**Ներածություն։ Ինչ է սերվլետը, JSP-ն և դրանց դերը web app-ում։**

Գրաֆիկական ինտերֆեյսով web application-ները վաղուց դադարել են ստատիկ լինելուց, իսկ դրանց դինմիկությունը ապահովելու համար պետք են համապատասխան միջոցներ։ Դինամիկության ապահովման միջոցներից են servlet-ների և JSP-ների օգտագործումը։

Servlet-ների կոնտեքստում կարևոր է web client-ի և web server-ի իրար հետ փոխազդեցության մեխանիզմը։ Web server ասելով հասկանում ենք որևէ ռեսուրս որը ունի մեքենայական մաս և դրա վրա աշխատող web server application` ծրագրային մաս, որ պատասխանատու է web client -ից request-ների ընդունման և respons-ների վերադարձման համար։

Web client – ը միջնորդ է user-ի ու server-ի միջև։ Որպես web client դիտարկում ենք բրաուզերը։ Բրաուզերը սերվերի հետ փոխազդեցությունից բացի ունի նաև այլ կարևոր ֆունկցի որը HTML կոդի ինտերպրետացիան է։ Web client-ի server-ի միջև շփումը մեծամասամբ տեղի է ունենում HTTP պրոտոկոլի միջոցով։ Բրաուզերը մշակում է server-ից ստացված response-ում պարունակվող տվյալները և վիզուալիզացնում է դրանք։

Բայց web server application – ը վերադարձնում է միայն ստատիկ տվյալներ, դինամիկության ապահովման համար անհրաժեշտ են օժանդակ application-ներ որոնց դերը կատարում են servlet-ները CGI (common gateway interface)։

Servlet-ը իրենից ներկայացնում է հատուկ տիպի Java class որը մշակում է request-ներ և վերադարձնում response` մշակման արդյունքը: Servlet-ի կլասը հանդիսանում է HttpServlet-ի ժառանգ։ Request-ների մշակման համար HttpServlet-ը ունի protected մեթոդներ։

doGet, doPost, doPut, doDelete, doHead։ Մեթոդները նախատեսված են համապատասխան տիպի http request-ների մշակման համար: Բոլոր մեթոդները որպես պարրամետրեր ստանում են 2 օբյեկտ՝ HttpServletRequest – պահում է request-ի մասին տեղեկատվություն, HttpServletResponse – կառավարում է response-ը։ Մեթոդներից ամենահաճախ օտագործվողները՝ doGet doPost մեթոդները են։

CGI servlet-ներին request-ների հղման որոշումներ կայացնող հատվածն է։

Servlet-ներով աշխատանքի ժամանակ HTML էջերի գեներացման համար օգտագործվում են JSP ֆայլերը։ JSP-ն java server page – հապավումն է։ JSP-ն ունի HTML-ի տեսք այն տարբերույամբ որ JSP-ում կարող ենք գրել Java լեզվով կոդ։ JSP-ի նպատակն է HTMl-ի Java-ինտեգրացիան։

**Container**

Java-ական լեզվով ասած servlet-ները չունեն main() մեթոդ։ Դրանք գտնվում են մեկ այլ application-ի կառավարման տակ որը կոչվում է կոնտեյներ։ Կոնտեյները իր վրա է վերցնում servlet-ի էկզեմպլյարի ստեղծման, ստացված request-ի մշակման համար առանձին thread-ի ստեղծման, servlet-ի doPost կամ doGet մեթոդների կանչի, դրա կյանքի ցիկլի կառավարման և այլ գործառույթներ։ Կոնտեյների օրինակ է Tomcat-ը: Երբ web server application-ը ստանում է request-ը այն փոխանցվում է ոչ թե միանգամից servlet-ին այլ փոխանցվում է կոնտեյներին որի մեջ և գտնվում է servlet-ը։

Կոնտեյերը՝

* Ապահովում է կապը web application – ի և servlet – ների միջև։
* Յուրաքանչյուր request-ի համար ստեղծում է առանձին thread, իսկ request-ի մշակման ավարտից հետոց thread-ը փակվում է։
* Կառավարում է servlet-ի կյանքի ցիկլերը, վերահսկում է servlet-ի էկզեմպլյարների ստեղծումը և ոչնչացումը։

Web application – ի կոնֆիգուրացիայի մասին տեղեկատվությունը պահվում է web.xml – ֆայլում։ Այս ֆայլի առկայությունը պարտադիր չէ բայց այն լայն կիրառությւն ունի կոնֆիգուրացիայի համար Ֆայլը պետք է գտնվի web-inf folder-ում։ Աշխատանքը սկսելիս կոնտեյները կարդում է web.xml ֆայլը և օգտագործում դրանում նշված config-ները։ web.xml – ը ունի էլեմենտներ որոնց միջոցով կարելի է սահմանել servlet-ների և դրանց կանչի url-ների միջև կապը։

<servlet>

<servlet-name></ servlet-name >

<servlet-class></ servlet-class >

</servlet>

<servlet-mapping>

<servlet-name></ servlet-name >

<url-pattern></ url-pattern>

</servlet-mapping>

Նշվում են url-ը որը հայտնի է web client-ին։ Deployment-ի անունը որը չպետք է համընկնի url-ի կամ սերվլետի ֆայլի անվան հետ և սերվլետի կլասսի անունը: Նշվում են 2 սկրիպտով՝ 1-ը client-ին հայտնի url-ն deployment-ի անվան հետ համադրելու և 2-ը deployment-ի անունը սերվլետի կլասսի հետ համադրելու համար։ Կոնտեյները հիմնվելով url-ի վրա կանչում է համապատասխան servlet-ը։

**MVC(Model view controller) կոնցեպտ**

Սերվլետներով aplication-ում գործում է MVC կոնցեպտը՝ այսինքն լոգիկան, վիզուալիզացիան և control-ը ապահովող հատվածների իրարից առանձնացումը։ Application-ը այս տարբերակով կոմպոնենտների բաժանելը դրանցում փոփոխությունը ավելի ճկուն է դարձնում։

Kepler project-ի կառուցվածքի նկարագրություն։

Կարևոր է նշել որ յուրաքանչյուր սերվլետ ունի միայն մեկ էկզեմպլյար։ Եթե կա 1-ից ավելի request կոնտեյները դրանցից յուրաքանչյուրի համար պարզապես ստեղծում է մի քանի thread սերվլետի նույն օբյկտին ուղղված։

Request-ի մշակումը ունի հետևյալ սխեման։ Client – ը ուղարկում է request-ը սրեվերին։ Կոնտեյներ ստանալով հարցումը ստեղծում է HttpServletRequest HttpServletResponse օբյեկտներ։ Կոնտեյները url-ի հիման վրա գտնում է համապատասխան սերվլետը ստեղծում է դրա համար նոր հոսք և կանչում է սերվլետի service() մեթոդը և դրան որպես արգումենտ փոխանցում է request և response օբյեկտները։ service() մեթոդը որոշում է թե doGet doPost և այլն մեթոդներից որը պետք է աշխատի և 2 օբյեկտները փոխանցում է այդ մեթոդին։ Սերվլետը response օբյեկտում ներառում է մշակման արդյունք հանդիսացող data-ն, որը container-ի և web server application-ի միջոցով վերադարձվում է web client-ին։ Սրանից հետո service() մեթոդի աշխատանքը ավարտվում է thread-ը փկվում է իսկ request և response օբյեկտները ոչնչանում են։

**Սերվլետի կյանքի ցիկլը**

Սերվլետի կյանքի ցիկլը սկսվում այն պահից երբ կոնտեյները start-ից հետո գտնում է սերվլետի կլասսը այն load է լինում, որից հետո կանչում է սերվլետի կոնստրուկտորը դրանից հետո init() մեթոդը մեկ անգամ։ Դրանից հետո սերվլետը անցնում է initialized վիճակի և արդեն պատրաստ է մշակել request-ներ։ Սերվլետի կյանքի ցիկլի մեծ մասը անցնում է այս հատվածում։ Վերջում կոնտեյները ավարտում է սերվլետ-ի կյանքի ցիկլը դրա destroy() մեթոդը կանչելով։