**What is Servlet**

Web application တစ္ခုကို ဖန္တီးရာမွာ အသံုးၿပဳတဲ့ technology တစ္ခုၿဖစ္တယ္။ အဲ့မွာ web applicaton တစ္ခုဆိုတာ ဘာလဲဆိုေတာ့ sever side ရယ္ dynamic web page တစ္ခုရွိတယ္ဆိုရင္ အဲ့တာကို web application လို႔ေခၚတယ္။

Servlet ဆိုတာကလည္း java ၿဖစ္တဲ့အတြက္ robust နဲ႔ scalable ၿဖစ္တယ္။ servlet မတိုင္ခင္တုန္းက CGI (Common Gateway Interface) scripting language ကို server side programming language တစ္ခုအေနနဲ႔ သံုးခဲ့ရတယ္။

Servlect API မွာ မ်ားၿပားလွတဲ့ interfaces နဲ႔ classes’ ေတြရွိတယ္ ဥပမာ Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc. တို႔ၿဖစ္ၾကတယ္

**Servlet ဆိုတာဘာလဲ**

Servlet ဆိုတာကို မ်ိဳးစံုအဓိပၸါယ္ဖြင့္လို႔ရတယ္။

* Web application တစ္ခုကို ဖန္တီးလို႔ရတဲ့ technology တစ္ခု။
* Documentation ပါ၀င္တဲ့ interfaces နဲ႔ classes မ်ားကို ေပးႏိုင္တဲ့ API တစ္ခု။
* မည္သည့္ servlet ကိုမဆိုဖန္တီးမယ္ဆိုရင္ servlet interface ကို implements လုပ္ကိုလုပ္ရမယ္ အဲ့တာေၾကာင့္ သူက interface တစ္ခုဆိုလည္း ဟုတ္တယ္။
* Servlet ဆိုတာ class တစ္ခုလည္း ဟုတ္တယ္ သူက မည္သည့္ request ကိုမဆို response ၿပန္ႏိုင္တယ္။
* Dynamic web page တစ္ခုကို ဖန္တီးဖို႔ရန္အတြက္ server ေပၚမွာ deployed လုပ္ၿခင္းခံရတဲ့ web component တစ္ခုလည္း ၿဖစ္တယ္။



**Web Application ဆိုတာဘာလဲ**

အလြယ္ေၿပာရမယ္ဆိုရင္ေတာ့ Web ေပၚမွာ run ႏိုင္တဲ့ application ကို web application လို႔ေခၚတယ္။ သူ႕ကို web components မ်ားၿဖစ္ၾကတဲ့ Servlet, JSP, Filter အစရွိတဲ့ web components မ်ားရယ္ HTML, CSS, and JavaScript အစရွိတဲ့ elements မ်ားနဲ႔ ေပါင္းစပ္ဖြဲ႕စည္းထားၿခင္း ၿဖစ္ပါတယ္။ အဲ့ web components ေတြက web server ေပၚမွာ execute လုပ္တယ္ ၿပီးရင္ HTTP request ကို response ၿပန္တယ္။

**CGI (Common Gateway Interface)**

CGI ဆိုတာကလည္း server နဲ႔အလုပ္တြဲလုပ္တဲ့ technology တစ္ခုၿဖစ္တယ္။ program ဆီကို http request မ်ားပို႔လႊတ္မယ္ process ေတြလုပ္မယ္ အစရွိသၿဖင့္ server side လုပ္ငန္းမ်ားကို လုပ္ေဆာင္ႏိုင္တယ္။



**CGI ရဲ႕ မေကာင္းတဲ့အခ်က္မ်ား**

* Client မ်ားလာရင္ response time ပိုၾကာလာေရာ။
* သူက request တစ္ခုခ်င္းစီတိုင္းအတြက္ process တစ္ခုစတယ္ ဒါေပမယ့္ web server မွာက စတဲ့ process limit က အကန္႔အသတ္ရွိတယ္။
* သူ႕ကို platform dependent language ေတြၿဖစ္တဲ့ C, C++, perl တို႔က သံုးၾကတယ္။ platform independent ၿဖစ္တဲ့ java အတြက္ သိပ္မကိုက္လွေပဘူး။

**Servlet ရဲ႕ ေကာင္းတဲ့အခ်က္မ်ား**

Servlet မွာက ေကာင္းတဲ့အခ်က္အမ်ားၾကီးရွိတယ္။ servlet container ကို လာတဲ့ multiple request မ်ားကို handle လုပ္ဖို႔အတြက္ web container က threads မ်ားကို ဖန္တီးလိုက္တယ္။အဲ့မွာ ေကာင္းက်ိဳးေတြ မ်ားစြာၿဖစ္ေပၚလာတယ္ ဘာေတြလဲဆိုေတာ့ thread ၿဖစ္တဲ့အတြက္ေၾကာင့္ common memory area တစ္ခုကို share ၿပီးသံုးႏိုင္တယ္ lightweight ၿဖစ္တယ္ thread ေတြၾကားထဲက communication cost က ေလ်ာ့က်သြားတယ္။



* **Performance ပိုေကာင္းလာတယ္**: ဘာလို႔လဲဆိုေတာ့ process မဟုတ္ပဲ thread ေတြကို သံုးထားလို႔ၿဖစ္တယ္။ request တစ္ခုခ်င္းစီတိုင္းအတြက္ thread တစ္ခုစီကို servlet က ဖန္တီးေပးလိုက္လို႔ၿဖစ္တယ္။
* **Portability**: java language ကို သံုးထားလို႔။
* **Robust**: JVM က servlet ကို manage လုပ္တဲ့အတြက္ ကၽြန္ေတာ္တို႔က memory မေလာက္တာတို႔ garbage collection တို႔အတြက္ စိတ္ပူစရာမလိုေတာ့ဘူး။
* **Secure**: Java language ကို သံုးထားလို႔။

**Web Terminology မ်ား**

1. **Website**
2. **HTTP**
3. **HTTP Requests**
4. **Get vs Post**
5. **Container**
6. **Server: Web vs Application**
7. **Content Type**

**1. Website**

Website မွာက ၂ မ်ိဳးရွိတယ္။

1. **Static website:** static website လုပ္ရတာက လြယ္ပါတယ္။ programming knowledge တို႔ database design တို႔ဘာမွ သိစရာမလိုဘူး။ static website ဆိုတာက page ေတြက အေသေတြ သူ႕ထဲက information က ဘာမွမေၿပာင္းလဲသြားဘူး။

2. **Dynamic website:** dynamic ၿဖစ္ေနတဲ့ web pages မ်ားကို စုစည္းထားၿခင္းၿဖစ္တယ္။ သူ႕ရဲ႕ content ေတြက dynamically ေၿပာင္းလဲေနၾကတယ္။ သူက content ေတြကို access လုပ္ေပးေနတယ္။အဲ့ content ေတြက ဘာေတြ

ထြက္လာလဲဆိုေတာ့ Database ဆီကို CMS ကတစ္ဆင့္ ထြက္လာၾကတာ။ အဲ့တာေၾကာင့္ တစ္ခုခုကို update လုပ္လိုက္မယ္ဆိုရင္ အဲ့တာေတြက လိုက္ေၿပာင္းသြားလိမ့္မယ္ ဆိုလိုတာက အေသမဟုတ္တဲ့အတြက္ေၾကာင့္။

Dynamic website ေတြက client side scripting သို႔မဟုတ္ server side scripting တစ္ခုခု ဒါမွမဟုတ္ ႏွစ္ခုစလံုးကို သံုးတယ္။

Client side scripting ဆိုတာက သူက user ရဲ႕ computer ေပၚမွာပဲ user ရိုက္ထည့္လိုက္တဲ့ input ေတြ အေပၚမွာပဲ content ေတြကို ထုတ္ေပးတယ္။ဆိုလိုတာက web browser က server ဆီကေန web page ကို download ဆြဲလိုက္တယ္ ၿပီးေတာ့ user ကိုၿပမယ့္ information ေတြကို page ထဲမွာပဲ code ေတြကို render လုပ္လိုက္ၿခင္းကို ဆိုလိုတယ္။ Server side scripting မွာက်ေတာ့ software က server ေပၚမွာ run တယ္။server ထဲမွာ processing လုပ္လို႔ၿပီးရင္ plain pages ေတြကို server ဆီကို ပိုု႔လႊတ္လိုက္တယ္။



|  |  |
| --- | --- |
| Static Website | Dynamic Website |
| Page တက္တိုင္းတက္လိုင္း ၿပတာ အတူတူပဲ | Page တက္တိုင္း ေၿပာင္းလဲေနတယ္ |
| Html သိရင္ ရၿပီ | Technology မ်ိဳးစံုနဲ႔ ေရးလို႔ရတယ္ |
| Request တိုင္းမွာ တူညီတဲ့ response ကိုပဲၿပန္တယ္ | Request တစ္ခုစီတိုင္းမွာ မတူညီတဲ့ response ကိုလည္း ထုတ္ေပးႏိုင္တယ္ |
| Update လုပ္ခ်င္တယ္ဆိုရင္ file ကို update လုပ္ၿပီး server ထဲသြားထည့္ရမယ္ | Page ထဲမွာပါတဲ့ server side code ေတြက page load လုပ္တိုင္းမွာ မတူညီတဲ့ content ေတြကို ထုတ္ေပးေနတယ္ |
| Static website က fix ၿဖစ္တယ္ အဲ့တာက သူရဲ႕ တစ္ခုတည္းေသာ ေကာင္းတဲ့အခ်က္ပဲ | သူ႕ရဲ႕ ေကာင္းတဲ့အခ်က္က CMS ကို ေကာင္းစြာလုပ္ေဆာင္ႏိုင္တယ္ |

**Http (Hyper Text Transfer Protocol)**

Client နဲ႔ Server ၾကားထဲက communication လုပ္ေပးတဲ့ data communication protocol တစ္ခုၿဖစ္တယ္။ Http ဆိုတာက TCP/IP ကိုအေၿခခံထားတဲ့ communication protocol တစ္ခုၿဖစ္ၿပီး image files, query results, HTML files အစရွိတာေတြကို world wide web ေပၚမွာ ေပးပို႔ဖို႔ အတြက္ သံုးတယ္။ သူ႕ရဲ႕ default port ကေတာ့ TCP 80 ၿဖစ္တယ္။



**Http ရဲ႕အေၿခခံက်ေသာ ၀ိေသသမ်ား**

* Web servers မ်ားနဲ႔ browser မ်ားၾကားထဲမွာ data မ်ားကို exchange လုပ္ေပးတဲ့ protocol တစ္ခုၿဖစ္တယ္။
* Request Response Protocol တစ္ခုၿဖစ္တယ္။
* သူက ယံုၾကည္ရတဲ့ TCP Connection မ်ားကို သံုးပါတယ္။သူ႕ရဲ႕ default protocol ကေတာ့ TCP 80 ပါ။
* သူက stateless ၿဖစ္တယ္။ဆိုလိုတာက request တစ္ခုပို႔လိုက္တိုင္း request အသစ္လို႔ပဲ စဥ္းစားခံရတယ္။server က user ကို ဘာမွအသိေပးထားၿခင္း မရွိဘူး။

**Http ရဲ႕ အေၿခခံက်ေသာ အခ်က္မ်ား**

သူ႕မွာ ၃ ခ်က္ရွိတယ္။

* **HTTP is media independent: ဆိုလိုတာက ဘယ္ media type မ်ိဳးၿဖစ္ပါေစ သူက handle လုပ္ႏိုင္တယ္။**
* **HTTP request ပို႔လိုက္ၿပီးတဲ့ေနာက္ server နဲ႔ user က connection ၿပတ္ေတာက္သြားတယ္ဆိုရင္ေတာင္ response ကို ေစာင့္ေပးတယ္။**
* **HTTP Stateless**

**HTTP ရဲ႕ အေၿခခံ architecture**



**Http Requests**

Http requests ဆိုတာက computer တစ္လံုးကေန web server တစ္ခုဆီသို႔ ပို႔လိုက္တဲ့ request မွာ စီထားတဲ့ interesting information ေတြပါ၀င္တယ္ အဲ့တာကို http requests မ်ားလို႔ေခၚတယ္။

User တစ္ေယာက္က server ဆီကို request တစ္ခုပို႔တဲ့အခါ ေအာက္ေဖာ္ၿပပါ information မ်ားပါ၀င္ပါတယ္။

* The Request-line
* The analysis of source IP address, proxy and port
* The analysis of destination IP address, protocol, port and host
* The Requested URI (Uniform Resource Identifier)
* The Request method and Content
* The User-Agent header
* The Connection control header
* The Cache control header

Http request method က **Requested URI (Uniform Resource Identifier) လုပ္ခိုင္းတဲ့ အတိုင္းလုပ္ေဆာင္ရတယ္။ အဲ့ method က case sensitive ၿဖစ္ရမယ္ upper case ၿဖစ္ရမယ္။**

**GET vs POST**

GET နဲ႔ POST ၾကားထဲမွာ မ်ားစြာေသာ ကြဲၿပားၿခားနားခ်က္မွာ ရွိပါတယ္။

|  |  |
| --- | --- |
| GET | POST |
| Data ကို limit amout နဲ႔ပဲ ေပးပို႔လို႔ရတယ္ ဘာလို႔လဲဆိုေတာ့ data ကို header ထဲမွာေပးပို႔ရလို႔။ | Data ကို large amount ေပးပို႔လို႔ရတယ္ ဘာလို႔လဲဆိုေတာ့ data ကို body ထဲမွာေပးပို႔ရလို႔။ |
| Data ေတြ စိတ္မခ်ရဘူး ဘာလို႔လဲဆိုေတာ့ url မွာအကုန္ၿမင္ေနရလို႔။ | Data ေတြက စိတ္ခ်ရတယ္ဘာလို႔လဲဆိုေတာ့ url မွာ မၿမင္ရလို႔။ |
| သူ႕ကို bookmark ထဲမွာ ထည့္သိမ္းထားလို႔ရတယ္။ | သူကေတာ့ bookmark ထဲမွာ ထည့္သိမ္းထားလို႔မရဘူး။ |
| သူက idempotent ၿဖစ္တယ္ အဲ့တာဘာေၿပာတာလဲဆိုေတာ့ second request ပို႔လိုက္လည္း first request ကို အေႏွာက္အယွက္မေပးဘူး first request က သူ႕အလုပ္ကို သူဆက္လုပ္ႏိုင္တယ္။ | သူကေတာ့ idempotent မၿဖစ္ဘူး။ |
| သူက ပိုထိေရာက္တယ္ အဲ့တာေၾကာင့္ post ထက္ပိုသံုးၾကတယ္။ | သူကေတာ့ သိပ္မထိေရာက္ဘူး အဲ့တာေၾကာင့္အသံုးနည္းတယ္။ |

သူတို႔ရဲ႕ အားသာခ်က္ အားနည္းခ်က္ကို သိသြားတဲ့အခါ သူတို႔ရဲ႕အသံုး၀င္ပံုကိုလည္း ပိုနားလည္သြားႏိုင္တယ္။

အဲ့တာက ဘာလဲဆိုေတာ့……

**GET** ကို resource ဆီက data မ်ားကို request လုပ္တဲ့အခါ သံုးတယ္။

**POST** ကို resource ဆီသို႔ processed လုပ္ထားေသာ data မ်ား submit လုပ္တဲ့အခါ သံုးပါတယ္။

**GET Request ကိုခြဲၿခမ္းစိပ္ၿဖာၿခင္း**

တစ္ကယ္တမ္းက်ေတာ့ query string ကို GET Request ရဲ႕ url အတြင္းမွာ ေပးပို႔ၿခင္းပဲၿဖစ္တယ္။

GET/RegisterDao.jsp?name1=value1&name2=value2

အဲ့တာက dataကို request header ထဲမွာ ပို႔တယ္လို႔လည္း ေၿပာလို႔ရတယ္။ဒါက default request တစ္ခုလည္း ၿဖစ္တယ္။



အခ်ိဳ႕ေသာ GET method ရဲ႕ feature ေတြကေတာ့…..

1. Browser history က်န္တယ္။
2. Bookmark ထဲမွာ ထည့္သိမ္းထားလို႔ရတယ္။
3. Cached က်န္တယ္။
4. Length restrictions ရွိတယ္။
5. Sensitive ၿဖစ္တဲ့ data ေတြအတြက္ မသံုးသင့္ဘူး။
6. Data ေတြယူဖို႔အတြက္ေလာက္ပဲ သံုးသင့္တယ္။

**POST Request ကိုခြဲၿခမ္းစိပ္ၿဖာၿခင္း**

တစ္ကယ္တမ္းက်ေတာ့ query string ကို GET Request ရဲ႕ HTTP message body အတြင္းမွာ ေပးပို႔ၿခင္းပဲၿဖစ္တယ္။

POST/RegisterDao.jsp HTTP/1.1

Host: www. javatpoint.com

name1=value1&name2=value2

အခ်ိဳ႕ေသာ POST method ရဲ႕ feature ေတြကေတာ့…..

Bookmark ထဲမွာ မွတ္ထားလို႔မရဘူး။

သူ႕မွာ Data length restriction မရွိဘူး။

Cached လံုး၀မရွိဘူး။

Browser history ထဲမွာ သူ႕ကိုလံုး၀ၿပန္ေဖာ္လို႔မရဘူး။



**Servlet Container**

Java EE applications မ်ားအတြက္ run time environment ကို ထုတ္လုပ္ေပးတယ္။ client က static web pages မ်ားမွာသာ server ဆီကို request ပို႔ႏိုင္တယ္။

Servlet Container ဆိုတာက web server ရဲ႕အစိတ္အပိုင္းတစ္ခုၿဖစ္ၿပီး သူ႕ကို သီၿခား process တစ္ခုအေနနဲ႔ run လို႔ရတယ္။



**Servlet Container States**

၃ ပိုင္းရွိပါတယ္။

1. Standalone: သူက typical java based server ေတြၿဖစ္တယ္။သူထဲမွာ ရွိတဲ့ servlet container နဲ႔ web server's တို႔ေတြက single program တစ္ခုရဲ႕ ၿပီးဆံုးေအာင္ လုပ္ေဆာင္ရမယ့္ အစိတ္အပိုင္းမ်ားၿဖစ္တယ္။ဥပမာ Tomcat က ကိုယ့္ဟာကုိယ္ run ႏိုင္တယ္။
2. In-process: သူကေတာ့ web server က separate ၿပဳလုပ္ၿခင္းခံထားရတယ္ ဘာလို႔လဲဆိုေတာ့ main server တစ္ခုရဲ႕ address space တစ္ခုတည္းမွာ မတူညီတဲ့ program တစ္ခုကို run ထားလို႔ၿဖစ္တယ္ (plugin တစ္ခုအေနနဲ႔) ။ဥပမာ JBoss ထဲမွာ Tomcat ကို run ၿခင္း။
3. Out-of-process: web server နဲ႔ servlet container တို႔ဟာ မတူညီတဲ့ program ေတြၿဖစ္ၾကတယ္။သူတို႔ထဲမွာ မတူညီတဲ့ program ေတြ run ေနၾကတယ္။ သူတို႔ၾကားထဲမွာ ဆက္သြယ္ဖို႔အတြက္ web server က plugin ကိုသံုးပါတယ္။အဲ့ plugin က servlet container ကို ေပးစြမ္းတယ္။

**Servlet Container လုပ္ေဆာင္ႏိုင္ေသာ operations မ်ား**

1. Life Cycle Management
2. Multithreaded Support
3. Object Pooling
4. Security

**Server: Web vs Application**

Server ဆိုတာ တကယ္ေတာ့ computer တစ္လံုး သို႔မဟုတ္ device တစ္ခုၿဖစ္တယ္။ သူက request ေတြကို လက္ခံႏိုင္တယ္ response ၿပန္ႏိုင္တယ္ ဘယ္သူ႕ဆီကိုလည္းဆိုေတာ့ client ေတြဆီကို။ network resources မ်ားကို manage လုပ္ဖို႔အတြက္နဲ႔ program သို႔မဟုတ္ software ေတြ run ဖို႔အတြက္ သူ႕ကိုသံုးတယ္။

သူ႕မွာ ၂ မ်ိဳးရွိတယ္။

1. Web Server
2. Application Server

**Web Server**

Web server မွာ web or servlet container တစ္ခုသာလွ်င္ပါ၀င္ပါတယ္။ သူ႕ကို servlet, jsp, struts, jsf အစရွိတဲ့ technology မ်ားနဲ႔ သံုးစြဲႏိုင္တယ္။ EJB မွာေတာ့ မသံုးပါဘူး။

အဲ့တာက computer တစ္လံုးၿဖစ္ၿပီးေတာ့ အဲ့ထဲမွာ web content ေတြကို သိမ္းဆည္းထားတယ္။ေယဘုယအားၿဖင့္ web server ကို website မ်ား host လုပ္ဖို႔ရန္အတြက္ သံုးတယ္ ဒါေပမယ့္ ဒီေနရာမွာ တစ္ၿခား web server မ်ားကိုလည္းသံုးႏိုင္ပါတယ္ FTP, email, storage, gaming အစရွိတဲ့ web server မ်ားၿဖစ္ၾကတယ္။

Web server မ်ားရဲ႕ ဥပမာကေတာ့ Apache Tomcat ၿဖစ္ပါတယ္။

**Web Server ရဲ႕အလုပ္လုပ္ပံု**

သူက နည္းလမ္း ၂ မ်ိဳးနဲ႔ client ဆီက request မ်ားကို response လုပ္ႏိုင္ပါတယ္။

Web server က script ကို အသံုးၿပဳၿခင္းအားၿဖင့္ database နဲ႔အလုပ္တြဲလုပ္ၿခင္းအားၿဖင့္ response ကို ထုတ္လႊတ္ပါတယ္။

Requested url နဲ႔ စပ္ဆက္ေနတဲ့ file မ်ားကို client ဆီကို ပို႔ေပးတယ္။



**အေရးၾကီးေသာအခ်က္မ်ား**

* အကယ္လို႔မ်ား requested Web page ကို client side မွာမေတြ႕ေတာ့ဘူးဆိုရင္ web server က 404 ဆိုတဲ့ http response ကို response ၿပန္ေပးလိမ့္မယ္။
* Web server က requested web page ကုိရွာေသာအခါ page ကို ေတြ႕ခဲ့မယ္ဆိုရင္ client ကို http response နဲ႔ page ကို ၿပန္ေပးလိမ့္မယ္။
* တစ္ၿခား Resource ဆီက client requests မ်ားလာခဲ့မယ္ဆိုရင္ web server က application server ဆီကို ခ်ိတ္ဆက္ၿပီး HTTP response ၿပဳလုပ္ဖို႔ရန္အတြက္ data မ်ားကို သိမ္းဆည္းပါတယ္။

**Application Server**

Application server မ်ားတြင္ web နဲ႔ EJB container မ်ားပါ၀င္ပါတယ္။သူကိုလည္း servlet, jsp, struts, jsf, ejb စတဲ့ technologies မ်ားနဲ႔ တြဲဖက္သံုးစြဲႏိုင္ပါတယ္။သူက component based product တစ္ခုၿဖစ္ၿပီးေတာ့ server centric architecture တစ္ခုရဲ႕ middle-tier ထဲမွာ အလုပ္ပါတယ္။

သူက State maintenance နဲ႔ security အတြက္ persistence နဲ႔ data access မ်ားနဲ႔အတူ middleware services မ်ားကို ေပးစြမ္းပါတယ္။



ထင္ရွားတဲ့ Application server မ်ားကေတာ့….

* JBoss - JBoss community ရဲ႕ open-source server တစ္ခုၿဖစ္တယ္။
* Glassfish - Oracle ရဲ႕ server ၿဖစ္တယ္။
* Weblogic - Oracle ရဲ႕ server ပဲ ဒါေပမယ့္ သူက ပိုၿပီး secure ၿဖစ္တယ္။
* Websphere - IBM ရဲ႕ server ၿဖစ္တယ္။

**Content Type**

Content type ကို MIME type (Multipurpose internet Mail Extension) လို႔လည္း သိထားၾကတယ္။သူက HTTP Header တစ္ခုၿဖစ္ၿပီးေတာ့ browser ကိုပို႔လႊတ္လိုက္တဲ့ အရာေတြက ဘာေတြလဲဆိုတာကို ေရးထားတဲ့ description မ်ားကို ေပးပါတယ္။

MIME က internet standard တစ္ခုၿဖစ္ၿပီးေတာ့ mail ရဲ႕ extending limited capabilities မ်ားအတြက္ အသံုးၿပဳပါတယ္။ဘယ္လိုနည္းလမ္းေတြနဲ႔လည္းဆိုေတာ့ sounds မ်ား images မ်ားထည့္သြင္းၿခင္း message ထဲတြင္း text မ်ားရိုက္ၿခင္း စတဲ့နည္းလမ္းေတြနဲ႔အသံုးၿပဳပါတယ္။

MIME ရဲ႕ features ေတြကေတာ့…

* non-ASCII characters မ်ားကို support ေပးပါတယ္။
* single message တစ္ခုတည္းမွာ multiple attachment ကို support ေပးပါတယ္။
* attachment ထဲမွာဆိုရင္လည္း executable audio, images နဲ႔ video files မ်ားအစရွိသည္မ်ားကို support ေပးပါတယ္။
* သူက unlimited message length ကို support ေပးပါတယ္။

**List of Content type**

သူ႔မွာ Content type မ်ားစြာရွိတယ္။ေတာ္ေတာ္ေလးကို မ်ားတယ္။အဲ့တာေၾကာင့္မေရးၿပေတာ့ပါ။

**Servlet Api**

 javax.servlet နဲ႔ javax.servlet.http packages မ်ားက servlet api အတြက္ interfaces နဲ႔ classes မ်ားကို ကိုယ္စားၿပဳပါတယ္။

javax.servlet package မွာ interfaces နဲ႔ classes အမ်ားၾကီး ပါ၀င္ပါတယ္။သူတို႔ေတြကို servlet သို႔မဟုတ္ web container က အသံုးၿပဳပါတယ္။သူတို႔မွာ ဘယ္ protocol ၿဖစ္ရမယ္လို႔ တိတိက်က်မရွိဘူး။

javax.servlet.http package မွာဆိုရင္ေတာ့ အေပၚက ေကာင္လိုပဲ interfaces နဲ႔ classes မ်ားစြာပါ၀င္ပါတယ္ ဒါေပမယ့္ သူကေတာ့ http requests မ်ားကိုပဲ တာ၀န္ယူလုပ္ေဆာင္ပါတယ္။

**javax.servlet package မွာရွိတဲ့ interfaces မ်ား**

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

**javax.servlet package မွာရွိတဲ့ classes မ်ား**

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

**javax.servlet.http packages တြင္ပါ၀င္ေသာ interfaces မ်ား**

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

**javax.servlet.http packages တြင္ပါ၀င္ေသာ classes မ်ား**

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

**Servlet Interface**

Servlet interface က method မ်ားကို define လုပ္တယ္ သူ႕ကို servlet အကုန္လံုးက implements လုပ္ရတယ္။

အဲ့မွာ Life cycle method ၃ ခုနဲ႔ non-life cycle method ၂ ခုေပးပါတယ္။ life cycle method ၃ ခုက servlet ကို initialize လုပ္ၿခင္း request မ်ားကို service လုပ္ၿခင္း နဲ႔ servlet ကို destroy လုပ္ၿခင္းတို႔ၿဖစ္ၾကပါတယ္။

**Method of Servlet Interface**

သူတို႔ကို Web container က invoke လုပ္ပါတယ္။

|  |  |
| --- | --- |
| Method | Description |
| **public void init(ServletConfig config)** | Servlet ကို initialize လုပ္တဲ့ method တစ္ခုၿဖစ္ၿပီး web container က သူ႕ကိုတစ္ခါပဲ invoke လုပ္တယ္။ |
| **public void service(ServletRequest request,ServletResponse response)** | Request အတြက္ response ၿပန္ေပးပါတယ္။web container ကို request လာတိုင္းလာတိုင္း အဲ့ method ကိုေခၚၿခင္းးခံရတယ္။ |
| **public void destroy()** | Servlect ကို destroy လုပ္တဲ့ေနရာမွာ သံုးတယ္။သူ႕ကိုလည္း တစ္ၾကိမ္ပဲေခၚတယ္။ |
| **public ServletConfig getServletConfig()** | Servlet Config Object ကို return ၿပန္ေပးပါတယ္။ |
| **public String getServletInfo()** | Servlect informationကို return ၿပန္ေပးပါတယ္ ဘယ္လိုဟာေတြလည္းဆိုေတာ့ writer တို႔ copyright တို႔ version အစရွိသၿဖင့္ေပါ့။ |

**Servlet example by implementing Servlet Interface**

ဒီ class ကို ေရးမယ္ဆိုရင္ သူက ပထမဆံုး ေရးမယ့္ class ၿဖစ္တဲ့အတြက္ေၾကာင့္ သူ႕မွာ project မရွိဘူး အဲ့တာေၾကာင့္ maven project တစ္ခုယူမယ္။servlect ေရးမွာၿဖစ္တဲ့အတြက္ servlect jar file မ်ားလိုအပ္လိမ့္မယ္။ servlet အတြက္ pom.xml မွာ ဘယ္လိုေရးရမလဲဆိုေတာ့…..

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>3.0-alpha-1</version>

<scope>provided</scope>

</dependency>

**public** **class** First **implements** Servlet {

//servletConfig ဆိုတာက javax.servlet package ရဲ႕ interface တစ္ခုၿဖစ္တယ္။

ServletConfig config = **null**;

**public** **void** init(ServletConfig config) {

**this**.config = config;

System.***out***.println("servlet is initialized");

}

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

res.setContentType("text/html");

PrintWriter out = res.getWriter();

out.print("<html><body>");

out.print("<b>hello simple servlet</b>");

out.print("</body></html>");

}

**public** **void** destroy() {

System.***out***.println("servlet is destroyed");

}

**public** ServletConfig getServletConfig() {

**return** config;

}

**public** String getServletInfo() {

**return** "copyright 2007-1010";

}

}

**Generic Servlet**

**Generic Servlet Class**

Generic Servlet Class က Servlet, ServletConfig နဲ႔ Serializable interfaces မ်ားကို implement လုပ္ပါတယ္။ service method ကလြဲလို႔ interfaces မ်ားရဲ႕ method အားလံုးကို implementation မ်ားကို ရရွိတယ္။

Generic Servlet Class က မည္သည့္ request type မ်ိဳးကိုမဆို handle လုပ္ႏိုင္တဲ့အတြက္ေၾကာင့္ သူ႕ကို protocol independent လုိ႔ေခၚႏိုင္တယ္။

မင္းက Generic Servlet Class ကို inheriting လုပ္ၿခင္းနဲ႔ေရာ service method ရဲ႕ implementation ကိုေပးၿခင္းအားၿဖင့္ေရာ generic servlet ကို ဖန္တီးႏိုင္တယ္။

**Methods of Generic Servlet Class**

**public void init(ServletConfig config)** servlet ကို initialize လုပ္ဖို႔အတြက္ သံုးတာ။

**public abstract void service(ServletRequest request, ServletResponse response)** user request ပို႔လိုက္တဲ့အခ်ိန္တိုင္းအလုပ္လုပ္တယ္။

**public void destroy()** servlet ကို destroy လုပ္ဖို႔အတြက္သံုးတယ္။

**public ServletConfig getServletConfig()** servletConfig return ၿပန္ဖို႔အတြက္သံုးတယ္။

**public String getServletInfo()** writer, copyright and version ကဲ့သို႔ေသာ servlet ရဲ႕ information မ်ားကို return ၿပန္ဖို႔ရန္အတြက္ သံုးတယ္။

**public void init()**

**public ServletContext getServletContext()** servletcontext ကို return ၿပန္ဖို႔အတြက္သံုးပါတယ္

**public String getInitParameter(String name)** ေပးလိုက္တဲ့ parameter name အတြက္ parameter value ကို return ၿပန္ေပးလိမ့္မယ္

**public Enumeration getInitParameterNames()** web.xml မွာ သတ္မွတ္ထားတဲ့ parameters values အားလံုးကို return ၿပန္ေပးလိမ့္မယ္

**public String getServletName()** servlet ရဲ႕ name ကို return ၿပန္ေပးလိမ့္မယ္

**public void log(String msg)** servlet log file ထဲမွာ ေပးလိုက္တဲ့ message ကို ေရးလိမ့္မယ္

**public void log(String msg,Throwable t)** servlet log file ထဲမွာ ရွင္းၿပတဲ့ message ကို ေရးလိမ့္မယ္

**public class First extends GenericServlet{**

**public void service(ServletRequest req,ServletResponse res)**

**throws IOException,ServletException{**

**res.setContentType("text/html");**

**PrintWriter out=res.getWriter();**

**out.print("<b>hello generic servlet</b>");**

**out.print("</body></html>");**

**}**

**}**

**HttpServlet**

**HttpServlet Class**

HttpServlet က GenericServlet class ကို depend လုပ္ထားတယ္ ၿပီးေတာ့ Serializable interface ကို implements လုပ္ထားတယ္။

**Method of HttpServlet Class**

|  |
| --- |
| **public void service(ServletRequest req,ServletResponse res)** request ကို protected service method ဆီကို ပို႔လႊတ္လိုက္တယ္ ဘယ္လိုနည္းနဲ႔လည္းဆိုေတာ့ request and response object ေတြကို http type အၿဖစ္ ေၿပာင္းလဲလိုက္ၿခင္းၿဖင့္ ပို႔လႊတ္လိုက္တယ္ |
| **protected void service(HttpServletRequest req, HttpServletResponse res)** service method က request ကိုလက္ခံတယ္ ၿပီးေတာ့ doXXX() method ဆီကို request မ်ားအား incoming http request typeအေပၚမူတည္ၿပီးေတ့ာ ပို႔လႊတ္လိုက္တယ္ |
| **protected void doGet(HttpServletRequest req, HttpServletResponse res) GET request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doPost(HttpServletRequest req, HttpServletResponse res)** **POST request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doHead(HttpServletRequest req, HttpServletResponse res) HEAD request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doOptions(HttpServletRequest req, HttpServletResponse res) OPTIONS request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doPut(HttpServletRequest req, HttpServletResponse res)** **PUT request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doTrace(HttpServletRequest req, HttpServletResponse res) TRACE request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected void doDelete(HttpServletRequest req, HttpServletResponse res) DELETE request ကို handle လုပ္တယ္ ၿပီးေတာ့ web container ကို invoke လုပ္တယ္** |
| **protected long getLastModified(HttpServletRequest req) HttpServletRequest ကို စၿပီး release လုပ္တဲ့ အခ်ိန္ကို ထည့္ေပးလိုက္တယ္(January 1,1970 GMT).** |

**Servlet Life Cycle**

Life Cycle of a Servlet

Servlet instance တစ္ခုရဲ႕ life cycle ကို web container က maintain လုပ္ပါတယ္။

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. Service method is invoked.
5. Destroy method is invoked.



ပံုအရဆိုရင္ servlet တစ္ခုမွာ အဆင့္ ၃ ဆင့္ရွိတယ္(new,ready and end). Servlet instance ကို create လုပ္လိုက္တယ္ဆိုရင္ အဲ့တာကို new state လို႔ေခၚတယ္။init method() ကိုေခၚၿပီးတဲ့ေနာက္မွာ servlet ဟာ ready state ကို ကူးေၿပာင္းသြားတယ္။အဲ့ ready state မွာ servlet က task မ်ားကို လုပ္ေဆာင္တယ္။web container က destroy() method ကို လွမ္းေခၚလိုက္ေသာအခါ end state ကိုေရာက္ရွိသြားတယ္။

**1. Servlet Class is loaded**

Class loader က servlet class ကို load လုပ္ဖို႔အတြက္ တာ၀န္ယူတယ္။ web container က servlet အတြက္ ပထမဆံုး request ေရာက္လာတဲ့အခါ servlet class ေတြကို load ဆြဲတင္တယ္။

**2. Servlet instance is created**

Servlet class ကို load ဆြဲတင္ၿပီးတဲ့ေနာက္ Web container က servlet တစ္ခုရဲ႕ instance ကို create လုပ္လိုက္တယ္။ servlet life cycle မွာ servlet instance ကို တစ္ၾကိမ္တစ္ခါတည္းသာ create လုပ္တယ္။

**3. init() method is invoked**

Servlet instance ကို create လုပ္ၿပီးတဲ့ေနာက္ web container က init method ကို လွမ္းေခၚလိုက္တယ္။ init() method က servlet ကို initialize လုပ္ဖို႔အတြက္ သံုးတယ္။init method က javax.servlet.Servlet interface ရဲ႕ life cycle method တစ္ခုလည္းၿဖစ္တယ္။

**public** **void** init(ServletConfig config) **throws** ServletException

**4. Service method is invoked**

Servlet က request ေတြကို လက္ခံရရွိတဲ့အခါ web container က service method ေတြကို request ေတြလက္ခံရရွိတဲ့အခ်ိန္တိုင္း လွမ္းလွမ္း invoke လုပ္ေနတယ္။ servlet ကို initialized မလုပ္ထားဘူးဆိုရင္

**public** **void** service(ServletRequest request, ServletResponse response)

**throws** ServletException, IOException

**5. Destroy method is invoked**

Service မွ servlet instance ကို remove မလုပ္ခင္ web container က destroy method ကို invoke လုပ္လိုက္တယ္။ အဲ့ method က servlet ကို resource ေတြကို clean up လုပ္ခိုင္းလိုက္တယ္ resource ဆိုတာက ဘယ္လိုဟာမ်ိဳးလည္းဆိုေတာ့ memory တို႔ thread တို႔ကို ရွင္းပစ္တာ ကိုဆိုလိုတယ္္။

**public** **void** destroy()

**Servlet Example**

**Servlet Example တစ္ခုကို create လုပ္ဖို႔အတြက္ step မ်ား**

Servlet example တစ္ခုကို create လုပ္ဖို႔အတြက္ အဆင့္ ၆ ဆင့္ရွိတယ္။Servlet example တစ္ခုကို နည္းလမ္း ၃ မ်ိဳးနဲ႔ ဖန္တီးႏိုင္တယ္။

1. Servlet interface ကို implement လုပ္ၿခင္းၿဖင့္လည္းေကာင္း
2. GenericServlet class ကို inheritance လုပ္ၿခင္းၿဖင့္ေသာ္လည္းေကာင္း
3. HttpServlet class ကို inheritance လုပ္ၿခင္းၿဖင့္ေသာ္လည္းေကာင္း servlet ကို create လုပ္ႏိုင္ပါတယ္။

User ေတြအသံုးအမ်ားဆံုးနည္းလမ္းကေတာ့ HttpServlet class ကို inheritance လုပ္တဲ့နည္းလမ္းပဲၿဖစ္တယ္ ဘာလို႔လဲဆိုေတာ့ သူက doGet(), doPost(), doHead() ကဲ့သို႔ေသာ http request specific method မ်ားကို ထုတ္ေပးႏိုင္လို႔ၿဖစ္တယ္။

အခု Example မွာဆိုရင္ Apache Tomcat Server ကို သံုးပါမယ္။ အဆင့္ ၃ ဆင့္ကေတာ့….

1. Directory structure တစ္ခု create လုပ္ၿခင္း။
2. Servlet တစ္ခု create လုပ္ၿခင္း။
3. Servlet ကို compile လုပ္မယ္။
4. Deployment descriptor တစ္ခုကို create လုပ္မယ္။
5. Server start မယ္ ၿပီးရင္ project ကို deploy လုပ္မယ္။
6. Servlet ကို access လုပ္မယ္။

**1. Directory structure တစ္ခု create လုပ္ၿခင္း**

Directory structure တစ္ခု create လုပ္တယ္ဆိုတာ အဲ့ေနရာမွာ မတူညီတဲ့ file type မ်ားကို ထည့္လိုက္ၿခင္းအားၿဖင့္ web container က information မ်ားကို ရရွိၿပီး client ဆီကို response ၿပန္ပါလိမ့္မယ္။

Sun Microsystem ဆိုတဲ့ အဖြဲ႕အစည္းက unique ၿဖစ္တဲ့ standard တစ္ခုကို သတ္မွတ္ေပးခဲ့တယ္ အဲ့ standard ကို ေရးတဲ့သူေတြအေနနဲ႔ လိုက္နာရမယ္။

Servlet class file ေတြက class folder ထဲမွာ ရွိကို ရွိရမယ္ ၿပီးေတာ့ web.xml ကလည္း WEB-INF ေအာက္မွာ ရွိရမယ္။



2. Servlet တစ္ခုကို create လုပ္ၿခင္း

အေပၚမွာ ေၿပာခဲ့သလိုပဲ နည္းလမ္း ၃ ခုနဲ႔ create လုပ္လို႔ရတယ္။ေအာက္မွာ ေဖာ္ၿပထားတဲ့ example မွာဆိုရင္ httpServlet class ကို extends လုပ္ထားတယ္။ဘာလို႔လဲဆိုေတာ့ doGet() method ရဲ႕ implementation ကို လိုခ်င္လို႔။မွတ္ထားရမွာက get request ဆိုတာက default request တစ္ခုၿဖစ္တယ္။

**import** javax.servlet.http.\*;

**import** javax.servlet.\*;

**import** java.io.\*;

**public** **class** DemoServlet **extends** HttpServlet {

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");// setting the content type

PrintWriter pw = res.getWriter();// get the stream to write the data

//writing html in the stream

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

pw.close();// closing the stream

}

}

**3. Servlet ကို compile လုပ္ၿခင္း**

Servlet ကို compile လုပ္တဲ့အခါ jar file ေတြ လိုအပ္တယ္။server ေပၚမူတည္ၿပီး jar file မ်ားလည္း ကြဲၿပားတယ္။

|  |  |
| --- | --- |
| Jar file | Server |
| servlet-api.jar | Apache Tomcat |
| weblogic.jar | Weblogic |
| javaee.jar | Glassfish |
| javaee.jar | JBoss |

Jar file ကို load လုပ္ဖုိ႔လည္း နည္းလမ္း ၂ မ်ိဳးရွိပါတယ္။

1. Set classpath
2. JRE/lib/ext folder ထဲမွာ jar file ကို paste လုပ္လိုက္တဲ့ နည္းလမ္းၿဖစ္ပါတယ္။

Java file ကိုေတာ့ မည္သည့္ folder မွာမဆိုထည့္လိုက္။ java file ကို compile လုပ္ၿပီးတဲ့ေနာက္မွာ servlet ရဲ႕ class file ကို **WEB-INF/classes** ထဲမွာ paste လုပ္လိုက္ရမယ္။

**4. Deployment descriptor (web.xml)**

Deployment descriptor ဆိုတာ xml file တစ္ခုၿဖစ္တယ္။သူ႕ဆီကေန web container က servlet ရဲ႕ information မ်ားကို ရယူတယ္။ web container က web.xml file မွ information မ်ားကို ယူဖို႔အတြက္ Parser ကို သံုးတယ္။ xml parser ေတြကေတာ့အမ်ားၾကီးပဲ such as SAX, DOM and Pull တို႔ၿဖစ္ၾကတယ္။

<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<servlet-class>DemoServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

**Description of the elements of web.xml file**

Web.xml file ထဲမွာ elements မ်ားစြာရွိႏိုင္တယ္။

**<web-app> application တစ္ခုလံုးကို ကိုယ္စားၿပဳပါတယ္။**

**<servlet> <web-app> ရဲ႕ sub element တစ္ခုၿဖစ္ၿပီးေတာ့ servlet ကိုကို္ယ္စားၿပဳပါတယ္။**

**<servlet-name> <servlet> ရဲ႕ sub element တစ္ခုၿဖစ္ၿပီးေတာ့ servlet name ကို ကိုယ္စားၿပဳပါတယ္။**

**<servlet-class> <servlet> ရဲ႕ sub element တစ္ခုၿဖစ္ၿပီးေတာ့ servlet class ကို ကိုယ္စားၿပဳပါတယ္။**

**<servlet-mapping> <web-app> ရဲ႕ sub element တစ္ခုၿဖစ္ၿပီးေတာ့ servlet ကို map လုပ္ဖို႔ရန္အတြက္ သံုးပါတယ္။**

**<url-pattern> <servlet-mapping> ရဲ႕ sub element တစ္ခုၿဖစ္ၿပီးေတာ့ servlet ကို invoke လုပ္ဖို႔ရန္အတြက္ client side မွာအသံုးၿပဳပါတယ္။**

**Server ကို start လုပ္မယ္ ၿပီးေတာ့ project ကို deploy လုပ္မယ္**

Tomcat server ကို start လုပ္ဖို႔ရန္အတြက္ apache-tomcat/bin folder ေအာက္က startup.bat ေပၚမွာ double click ႏွိပ္လိုက္ပါ။

**Apache tomcat server အတြက္ one time configuration**

1. JAVA\_HOME or JRE\_HOME ကို environment variable မွာထည့္မယ္ (server ကို စဖို႔အတြက္)
2. Tomcat ရဲ႕ port number ကို change လုိက္ပါ ဘာလို႔လဲဆိုေတာ့ ေနာက္ server တစ္ခုက တူညီတဲ့ port မွာ run ေနမွာစိုးလို႔။

**Servlet running manually**

1. Tomcat server ကို start လုပ္ဖို႔ရန္အတြက္ apache-tomcat/bin folder ေအာက္က startup.bat ေပၚမွာ double click ႏွိပ္လိုက္ပါ။

2. Apache tomcat ရဲ႕ port number ကို ဘယ္လိုခ်ိန္းမလဲ

Server.xml file ကို notepad နဲ႔ ဖြင့္လိုက္ပါ။အဲ့ file က ဘယ္ထဲမွာရွိလဲဆိုေတာ့ apache-tomcat-8.0.36\conf ဆိုတဲ့ folder ေအာက္မွာရွိတယ္။ <Connector port="8080" မွာ 8080 ေနရာမွာ 9999 လို႔ change လုိက္ပါ။

**5. Project ကို ဘယ္လို deploy လုပ္မလဲ**

ကိုယ့္ Servlet project ကို copy ကူးၿပီး apache-tomcat-8.0.36\webapps ေအာက္မွာ paste လုပ္လိုက္ပါ။

* သို႔ေသာ္ တစ္ၿခားနည္းလမ္းမ်ားလည္း ရွိပါေသးတယ္။
* context(project) folder ကို webapps directory ထဲသို႔ copy ကူးၿခင္း
* webapps folder ထဲသို႔ war folder ကို copy ကူးၿခင္း
* server ရဲ႕ folder path ကို ေရြးေပးလိုက္ၿခင္း
* server ရဲ႕ war file ကို ေရြးေပးလိုက္ၿခင္း တို႔ ၿဖစ္ၾကပါတယ္။

ဒီမွာေတာ့ ကၽြန္ေတာ္တို႔ ပထမနည္းကို သံုးလိုက္ပါတယ္။

**6. Servlet ကို ဘယ္လို access လုပ္မလဲ**

Browser ကို ဖြင့္လိုက္ၿပီးေတာ့ သူ႕ address bar မွာ ရိုက္ထည့္ရမယ္။ရိုက္ရမယ့္ format ကေတာ့

http://hostname:portno/contextroot/urlpatternofservlet

http://localhost:9999/demo/welcome

ဒါကိုေတာ့ သိပ္မသံုးပါဘူး theory သေဘာအရသာ ၿပသၿခင္းၿဖစ္တယ္။လက္ေတြ႕မွာေတာ့ ide နဲ႔သာ servlet ကိုေရးသားပါတယ္။

**Servlet ဘယ္လို အလုပ္လုပ္သလဲ**

**War file**

ပထမဆံုးအၾကိမ္ Servlet ဆီကို request ေရာက္လာၿပီဆိုရင္ server က လွမ္း check တယ္။ပထမဆံုးအၾကိမ္ servlet ဆီကို request ေရာက္လာၿပီဆိုရင္ server က လွမ္း check လိုက္တယ္။

Yes ၿဖစ္ခဲ့တယ္ဆိုရင္ ေအာက္က ၃ ဆင့္ကို လုပ္ေဆာင္တယ္

1. servlet class ကို load ဆြဲတင္တယ္
2. servlet class ကို instantiate လုပ္တယ္
3. servletConfig object ကို passing လုပ္ၿပီး init method ကို လွမ္းေခၚတယ္။

No ၿဖစ္ခဲ့တယ္ဆိုရင္ေတာ့

1. request and response object မ်ားကို passing လုပ္ၿပီး service method ကို လွမ္းေခၚတယ္။

Servlet ကို remove လုပ္ခ်င္တယ္ဆိုရင္ေတာ့ web container က destroy method ကိုလွမ္းေခၚလိုက္တယ္

ဘယ္လိုအခ်ိန္မ်ိဳးေတြလည္းဆိုေတာ့ Server ကို ရပ္ခ်င္တာတို႔ undeploying project လုပ္တဲ့အခ်ိန္မ်ိဳးၿဖစ္တယ္။

**Web container က servlet request ကို ဘယ္လို handle လုပ္သလဲ**

Web container ဆိုတာက request ေတြကို handle လုပ္ဖို႔တာ၀န္ယူရတယ္။

web.xml ထဲမွာ servlet နဲ႔ request ေတြကို map လုပ္ေပးတယ္

* request တစ္ခုေရာက္လာရင္ အဲ့ request အတြက္ request and response objects မ်ားကို create လုပ္တယ္
* thread ေပၚမွာ service method ကို ေခၚေပးတယ္
* pulic service method က protected service method ကို အတြင္းက်က် လွမ္းေခၚလိုက္တယ္
* protected service method က request ရဲ႕ type အေပၚမူတည္ၿပီးေတာ့ doGet method ကို လွမ္းေခၚတယ္
* doGet method က response ကို ထုတ္လိုက္ၿပီးေတာ့ client ဆီကို ပို႔လႊတ္လိုက္တယ္
* response ကိုပို႔လႊတ္ၿပီးေနာက္မွာေတာ့ web container က request ကို delete လုပ္တယ္ ၿပီးေတာ့ objects ေတြကို response ၿပန္တယ္။ thread ေတြက thread pool မွာရွိရင္ရွိႏိုင္သလို delete လုပ္ရင္လည္း လုပ္ခံရႏိုင္တယ္။အဲ့တာကေတာ့ သူ႕ server ရဲ႕ implementation အေပၚမူတည္တယ္။

**public service method ထဲမွာ ဘာေရးတာလဲ**

public service method က ServletRequest object ကို HttpServletRequest type အၿဖစ္ နဲ႔ ServletResponse object ကို HttpServletResponse type အၿဖစ္ convert လုပ္လိုက္တယ္။ အဲ့ေနာက္မွာေတာ့ အဲ့ objects မ်ားကို passing လုပ္ၿပီးေတာ့service method မ်ားကို လွမ္းေခၚပါတယ္။

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

HttpServletRequest request; HttpServletResponse response;

try {

request = (HttpServletRequest) req;

response = (HttpServletResponse) res;

} catch (ClassCastException e) {

hrow new ServletException("non-HTTP request or response"); }

service (request, response) ;}

**protected service method ထဲမွာ ဘာေရးထားတာလဲ**

protected service method က request က ဘာ type လည္းဆိုတာ လွမ္း check တယ္ တကယ္လို႔ type က get ၿဖစ္ေနတယ္ဆိုရင္ doGet method ကို လွမ္းေခၚလိုက္ၿပီး type က post ၿဖစ္ေနတယ္ဆိုရင္ doPost method ကို လွမ္းေခၚလိုက္တယ္။

protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String method = req.getMethod();

if(method.equals("GET")) {

long lastModified = getLastModified(req);

if(lastModified == -1L) {

doGet(req, resp);

}

....

//rest of the code

}

}

**War file**

war (web archive) file မွာ web project တစ္ခုရဲ႕ file မ်ားပါ၀င္ပါတယ္။ သူ႕ထဲမွာ servlet, xml, jsp, image, html, css, js အစရွိတဲ့ file မ်ားပါ၀င္ပါတယ္။

**war file ဆိုတာဘာလဲ**

war ထဲမွာ ပါ၀င္တာေတြကေတာ့ web application တစ္ခုထဲမွာ ပါ၀င္တဲ့ contents အားလံုးပဲၿဖစ္တယ္။သူက file transferring လုပ္တဲ့ ၾကာခ်ိန္ကို ေလ်ာ့ခ်ေပးပါတယ္။

**Advantage of war**

saves time: war file ဆိုတာက files ေတြအားလံုးကို single unit တစ္ခုအၿဖစ္ ေပါင္းစပ္ထားၿခင္းၿဖစ္တယ္

အဲ့တာေၾကာင့္ သူက အခ်ိန္နည္းနည္းပဲယူတယ္ client ကေန server ဆီကို file ေတြ transferring လုပ္တဲ့အခါမ်ိဳးမွာေပါ့။

**war file ကို ဘယ္လို create လုပ္ၾကမလဲ**

project ကို right click ႏွိပ္ၿပီး export မွာ war ကို ေရြးၿပီး export ထုတ္လိုက္ရံုပဲ။

**war file ကို ဘယ္လို deploy လုပ္မလဲ**။

နည္းလမ္း ၂ မ်ိဳးရွိတယ္။

1. By server console panel
2. Srver ရဲ႕ specific folder ထဲကို war file ကို manually သြားထည့္ၿခင္း

**war file ကို ဘယ္လို extract ေၿဖမလဲ**

သူ႕ရဲ႕ command line ကေတာ့

jar -xvf projectname.war

**web.xml ထဲက welcome-file-list**

web-app ရဲ႕ welcome-file-list element ကို welcome files မ်ားရဲ႕ list တစ္ခုကို သတ္မွတ္ဖို႔ရန္အတြက္ အသံုးၿပဳပါတယ္။သူ႕ရဲ႕ sub element ကေတာ့ welcome-file ၿဖစ္ၿပီးေတာ့ သူ႕ကို welcome file သတ္မွတ္ဖို႔သံုးပါတယ္။

ဘယ္ file name ကိုမွ ေရးမထားဘူးဆိုရင္ေတာ့ server က automatically welcome file တစ္ခုကို လွမ္းေခၚလိုက္တယ္။

server ရဲ႕ default ရွာတဲ့ welcome file ေတြကေတာ့

* + 1. index.html
    2. index.htm
    3. index.jsp တို႔ၿဖစ္ၾကတယ္။

အဲ့တာေတြကိုမွ မေတြ႕ဘူးဆိုရင္ေတာ့ server က 404 error တက္သြားလိမ့္မယ္။



ကိုယ့္က welcom file ကိုလည္း web.xml ထဲမွာ သတ္မွတ္ထားတယ္ ၿပီးေတာ့ default welcome file ေတြလည္း ရွိတယ္ဆိုရင္ေတာ့ ဦးစားေပးတာက welcome-file ကို ဦးစားေပးလိမ့္မယ္။

အကယ္လို႔မွ web.xml ထဲမွာ welcome file ကို မသတ္မွတ္ခဲ့ဘူးဆိုရင္ေတာ့ index.html ကိုအရင္ရွာလိမ့္မယ္ ၿပီးေတာ့ index.htm ကို ရွာမယ္ မေတြ႕ဘူးဆိုေတာ့မွ ေနာက္ဆံုး index.jsp file ကို လိုက္ရွာလိမ့္မယ္။

**web.xml**

<web-app>

....

<welcome-file-list>

<welcome-file>home.html</welcome-file>

<welcome-file>default.html</welcome-file>

</welcome-file-list>

</web-app>

အခု ၿပထားတဲ့ code အရဆိုရင္ေတာ့ home.html နဲ႔ default.html က welcome file ၿဖစ္လာလိမ့္မယ္။

အကယ္လို႔ welcome file ကို သတ္မွတ္ခဲ့မယ္ဆိုရင္ေတာ့ http://localhost:8888/myproject လို႔ေခၚလိုက္တာနဲ႔ ကိုယ့္ရဲ႕ welcome file ကို browser မွာတန္းၿမင္ရလိမ့္မယ္။

**Servlet with IDE**

**Creating Servlet with Eclipse IDE**

1. Create a dynamic web project (project's name format - TechnologyFirstProject eg.ServletFirstProject)



အဲ့တာကို check ခဲ့ဖို႔မေမ့နဲ႔ ေနာက္မို႔ဆို web.xml ပါလာမွာ မဟုတ္ဘူး

2.Create an html file (WebContent folder ေပၚမွာ right click ႏွိပ္ၿပီး html file ယူလိုက္)

<form action=*"Hello"*>

Name:<input type=*"text"* name=*"user"* />

<input type=*"submit"* value=*"go"* />

</form>

1. Create a servlet
2. Start my eclipse tomcat server and deploy project

အဲ့မွာ Servlet ကို create လုပ္ရတာ နည္းနည္းရွဳပ္တဲ့အတြက္ေၾကာင့္မို႔လို႔ detail ေရးၿပထားတယ္။

- Project ကို right click ႏွိပ္ၿပီး new > other > servlet





အဲ့ေကာင္က web.xml ထဲက

**<url-pattern>/Hello</url-pattern>**

တကယ္တန္းနာမည္က Hello ပါ HelloWorld မဟုတ္ပါ



Inherited abstract methods ကိုအမွန္ၿခစ္ၿဖဳတ္လိုက္ရင္ တစ္ၿခား check out မ်ားလည္း dsiable ၿဖစ္သြားလိမ့္မယ္ အစအဆံုးကိုယ့္ဟာကိုေရးလိုက္တာ ပိုေကာင္းမယ္

Servlet ရဲ႕ code

@WebServlet("Hello")

**public** **class** Hello **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String name = request.getParameter("user");

out.print("Hello " + name);

out.close();

}

Web.xml ရဲ႕ code

<web-app အမ်ားၾကီးေရးထားတာေတာ့ရွိတယ္>

<servlet>

<description>This is the description of my J2EE component.</description>

<display-name>This is the display name of my J2EE component</display-name>

<servlet-name>Hello</servlet-name>

Servlet name သူက subfolder ထဲမွာဆိုရင္ path ပါထည့္ေပး(xxx.Hello)

<servlet-class>Hello</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Hello</servlet-name>

<url-pattern>/Hello</url-pattern>

Servlet ကို တစ္ၿခားေနရာေတြက ေခၚတဲ့အခ်ိန္မွာ သတ္မွတ္ေပးလိုက္တဲ့ နာမည္

</servlet-mapping>

<welcome-file-list>

<welcome-file>MyHtml.html</welcome-file>

</welcome-file-list>

</web-app>

Output က





url က ဒီလိုထြက္သြားတယ္

<http://localhost:9090/ServletFisrtProjet/Hello?user=Lyan+Kyaw>

**Servlet Request**

**Servlet Request Interface**

Servlet request ရဲ႕ object တစ္ခုကို servlet ဆီကို client request information မ်ားေပးပို႔ရန္အတြက္ သံုးပါတယ္။အဲ့ information ေတြက ဘာေတြလဲဆိုေတာ့….content type, content length, parameters names နဲ႔ values, header information, attributes စသည္တို႔ၿဖစ္ၾကပါတယ္။

**Methods of Servlet Request Interface**

|  |  |
| --- | --- |
| Method | Description |
| **public String getParameter(String name)** | Param name ထည့္ေပးလိုက္ၿပီး Parameter တစ္ခုရဲ႕ value တစ္ခုကိုလိုခ်င္ရင္ သံုးတယ္။ |
| **public String[] getParameterValues(String name)** | အေပၚက method ကို list နဲ႔ဆြဲထုတ္တဲ့ပံုစံမ်ိဳး။သူ႕ကို multi select list box မွာ သံုးပါတယ္။ |
| **java.util.Enumeration getParameterNames()** | Request parameter names အားလံုးရဲ႕ enumeration ကို return ၿပန္ခ်င္ရင္ သံုးပါတယ္။ |
| **public int getContentLength()** | Request entity data ရဲ႕ size ကို return ၿပန္ေပးတယ္ မသိဘူးဆိုရင္ေတာ့ -1 ကို ၿပန္လိမ့္မယ္။ |
| **public String getCharacterEncoding()** | Request ရဲ႕ input အတြက္ encoding လုပ္ထားတဲ့ character set ကို return ၿပန္ |
| **public String getContentType()** | Null or request entity data ရဲ႕ internet media type ကို return ၿပန္ |
| **public ServletInputStream getInputStream() throws IOException** | Request body ထဲမွာ binary data ကို reading လုပ္ဖို႔ရန္အတြက္ input stream တစ္ခုကို return ၿပန္ေပး |
| **public abstract String getServerName()** | Server ရဲ႕ hostname ကို return ၿပန္ေပးပါတယ္ အဲ့တာက request ကို လက္ခံေပးပါတယ္။ |
| **public int getServerPort()** | Port number ကို return ၿပန္ေပးပါတယ္ |

**RequestDispatcher**

# **RequestDispatcher in Servlet**

RequestDispatcher interface က html , servlet or jsp စေသာ resource တစ္ခုသို႔ request တစ္ခု ေစလႊတ္ၿခင္းကို ၿပဳလုပ္ပါတယ္။ ဒီ interface က တစ္ၿခား resource ရဲ႕ content ပါ၀င္လာေစရန္လည္း အသံုးၿပဳႏိုင္တယ္။ servlet collaboration ရဲ႕ way တစ္ခု ၿဖစ္တယ္။

RequestDispatcher interface ထဲမွာ method ႏွစ္ခု ရွိတယ္။

# **Methods of RequestDispatcher interface**

1. **public void forward(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:** server ေပၚမွာ servlet တစ္ခုကေန အၿခား resource (servlet, JSP page, or HTML file) တစ္ခုဆီကို forwards a request လုပ္ပါတယ္။
2. **public void include(ServletRequest request,ServletResponse response)throws ServletException,java.io.IOException:** response ထဲမွာ resource (servlet, JSP page, or HTML file) ရဲ႕ content ေတြကို ပါ၀င္ေစတယ္။



Second servlet ရဲ႕ response က client ဆီကို ပို႔ခံရတယ္ ဒါေပမယ့္ first servlet ရဲ႕ response ရဲ႕ response ကိုေတာ့ user ဆီမွာ မၿပဘူး။



Second servlet ရဲ႕ response က first servlet ရဲ႕ response ကို client ဆီကို ပို႔လႊတ္တဲ့အခါ တစ္ပါတည္း ပါသြားတယ္ သီးၿခား response ၿခင္းမရွိဘူး။

**RequestDispatcher ကို ဘယ္လိုရႏိုင္မလဲ**

ServletRequest interface ရဲ႕ getRequestDispatcher() method က RequestDispatcher object ကို return ၿပန္ေပးပါတယ္။

Syntax

public RequestDispatcher getRequestDispatcher(String resource);

Example

RequestDispatcher rd=request.getRequestDispatcher("servlet2");

//servlet2 က second servlet ရဲ႕ url-pattern ၿဖစ္တယ္။

rd.forward(request, response);//method က include or forward

**Example of RequestDispatcher interface**

Process flow က user ရိုက္ထည့္လိုက္တဲ့ password ကို စစ္မယ္။ password မွန္ခဲ့ရင္ forward to welcomeServlet လုပ္ၿပီးေတာ့ မွားခဲ့ရင္ error message ၿပလိမ့္မယ္။ ဒီ example မွာ hardcoded information ကို checking လုပ္မယ္။

ဒီ example မွာ file ၄ file ပါမယ္။

1. index.html file: user ဆီက input ရဖို႔
2. Login.java file: servlet တစ္ခုကို process လုပ္ဖို႔နဲ႔ မွန္ခဲ့ရင္ welcome servlet ဆီကို forward လုပ္ဖို႔
3. WelcomeServlet.java file: servlet တစ္ခုၿဖစ္ၿပီးေတာ့ message ၿပဖို႔
4. web.xml file:



**1. login.html**

<form action=*"servlet\_with\_ide/Hello"* method=*"get"*>

Name:<input type=*"text"* name=*"user"* />

<input type=*"submit"* value=*"go"* />

</form>

**2. loginServlet.java**

**public** **class** LoginServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("userName");

String p = request.getParameter("userPass");

**if** (p.equals("servlet")) {

RequestDispatcher rd =request.getRequestDispatcher("welcomeServlet");

// getRequestDispatcher method ရဲ႕ parameter က servlet name ၿဖစ္လို႔ရသလို page name လည္းၿဖစ္လို႔ရတယ္

rd.forward(request, response);

} **else** {

out.print("Sorry UserName or Password Error!");

RequestDispatcher rd = request.getRequestDispatcher("/request\_dispatcher/login.html");

rd.include(request, response);

}

}

**3. WelcomeServlet.java**

**public** **class** WelcomeServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n=request.getParameter("userName");

out.print("Welcome "+n);

}

}

**4. web.xml**

<!-- request\_dispatcher -->

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>request\_dispatcher.LoginServlet</servlet-class>

</servlet>

<servlet>

<servlet-name>WelcomeServlet</servlet-name>

<servlet-class>request\_dispatcher.WelcomeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/loginServlet</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>WelcomeServlet</servlet-name>

<url-pattern>/welcomeServlet</url-pattern>

</servlet-mapping>

အခု Example က forward method မွာ ၿပထားတဲ့ပံုထားက အတုိင္းပဲ request ကလာတယ္ loginServelt ဆိုတဲ့ servlet1 မွာ process လုပ္ၿပီး clinet ကို response မၿပန္ပဲ welcomeServlet ကို forward လိုက္ၿပီးတဲ့ေနာက္မွ welcomeServlet ကမွ response ၿပန္လိုက္တယ္ ဘယ္ကိုၿပန္လဲဆိုေတာ့ loginServlet ကို ၿပန္လိုက္တယ္ reponse ကို loginServlet က သူ႕response ၿပန္ခ်ိန္မွာ တစ္ပါတည္း ၿပန္ေပးလိုက္တာ welcomeServlet က သီးသန္႔မၿပန္ဘူး။

**Send Redirect**

# **SendRedirect in servlet**

HttpServletResponse interface ရဲ႕ **sendRedirect()** method က အၿခား resource ကို redirect response လုပ္ဖို႔အတြက္ သံုးပါတယ္။သူက client side မွာ အလုပ္လုပ္တယ္ ဘာလို႔လဲဆိုေတာ့ တစ္ၿခား request ကိုၿပဳလုပ္ဖို႔ရန္ browser ရဲ႕ url bar ကို သံုးတယ္။ အဲ့တာေၾကာင့္ သူက server ရဲ႕ အတြင္းပိုင္းနဲ႔အၿပင္ပိုင္းမွာ လုပ္ေဆာင္ႏိုင္တယ္။

# **forward() and sendRedirect() method ဘာကြာလဲ**

|  |  |
| --- | --- |
| Forword() method | sendRedirect() method |
| Forward method က server side မွာ အလုပ္လုပ္တယ္ | SendRedirect method က client side မွာအလုပ္လုပ္တယ္ |
| တူညီတဲ့ request နဲ႔ response object ကို အၿခား servlet ကို ပို႔တယ္ | သူက အၿမဲတမ္း request အသစ္တစ္ခုပို႔တယ္ |
| Server မွာပဲ လုပ္ေဆာင္ႏိုင္တယ္ | Server အတြင္းထဲမွာေရာ အၿပင္မွာပါ လုပ္ေဆာင္ႏိုင္တယ္ |
| Example: request.getRequestDispacher("servlet2").forward(request,response); | Example: response.sendRedirect("servlet2"); |

# **Syntax of sendRedirect() method**

public void sendRedirect(String URL)throws IOException;

# **Example of sendRedirect() method**

response.sendRedirect("http://www.javatpoint.com");

# **Full example of sendRedirect method in servlet**

အခု example မွာ google server ဆီကို request တစ္ေၾကာင္းကို redirect လုပ္လိုက္မယ္။

**googleSearcher.html**

<body>

<form action=*"googleSearchServlet2"*>

<input type=*"text"* name=*"name"* />

<input type=*"submit"* value=*"Google Search"* />

</form>

</body>

**GoogleSearchServlet2.java**

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

String name = request.getParameter("name");

response.sendRedirect("https://www.google.co.in/#q=" + name);

}

**Pom.xml**

<servlet>

<servlet-name>GoogleSearchServlet2</servlet-name>

<servlet-class>send\_redirect.GoogleSearchServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>GoogleSearchServlet2</servlet-name>

<url-pattern>/googleSearchServlet2</url-pattern>

</servlet-mapping>

<welcome-file>send\_redirect/googleSearcher.html</welcome-file>

**ServletConfig**

# **ServletConfig Interface**

Servlet တစ္ခုခ်င္းစီတိုင္းအတြက္Web container က servlet object တစ္ခုကို create လုပ္တယ္။web.xml file မွတစ္ဆင့္ configuration information ကို ရရွိရန္ အဲ့ object ကို သံုးႏိုင္တယ္။ web.xml file မွတစ္ဆင့္ configuration information ကို modify လုပ္တယ္ဆိုရင္ ကၽြန္ေတာ္တို႔က servlet ကို change စရာမလိုပါဘူး။ အဲ့တာေၾကာင့္ specific content ကို တစ္ၾကိမ္မွ တစ္ၾကိမ္ modify လုပ္ခဲ့မယ္ဆိုရင္ servletConfig က web application တစ္ခုကို manage လုပ္ဖို႔ ပိုမို လြယ္ကူေစတယ္။

# **ServletConfig ရဲ႕ အားသားခ်က္မ်ား**

Web.xml file ထဲက information ေတြကို modify လုပ္ခဲ့တယ္ဆိုရင္ေတာင္ servlet ကိုၿပင္စရာမလိုဘူး။

Html file နဲ႔ xml file ကို တိုက္ရိုက္ခ်ိတ္ထားတာ servlet ကသီးၿခားရွိေနတာ အဲ့တာေၾကာင့္ servlet ကို ၿပင္စရာမလိုတာ။

# **Methods of ServletConfig interface**

1. public String getInitParameter(String name):Returns the parameter value for the specified parameter name.
2. public Enumeration getInitParameterNames():Returns an enumeration of all the initialization parameter names.
3. public String getServletName():Returns the name of the servlet.
4. public ServletContext getServletContext():Returns an object of ServletContext.

# **servletConfig ရဲ႕ object ကို ဘယ္လိုယူမလဲ**

Servlet interface ရဲ႕ **getServletConfig() method** က servletConfig ရဲ႕ object ကို return ၿပန္ေပးပါတယ္။

# **Syntax of getServletConfig() method**

public ServletConfig getServletConfig();

# **Example of getServletConfig() method**

ServletConfig config=getServletConfig();

Servlet တစ္ခုအတြက္ initialization parameter ကိုေရးတဲ့ syntax

Init-param ဆိုတာက servlet ရဲ႕ sub-element တစ္ခုၿဖစ္တယ္။

<web-app>

<servlet>

<init-param>

<param-name>parametername</param-name>

<param-value>parametervalue</param-value>

</init-param>

</servlet>

</web-app>

# **Example of ServletConfig to get initialization parameter**

အခု Example မွာ web.xml ကေန initialization parameter (**တစ္ခု**)ကို ရယူမယ္ ၿပီးရင္ servlet မွာ အဲ့ information ကို ရိုက္ထုတ္မယ္။

*ServletConfig1.java*

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

ServletConfig config = getServletConfig();

String driver = config.getInitParameter("driver");

out.print("Driver is: " + driver);

out.close();

}

*servletConfig1.html*

<a href="servletConfig1">click here</a>

*Web.xml*

<servlet>

<servlet-name>ServletConfig1</servlet-name>

<servlet-class>servlet\_config.ServletConfig1</servlet-class>

<init-param>

<param-name>driver</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>ServletConfig1</servlet-name>

<url-pattern>/servletConfig1</url-pattern>

</servlet-mapping>

<welcome-file>servlet\_config/servletConfig1.html</welcome-file>

# **Example of ServletConfig to get initialization parameter**

အခု Example မွာ web.xml ကေန initialization parameter (**အကုန္လံုး**)ကို ရယူမယ္ ၿပီးရင္ servlet မွာ အဲ့ information ကို ရိုက္ထုတ္မယ္။

*ServletConfig2.java*

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

ServletConfig config = getServletConfig();

Enumeration<String> e = config.getInitParameterNames();

String str = "";

while (e.hasMoreElements()) {

str = e.nextElement();

out.print("<br>Name: " + str);

out.print(" value: " + config.getInitParameter(str));

}

out.close();

}

*servletConfig2.html*

<a href="servletConfig2">click here</a>

*Web.xml*

<servlet>

<servlet-name>ServletConfig2</servlet-name>

<servlet-class>servlet\_config.ServletConfig2</servlet-class>

<init-param>

<param-name>username</param-name>

<param-value>system</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>ServletConfig2</servlet-name>

<url-pattern>/servletConfig2</url-pattern>

</servlet-mapping>

<welcome-file>servlet\_config/servletConfig2.html</welcome-file>

**ServletContext**

**Servlet Content Interface**

Servlet context ကို web container က project ကို deploy လုပ္ေနခ်ိန္မွာ create လုပ္လိုက္တယ္။ web.xml ထဲက configuration ကို ရရွိဖို႔ရန္ အသံုးၿပဳပါတယ္။ web project တစ္ခုလံုးမွာ servletContext object တစ္ခုသာလွ်င္ ရွိရမယ္။

အကယ္လို႔မ်ား servlet ေတြအမ်ားၾကီးက web.xml file ထဲက information ေတြကို share ၿပီး သံုးတယ္ဆိုရင္ ပိုေတာင္အဆင္ေၿပပါတယ္ ဘာလုိ႔လဲဆိုေတာ့ information ေတြကို **<context-param>** element ကိုသံုးၿပီး ေရးထားလို႔ပါ။

**Servlet Context ရဲ႕ အားသာခ်က္မ်ား**

Easy to maintain- ဘာလို႔လဲဆိုေတာ့ information ေတြကို share တာက web.xml အဲ့ေတာ့ information ေတြကို change မယ္ဆိုရင္ web.xml ကိုပဲ change စရာလိုၿပီး servlet file ေတြကို လိုက္ change ေနစရာမလိုေတာ့ဘူး။

**Servlet Context ကို အသံုးၿပဳပံုမ်ား**

1. ServletContent ရဲ႕ object က servlet နဲ႔ container ၾကားထဲမွာ interface တစ္ခုကို ေပးတယ္။
2. Web.xml file ထဲက configuration information မ်ားကို ယူဖို႔သံုး
3. Web.xml file ထဲက attribute ေတြကို ရယူဖို႔ သို႔ ဖယ္ရွားဖို႔အတြက္ သံုး
4. inter-application communication ကို ေပးဖို႔အတြက္ သံုး



**ServletContext interface ရဲ႕ အသံုးမ်ားေသာ method မ်ား**

**public String getInitParameter(String name): သတ္မွတ္ထားတဲ့ parameter name အတြက္ parameter values ေတြကို return ၿပန္ေပးပါတယ္။**

**public Enumeration getInitParameterNames():** **context's initialization parameters မ်ားရဲ႕ name မ်ားကို return ၿပန္ေပးပါတယ္။**

1. **public void setAttribute(String name,Object object): application scope အတြင္းမွာ give objects ကို set လုပ္တယ္။**
2. **public Object getAttribute(String name): သတ္မွတ္ထားတဲ့ name အတြက္ attribute ကို return ၿပန္ေပးပါတယ္။**
3. **public Enumeration getInitParameterNames(): String objects မ်ားရဲ႕ enumeration တစ္ခုကဲ့သို႔** context's initialization parameters မ်ားရဲ႕ names မ်ားကို return ၿပန္ေပးပါတယ္။
4. **public void removeAttribute(String name): sevlet context မွ given name မ်ားနဲ႔ attribute မ်ားကို return ၿပန္ေပး။**

**ServletContext interface ကို ဘယ္လိုရႏိုင္သလဲ**

1. **ServletConfig interface ရဲ႕ getServletContext() method က ServletContext ရဲ႕ object ကို reutn ၿပန္ေပးလိမ့္မယ္။**
2. **GenericServlet class ရဲ႕ getServletContext() method က ServletContext ရဲ႕ object ကို reutn ၿပန္ေပးလိမ့္မယ္။**

**Syntax of getServletContext() method**

**public ServletContext getServletContext()**

**Example of getServletContext() method**

**ServletContext application=getServletConfig().getServletContext();**

**ServletContext application=getServletContext();**

Context scope ထဲမွာ ရွိတဲ့ initialization parameter ကို ေပးတဲ့ syntax

context-param element , web-app ရဲ႕ sub elements တစ္ခု သူ႕ကို application scope ထဲမွာရွိတဲ့ initialization parameter ကို သတ္မွတ္ဖို႔ရန္ အသံုးၿပဳႏိုင္ပါတယ္။ param-name and param-value တို႔ကေတာ့ context-param ရဲ႕ sub-elements မ်ားၿဖစ္ၾကပါတယ္။ param-name element က param name ကို သတ္မွတ္ၿပီးေတာ့ param-value element ကေတာ့ param value ကို သတ္မွတ္ေပးပါတယ္။

<web-app>

 ......

<context-param>

<param-name>parametername</param-name>

<param-value>parametervalue</param-value>

</context-param>

......

</web-app>

**Example of ServletContext to get the initialization parameter**

ဒီ example မွာဆိုရင္ web.xml မွ initialization parameter မ်ားကို ရယူမယ္ ၿပီးေတာ့ အဲ့တာေတြရဲ႕ value ကို print ထုတ္မယ္။ မွတ္ထားရမွာက servletContext ရဲ႕ object က application scope ကို ကိုယ္စားၿပဳပါတယ္။ အဲ့တာေၾကာင့္ web.xml မွ parameter ရဲ႕ value ကို change ခဲ့မယ္ဆိုရင္ servlet အကုန္လံုးက change ထားတဲ့ value ကိုပဲ ရရွိလိမ့္မယ္။ဆိုလိုတာက servlet ကို လိုက္ change ေနစရာမလိုေတာ့ဘူး။ အဲ့တာေၾကာင့္ web.xml မွာရွိတဲ့ servlet ေတြအတြက္ လိုအပ္တဲ့ common information မ်ားအတြက္ ေတာ္ေတာ္အသံုးတည့္ပါတယ္။

**Html က အေပၚက servletConfig နဲ႔အတူတူပဲ မေရးေတာ့ပါ**

**ServletContext1.java**

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter pw = res.getWriter();

// creating ServletContext object

ServletContext context = getServletContext();

// Getting the value of the initialization parameter and printing it

String driverName = context.getInitParameter("dname");

pw.println("driver name is=" + driverName);

pw.close();

}

**Web.xml**

<servlet>

<servlet-name>ServletContext1</servlet-name>

<servlet-class>servlet\_context.ServletContext1</servlet-class>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<servlet-mapping>

<servlet-name>ServletContext1</servlet-name>

<url-pattern>/servletContext1</url-pattern>

</servlet-mapping>

<welcome-file>servlet\_context/servletContext1.html</welcome-file>

**Example of ServletContext to get all the initialization parameters**

Second example မွာက်ေတာ့ web.xml မွ initialization parameter အကုန္လံုးကို ရယူမယ္ ၿပီးေတာ့ အဲ့လိုရယူဖို႔အတြက္ servlet class ထဲမွာ getInitParameterNames() method ကို သံုးကို သံုးရမယ္။

**ServletContext2.java**

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

res.setContentType("text/html");

PrintWriter out = res.getWriter();

ServletContext context = getServletContext();

Enumeration<String> e = context.getInitParameterNames();

String str = "";

**while** (e.hasMoreElements()) {

str = e.nextElement();

out.print("<br> " + context.getInitParameter(str));

}

}

**Web.xml**

<servlet>

<servlet-name>ServletContext2</servlet-name>

<servlet-class>servlet\_context.ServletContext2</servlet-class>

</servlet>

<context-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</context-param>

<context-param>

<param-name>username</param-name>

<param-value>system</param-value>

</context-param>

<context-param>

<param-name>password</param-name>

<param-value>oracle</param-value>

</context-param>

<servlet-mapping>

<servlet-name>ServletContext2</servlet-name>

<url-pattern>/servletContext2</url-pattern>

</servlet-mapping>

<welcome-file>servlet\_context/servletContext2.html</welcome-file>

**Attribute in Servlet**

**Attribute in servlet**

Attribute in servlet ဆိုတာက object တစ္ခု သူ႕ကို **1.request scope 2.session scope 3.application scope** ဆိုတဲ့ scope ၃ မ်ိဳးထဲမွာ set လုပ္လို႔ရတယ္ get လုပ္လို႔ရတယ္ remove လုပ္လို႔ရတယ္။

servlet programmer ေတြက servlet attribute ကို သံုးၿပီးေတာ့ servlet တစ္ခုကေန အၿခားတစ္ခုသို႔ information မ်ားကို pass လုပ္ႏိုင္တယ္။အဲ့လိုလုပ္ႏိုင္တဲ့အတြက္ေၾကာင့္ class တစ္ခုကေန ေနာက္တစ္ခုသို႔ တူညီတဲ့ object မ်ားကို ထပ္ကာထပ္ကာ reuse လုပ္ကာ အသံုးၿပဳႏိုင္ပါတယ္။

**1. ServletRequest**

**2. HttpSession**

**3. ServletContext interface မ်ားရဲ႕ attribute specific methods**

attribute specific methods မ်ားကို ေအာက္မွာေၿပာထားပါတယ္။

**public void setAttribute(String name,Object object): application scope ထဲမွာ ေပးထားေသာ object ကို ထည့္ပါတယ္။**

**public Object getAttribute(String name): သတ္မွတ္ထားေသာ name အတြက္ attribute ကို return ၿပန္ေပးတယ္။**

**public Enumeration getInitParameterNames():context's initialization parameters မ်ားရဲ႕ names မ်ားကို string objects မ်ားရဲ႕ enumeration တစ္ခုအၿဖစ္ return ၿပန္ေပးပါတယ္။**

**public void removeAttribute(String name): string name ကို parameter ေပးလိုက္တယ္ servlet context မွ attribute ကို remove လုပ္ေပးတယ္။**

**Example of ServletContext to set and get attribute**

ဒီ example မွာ servlet1 ကေန attribute ကို ထည့္ေပးလိုက္တယ္ servlet 2 ကေန ၿပန္ဖမ္းၿပီး output ၿပန္ၿပေပးတယ္။

**AttributeServlet.java**

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) {

**try** {

res.setContentType("text/html");

PrintWriter out = res.getWriter();

ServletContext context = getServletContext();

//attributeServlet ရဲ႕ name က company value က IBM ကိုထည့္ေပးလိုက္တယ္

context.setAttribute("company", "IBM");

out.println("Welcome to first servlet");

//ဒီ code က link ႏွိပ္လိုက္ရင္ servlet ကို ကူးတဲ့ code

out.println("<a href='attributeServlet2'>visit</a>");

out.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**AttributeServlet2.java**

**public** **void** doGet(HttpServletRequest req, HttpServletResponse res) {

**try** {

res.setContentType("text/html");

PrintWriter out = res.getWriter();

ServletContext context = getServletContext();

// attributeServlet2 က attributeServlet1 ရဲ႕ attribute ကို value ကို ရဖို႔ name နဲ႔ လွမ္းေခၚလိုက္တယ္။အဲ့ဆို value ပါလာမယ္

String n = (String) context.getAttribute("company");

out.println("Welcome to " + n);

out.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**Web.xml**

<servlet>

<servlet-name>AttributeServlet</servlet-name>

<servlet-class>attribute.AttributeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AttributeServlet</servlet-name>

<url-pattern>/attributeServlet</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>AttributeServlet2</servlet-name>

<servlet-class>attribute.AttributeServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AttributeServlet2</servlet-name>

<url-pattern>/attributeServlet2</url-pattern>

</servlet-mapping>

<welcome-file>attribute/attribute.html</welcome-file>

**ServletConfig နဲ႔ ServletContext ၾကားက ကြာၿခားခ်က္**

servletConfig က single servlet ကို ရည္ရြယ္တယ္ ဒါေပမယ့္ servletContext က web application တစ္ခုလံုးကို ရည္ရြယ္တယ္။

**Servlet Advance**

Session Tracking

**Session Tracking in Servlet**

Session ဆိုတာက ၾကာခ်ိန္ပဲ။

Session tracking ဆိုတာက user တစ္ေယာက္ရဲ႕ data ေတြကို သိမ္းထားေပးဖို႔ နည္းလမ္းတစ္ခုၿဖစ္တယ္။ servlet ထဲက session management လို႔ သိၾကတယ္။

http protocal က stateless ၿဖစ္တဲ့အတြက္ေၾကာင့္ သူ႕ကို session tracking techniques ကို သံုးၿပီး state ကို ထိန္းသိမ္းေပးဖို႔ လိုအပ္တယ္။ server ဆီကို user request ပို႔လိုက္တိုင္းမွာ server က request ကို new request လို႔ပဲ ဆက္ဆံတယ္။ အဲ့တာေၾကာင့္ ကၽြန္ေတာ္တို႔က user တစ္ေယာက္ခ်င္းဆီကို မွတ္မိေအာင္ state ကို သိမ္းေပးထားဖို႔ လိုအပ္တယ္။



**ဘာေၾကာင့္ session tracking ကို သံုးတာလဲ**

User တစ္ေယာက္ခ်င္းစီတိုင္းကို server က မွတ္မိေစခ်င္လို႔။

Session Tracking Techniques

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

**1) Cookies in Servlet**

**Cookies in Servlet**

Cookie ဆိုတာက information ရဲ႕ ေသးငယ္တဲ့ အစိတ္အပိုင္းၿဖစ္တယ္။ သူက multiple client requests မ်ားၾကားတြင္ တည္ရွိေနတယ္။

Cookie တစ္ခုမွာ နာမည္ရွိတယ္ vale ရွိတယ္ optional attributes ေတြရွိတယ္ ဘယ္လိုဟာေတြလဲဆိုေတာ့ comment, path and domain qualifiers, a maximum age, and a version number.

**Cookie ဘယ္လုိအလုပ္လုပ္လဲ**

Default အားၿဖင့္ server က request တစ္ခုလာတိုင္း request အသစ္လို႔ပဲ စဥ္းစားတယ္။ အခု cookie techniques မွာဆိုရင္ၿဖင့္ servlet ကို response ၿပန္ရာမွာ အဲ့ cookie ကို ထည့္ေပးလိုက္တာ။ အဲ့ေတာ့ သူ႕ကို browser ရဲ႕ cache ထဲမွာ သိမ္းေပးထာေရာ အဲ့ေနာက္မွာေတာ့ user က request ပို႔လိုက္တယ္ဆိုရင္ အဲ့ request မွာ cookie ကို default သြားၿပီး ေပါင္းေပါင္းေပးေနမယ္။ အဲ့လိုနဲ႔ ဒီ user က old user ဆိုတာကို သိေစတယ္။

**Type of Cookie**

1. Non-persistent cookie
2. Persistent cookie

**Non-persistent cookie** : သူက single session အတြက္သာလွ်င္ အဆင္ေၿပတယ္။ user က browser ကို close လုပ္လိုက္ရင္ သူက auto ပ်က္သြားေရာ

**Persistent cookie** : သူက multiple seeion အတြက္အဆင္ေၿပတယ္။ user က browser ကို close လုပ္လိုက္လည္း မပ်က္ဘူး။ user က logout သို႔ sign out လုပ္မွပဲ ပ်က္လိမ့္မယ္။

**Cookies ရဲ႕ အားသာခ်က္မ်ား**

1. State ကို ထိန္းသိမ္းဖို႔အတြက္ အရိုးရွင္းဆံုးနည္းလမ္းၿဖစ္တယ္။
2. Cookie ကို maintain လုပ္တယ္ဆိုတာ client side မွာ လုပ္ၿခင္းၿဖစ္တယ္။

**Cookie ရဲ႕ အားနည္းခ်က္မ်ား**

1. Browser မွာ cookie က disable ၿဖစ္ေနတယ္ဆိုရင္ အဲ့ေကာင္က အလုပ္လုပ္မွာ မဟုတ္ဘူး။
2. Cookie ထဲမွာ textual information သာလွ်င္ ထည့္ေပးလိုက္လို႔ရတယ္။

Note: gmail က login အတြက္ cookie technique ကို သံုးထားတာၿဖစ္တယ္။ cookie ကို သာ disable လုပ္ၾကည့္ gmail ကို သံုးလို႔ရေတာ့မွာ မဟုတ္ဘူး။

**Cookie Class**

**javax.servlet.http.Cookie** class က cookie ကို သံုးဖို႔အတြက္ function ေတြကို ထုတ္ေပးပါတယ္။

**Cookie class ရဲ႕ constructor**

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| Cookie() | Cookie တစ္ခုကို constructs လုပ္တယ္။ |
| Cookie(String name, String value) | Cookie တစ္ခုကို specific ၿဖစ္တဲ့ name နဲ႔ value နဲ႔ constructs လုပ္တယ္။ |

Cookie class ရဲ႕ အသံုး၀င္ေသာ methods မ်ား

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void setMaxAge(int expiry) | Second ေတြအတြင္းမွာ cookie ရဲ႕ maximum ၿဖစ္တဲ့ age ကို set လုပ္တယ္။ |
| public String getName() | Cookie ရဲ႕ name ကို return ၿပန္ေပးတယ္။ အဲ့ cookie ကို create လုပ္ၿပီးကတည္းက name ကို ခ်ိန္းလို႔ မရေတာ့ဘူး။ |
| public String getValue() | Cookie ရဲ႕ value ကို return ၿပန္ေပးတယ္။ |
| public void setName(String name) | Cookie ရဲ႕ name ကို change ေပးတယ္။ |
| public void setValue(String value) | Cookie ရဲ႕ value ကို change ေပးတယ္။ |

**Cookies ကို သံုးၿခင္းအတြက္ အၿခားလိုအပ္ေသာ method မ်ား**

1. **public void addCookie(Cookie ck):** HttpServletResponse interface ရဲ႕ method တစ္ခု သူ႕ကို response object ထဲမွာ cookie ေတြကို ေပါင္းထည့္ဖို႔အတြက္ သံုးပါတယ္။
2. **public Cookie[] getCookies():**HttpServletRequest interface ရဲ႕ method တစ္ခု သူ႕ကို browser မွ cookies ေတြအားလံုး return ၿပန္ဖို႔အတြက္ သံုးတယ္။

**Cookie ကို ဘယ္လို create လုပ္မလဲ**

Cookie ck=**new** Cookie("user","sonoo jaiswal");//creating cookie object

response.addCookie(ck);//adding cookie in the response

**Cookie ကို ဘယ္လို delete လုပ္မလဲ**

သူ႕ကို logout နဲ႔ signout လုပ္တဲ့ေနရာေတြမွာ သံုးတယ္။

Cookie ck=**new** Cookie("user","");//deleting value of cookie

ck.setMaxAge(0);//changing the maximum age to 0 seconds

response.addCookie(ck);//adding cookie in the response

**Cookie ကို ဘယ္လိုရမလဲ**

Cookie ck[]=request.getCookies();

**for**(**int** i=0;i<ck.length;i++){

out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie

}

**Simple Example of Servlet Cookies**

ပထမဆံုး html from တစ္ခုတက္လာၿပီး user name ကိုေတာင္းမယ္ အဲ့ user name ကို cookie ထဲမွာ သိမ္းမယ့္အလုပ္ကို servlet 1 မွာ လုပ္မယ္ ၿပီးရင္ servlet 2 မွာ cookie ထဲမွာ သိမ္းထားတဲ့ username ကို ထုတ္ၿပီး ၿပန္ၿပမယ္။



**Index.html**

<form action=*"cookieInServlet1"* method=*"post"*>

Name:<input type=*"text"* name=*"userName"* /><br />

<input type=*"submit"* value=*"go"* />

</form>

**CookieInServlet1.java**

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("userName");

out.print("Welcome " + n);

Cookie ck = **new** Cookie("uname", n);// creating cookie object

response.addCookie(ck);// adding cookie in the response

out.print("<form action='cookieInServlet2' method='POST'>");

out.print("<input type='submit' value='go'>");

out.print("</form>");

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**CookieInServlet2.java**

**public** **void** doPost(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

Cookie ck[] = request.getCookies();

out.print("Hello " + ck[0].getValue());

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**Web.xml**

<servlet>

<servlet-name>CookieInServlet1</servlet-name>

<servlet-class>cookie\_in\_servlet.CookieInServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CookieInServlet1</servlet-name>

<url-pattern>/cookieInServlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>CookieInServlet2</servlet-name>

<servlet-class>cookie\_in\_servlet.CookieInServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CookieInServlet2</servlet-name>

<url-pattern>/cookieInServlet2</url-pattern>

</servlet-mapping>

<welcome-file>cookie\_in\_servlet/index.html</welcome-file>

**Cookies: Login & Logout**

**Servlet Login and Logout Example using Cookies**

Cookie ဆိုတာက client side မွာ data ကို သိမ္းဆည္းေပးတဲ့ အမ်ိဳးအစားတစ္ခုၿဖစ္တယ္။

အခု example မွာ link သံုးခုပါတယ္ 1.login 2.logout 3.profile ။ user က login မ၀င္ပဲ profile ကို သြားလို႔မရသလို logout လုပ္ၿပီး သြားၿပီရင္လည္း profile ကို သြားလို႔မရရဘူး login ၿပန္၀င္ရမယ္။

ပါ၀င္သာ file မ်ား

1. index.html
2. link.html
3. login.html
4. LoginServlet.java
5. LogoutServlet.java
6. ProfileServlet.java
7. web.xml

**ဒီ example မွာ web.xml မွ servlet ကို html မွ ေခၚပံုေခၚနည္းေၿပာင္းထားပါသည္။**

**index.html**

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Servlet Login Example</title>

</head>

<body>

<h1>Welcome to Login App by Cookie</h1>

<a href=*"cookie\_login\_logout\_profile/login.html"*>Login</a>|

<a href=*"/ServletFirstProject/cookieLogoutServlet"*>Logout</a>|

<a href=*"/ServletFirstProject/cookieProfileServlet"*>Profile</a>

</body>

</html>

**Link.html**

<a href=*"cookie\_login\_logout\_profile/login.html"*>Login</a>

|

<a href=*"/ServletFirstProject/cookieLogoutServlet"*>Logout</a>

|

<a href=*"/ServletFirstProject/cookieProfileServlet"*>Profile</a>

<hr>

**Login.html**

<form action=*"/ServletFirstProject/cookieLoginServlet"* method=*"post"*>

Name:<input type=*"text"* name=*"name"*><br>

Password:<input type=*"password"* name=*"password"*><br>

<input type=*"submit"* value=*"login"*>

</form>

**CookieLoginServlet.java**

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("cookie\_login\_logout\_profile/link.html").include(request, response);

String name = request.getParameter("name");

String password = request.getParameter("password");

**if** (password.equals("admin123")) {

out.print("You are successfully logged in!");

out.print("<br>Welcome, " + name);

Cookie ck = **new** Cookie("name", name);

response.addCookie(ck);

} **else** {

out.print("sorry, username or password error!");

request.getRequestDispatcher("cookie\_login\_logout\_profile/login.html").include(request, response);

}

out.close();

}

**CookieLogoutServlet**

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("cookie\_login\_logout\_profile/link.html").include(request, response);

Cookie ck = **new** Cookie("name", "");

ck.setMaxAge(0);

response.addCookie(ck);

out.print("you are successfully logged out!");

}

**CookieProfileServlet**

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("cookie\_login\_logout\_profile/link.html").include(request, response);

Cookie ck[] = request.getCookies();

**if** (ck != **null**) {

String name = ck[0].getValue();

**if** (!name.equals("") || name != **null**) {

out.print("<b>Welcome to Profile</b>");

out.print("<br>Welcome, " + name);

}

} **else** {

out.print("Please login first");

request.getRequestDispatcher("cookie\_login\_logout\_profile/login.html").include(request, response);

}

out.close();

}

**Web.xml**

<servlet>

<servlet-name>CookieLoginServlet</servlet-name>

<servlet-class>cookie\_login\_logout\_profile.CookieLoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CookieLoginServlet</servlet-name>

<url-pattern>/cookieLoginServlet</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>CookieProfileServlet</servlet-name>

<servlet-class>cookie\_login\_logout\_profile.CookieProfileServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CookieProfileServlet</servlet-name>

<url-pattern>/cookieProfileServlet</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>CookieLogoutServlet</servlet-name>

<servlet-class>cookie\_login\_logout\_profile.CookieLogoutServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>CookieLogoutServlet</servlet-name>

<url-pattern>/cookieLogoutServlet</url-pattern>

</servlet-mapping>

<welcome-file>cookie\_login\_logout\_profile/index.html</welcome-file>

**2) Hidden Form Field**

**2) Hidden Form Field**

User တစ္ေယာက္ရဲ႕ state ကို သိမ္းထားဖို႔အတြက္ a hidden (invisible) textfield တစ္ခုကို အသံုးၿပဳပါတယ္။ ဒီကိစၥမွာဆိုရင္ ဒီ hidden form field မွာ information သိမ္းမယ္ အဲ့ hidden filed ကို တစ္ၿခား servlet ကေနရမယ္။ page ေတြမွာ submit button ကိုႏွိပ္မယ္ ၿပီးေတာ့ browser ေပၚမွာ depends မၿဖစ္ခ်င္ဘူးဆိုရင္ hidden form က ပိုေကာင္းတဲ့ နည္းလမ္းၿဖစ္လိမ့္မယ္။

<input type="hidden" name="uname" value="Vimal Jaiswal">

**Hidden form field ရဲ႕ real application**

Website တစ္ခုရဲ႕ comment form မွာ အသံုးမ်ားတယ္။ ဘယ္လိုမ်ိဳးလဲဆိုေတာ့ hidden field ထဲမွာ page id or page name ကို သိမ္းတယ္ အဲ့တာမွာ page တစ္ခုခ်င္းစီတုိင္းကို uniquely identified လုပ္လို႔ရမယ္။

**Hidden form field ရဲ႕ အားသာခ်က္မ်ား**

1. Cookie ကို ဂရုစိုက္စရာမလိုေတာ့ဘူး disable ၿဖစ္ၿဖစ္ enable ၿဖစ္ၿဖစ္။

**Hidden form field ရဲ႕ အားနည္းခ်က္မ်ား**

1. သူက server side မွာ အလုပ္လုပ္တာ။
2. Page တစ္ခုခ်င္းစီတိုင္းအတြက္ အပို form တစ္ခုအၿမဲလိုေနတယ္။
3. စာသားပဲ သိမ္းလို႔ရမယ္။

**Example of using Hidden Form Field**

ဒီ example မွာဆိုရင္ user ရဲ႕ name ကို hidden textfield ထဲမွာ သိမ္းၿပီး တစ္ၿခား servlet ဆီကို ပို႔ေပးမယ္။



**Index.html**

<form action=*"hidden\_form\_field/HiddenFormField1"*>

Name:<input type=*"text"* name=*"userName"*/><br/>

<input type=*"submit"* value=*"go"*/>

</form>

**HiddenFormField1.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("userName");

out.print("Welcome " + n);

// creating form that have invisible textfield

out.print("<form action='HiddenFormField2'>");

out.print("<input type='hidden' name='uname' value='" + n + "'>");

out.print("<input type='submit' value='go'>");

out.print("</form>");

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**HiddenFormField2.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Getting the value from the hidden field

String n = request.getParameter("uname");

out.print("Hello " + n);

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**Web.xml**

<servlet>

<servlet-name>HiddenFormField1</servlet-name>

<servlet-class>hidden\_form\_field.HiddenFormField1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HiddenFormField1</servlet-name>

<url-pattern>/hidden\_form\_field/HiddenFormField1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>HiddenFormField2</servlet-name>

<servlet-class>hidden\_form\_field.HiddenFormField2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HiddenFormField2</servlet-name>

<url-pattern>/hidden\_form\_field/HiddenFormField2</url-pattern>

</servlet-mapping>

<welcome-file>hidden\_form\_field/index.html</welcome-file>

**3)URL Rewriting**

ကၽြန္ေတာ္တို႔က next servlet သို႔ next resource ရဲ႕ URL ဆီသို႔ token တစ္ခု သို႔ identifier တစ္ခုကို append လုပ္ၾကတယ္။ ကၽြန္ေတာ္တို႔က name/value အတြဲလိုက္ေတြကို အခုလို format သံုးၿပီးေတာ့ ပိုေပးလို႔ရတယ္။

**url?name1=value1&name2=value2&??**

Name နဲ႔ value ၾကားထဲမွာေတာ့ equal sign နဲ႔ၿခားထားၿပီးေတာ့ parameter တစ္ခုနဲ႔ တစ္ခုၾကားထဲမွာေတာ့ & sign နဲ႔ၿခားတယ္။ user ေတြက hyperlink ကို click နွိပ္တဲ့အခါ parameter pairs ေတြက server ဆီကို သြားလိမ့္မယ္။ server တစ္ခုစီကေန getParameter() method ကိုသံုးၿပီးေတာ့ parameter value တစ္ခုကို ရရွိႏိုင္တယ္။

**URL Rewriting ရဲ႕အားသာခ်က္မ်ား**

1. Cookie ကို ဂရုစိုက္စရာမလိုေတာ့ဘူး။
2. Page တစ္ခုခ်င္းစီတိုင္းအတြက္ form အပိုတစ္ခုမလိုေတာ့ဘူး။

**URL Rewriting ရဲ႕အားနည္းခ်က္မ်ား**

1. Link ေတြနဲ႔ပဲ အလုပ္လုပ္တယ္။
2. Text သီးသန္႔ပဲ ပို႔လို႔ရမယ္။

**Example of using URL Rewriting**

ဒီ example မွာ link ကို သံုးၿပီး user ရဲ႕ state ကို maintain လုပ္မယ္။ အဲ့တာရဲ႕ ရည္ရြယ္ခ်က္က user ရဲ႕ name ကို query string ထဲမွာ append လုပ္ခ်င္လို႔ ၿပီးေတာ့ ေနာက္ page ေရာက္ရင္ အဲ့ query string ကေန value ကို ၿပန္ရယူမယ္။

**Index.html**

<form action=*"hidden\_form\_field/HiddenFormField1"*>

Name:<input type=*"text"* name=*"userName"*/><br/>

<input type=*"submit"* value=*"go"*/>

</form>

**UrlRewritingServlet1**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("userName");

out.print("Welcome " + n);

// appending the username in the query string

out.print("<a href='urlRewritingServlet2?uname=" + n + "'>visit</a>");

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**UrlRewritingServlet2**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// getting value from the query string

String n = request.getParameter("uname");

out.print("Hello " + n);

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**Web.xml**

<servlet>

<servlet-name>UrlRewritingServlet1</servlet-name>

<servlet-class>url\_rewriting.UrlRewritingServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>UrlRewritingServlet1</servlet-name>

<url-pattern>/url\_rewriting/urlRewritingServlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>UrlRewritingServlet2</servlet-name>

<servlet-class>url\_rewriting.UrlRewritingServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>UrlRewritingServlet2</servlet-name>

<url-pattern>/url\_rewriting/urlRewritingServlet2</url-pattern>

</servlet-mapping>

<welcome-file>url\_rewriting/index.html</welcome-file>

**Session Tracking**

**HttpSession interface**

ဒီ case မွာဆိုလု႔ိရွိရင္ container က user တစ္ေယာက္ခ်င္းစီအတြက္ session id ကို create လုပ္ေပးတယ္။ container က user ေတြကို identify လုပ္ဖို႔ရန္ အဲ့ id ကို သံုးတယ္။ HttpSession ရဲ႕ object တစ္ခုကို tasks နွစ္ခုလုပ္ဖို႔ရန္ သံုးႏိုင္တယ္။

1. Bind objects
2. Session တစ္ခုရဲ႕ အေၾကာင္းကို view and manipulate လုပ္လို႔ရတယ္။ ဘယ္လိုဟာေတြလဲဆိုေတာ့ session identifier, creation time, and last accessed time လိုဟာမ်ိဳးေတြ။



**HttpSession object ကို ဘယ္လိုရႏိုင္မလဲ**

 HttpServletRequest interface က HttpSession ရဲ႕ object ကို ရရွိရန္ methods ႏွစ္ခုေပးထားပါတယ္။

1. **public HttpSession getSession():** အဲ့ request နဲ႔ သက္ဆိုင္တဲ့ session ကို return ၿပန္ေပးတယ္ သို႔ request မွာ session မရွိခဲ့ဘူးဆိုရင္ အသစ္တစ္ခုကို create လုပ္ေပးတယ္။
2. **public HttpSession getSession(boolean create):** အဲ့ request နဲ႔သက္ဆိုင္တဲ့ current HttpSession ကို return ၿပန္ေပးတယ္ သို႔ တကယ္လို႔ current session မရွိခဲ့ဘူးဆိုရင္ အသစ္ေဆာက္ၿပီး ၿပန္မယ္။

**HttpSession interface ရဲ႕ အသံုးမ်ားေသာ methods မ်ား**

1. **public String getId(): id string ကို return ၿပန္ေပးတယ္။**
2. **public long getCreationTime(): session ကို create လုပ္လိုက္တဲ့ အခ်ိန္ကို milliseconds နဲ႔တိုင္းတာေပးတယ္။**
3. **public long getLastAccessedTime(): အဲ့ session နဲ႔စပ္ဆက္ေနတဲ့ request ေနာက္ဆံုးလာခဲ့တဲ့အခ်ိန္ကို တိုင္းတာေပးတယ္။**
4. **public void invalidate():**Invalidates this session then unbinds any objects bound to it.

**Example of using HttpSession**

Servlet တစ္ခုထဲမွာရွိတဲ့ session တစ္ခုထဲကို attribute တစ္ခု ထည့္မယ္ၿပီးေတာ့ တစ္ၿခား servlet မွ session scope ကေန ၿပန္ယူမယ္။ session scope ထဲမွာ attribute ကိုထည့္ရန္ HttpSession interface ရဲ႕ setAttribute() method ကို သံုးရမယ္ ၿပီးေတာ့ attribute ကို ယူဖို႔ရန္ getAttribute method ကိုသံုးရမယ္။

**Index.html**

<form action=*"http\_session/httpSessionServlet1"*>

Name:<input type=*"text"* name=*"userName"*/><br/>

<input type=*"submit"* value=*"go"*/> </form>

**HttpSessionServlet1.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("userName");

out.print("Welcome " + n);

HttpSession session = request.getSession();

session.setAttribute("uname", n);

out.print("<a href='httpSessionServlet2'>visit</a>");

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**HttpSessionServlet2.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) {

**try** {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession(**false**);

String n = (String) session.getAttribute("uname");

out.print("Hello " + n);

out.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

**Web.xml**

<servlet>

<servlet-name>HttpSessionServlet1</servlet-name>

<servlet-class>http\_session.HttpSessionServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HttpSessionServlet1</servlet-name>

<url-pattern>/http\_session/httpSessionServlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>HttpSessionServlet2</servlet-name>

<servlet-class>http\_session.HttpSessionServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HttpSessionServlet2</servlet-name>

<url-pattern>/http\_session/httpSessionServlet2</url-pattern>

</servlet-mapping>

**Servlet HttpSession Login and Logout Example**

ကၽြန္ေတာ္တို႔က HttpSession instance ေပၚမွာ objects မ်ားကို bind ႏိုင္တယ္။ setAttribute နဲ႔ getAttribute method ေတြကို သံုးမယ္။ ဒီမွာ database ကိုမသံုးပဲ login logout application တစ္ခုကို ေရးပါမယ္။

ဘာသတိထားရမလဲဆိုေတာ့ ပို႔လိုက္တဲ့ request ေတြနဲ႔ web.xml က configure လုပ္ထားတဲ့ url ေတြနဲ႔ကိုက္ေအာင္ သတိထားရမယ္။

**Index.html**

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Servlet Login Example</title>

</head>

<body>

<h1>Login App using HttpSession</h1>

<a href=*"http\_session\_login\_logout\_profile/login.html"*>Login</a>|

<a href=*"http\_session\_login\_logout\_profile/logoutHttpSessionServlet"*>Logout</a>|

<a href=*"http\_session\_login\_logout\_profile/profileHttpSessionServlet"*>Profile</a>

</body>

</html>

**Link.html**

<a href=*"login.html"*>Login</a> |

<a href=*"logoutHttpSessionServlet"*>Logout</a> |

<a href=*"profileHttpSessionServlet"*>Profile</a>

**Login.html**

<form action=*"loginHttpSessionServlet"* method=*"post"*>

Name:<input type=*"text"* name=*"name"*><br>

Password:<input type=*"password"* name=*"password"*><br>

<input type=*"submit"* value=*"login"*>

</form>

**LoginHttpSessionServlet.java**

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("link.html").include(request, response);

String name = request.getParameter("name");

String password = request.getParameter("password");

**if** (password.equals("admin123")) {

out.print("Welcome, " + name);

HttpSession session = request.getSession();

session.setAttribute("name", name);

} **else** {

out.print("Sorry, username or password error!");

request.getRequestDispatcher("login.html").include(request, response);

}

out.close();}

**LogoutHttpSessionServlet.java**

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("link.html").include(request, response);

HttpSession session = request.getSession();

session.invalidate();

out.print("You are successfully logged out!");

out.close();

}

**ProfileHttpSessionServlet.java**

**protected** **void** doGet(HttpServletRequest request, HttpServletResponse response)

**throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

request.getRequestDispatcher("link.html").include(request, response);

HttpSession session = request.getSession(**false**);

**if** (session != **null**) {

String name = (String) session.getAttribute("name");

out.print("Hello, " + name + " Welcome to Profile");

} **else** {

out.print("Please login first");

request.getRequestDispatcher("http\_session\_login\_logout\_profile/login.html").include(request, response);

}

out.close();

}

**Web.xml**

<servlet>

<description></description>

<display-name>LoginHttpSessionServlet</display-name>

<servlet-name>LoginHttpSessionServlet</servlet-name>

<servlet-class>http\_session\_login\_logout\_profile.LoginHttpSessionServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginHttpSessionServlet</servlet-name>

<url-pattern>/http\_session\_login\_logout\_profile/loginHttpSessionServlet</url-pattern>

</servlet-mapping>

<servlet>

<description></description>

<display-name>ProfileHttpSessionServlet</display-name>

<servlet-name>ProfileHttpSessionServlet</servlet-name>

<servlet-class>http\_session\_login\_logout\_profile.ProfileHttpSessionServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ProfileHttpSessionServlet</servlet-name>

<url-pattern>/http\_session\_login\_logout\_profile/profileHttpSessionServlet</url-pattern>

</servlet-mapping>

<servlet>

<description></description>

<display-name>LogoutHttpSessionServlet</display-name>

<servlet-name>LogoutHttpSessionServlet</servlet-name>

<servlet-class>http\_session\_login\_logout\_profile.LogoutHttpSessionServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LogoutHttpSessionServlet</servlet-name>

<url-pattern>/http\_session\_login\_logout\_profile/logoutHttpSessionServlet</url-pattern>

</servlet-mapping>

<welcome-file>http\_session\_login\_logout\_profile/index.html</welcome-file>

**Event and Listener in Servlet**

**Event and Listener in Servlet**

Object တစ္ခုရဲ႕ state changing ၿဖစ္တာကို event လို႔ေခၚတယ္။ ဘယ္လိုဟာမ်ိဳးလဲဆိုေတာ့ exception ေတြတက္ႏိုင္တဲ့ case ေတြမွာပဲ အေရးၾကီးတဲ့ process ကို လုပ္ေဆာင္ႏိုင္ၿခင္းလည္းၿဖစ္တယ္ ဥပမာ counting total and current logged-in users တို႔ project ကို deploy လုပ္တဲ့အခ်ိန္မွာ database မွာ tables ေတြကို create လုပ္တာတို႔ database connection object ကို create လုပ္တာမ်ိဳးတို႔ ၿဖစ္ၾကတယ္။  javax.servlet နဲ႔ javax.servlet.http packages ေတြထဲမွာ many event classes နဲ႔ listener interfaces တို႔ပါ၀င္ၾကတယ္။

**Event classes**

1. ServletRequestEvent
2. ServletContextEvent
3. ServletRequestAttributeEvent
4. ServletContextAttributeEvent
5. HttpSessionEvent
6. HttpSessionBindingEvent

**Event interfaces**

1. ServletRequestListener
2. ServletRequestAttributeListener
3. ServletContextListener
4. ServletContextAttributeListener
5. HttpSessionListener
6. HttpSessionAttributeListener
7. HttpSessionBindingListener
8. HttpSessionActivationListener

**1. ServletContextEvent and ServletContextListener**

Web application တစ္ခုကို server ေပၚမွာ deploy လုပ္တဲ့အခ်ိန္မွာ ServletContextEvent က notify လုပ္ၿခင္းခံရတယ္။ web application ကို deploy လုပ္တဲ့အခ်ိန္မွာ အခ်ိဳ႕ေသာ action ေတြကို လုပ္ေဆာင္ခ်င္တယ္ဆိုရင္ ဘယ္လိုမ်ိဳးလဲဆိုေတာ့ database connection ကို create လုပ္တာတို႔ project မွာရွိတဲ့ table အားလံုးကို create လုပ္တာတို႔ အစရွိသၿဖင့္။အဲ့လိုလုပ္ခဲ့မယ္ဆိုရင္ ServletContextListener interface ကို implement လုပ္ၿပီး သူ႕ရဲ႕

method ေတြကိုု ယူသံုးလို႔ရပါတယ္။

**Constructor of ServletContextEvent class**

ServletContextEvent class မွာ constructor တစ္ခုပဲရွိတယ္။ ServletContext instance လုပ္ၿပီးတဲ့ေနာက္မွာ web container က ServletContextEvent ရဲ႕ instance ကို create လုပ္တယ္။

public ServletContext getServletContext(): returns the instance of ServletContext.

**Method of ServletContextEvent class**

ServletContextListener interface မွာ method ႏွစ္ခုရွိတယ္။ database connection create လုပ္တဲ့ action လိုဟာမ်ိဳးေတြမွာ သူ႕ကို implement မလုပ္လို႔ိကို မရဘူး။

public void contextInitialized(ServletContextEvent e): is invoked when application is deployed on the server.

public void contextDestroyed(ServletContextEvent e): is invoked when application is undeployed from the server.

**Example of ServletContextEvent and ServletContextListener**

Emp32 table ထဲကေန data ကို retrieve လုပ္ပါမယ္။ အဲ့တာကို လုပ္ေဆာင္ဖို႔အတြက္ connection object ကို listener class ထဲမွာ create လုပ္ရမယ္ ၿပီးေတာ့ အဲ့ connection object ကို servlet ထဲမွာ သံုးမယ္။(တကယ္မစမ္းဘူး)

**MyListener.java**

**public** **void** contextInitialized(ServletContextEvent event) {

**try** {

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "oracle");

// storing connection object as an attribute in ServletContext

ServletContext ctx = event.getServletContext();

ctx.setAttribute("mycon", con);

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **void** contextDestroyed(ServletContextEvent arg0) {}

**FetchData.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

**try** {

// Retrieving connection object from ServletContext object

ServletContext ctx = getServletContext();

Connection con = (Connection) ctx.getAttribute("mycon");

// retieving data from emp32 table

PreparedStatement ps = con.prepareStatement("select \* from emp32", ResultSet.***TYPE\_SCROLL\_SENSITIVE***,

ResultSet.***CONCUR\_UPDATABLE***);

ResultSet rs = ps.executeQuery();

**while** (rs.next()) {

out.print("<br>" + rs.getString(1) + " " + rs.getString(2));

}

con.close();

} **catch** (Exception e) {

e.printStackTrace();

}

out.close();

}

**Index.html**

<a href=*"servlet1"*>fetch records</a>

**Example of ServletContextListener to create table of a project**

ဒီ example ကေတာ့ project ထဲမွာ table ေဆာက္တာေပါ့။ဒါေပမယ့္ example ၿပၿခင္းသာၿဖစ္တယ္ တကယ္ေတာ့ manually လိုက္ေဆာက္စရာမလိုဘူး။တကယ္တမ္းလဲ မစမ္းခဲ့ဘူး။

**public** **void** contextInitialized(ServletContextEvent arg0) {

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

String query="create table emp32(id number(10),name varchar2(40))";

PreparedStatement ps=con.prepareStatement(query);

ps.executeUpdate();

System.***out***.println(query);

}**catch**(Exception e){e.printStackTrace();}

}

**public** **void** contextDestroyed(ServletContextEvent arg0) {

System.***out***.println("project undeployed");

}

**2. HttpSessionEvent and HttpSessionListener**

Session object ကို change လိုက္တဲ့အခါ HttpSessionEvent ကို notify လုပ္ၿခင္းခံရတယ္။အဲ့ event အတြက္ သက္ဆိုင္တဲ့ interface ကေတာ့ HttpSessionListener ၿဖစ္ပါတယ္။ အဲ့ event မွာ ကၽြန္္ေတာ္တို႔က တစ္ခ်ိဳ႕ operations ေတြကို လုပ္ေဆာင္ႏိုင္ပါတယ္ ဘယ္လိုဟာမ်ိဳးလဲဆိုေတာ့ counting total နဲ႔ current logged-in users

user details ရဲ႕ log တစ္ခုကို maintaing ဘယ္လိုဟာမ်ိဳးလဲဆိုေတာ့ login time နဲ႔ logout time လိုဟာမ်ိဳး။

**HttpSessionListener interface ရဲ႕ methods မ်ား**

HttpSessionListener interface ထဲမွာ methods ႏွစ္ခုရွိတယ္။

1. **public void sessionCreated(HttpSessionEvent e)**: is invoked when session object is created.
2. **public void sessionDestroyed(ServletContextEvent e)**: is invoked when session is invalidated.

**Example of HttpSessionEvent and HttpSessionListener to count total and current logged-in users**

1. Index.html: user ဆီကေန input ကိုရရန္
2. MyListener.java: ဒီ listener က total ကို count လုပ္တယ္ current logged-in users လုပ္တယ္ servletContext object ထဲမွာ အဲ့ information ကို attribute အေနနဲ႔ သိမ္းဆည္းမယ္။
3. First.java: servlet class တစ္ခု သူက session ကို create လုပ္တယ္ ၿပီးေတာ့ total ကို print လုပ္တယ္ ၿပီးေတာ့ current logged-in users လုပ္တယ္။
4. Logout.java: servlet class တစ္ခု သူက session validate လုပ္တယ္။

*(သတိထားရမွာက listener class ကို web.xml ထဲမွာ ေၾကညာေပးရမယ္)*

**HttpSessionLoginLogoutEventServlet1.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String n = request.getParameter("username");

out.print("Welcome " + n);

HttpSession session = request.getSession();

session.setAttribute("uname", n);

// ServletContext object က data ကို retrieving လုပ္လိုက္တယ္

ServletContext ctx = getServletContext();

**int** t = (Integer) ctx.getAttribute("totalusers");

**int** c = (Integer) ctx.getAttribute("currentusers");

out.print("<br>total users= " + t);

out.print("<br>current users= " + c);

out.print("<br><a href='httpSessionLoginLogoutEventServlet2'>logout</a>");

out.close();

}

**HttpSessionLoginLogoutEventServlet2.java**

**public** **void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession(**false**);

session.invalidate();// invalidating session

out.print("You are successfully logged out");

out.close();

}

**HttpSessionLoginLogoutListener.java**

**public** **class** HttpSessionLoginLogoutListener **implements** HttpSessionListener {

ServletContext ctx = **null**;

**static** **int** *total* = 0, *current* = 0;

**public** **void** sessionCreated(HttpSessionEvent e) {

*total*++;

*current*++;

ctx = e.getSession().getServletContext();

ctx.setAttribute("totalusers", *total*);

ctx.setAttribute("currentusers", *current*);

}

**public** **void** sessionDestroyed(HttpSessionEvent e) {

*current*--;

ctx.setAttribute("currentusers", *current*);

}

}

**Index.html**

<form action=*"http\_session\_event\_listener/httpSessionLoginLogoutEventServlet1"*>

Name:<input type=*"text"* name=*"username"*><br>

Password:<input type=*"password"* name=*"userpass"*><br>

<input type=*"submit"* value=*"login"*/>

</form>

Web.xml

<servlet>

<description></description>

<servlet-name>HttpSessionLoginLogoutEventServlet1</servlet-name>

<servlet-class>http\_session\_event\_listener.HttpSessionLoginLogoutEventServlet1</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HttpSessionLoginLogoutEventServlet1</servlet-name>

<url-pattern>/http\_session\_event\_listener/httpSessionLoginLogoutEventServlet1</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name>HttpSessionLoginLogoutEventServlet2</servlet-name>

<servlet-class>http\_session\_event\_listener.HttpSessionLoginLogoutEventServlet2</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HttpSessionLoginLogoutEventServlet2</servlet-name>

<url-pattern>/http\_session\_event\_listener/httpSessionLoginLogoutEventServlet2</url-pattern>

</servlet-mapping>

<listener>

<listener-class>http\_session\_event\_listener.HttpSessionLoginLogoutListener</listener-class>

</listener>

***(Servlet Events and Listeners ထဲမွာရွိတဲ့ တစ္ၿခား topic ေတြက သီးသန္႔example ေတြမရွိတဲ့အတြက္ ေက်ာ္လိုက္တယ္)***

**Filter**

**Servlet Filter**

Filter ဆိုတာ object တစ္ခုၿဖစ္ၿပီးေတာ့ request တစ္ခုရဲ႕ preprocessing and postprocessing မွာ သူ႕ုကို invoke လုပ္တယ္။ သူ႕ကို filter task ေတြအမ်ားဆံုးအသံုးၿပဳတယ္ ဘယ္လိုဟာေတြလဲဆိုေတာ့ conversion, logging, compression, encryption and decryption, input validation လို ဟာမ်ိဳးေတြ။ servlet filter က pluggable ၿဖစ္တယ္ ဘယ္လိုလဲဆိုေတာ့ သူ႕ကို web.xml ထဲမွာ configure ခ်ထားတာ အဲ့တာေၾကာင့္ သူ႕ကို ၿဖဳတ္ခ်င္ရင္ web.xml ထဲမွာ ၿဖဳတ္လိုက္ရံုပဲ servlet ကို ထိစရာမလိုဘူး။



***Note: Servlet နဲ႔မတူတာက filter တစ္ခုက တစ္ၿခား filter တစ္ခုကို dependency မရွိဘူး။***

**Filter ကုိ အသံုးၿပဳပံု**

* ၀င္လာတဲ့ Request ေတြအကုန္လံုးကို record လုပ္ထားတယ္
* Request ကို ထုတ္လႊတ္လိုက္တဲ့ computers ရဲ႕ ip address ကို logs ရိုက္ထားလို႔ရတယ္
* Conversion ေၿပာင္းလဲၿခင္းမ်ား
* Data compression
* Encryption and decryption
* Input validation

**Filter ရဲ႕အားသာခ်က္မ်ား**

* Filter က pluggable ၿဖစ္တယ္။
* Filter တစ္ခုက တစ္ၿခား resource တစ္ခုမွာ dependency မရွိဘူး
* Maintenance လုပ္ရတာ သက္သာတယ္

**Filter API**

Servlet လိုပဲ filter မွာလည္း သူ႕ကိုယ္ပိုင္ API တစ္ခုရွိတယ္။ javax.servlet package မွာ Filter API ရဲ႕ interfaces ၃ ခု ပါ၀င္တယ္။

1. Filter
2. FilterChain
3. FilterConfig

**1) Filter interface**

Filter တစ္ခုကို create လုပ္မယ္ဆိုရင္ မင္းက filter interface ကို implements လုပ္ကို လုပ္ရမယ္။ Filter interface က filter တစ္ခုအတြက္ life cycle methods ေတြကို ထုတ္ေပးပါတယ္။

|  |  |
| --- | --- |
| Method | Description |
| public void init(FilterConfig config) | တစ္ခါပဲေခၚမယ္ သူက filter ကို initialize လုပ္ဖို႔ရန္သံုးပါတယ္။ |
| public void doFilter(HttpServletRequest request,HttpServletResponse response, FilterChain chain) | User က resource ဆီကို request ပို႔လိုက္တဲ့အခါတိုင္း doFilter() method ကုိ လွမ္းလွမ္းေခၚတယ္။ filter က အဲ့ method နဲ႔ mapped လုပ္ထားတယ္။ filtering tasks မ်ားကို လုပ္ေဆာင္ရန္သူ႕ကို သံုးပါတယ္။ |
| public void destroy() | Service မွာ filter ကို ၿဖဳတ္ခ်င္ရင္ သူ႕ကို တစ္ခါေခၚတယ္။ |

**2) FilterChain interface**

FilterChain ရဲ႕ object က chain ထဲမွာရွိတဲ့ ေနာက္ filter or resource ကို invoke လုပ္ရင္ တာ၀န္ယူလုပ္ေဆာင္ပါတယ္။ အဲ့ object က Filter interface ရဲ႕ doFilter method ကို ၿဖတ္သန္းသြားတယ္။ FilterChain interface မွာ method တစ္ခုပဲ ပါ၀င္ပါတယ္။

1. public void doFilter(HttpServletRequest request, HttpServletResponse response): သူက ေနာက္ filter သို႔ resource ကို ၿဖတ္သန္းသြားတယ္။

**Filter ကို သတ္မွတ္ႏိုင္သလဲ**

Filter create လုပ္တာကလည္း servlet နဲ႔ အတူတူပဲ elements of filter နဲ႔ filter-mapping ဆိုၿပီး ပါမယ္။

<web-app>

<filter>

<filter-name>...</filter-name>

<filter-class>...</filter-class>

</filter>

<filter-mapping>

<filter-name>...</filter-name>

<url-pattern>...</url-pattern>

</filter-mapping>

</web-app>

**Filter ရဲ႕ example**

Information ကို ရိုးရိုးရွင္းရွင္း display ၿပမယ္ request ရဲ႕ post processing လုပ္ၿပီးတဲ့အခ်ိန္က်ရင္ filter ကို auto invoked လုပ္မယ္။

**index.html**

<a href="simpleFilterServlet">click here</a>

**SimpleFilter.java**

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)

throws IOException, ServletException {

PrintWriter out = resp.getWriter();

out.print("filter is invoked before");

chain.doFilter(req, resp);// servlet ဆီကိုသြားတယ္

//servlet ဆီကေနၿပန္လာတယ္

out.print("filter is invoked after");

}

public void destroy() {

}

**SimpleFilterServlet.java**

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("<br>welcome to servlet<br>");

}

**Web.xml**

<servlet>

<servlet-name>SimpleFilterServlet</servlet-name>

<servlet-class>filter.simple\_filter.SimpleFilterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>SimpleFilterServlet</servlet-name>

<url-pattern>/simpleFilterServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>SimpleFilter</filter-name>

<filter-class>filter.simple\_filter.SimpleFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>SimpleFilter</filter-name>

<url-pattern>/simpleFilterServlet</url-pattern>

</filter-mapping>

<welcome-file>filter/simple\_filter/index.html</welcome-file>

Note: fiter ရဲ႕ url-pattern နဲ႔ servlet ရဲ႕ url-pattern တူကိုတူရမယ္။

**Authentication Filter**

Filter ထဲမွာ authentication ကို လုပ္လို႔ရတယ္။ filter class ထဲမွာ user ရိုက္ထည့္လိုက္တဲ့ password ကို စစ္ေဆးႏိုင္တယ္။ မွန္ကန္တဲ့ password ၿဖစ္ခဲ့မယ္ဆိုရင္ေတာ့ request ကို သက္ဆိုင္ရာ servlet ဆီသို႔ forword လုပ္လိုက္မယ္ မွားခဲ့မယ္ဆိုရင္ေတာ့ error message ၿပမယ္။

**Filter ကို သံုးၿပီးေတာ့ user ကို authenticating စစ္တဲ့ Example**

**Index.html**

<form action="authenticationFilterServlet">

Name:<input type="text" name="name"/><br/>

Password:<input type="password" name="password"/><br/>

<input type="submit" value="login" />

</form>

**AuthenticationFilter.java**

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)

throws IOException, ServletException {

PrintWriter out = resp.getWriter();

String password = req.getParameter("password");

if (password.equals("admin")) {

chain.doFilter(req, resp);// sends request to next resource

} else {

out.print("username or password error!" );

//ဒီ code က သူ႕ website မွာက်န္ခဲ့တယ္ အဲ့တာ မပါရင္ html ကို browser က ui အေနနဲ႔ မၿပႏိုင္ေတာ့ဘူး

resp.setContentType("text/html");

req.getRequestDispatcher("filter/authentication\_filter/index.html").include(req, resp);

}

}

public void destroy() {}

**AuthenticationFilterServlet.java**

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("welcome ADMIN");

out.close();

}

**Web.xml**

<servlet>

<servlet-name>AuthenticationFilterServlet</servlet-name>

<servlet-class>filter.authentication\_filter.AuthenticationFilterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>AuthenticationFilterServlet</servlet-name>

<url-pattern>/authenticationFilterServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>AuthenticationFilter</filter-name>

<filter-class>filter.authentication\_filter.AuthenticationFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>AuthenticationFilter</filter-name>

<url-pattern>/authenticationFilterServlet</url-pattern>

</filter-mapping>

<welcome-file>filter/authentication\_filter/index.html</welcome-file>

**FilterConfig**

Web container က FilterConfig object တစ္ခုကို create လုပ္တယ္။ အဲ့ object က web.xml file က configuration information ေတြကို ရယူဖို႔ရန္ အသံုးၿပဳတယ္။

**FilterConfig interface ရဲ႕ methods မ်ား**

1. public void init(FilterConfig config): filter ကို initialize လုပ္ရန္ဒီ method ကို သံုးတယ္။
2. public String getInitParameter(String parameterName): parameter name ေပးလိုက္ရင္ parameter value ကို return ၿပန္ေပးလိမ့္မယ္။
3. public java.util.Enumeration getInitParameterNames():parameter names အားလံုးပါ၀င္တဲ့ enumeration တစ္ခုကို return ၿပန္ေပးတယ္။
4. public ServletContext getServletContext():ServletContext object ကို return ၿပန္ေပးတယ္။

**FilterConfig ရဲ႕ example**

**Index.html**

<a href="filterConfigServlet">click here</a>

**FilterConfigFliter.java**

public void init(FilterConfig config) throws ServletException {

this.config = config;

}

public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws IOException, ServletException {

PrintWriter out = resp.getWriter();

//parameter ေပးလိုက္ရင္ အဲ့ parameter နဲ႔တူတဲ့ param element ကို web.xml မွာရွာလိမ့္မယ္

String s = config.getInitParameter("construction");

if (s.equals("yes")) {

out.print("This page is under construction");

} else {

chain.doFilter(req, resp);// sends request to next resource

}

}

public void destroy() {}

**FilterConfigServlet.java**

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("<br>welcome to servlet<br>");}

**web.xml**

<servlet>

<servlet-name>FilterConfigServlet</servlet-name>

<servlet-class>filter.filter\_config.FilterConfigServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>FilterConfigServlet</servlet-name>

<url-pattern>/filterConfigServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>FilterConfigFliter</filter-name>

<filter-class>filter.filter\_config.FilterConfigFliter</filter-class>

<init-param>

<param-name>construction</param-name>

<param-value>no</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>FilterConfigFliter</filter-name>

<url-pattern>/filterConfigServlet</url-pattern>

</filter-mapping>

<welcome-file>filter/filter\_config/index.html</welcome-file>

**Useful Filter Examples**

**Filter ကေနရိုးရိုး response တစ္ခုပဲၿပန္တဲ့ example**

**responseByFilterOnly.html**

<a href="responseByFilterOnlyServlet">click here</a>

**ResponseByFilterOnly.java**

public void init(FilterConfig arg0) throws ServletException {}

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

PrintWriter out = res.getWriter();

out.print("<br/>this site is underconstruction..");

out.close();

}

public void destroy() {}

**ResponseByFilterOnlyServlet.java**

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.print("<br>welcome to servlet<br>");

}

**Web.xml**

<servlet>

<servlet-name>ResponseByFilterOnlyServlet</servlet-name>

<servlet-class>filter.useful\_example.ResponseByFilterOnlyServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ResponseByFilterOnlyServlet</servlet-name>

<url-pattern>/responseByFilterOnlyServlet</url-pattern>

</servlet-mapping>

<filter>

<filter-name>ResponseByFilterOnly</filter-name>

<filter-class>filter.useful\_example.ResponseByFilterOnly</filter-class>

</filter>

<filter-mapping>

<filter-name>ResponseByFilterOnly</filter-name>

<url-pattern>/responseByFilterOnlyServlet</url-pattern>

</filter-mapping>

<welcome-file>filter/useful\_example/responseByFilterOnly.html</welcome-file>

Page တစ္ခုထဲကို ၀င္ၾကည့္တဲ့ လူအေရအတြက္ကို count လုပ္တဲ့ fiter

countByUser.html