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

Գրաֆիկական ինտերֆեյսով web application-ները դինամիկ են, իսկ դրանց դինմիկությունը ապահովելու համար պետք են համապատասխան տեխնոլոգիաներ։ Նման տեխնոլոգիաներից են servlet-ները և JSP-ները։

**(slide 2)** 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 ծրագրային մաս՝ web server application, որը 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 փոխազդեցությունը մեծամասամբ տեղի է ունենում HTTP պրոտոկոլի միջոցով։ Բրաուզերը մշակում է server-ից ստացված response-ում պարունակվող տվյալները և վիզուալիզացնում է դրանք (slide 2)։

**(slide 3)** Բայց web server application – ը վերադարձնում է միայն ստատիկ տվյալներ, այսինքն դինամիկության ապահովման համար անհրաժեշտ են օժանդակ application-ներ որոնց դերը կատարում են servlet-ները (CGI (common gateway interface)) (slide 3):

**(slide 4)** Servlet-ը իրենից ներկայացնում է հատուկ տիպի Java class որը մշակում է request-ներ և վերադարձնում response` մշակման արդյունքը: Servlet – ների ստեղծման համար անհրաժեշտ կլասսները և ինտերֆեյսները տրամադրում է javax.servlet pacage – ը։ Servlet-ի կլասը պետք է լինի HttpServlet-ի ժառանգ (ցույց տալ ամբողջ հիերարխիան, և միացման համար անհրաժեշտ dependesy-ները)։ Servlet-ի կլասը Request-ների մշակման համար իր swuper class – ից ժառանգում է protected մեթոդներ (slide 4)։

**(slide 5)** 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 (slide 5)։ Մեթոդներից ամենահաճախ օտագործվողները՝ doGet, doPost մեթոդները են: (app – ում որևէ սերվլետի վրա ցույց տալ override արված doGet – ի կամ doPost – ի օրինակ) (slide 5):

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

**Container**

**(slide 6)** 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, http request – ի տիպին համապատասխան 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։ **(slide 6)**

**(slide 7)** Request-ի մշակումը ունի հետևյալ սխեման։ Client – ը ուղարկում է request-ը սրեվերին։ Կոնտեյները 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 HttpServlet – ից ժառանգված 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։**(slide 7)**

**(slide 8)** Web application – ի կոնֆիգուրացիայի մասին տեղեկատվությունը պահվում է web.xml – ֆայլում։ Այս ֆայլի առկայությունը պարտադիր չէ բայց այն լայն կիրառությւն ունի կոնֆիգուրացիայի համար: web.xml չօգտագործելու ալընտրանքը անոտացիաների օգտագործումն է։

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, error – ների, exception – ների մշակման և application – ի շատ այլ config – ներ։ (app - ի վրա ցույց տալ .xml – ի mapping – ի օրինակ) mapping – ի այլընտրանքը @WebServlet անոտացիան է որի value ատրիբուտում նշվում է համապատասխան url – ն։ (app - ի վրա ցույց տալ @WebServlet անոտացիան)

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-րդը deployment-ի անունը սերվլետի կլասսի հետ համադրելու համար։ Կոնտեյները հիմնվելով url-ի վրա կանչում է համապատասխան servlet-ը։

web.xml – ը տալիս է նաև error-ների և exception-ներ մշակման համար նշվում է error – ի ստատուս կոդը կամ exception – ի կլասսը և location – ը։ (application - ի վրա ցույց տալ .xml – ի error, exception – ի մասը։ Հավաքել ոչ վալիդ url)

Ընդհանրապես .xml – ի միջոցով app- ն config անելու հնարավորությունները շատ-շատ են ու դրանցից մի քանիսին որոնք որ օգտագործել եմ ընթացքում էլի ակնդրադառնամ։ **(slide 8)**

**(slide 9)** 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։ **(slide 9)**

**Request parameters**

**(slide 10)** HttpServletRequest օբյեկտի միջոցով կարող ենք ստանալ request-ի պարրամետրեր։ Դրա համար կան HttpServletRequest օբյեկտի հետևյալ մեթոդները՝

getParameter(String param)

getParameterValues(String param) եթե նույն անունով արամետրեր են։

Get request-ների ժամանակ պարրամետրերը փոխանցվում են url-ի միջոցով որը ունի հետևայլ տեսքը՝

ռեսուրսիԱնվանում?պարամետր1=value1& պարամետր2=value2

(application – ի BookSection.jsp - ի վրա ցույց տալ գրքի վրա edit book – ի ժամանակ ուղարկվող url – ն որի մեջ նշվում է ընտրված գրքի ինդեքսը և որի միջոցով որոշվում է թե որ գիրքն է ընտրվել, BookEdit.jsp – ում գրքի ջնջելը հաստատելու yes կամ no պարրամետրերի օգտագործումը BookDeleteController - ում)։

Html ֆորմաներից արժեքներ ստանալու համար html էլէմենտում նշվում է name ատրիբուտը, որից հետո request.getParameter(nameValue)

(app – ում ցույց տալ Registration.jsp – ի name ատտրիբուտները և RegistrationController – ում դրանց օգտագործումը) **(slide 10)**

**Servlet Initialization Options**

**(slide 11)** Սերվլետների պարամետրեր կարելի է փոխանցել նաև կոնֆիգուրացիոն web.xml ֆայլի միջոցով։

Բոլոր սերվլետներին պարամետր փոխանցելու համար`

Պարամետրը սերվլետում կարող ենք ստանալ ServletContext տիպի օբյեկտի միջոցով, որը կարող ենք ստանալ GenericServlet կլասսից ժառտանգված getServletContext() մեթոդի միջոցով։ ServletContext - ը ինտերֆեյս է որը տրամադրում է մեթոդներ որոնք սերվելտը կարող է օգտագործել իր կոնտեյների հետ կապի համար։ Յուրաքանչյուր application-ի համար գյություն ունի միայն մեկ կոնտեքստ։

ServletContext տիպի օբյեկտի getInitParameter() մեթոդի միջոցով ստանում ենք պարամետրը։

(app – ի մեջ որպես կոնտեքստի պարրամետր նշվում է տվյալների բազային միանալու url – ն, որը օգտագործվում է lisener կլասսի մեջ)

config-ներից պարամետր կարելի է փոխանցել նաև միայն կոնկրետ սերվլետին՝

Այս դեպքում պարամետրը կարող ենք ստանալ՝ ServletConfig տիպի օբյեկտի միջոցով որը կարող ենք ստանալ GenericServlet կլասսից ժառտանգված getServletConfig() մեթոդի միջոցով։ ServletConfig – ը ինտերֆեյս է որի միջոցով սերվլետը կարող է ստանալ config-ներ

ServletConfig տիպի օբյեկտի getInitParameter() մեթոդի միջոցով ստանում ենք պարամետրը։

Ինտերֆեյսների ռեալիզացիաները ապահովում է կոնտեյները։ **(slide 11)**

**MVC(Model view controller) կոնցեպտ**

**(slide 12)** Սերվլետներով aplication-ը նույնպես պետք է կառուցել հիմնվելով MVC կոնցեպտի վրա՝ այսինքն լոգիկան, վիզուալիզացիան և control-ը ապահովող հատվածների իրարից առանձնացումը։ Application-ը այս տարբերակով կոմպոնենտների բաժանելը դրանցում փոփոխությունը ավելի ճկուն է դարձնում։ (app-ի կառուցվածքի նկարագրություն) **(slide 12)**

**Redirect and Forwarding**

**(slide 13)** Web ծրագրավորման մեջ ամենահաճխ օգտագործվող գործողություններից մեկն է վերահասցաեագրումը դեպի այլ ռեսուրս։ Դրա համար կա 2 մեխանիզմ` Redirect ու Forward:

Redirec – ի ժամանակ սերվերը client – ին ուղարկում է response 302 կամ 301 ստատուս կոդով։ Response – ում նշվում է հասցեն թե դեպի որ ռեսուրս (url) պետք է client - ը ուղարկի նոր request: Վերահասցեագրումը կարող է լինել ինչպես սերվերի ներսում այնպես էլ դեպի արտաքին ռեսուրս։ Վերահասցեագրումը տեղի է ունենում HttpServletResponse օբյեկտի sendRedirect() մեթոդով որին որպես պարրամերտ փոխանցվում է վերահասցեագրման հասցեն։ (app – ում ցույց տալ սերվլետների մեթոդները redirect - ով)

Forwardi – ի դեպքում վերահղումը ամբողջությամբ տեղի է ունենում սերվերի վրա։ Client – ը ստանում է սովորական response 200 ստատուս կոդով։ Այսինքն սերվերը վերահղում է request – ը սերվերի ներքին այլ url հասցեով։

RequestDispatcher-ի forward() մեթոդի միջոցով միջոցով կարող ենք reuqest – ը սերվլետից վերահղել դեպի այլ սերվլետ, html կամ jsp էջի։ Կարևոր է ընդգծել որ սա վերահղում է այլ ոչ թե վերահասեագրում։

RequestDispatcher – ի օբյեկտ կարող ենք ստանալ ServletContext-ից կամ HttpServletRequest օբյեկտից՝ 2 դեպքում էլ getRequestDispatcher() մեթոդով։

Ռեսուրսը որին պետք է կատարվի վերահղումը նշվում է getRequestDispatcher() – ի պարրամետրում։ Որից հետո forward -ին փոխանցվում են request և response օբյեկտները request – ի վերահղման և դրա հետագա մշակման համար։ (app – ում ցույց տալ սերվլետների doGet մեթոդները forward - ով)

Redirect – ի և forward – ի տարբերությունները

* Redirect – ը տեղի է ունենում կլիենտի մոտ, բրաուզերը կատարում է նոր request, Forward – ը տեղի է ունենում սերվերի վրա
* Redirect – ի ժամանակ url – ն բրաուզերում փոխվում է, իսկ forward – ի ժամանակ չի փոխվում։

**(slide 13)**

**Session**

**(slide 14)** Server – ը request – ների մասին տվյալներ չի պահում, բայց շատ հաճախ առաջանում է անհրաժեշտություն որ request – ների մասին տվյալներ պահվեն, նաև այդ տվյալները փոխանցվեն մեկը մյուսին։ Այս խնդիրը լուծվում է սեսսիայի միջոցով։ Սեսսիա ասելով հասկանում ենք որոշակի ժամանակահատվածում նույն client-ի կողմից իրականացված request-ների հաջորդականություն։ sission-ի հետ աշխատելու համար ունենք HttpSession ինտերֆեյսը, որը օգտագործվում է նույն client-ի կողմից իրականացված տարբեր request-ների միջև տվյալներ փոխանցելու կամ տվյալները պահպանելու համար։ Տվյալները պահում են ամբողջ սեսսիայի ընթացքում կոնկրետ client-ի համար։

Քանի որ յուրաքանչյուր client-ի համար սեսսիան առանձին է ապա պետք է մեխանիզմ որով կոնտեյները կկարողանա նույնականացնել client-ին։ Այդ նպատակով գեներացվում է ունիկալ id և response-ի հետ միասին այն ուղարկվում է client-ին։ Client-ը նույ id-ն հետ է վերադարձնում request-ի հետ և կոնտեյները կապում է համապատասխան սեանսը request-ի հետ։ Id-ների միմյանց հետ փոխանակումը տեղի է ունենում cookie ֆայլերի միջոցով, որոնք պահվում են բրաուզերում։

Սեսսիայի հետ աշխատանքի համար HttpSession ինտերֆեյսը տրամադրում է մի շարք մեթոդներ։ (app – ի վրա ատտրիբուտներից ցույց տալ լոգին եղած user – ի set, get, remove, invalidate հատվածները) **(slide 14)**

**Սերվլետի կյանքի ցիկլը**

**(slide 15)** 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։ destroy() մեթոդի կանչի պահի որոշման ալգորիթմը ինդիվիդւալ է յուրաքանչյուր կոնտեյների համար։ **(slide 15)**

**Listeners**

**(slide 16)** Սերվլետի կյանքի ցիկլերի կոնտեքստում կարևոր դեր ունեն listener-ները։ Listener-ները իրենցից ներկայացնում են ինտերֆեյսներ որոնք ունեն համապատասխան մեթոդներ՝ սերվլետի հետ տեղի ունեցող որոշակի իրադարձությունների հետևելու համար։ Listener class-ը պետք է իմպլեմենտացնի անհրաժեշտ ինտերֆեյսը, ռեալիզացնի դրա մեթոդները կամ հաշվի առնելով որ բոլորը default են՝ մեթոդներից որևէ մեկը։ Որից հետո պետք է նշվի Weblistener անոտացիայով կամ Xml ֆայլում:

(app – ում ցույց տալ context listener – ի initialized – ի ժամանակ db connection – ի set – ի, Controller կլասսներում ատրիբուտի օգտագործման և destroy – ի ժամանակ close – ի օրինակները)։

Կան հետևյալ Listener-ների ինտերֆեյսերը։

|  |  |
| --- | --- |
| ServletRequestListener  Հետևում է Request-ներին | requestDestroyed(ServletRequestEvent sre) requestInitialized(ServletRequestEvent sre) |
| ServletRequestAttributeListener  Հետևում է Request-ների ատրիբուտներ փոփոխմանը | attributeAdded(ServletRequestAttributeEvent srae)  attributeRemoved(ServletRequestAttributeEvent srae)  attributeReplaced(ServletRequestAttributeEvent srae) |
| ServletContextListener  Երբ app – ն deploy է լինում սերվերկ վրա։ | contextInitialized(ServletContextEvent sce)  contextDestroyed(ServletContextEvent sce) |
| ServletContextAttributeListener  Հետևում է Context-ի ատրիբուտներ փոփոխմանը | attributeAdded(ServletContextAttributeEvent event)  attributeRemoved(ServletContextAttributeEvent event) attributeReplaced(ServletContextAttributeEvent event) |
| HttpSessionListener  Հետևում է սեսիայի օբյեկտի փոփոխություննեըին | sessionCreated(HttpSessionEvent se)  sessionDestroyed(HttpSessionEvent se) |
| HttpSessionAttributeListener  Հետևում է Sission-ի ատրիբուտներ փոփոխմանը | attributeAdded(HttpSessionBindingEvent event)  attributeRemoved(HttpSessionBindingEvent event)  attributeReplaced(HttpSessionBindingEvent event) |
| HttpSessionBindingListener  Երբ օբյեկտը կցվում կամ անատվում է սեսիայից | valueBound(HttpSessionBindingEvent event)  valueUnbound(HttpSessionBindingEvent event) |
| HttpSessionActivationListener  Սեանսի ակտիվացման կամ պասիվացման ժամանակ | sessionWillPassivate(HttpSessionEvent se)  sessionDidActivate(HttpSessionEvent se) |

**(slide 16)**

**Filter**

**(slide 17)** Ֆիլտրը իրենից ներկայացնում է օբյեկտ որը կանչվում է request – ի մշակումից առաջ և կատարում է ֆիլտրացիայի գործողություններ։ Կոդի մակարդակում ֆիլտրը կլասս է որը իմպլեմենտացնում է Filter ինտերֆեյսը և ստանում է init(), destroy() և doFilter մեթոդները։ Ինչպես սերվլետներին՝ ֆիլտրերին նունպես պետք է mapping, որը xml – ում գրվում է նույն տեսքով ինչ որ սերվլետի համար։ (app – ում ցույց տալ .xml – ի ֆիլտրերի mapping - ը) Ֆիլտրի միջոցով կարող ենք մինչև url հասցեով անցնելը կատարել որոշակի գործողություններ; init() և destroy() մեթոդնրը կանչվում են մեկ անգամ ֆիլտրի ինիցիլիզացիայի ու ոչնչացման ժամանակ։ doFilter() – ը կանչվում է ամեն անգամ երբ request է ուղարկվում այն ռեսուրսին որի վրա դրված է ֆիլտրը։ doFilter – ը ստանում FilterChain տիպի օբյեկտ որը պատասխանատու է հաջորդ ֆիլտրի կամ ռեսուրսի կանչի համար։ (app – ում ցույց տալ cahce-control – ի (CacheControlFilter) logout - ի հետո սեղմել back – ը։ user – ի ստուգման համար էջերը միայն լոգին եղած user – ին հասանելի դարձնելու (UserAuthenticateFilter) ֆիլտրերը, logout – ից հետո հավաքել secure page – ի url) Ֆիլտրերի կանչի հերթականությունը որոշվում է web.xml-ում սահմանված հաջորդականությամբ։ **(slide 17)**

**JSP**

**(slide 18)** 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 տարբերությամբ որ JSP-ում կարող ենք գրել Java լեզվով կոդ։ JSP-ի նպատակն է HTMl-ի Java-ինտեգրացիան։

Տեխնոլոգիա է որը հնարավորություն է տալիս ստեղծել դինամի վեբ էջեր։ Որպեսզի սերվերը կարողանա մշակել JSP ֆայլերը պահանջվում է engine(движок) որը կոչվում է նաև JSP կոնտեյներ։ Օրինակ Apache Tomcat-ը միաժամանակ հանդիսանում է և web server application, web container, jsp container:JSP չօգտագործելու դեպքում ստիպված պետք է լինենք ամբող HTML կոդը գրել սերվլետների request-ները սպասարկող մեթոդներում և response օբյեկտից getWriter() մեթոդով ստանալ PrintWriter-ի օբյեկտ, որի printLn() մեթոդի միջոցով ուղարկել response: **(slide 18)**

**(slide 19)** Ինչ է անում կոնտեյներ JSP-ի հետ

Կոնտեյները Jsp-ն թարգմանում է java կոդի և վերածում լիարժեք սերվլետի։

* Սկզբից կարդում է դիրեկտիվները
* Ստեղծում է HttpServlet կլասսից ժառանգվող սուբկլասս (Tomcat 5 – ից սկսած HttpJspBase)։ JSP-ից ստեղծված սերվլետի կլասսը իմպլեմենտացնում է նաև HttpJspPage ինտերֆեյսը որը ժառանգում է JspPage ինտերֆեյսից։
* Եթե կան իմպորտներ նորաստեղծ կլասսում դրանք ներառվում են ամենասկզբից
* Եթե կան expression-ով հայտարարված փոփոխականներ կամ մեթոդներ գրում է դրանք կլասսի մեջ։
* Ստեղծում է սերվիս մեթոդ \_jspService() որին աշխատացնում է ծնող կլասսի service() մեթոդը և որպես արգումենտ ստանում է request response: Այս ընթացքում կոնտեյները ստեղծում է նաև jsp -ում հասանելի բոլոր ոչ ակնհայտ օբյեկտները։
* Միավորում է html-ի մնացած մասը նորաստեղծ կլասսի հետ և արդյունքը դուրս է բերում PrintWriter-ի միջոցով։

jspInit, jspdestroy \_jspService(overrde չենք կարող անել քանի որ կոնտեյները ավտոմատ արդեն ստեղծում է նույն անունով մեթոդ) որոնք կանչում է կոնտեյները։ մեթոդները կանչվում են սերվլետի համապատասխանաբար init destroy service մեթոդներից: **(slide 19)**

**JSP սինտաքս**

**(slide 20)** JSP-ի սինտաքսը փաստացի բաժանվում է 2 մասերի html կոդ և java կոդ։ JSP-ում java կոդի տաղադրման նամար օգտագործվում է հետևայալ հիմնական էլէմենտները։

<% %> skriplet

<%= %> expression դառնում է արգումենտ out.print()-ի համար, դրա համար կետ-ստորակետ չի դրվում

<%! %> declaration (փոփոխականները լոկալից դառնում են էկզեմպլյարի), կարող են լինել ստատիկ փոփոխականներ կամ մեթոդներ սլայդի վրա ցույց տալ պարզեցված կլասսի օրինակը

<%@ %> directive իմպորտների և այլ instruction-ների համար, ունի 3 դիրեկտիվ page,  include, taglib

<%-- --%> մեկնաբանություններ **(slide 20)**

**(slide 21)** Սկրիպլետներում հայտարարված փոփոխականները կոմպիլացված սերվլետ կլասսում բոլորը լոկալ գտնովում են \_jspService() – ի մեջ են բացի declaration - ի մեջ գրվածները որոնք դուրս են գալիս \_ jspService() – ի միջից և հայտարարվում են ստեղծված սերվլետ կլասսում, այսինքն կարող են լինել փոփոխականներ կամ մեթոդներ նաև կարող են լինել static։ Expression – ները փոխանցվում են որպես printLn() – ի արգումենտ, այդ պատճառով նաև expression – ներից հետո կետ-ստորակետը compile error է։ **(slide 21)**

**JSP-ի կյանքի ցիկլը**

**(slide 22)** Երբ կոնտեյները գեներացնում է jsp-ից սերվլետի կլասս այն իմպլեմենտացնում է HttpJspPage ինտերֆեյսը։ jspInit, jspDestroy, \_jspService

Jsp ֆայլը deploy է արվում որպես app-ի մաս և քանի դեռ այն չի կանչվել դրա հետ դեռևս ոչինչ տեղի չի ունենում։

Ցանկացած հղում որի ժամանակ կանչվում է jsp - ն կոնտեյները թարգմանում է .jsp-ն ․java ֆայլի, եթե jsp – ում կան սինտաքսային սխալներ դրանք ի հայտ են գալիս այս փուլում։ Հետո .java-ն կոմպիլացվում է .class-ի որը load է լինում։ Կոնտեյները ստեղծում է սերվլետի էկզեմպլյար և աշխատացնում է jspInit մեթոդը։ Դրանից հետո request-ների համար բացվում են thread-ներ և դրանց սպասարկման համար աշխատում է \_jspService մեթոդը։

Քանի որ jspInit()-ը կանչվում է սերվլետի init()-ի միջոցով ապա init()-ի overriding-ի ժամանակ հասանելի կլինեն ServletContext ServletConfig տիպի օբյեկտները։ **(slide 22)**

**Jsp – ում հասանելի են հետևյալ օբյեկտները՝**

**(slide 23)** JspWriter out

HttpServletRequest request

HttpServletResponse response

HttpSession session

ServletContext application

ServletConfig config

Throwable exception

PageContext pageContext

Object page

**(slide 23)**

**(slide 24)** Jsp-ն սերվլետից տվյալներ կարող է ստանալ ատրիբուտների միջոցով։ Jsp-ում սերվլետից տվյալներ ստանալու համար օգտագործվում են հետևյալ օբյեկտները՝ request, session, application:

Ատրիբուտներ սահմանելու համար հասանելի է նաև pageContext օբյեկտը։ pageContext – ի get set attribute մեթոդները ընդունում են նաև scope-ի արգումենտ, այսինքն pageContext -ով կարող ենք ատրիբուտներ սահմանել և ստանալ նաև Request Session Application-ում PageContext.APPLICATION\_SCOPE, PageContext.SESSION\_SCOPE

JSP-ին նույնպես կարելի է web.xml-ից փոխանցել պարամետրեր։

Jsp-ին կարելի է պարամետրեր փոխանցել նաև config-ներից ինչպես սերվելետի դեպքում։ **(slide 24)**

<servlet>

      <servlet-name>HelloServlet</servlet-name>

      <jsp-file>HelloServlet.jsp</jsp-file >

      <init-param>

        <param-name>message</param-name>

        <param-value>Hello Servlets</param-value>

      </init-param>

   </servlet>

<servlet-mapping>

      <servlet-name> HelloServlet </servlet-name>

      <url-pattern>/HelloServlet.jsp </url-pattern>

   </servlet-mapping >

**EL (expression language)**

**(slide 25)** El-ը իրենից ներկայացնում է կոմպատկ սինտաքս jsp-ում որը նախատեսված է java կոդի կանչի համար։ օբյեկտներին և դրանց property-ներին հասանելիության համար։

նշվում է ${-----}

El-ը փնտրում է փակագծերում նշված փոփոխականը բոլոր հասանելի կոնտեքստներում հետևյալ հերթականությամբ՝

Page context -> request context -> session context -> application context: Գտնված րժեքը կոնվերտացվում է տողի և ներդրվում է էջում։ Փոփոխականը կոնկրետ կոնտեքստից ստանալու համար՝  pageScope, requestScope, sessionScope, applicationScope:

El-ի մեջ նշվող առաջին փոփոխականը կամ ոչ ակնհայտ հայտարարված օբյեկտ է կամ ատտրիբուտ։ Եթե el – ի մեջ օգտագործում ենք կետ ապա ձախ փոփոխականը պետք է լինի կամ map կամ bean, իսկ աջ կողմինը կամ key property:

El – ում property – ին կարող ենք դիմել նաև ${person[“name”]} ձևով։ նշված սինտաքսը ավելի արդյունավետ է քանի որ ձախ հատվածում կարող է լինել օր մասիվ, լիստ։ []-ի դեպքում ձախ մասը կարող է լինել վերը նշվածներից բոլորը։ Եթե ձախ կողմում ունենք մասսիվ կամ լիստ ապա տողային արտահայտությամբ նշված թվերը աջ կողմում ձևափոխվում են int – ի։ Եթե []- ի մեջ առանց չակերտների նշվում է արժեք ապա կոնտեյները առաջին հերթին կփնտրի աշված անունով ատտրիբուտ։

El – ում կարելի է օգտագործել նաև այլ օպերատորներ ինչպիսիք են թվաբանական (+, -, \*, /, %), տրամաբանական (&&, ||, !), համեմատության:

Execption-ներից խուսափլու համար el – ը մշակում null – երեը։ Օր եթե ատտրիբուտը գոյություն չունի թվաբանական գործողության ժամանակ այն ընդունում է 0 արժեք իսկ բուլյան գործողությունների դեպքում false:

El-ը անջատելու կարիք կարող է առաջանալ եթե օրինակ անհրաժեշտ է կոդի մեջ ներառել el սինտաքսին նման հատվածներ։ Դրա համար .xml – ում <el-ignored> էլեմենտին տալիս ենք true արժեք։ **(slide 25)**

**Implicit objects inside EL**

**(slide 26)**

El – ու կան ոչ ակնհայտ օբյեկտներ, որոնք իրենցից ներկայացնում են map – եր բացառությամբ pageContext օբյեկտի։

**(slide 26)**

**Jsp & beans**

**(slide 27)**

MVC կոնցեպտով application-ները շատ են կախված ատրիբուտներից։ Այս պարագայում եթե ատտրիբուտը օբյեկտ է ապա դրա property – ներին հասանելության համար կպահանջվի ավելի շատ Java կոդ։ Դրա համար Jsp-ում Java bean – երի հետ աշխատելու համար կան այլ մեխանիզմներ՝ Jsp – ում java կոդի ծավալը քչացնելու համար։ Bean – ը java կլասս է որն ունի default constructor, private fields և դրանց համար getter setter մեթոդներ։

jsp:useBean միջոցով կարող ենք jsp – ի մեջ հայտարարել ատտրիբուտ օբյեկտը և այն օգտագործել։ <jsp:useBean/>-ում նշված օբյեկտը եթե որպես ատտրիբուտ չկա ապա կոնտեյները այն ստեղծում է ավտոմատ։ Այս դեպքում \_jspService() մեթոդում գեներացվում է if պայման որը ստուգում է օբյեկտի առկայությունը։

Պոլիմորֆիզմի հնարավորություններից օգտվելու համար կարող ենք ավելացնել type ատտրիբուտը։ Եթե նշվի միայն տիպը առանց կլասսի ապա նորմալ կաշխատի եթե ատտրիբուտ հանդիսացող օբյեկտը գոյություն ունի, եթե գոյութնուն չունի և նոր պետք է ստեղծվի ապա կունենանք InstantiationException: Չենք կարող նշել միայն տիպը առնց կլասսի։

jsp:useBean – ը կարող է ունենալ մարմին։ Այս դեպքում մարմինը կաշխատի եթե ատտրիբուտը գոյությւոն չունի և նոր է ստեղծվում։

jsp:useBean – ի մեջ հայտարարված օբյեկտի դաշտերը կարող ենք set անել <jsp:сetproperty> - ով և get անել <jsp:getproperty> - ով։

<jsp:getproperty name=”person ” property=”name”/>

<jsp:setProperty name="person" property="name" value="Fred" />

Եթե value – ի արժեքը ստանում ենք html ֆորմայից կամ պարրամետրերից ապա value – ի փոխարեն կարող ենք նշել param=”html input-ի name ատտրիբուտ”, իսկ եթե անունները համընկնում են ապա param նշելու կարիք ևս չի լինի և կվերագրվի input – ից եկած արժեքը։ Եթե բոլոր input – ների և property – ների անունները համընկնում են ապա պետք է նշել property=”\*” և արժեքները ավտոմատ կվերագրվեն։

Եթե սկոպը չի նշվում այն ավտոմատ համարվում է page:

Bean թեգերը input-ներից ստացված պրիմիտիվ տիպերը ավտոմատ քաստ են անում օբյեկտի դաշտի տիպերի եթե դրանց անունները համընկնում են։

Եթե bean- ի դաշտը ոչ թե պրիմիտիվ է այլ օբյեկտ ապա getProperty – ն այս դեպքում չի աշխատի։ Նշված խնդիրը կլուծվի EL-ի միջոցով։ Օր․՝ ${person.dog.name}։

Jsp – ն մեկ այլ jsp – ում ներառելու համար գոյություն ունի <jsp:include page=” Header.jsp” /> էլէմենտը։ Նույն գործողությունը կարելի է կատարել նաև դիրեկտիվի միջոցով՝ <%@ include file="Header.jsp"%> ֆայլի ցանկացած հատվածում տեղադրելով։ **(slide 27)**