪 Có sẵn để sử dụng.

**6) Directive:**

- Giúp cung cấp thông tin page về file jsp .

- Khai báo trên jsp yêu cầu container 🡪 convert 🡪file servlet chính xác

- Syntax <% @ = %> 🡪 Viết trong 1 dòng.

-Req các service từ container / jsp engine.

- Có 3 loại : + page

+ include

+ taglib

- Page = import package ( hỗ trợ section, import library)

+ Include = RequestDispatcherInclude ( servlet ob) 🡪 Convert chĩ trả một file jsp thay vì nhiều jsp .

+ taglib = dùng thư viện khác.

**7) Life Cycle of Jsp:**

- Có 4 giai đoạn:

**a. Biên dịch( compilation)**: khi Br gửi Req jsp cho web server công cụ jsp engine sẽ kiểm tra xem nó có được tạo mới hoặc sữa chửa gì không.

+ Có : Công cụ jsp , biên dịch trang.

+ Không : Lấy servlet đã biên dịch và thực hiện.

**\* Qúa trình biên dịch: phân tích cú pháp jsp chuyển jsp 🡪 servlet biên dịch servlet**.

**- Khởi tạo JSP( -jsp init())** : được thực hiện ở Req đầu tiên = servlet init()

-Thực thi JSP( - jsp service()): Method có http ServletRequest và HttpServletRespone là tham số chạy một lần cho 1 Req.

-Hủy jsp(-jsp destroy()) = Servlet destroy().

**Session**

\*Review:

-Container cung cấp nơi để các thành phần trong app giao tiếp vs nhau ⬄ Scope.

- Session 🡪 tạo ra bởi mong muốn của user.

1 session scope = 1 user.

**1) URL Rewriting**: Delete.

**2)Hidden form**: Cập nhật dòng trên view.

**3)Session Tracking + Cookie**: Bằng hành động của user lưu trữ ở phía server,đảm bảo tác vụ: remember,remind,….

**4)Shopping cart**: người dùng mua nhiều đồ => hệ thống không quên.

**5)Error Handle:**

- Bắt lỗi nhưng chưa xử lí lỗi.

PrintStackTrace() 🡪 in lỗi ra console

- Xác định lỗi : + System

+ User

🡪 Toàn bộ hành vi thao tác trên máy tính, app sẽ ghi nhận lại để sữa chữa lỗi.

**-Error**: + System

Đều phải ghi nhận

+ User

-Web App = Không biết user🡪 không biết lỗi + không thể truy lung lỗi 🡪 toàn bộ thao tác phải được ghi nhận trên hệ thống = ghi Log file -> Phân tích 🡪 phát sinh behavior user.

- Facebook + Google: Collection data user.

🡪 Phân tích 🡪 Xác định xu hướng.

🡪 Bán cho công ty về kinh doanh thương mại 🡪 Nên đầu tư lĩnh vực nào.

**Requirement**

1) Trên lưới – Có thể xóa dòng bất kì 🡪 Sau khi delete 🡪 lưới và dữ liệu được cập nhật.

2) Trên lưới : Có thể update password 🡪 Lưu trữ của 1 dòng cụ thể 🡪 Cập nhật .

3)Remember password: Ở lần đầu tiên sau khi login thành công 🡪 Lần 2: vào thẳng trang web 🡪 Shoe inFor.

4) Shopping cart: Add, view, remove,…

5) Thêm mới 1 account vào System:

+ Thành công 🡪 login

+ Thất bại 🡪 Show tại sao error.

6) Delete:

**\* Lưu ý:**

-Toàn bộ trang phải có welcome.

-Shopping: Gio hàng : chọn hàng 🡪 Add vào giỏ 🡪 view remove giỏ 🡨 giỏ.

- Đăng kí : thành công 🡪 login

Không thành công 🡪 báo lỗi.

**\* Account:**

- Hệ thống có các loại account để quản lí account như :

a.Supper admin:

- Mỗi app chỉ có 1 supper admin.

- Khi deploy app complete 🡪 supper admin cấp quyền + tạo admin.

- Admin tạo account ( staff)

- Supper Admin: Show toàn bộ account( - supper admin) , không thể thấy account chính mình.

- Admin: thấy được account admin tương tự , không làm dc gì trên = account đó.

**\* Qui tắc:**

- Admin không xóa admin.

-Admin không xóa chính mình.

- Admin không được thay đổi quyền của các Admin khác.

🡪 Admin dc cấp Admin.

- Supper Admin chỉ sữ dụng quyền khi Admin có vấn đề.

**\*Phân quyền trong DB:**

-DB không thực sự thuộc quyền sa:

- một table /DB có phân quyền riêng.

-Backup luôn thay đổi : phần quan trọng vẫn còn ( đảm bảo)

**\* Cơ chế Session Tracking:**

- Đáp ứng nhiều hơn một Req liên tục nhau ( Successive Req) 🡪 Duy trì thông tin người dùng ở phía container để đáp ứng nhiều hơn một Req.

=> Duy trì thông tin của người dùng ở server để đáp ứng nhiều hơn một Req.

**\*Successive:**

-Req liên tiếp nhau trong khoảng thời gian session còn tồn tại ⬄ send Req 🡪 liên tiếp: Không theo khái niệm ( gõ mạch liên tục).

- Định kì Req tới sao cho thời gian Session Scope vẫn còn tồn tại.

**\* URL Rewriting:**

- Chia thành 2 phần :

URL 🡨 Rewriting

-Thành phần ReqMess liên quan URL: Reqline : Xác định tài nguyên phía server để xử lí Req.

- Server/Container 🡪 Không nhớ client đã làm gì => Giúp Server hoặc Container nhớ client đã làm gì => Nhắc ( Remind).

\* Add URL:

- Syntax: URL ở Query – String.

- Nhắc cho server nhớ client đã làm gì = cách truyền thêm parameter vào đường dẫn url.

- Phương thức tĩnh ( static)

- ? Truyền parameter nào là do developer viết sẵn.

- Gía trị thay đổi theo đường truyền.

- Không thể đón nhận giá trị nhập liệu tại thời điểm run time.

- href = “ url ? parameter”

+ Gía trị có thể thay đổi.

+ Tên cố định.

**\* Disadvantage URL Rewriting:**

**⬄ GET.**

**1.Phân tích bài toán:**

- Xóa 1 dòng cần truyền về server primary key.

- Delete link 🡪 gửi về server 🡪 gọi tài nguyên có định dạng.

- DispatchController phân biệt hành động thông qua giá trị nút ( btAction).

- Xóa: Phải có username = giá trị cần xóa.

- DispatchController ? btAction = delete & username = data cần xóa.

- **Click link delete/update** 🡪 **Send Server** 🡪 **Send Servlet/Container** 🡪**Send Servlet chức năng** 🡪 **thực hiện thành phần DAO + Connect DB** 🡪 **thực hiện** 🡪 **trả về servlet**.

**2.Sơ đồ:**

- Click 🡪 Br 🡪Server 🡪 Servlet <-> DAO <-> DB.

**3.Refesh giao diện:**

- Giao diện thực hiện chức năng delete do thực hiện chức năng Search để có.

- Xác định chức năng 🡪 Có được giao diện 🡪 Gọi lại chức năng đó thêm một lần nữa

- Kích hoạt search form ở client.

- Click Search 🡪Form Search không truyền về server 🡪 Chỉ truyền ReqMess url khi click search 🡪 Reqline: URL.

- DispatchController ? btAction = search & txtSearch Value = giá trị đã nhập , đã search bằng gì ?

- Đẩy về Server 🡪 xử lí Complete 🡪 Gía trị search không được lưu trữ.

- Gía trị vẩn tồn tại trong Req Scope. ( sử dụng forward) 🡪 Khi render 🡪 Chỉ cần Remind = cách.

**Data Source**

- Context . xml: Hỗ trợ connect DB mà không phụ thuộc vào bất kì ràng buộc toàn bộ nội dung 🡪 Gọi là Data Source.

**\* Data Source:**

- Tên Luận Lý ( logical name): dùng tương tác toàn bộ thành phần.

- Binding: DB, IP, Port,Security,…..

-Tại thời điểm Binding vs đối tượng nào 🡪 Đang tương tác vs đối tượng đó.

- Là thành phần chính trong app để refend 🡪 thành phần nguồn dữ liệu cụ thể để thao tác.

-Config Data Source:  
 - Define data Source 🡪 trong context.xml

- Map data source 🡪 application 🡪 trong web.xml

- Sử dụng data source để xử lí connect DB.

- Có trước khi app tồn tại:

+ War 🡪 deploy 🡪 Đọc context . xml đầu tiên

+ Nơi xác định toàn bộ thành phần liên quan đến app run như thế nào , các thành phần thiết lập cơ bản của app : Web .xml