**Reading HTML Form Data**

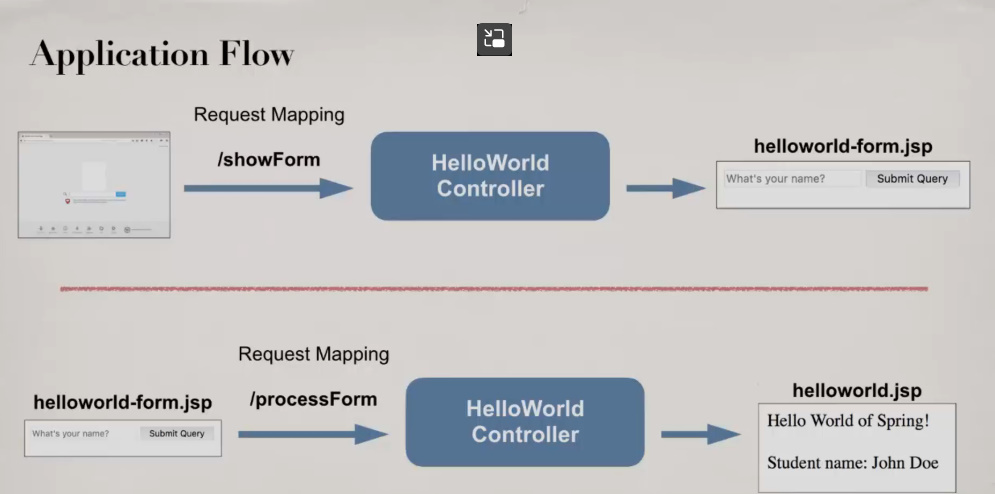
Asta se refera la obtinerea datelor scrise de un user intr-o forma.

**Exemplu de functionare**

**Cream o forma si un raspuns la ea cu o pagina. O forma care odata e completata si se apasa pe buton, apare o pagina de confirmare.**

De ex, pentru a afisa forma cand se acceseaza linkul, se va crea un Request Mapping /showForm, care va fi trimis si gestionat de Controller, si va retura numele paginii helloworld-form.jsp si va fi apoi afisata pagina unde sa introduci numele.

Odata ce introducem numele si apasam pe butonul Submit, se creaza o noua Request Mapping, de ex /processForm care va ajungel la Controller si acesta va returna si afisa o pagina de raspuns.



De exemplu, daca cream o metoda cu anotatia @RequestMapping

@RequestMapping("/showForm")  
public String showForm(){  
 return "helloworld-form";  
}



Datorita beanului adaugat in configapp, spring va adauga prefixul si suffixul, si deci ceea ce e returnat va ajunge ca: WEB-INF/view/helloworld-form.jsp

**Trimiterea de Request de catre HTML**

* Pentru ca un atribut oarecare HTML, sa trimita un request si sa fie procesat de vreo metoda ce are @RequestMapping(“/ceva”), se foloseste atributul action si method=”GET”

<**form** action="processForm" method="GET">

Aceasta forma, la trimiterea datelor, va expedia Request cu numele “processForm” si va fi cautat prin metodele avute cu @RequestMapping

* Putem utiliza si atributul name la <input type....> pentru a putea mai apoi afisa datele trimise de forma. De exemplu:

<form action="processForm" method="GET">  
 <input type="text" placeholder="Enter your name" name="studentName">  
 <input type="submit" value="Send">  
</form>

Name=”studentName”, acesta e numele zonei text, sau numele obiectului String pe care ea il va crea si trimite la Controller, si deci daca vom dori sa afisam datele trimise de ea, vom folosi obiectul param. care stocheaza datele venite din link la request: ${param.studentName}in noua pagina

**Param** – e un obiect standart,care pastreaza obiecte de tip String ce au fost trimisi ca parmetri in link, cu datele din forma, fiecare name=”” exprimand numele obiectului(parametru) String din el. El este folosit pentru a ne referi la parametrii ce pastreaza toate datele din forma trimise de pagina HTM, iar studentName e prametru sau obiectul de tip String primit de la field. Daca nu se scriem param. , atunci Spring va intelege ca ne referim la un atribut, nu param object care stocheaza parametrii. HttpServletRequest doar creaza o referinta catre param, asa cum param a venit de la view prin HTTP. Nu creaza pagina un Model, ci trimite datele prin HTTP.Spring MVC stie cum sa le interpreteze. Spring s-a asigurat sa creeze un obiect de tip param ce contine Stringuri cu toti parametrii veniti in request

Pentru ca ${} sa functioneze, trebuie sa adaugam la paginile HTML la inceput

<%@**page** isELIgnored="false" %>

* Putem folosi si atributul <a href=”Request”> care tot va trimite un Request, el din href

**Adding Data to Spring Model**

* Model(cel din schema) – obiect, numit si container, care transporta date de la Controller la View, nu si invers! View nu creaza Model.
* Paginile jsp pot accesa orice date din Model, dar nu creaza unul, ci trimit datele prin HTTP la Controller
* Pentru a putea primi si trimite date, avem nevoie ca metoda noastra sa aiba cel putin un parametru din acesti 2 parametri:

**HttpServletRequest request** – acest parametru citeste datele din forma trimise de catre pagina HTML, creand o referinta la obiectul param, creat de spring pe baza la datele venite de la pagina prin HTTP, ce corespund formei. Poate sa nu fie disponibil si nici gasit, de aceea adaugam in Pom.xml:

<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>3.1.0</version>  
 <scope>provided</scope>  
</dependency>

Apoi reload la Maeven din dreapta.

**Model model** – un obiect model simplu. Cand el e declarat ca parametru, initial este gol, caci inca nu are date puse in el, dar e initializat, asa cum va fi un model object ce va fi trimis la pagina. Model e creat doar de Spring MVC, deci doar in Controller. View nu pote crea un Model, dar trimite datele prin HTTP. Acest Model object doar pune atribute (obiecte) in Model obj al Controller, avand o referinta la el.

* Pentru a obtine datele din parametri, veniti de la pagina prin HTTP, la care are referinta HttpServletRequest object, folosim metoda **getParameter(“FieldName”)** – unde “FieldName” e numele unui parametru sau obiect creat de field din form,pastrat in param, ca de ex <input type=”text” name=”FieldName”/>. Aceasta metoda cauta obiectul/parametrul “FiledName” in obiectul param, la care are referinta, si il returneaza. Adica pentru fiecare tag ce are name=”” in forma, se va trimite un param in link la request, ele apoi prin HTTP, si spring le va converti in String sau String[] si le pastreaza sub ID sau Parametru/Nume identic cu cel din name=”” in param. Metoda returneaza mereu **String sau String[] in unele situatii**
* Pentru a adauga date in model object, folosim metoda **addAtribute(“Name”,String),** unde “Name” poate fi orice text sau nume al obiectului pe care il adaugam in Model obj al Controller. Aici deja se adauga chiar un atribut sau obiect in Model la Controller. Il vom folosi mai apoi in program, asa cum un model object poate stoca mai multe date, are nevoie sa le identifice cumva,sa le dea nume, si asta se face prin acest parametru, exact cum face la un field cu name=””. Putem folosi si un name deja existent, caci oricum Model Object creat e unul nou, creat de Controller ce adauga obiecte in Model al Controller ce se va duce la pagina din return, adica paginii de raspuns.
* Pentru a afisa aceste date pe o pagina HTML, folosim **${AttributName}.**

**Nu mai e nevoie de param.AtributName, caci param. e folosit cand ne referim la parametrii ce contin date de la o pagina WEB , el fiind un obiect default ce anume contine datele inregistrate de la pagina, si attributName fara param., face referire direct la un atribut venit de la Controller, si deja va spune ca vrem sa accesam obiectul AttributName venit de la Controller, care e de tip String aici. Chiar putem folosi si ${AttributName.uppercase()}.**

* Putem adauga oricate date dorim intr-un object model:

addAtribute(“Name1”,String1)

addAtribute(“Name2”,String2)

addAtribute(“Name3”,String3)

**HTTPServletRequest vs Model**

* **HTTPServletRequest** contine o referinta la obiectul param creat de Spring pe baza la datele din link de la pagin web,de la forma anume, si contine deci strict doar datele care sunt trimise de forma din pagina ce a trimis requestul.

@RequestMapping("/process")  
public String process(HttpServletRequest servletRequest, Model model){  
 System.*out*.println(servletRequest.getParameter("name"));  
  
 return "hello";  
}

HttpServletRequest contine doar datele formei care a trimis requestul “/process”,asa cum are referinta la obiectul param venit in Model obj de la pagina, si atat, mai mult nimic**,deci contine doar parametrii din link, nu si atribute sau ceva de genul.** Acum, pagina “hello” va putea folosi datele din acest HttpServletRequest pentru a le afisa, prin ${param.name}. **Cand se returneaza pagina, se creaza un Model Object ce contine tot ce am setat noi in model object ca parametru in Controller si tot ce este in obiectul param, la care HttpServletRequest are referinta. Atentie, atributele venite de la pagina print HTTP nu se trimit automat catre noua pagina fara @ModelAttribute.Ele nu sunt trimise automat ca param!Insa datele formei se trimit si fara un parametru de tip HttpServletRequest, asa cum el doar modifica obiectul param deja existent in Model obj de la pagina** [**venit.@ModelAttribute**](mailto:venit.@ModelAttribute) **spune ca se va prelua atributul cu numele dat, venit de la View catre Controller prin HTTP, si se va pune in noul Model creat pentru viitoarea pagina. Datele parametrilor automat se pun, dar a atributelor nu.**

Param.nume spune ca se va folosi un parametru, nu atribut. Acum, daca pagina “hello” are si ea o forma ce va trimite date, de ex daca are un camp cu numele “test”, forma va trimite un nou request prin HTTP, cu date asbolut noi, fara ceea ce a continut Model object venit de la Controller, si noul obiect param va lua date doar din forma curenta, cele vechi din vechiul param nemaiexistand in response al paginii, adica ele nu mai sunt trimise in HTTP. Deci, aici deja nu mai exista obiectul/field “name” care a existat in metoda precedenta, numai daca forma care a trimsi requestul nu avea si ea un camp cu numele “name”. Response al paginii, adica datele trimise la Controller prin HTTP, nu mai are atributele ce au putut fi folosite in pagina, din Model Object venit de la Controller atunci, numai daca avem o forma ce are modelAttribute, ce se asigura sa il trimita prin HTTP, dar oricum ca un nou obiect. Pagina deja poate accesa atributele lui Model Object venit de la noul Controller, cu @ModelAttribute, si tot aceasta anotatie se asigura ca **ele sa fie trimise, asa cum returneaza o referinta la acel atribut venit de la pagina prin HTTP, si o adauga in Model obj al Controller.**

@RequestMapping("/process2")  
public String process2(HttpServletRequest servletRequest){  
 System.*out*.println(servletRequest.getParameter("test"));

System.*out*.println(servletRequest.getParameter("name"));  
  
 return "hello2";  
}

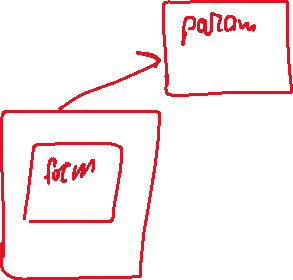
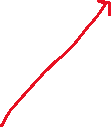
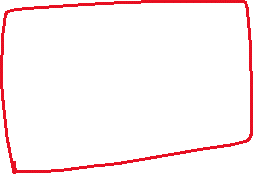
* Putem sa adaugam atribute cu HttpservletRequest, caci pagina “hello2” va putea accesa acest atribut adaugat in Model obj al Controller ce a venit la ea de HttpservletRequest. Object Model ce va veni la ea va contine ce are param + model object pus de noi + obiectele cu @ModelAttribute ca parametri. Nu putem adauga parametri in el, caci parametrii vin strict de la forma, dar putem adauga atribut in schimb cu **servlerRequest.setAttribute(“nume”,obiect);**
* In pagina hello2 putem accesa parametrii prin ${param.nume} si atributele prin ${atribut}. Daca param. lipseste, spring va intelege ca e vorba de atribut, nu parametru.
* Putem folosi si metodele **getParameter()** si **getAttribute()** pentru a obtine vreun parametru sau atribut a lui param la care are referinta.

Cand obiectul HttpServlerRequest abea e setat de Model Object venit, el poate avea atribute numai daca noi le setam in pagina special pentru el **sau daca forma va trimite vreun atribut, prin taguri speciale in param obiect**. Attribut din modelAttribute al formei nu e pus ca attribut al formei, ci e pur si simplu pus in Model object al paginii ce se duce la Controller, **deci HttpServlerRequest nu il poate accesa!**

* **Model** – Un obiect de tip Model ca parametru mereu este instantiat si este gol. El nu preia absolut niciun atribut de la pagina ce a trimis requestul, ci doar parametrii. Insa, el va fi trimis paginii din return.

**addAttrbibute(“name”,obj)** – e folosit pentru a a adauga un attribut in Model obj de la Controller ce e trimis la pagina din return, ce poate fi accesa cu ${obj};

Atentie! Si Model si HttpServletRequest au metode pentru a adauga atribute. Ambele le accesam prin ${numeAttribute}. Putem adauga atribute cu Model model deja existente, asa cum el va trimite pur si simplu o noua referinta catre atributul deja existent.



**Cum returnam atribute**

* Dupa cum s-a spus mai sus, atributele nu trec automat de la o pagina la alta. De ex:
* @Controller  
  public class App {  
   @RequestMapping("/")  
   public String index(Model model){  
   model.addAttribute("test1","Index");  
    
   return "index";  
   }  
    
   @RequestMapping(value="/hello2",method = RequestMethod.*GET*)  
   public String hello(@ModelAttribute("test1")String s){  
   System.*out*.println(s);  
    
   return "hello2";  
   }  
  }

Aici, metoda “index”, cand va returna pagina, va crea un HTTP response ce va prelua atributele care au fost create in el de Model object al nostru, si il va trimite paginii “index”. Pagina index va putea utiliza atributul “test1” din acest HTTP response.Daca ar fi fost si un obiect HttpServlerRequest ca parametru, si ar fi adaugat atribute, Model object al Controllerului ar fi adauga atributele sale. Dupa ce pagina “index” va da vreun request catre metode “hello”, sa zicem ca fara forma, atunci ea va crea un nou HTTP response, dar acest response pe care il creaza orice pagina nu copie niciodata Atributele sau param de la Model Object venit de la controller! Insa, daca request e trimis de vreo <form:form action=”hellp2” modelAttribute=”test1”> ,atunci forma se va asigura ca HTTP Response creat de pagina sa contina si atributul “test1”,cu un nou obiect, si atunci meoda hello ar putea sa foloseasca atributul “test1”. **Numai @ModelAttribute poate accesa aceste atribute, nu si HttpServletRequest, si anume el se va asigura ca referinta a acestui atribut sa se adauge in Model de la Controller,ce se va trimite la noua pagina, caci atributele venite de pagina nu se trimit automat ca Model object setat de noi si param!**

* Multe atribute sunt trimise pentru a afisa date, nu pentru a colecta si a reintoarce, de aceea nu se adauga si atributele din Model object venit de la Controller in HTTP Response creat de pagina, ca sa nu fie prea multe si inutile.
* Cand e folosita anotatia data, in Model object de la controller se creaza atributul cu numele dat ce va avea referinta la cel din HTTP Response venit de la pagina, si ModelAttribute va avea la el. Aceste atribute sunt ca niste HashMap de forma nume : referinta

**Adaugare la CSS/JS etc. in MVC project**

* **nu uitam de / la inceput si final**
* **href=”/adresa/” de multe ori nu merge, si e preferabil href=”<c:url value=”/>”**

**mvc:resources**

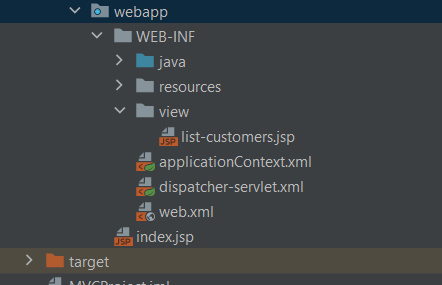
1. In webapp cream un folder cu orice nume,de ex resources, si punem in el foldere la css,images, js etc.
2. In applicationContext.xml adugam

<mvc:resources mapping="/p/\*\*" location="/WEB-INF/resources/"/>

**mapping** – aici indicam care va fi request pentru care se va cauta ceea ce e in location

**location** – aici indicam propriu zis care este locatia la care se refera mapping

Deci, daca de ex vom avea href=”/p/styles.css”, MVC va cauta in loc de /p/ in /WEB-INF/resources/, nu pur si simplu va inlocui /p/ cu asta



**SAU**

Pentru a accesa acum calea unde se afla folderul, putem folosi:

**${pageContext.request.contextPath}/WEB-INF/resources/CSS/styles.css**

**<link rel="stylesheet" type="text/css"**

**href="${pageContext.request.contextPath}/WEB-INF/CSS/CSS.css**

**">**

**<img src="${pageContext.request.contextPath}/resources/images/spring-logo.png" />**

* **${pageContext.request.contextPath} – se refera la numele aplicatiei, daca e furnizat vreunul.**

**Sau**

Utilizam pur si simplu

"../../resources/css/styles.css"

**SAU!!!**

Spring Security sau MVC poate trata toate aceste / ca normal request, si deci sa nu poata citi fisierele noatre. In asa caz,nu mai putem lucra cu WEB-INF folder, asa ca trebuie sa il abandonam.Deci lasam in context app neaparat mvc:resources!

<mvc:resources mapping="/resources/\*\*" location="/resources/"/>

Si vom adauga css neaparat asa:

<link rel="stylesheet" type="text/css" href="<**c:url** value="/resources/CSS/styles.css"/>"/>

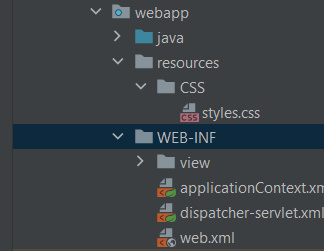
Pentru asta, in JSP mai adaugam

<%@ **taglib** prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

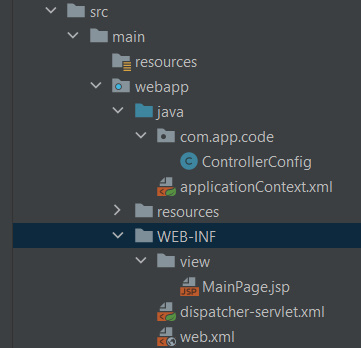
Si in pom.xml

<dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
</dependency>

Dar vom crea un folder resources in webapp folder, si acolo vom adauga CSS si tot ce avem nevoie.



applicationContext poate fi si in java folder, nu se va schimba nimic,



**@RequestParam(“Name”)**

Putem obtine datele de la HTTP Response ce vine de la pagina HTML fara sa cream in functie un parametru de tip HttpServletRequest. ce sa aiba referinta la param Putem citi orice data direct de la param prin anotatia **@RequestParam(“Name Field”).**

O punem in fata unei variabile parametru si Spring va face automat atribuirea.

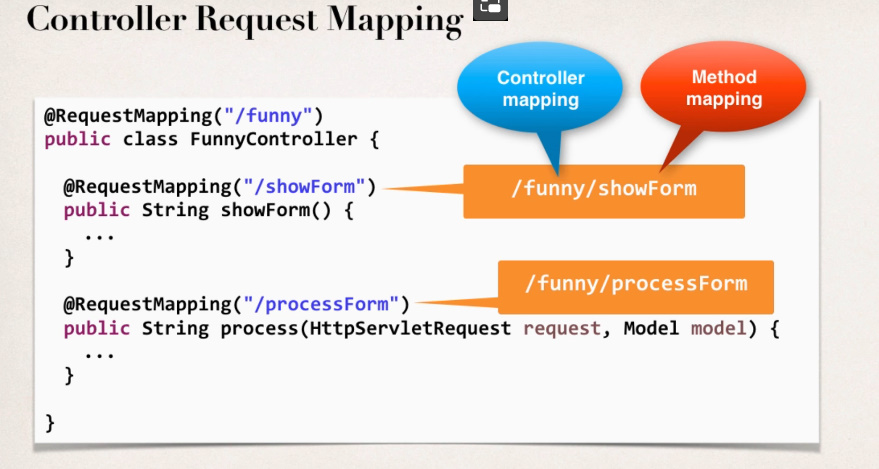
@RequestParam(“Name”) => request.getParameter(“Name”);

Unde request e de tip HttpServlerRequest

**@RequestParam poate fi folosi doar la parametrii metodei!**

**Controller Request Mapping**

Putem folosi @RequestMapping(“Nume”) nu doar la metode din interiorul unei clase @Controller, dar si la o clasa. Daca o punem la o clasa, pastram mereu si @Controller





Clasa de mai sus va face ca orice metoda sa aiba un RequestMapping de la Requestul /funny

* /funny – controller mapping
* Celelalte dupa /funy/ sunt method mapping

/funny e **ca** un folder, si /sowForm si /processForm sunt **ca** subfoldere din el.

Acest lucru e necesar, deoarece am putea avea anumite conflicte cu alte RequestMapping. De ex:

Fie clasa

@Controller  
public class Silly {  
  
 @RequestMapping(value="/form", method = RequestMethod.*GET*)  
 public String error(){  
 return "main";  
 }  
}

si clasa

@Controller  
public class HelloWorld {  
  
 @RequestMapping(value="/", method = RequestMethod.*GET*)  
 public String setMain(){  
 return "main";  
 }  
  
 @RequestMapping(value="/form", method = RequestMethod.*GET*)  
 public String setForm(){  
 return "form";  
 }  
  
  
}

Ambele au o metoda ce are @RequestMapping(value="/form", method = RequestMethod.*GET*)  
dar ambele returneaza o pagina diferita. La rulare, TomCat va da o eroare cu ambiguitate.

Pentru a rezolva ambiguitatea, avem nevoie de @RequestMapping() dupa @Controller la o clasa.

@Controller

@RequestMapping(“/hello”)



Parent mapping care se refera ca la un director si spune ca toate metodele ce vor avea @RequestMapping se vor referi mereu la parintele /hello. Adica fara a mai scrie /hello in fata lor, ele automat vor proveni doar de la el.

**La Metodele care au aceasta anotatie avem deja submapping**

**Pentru a rezolva amiguitatea de mai sus, facem:**

@Controller  
@RequestMapping("/hello")  
public class Silly {  
 @RequestMapping(value="/form", method = RequestMethod.*GET*)  
 public String error(){  
 return "main";  
 }  
  
}

Astfel, ambele clase de mai sus au o metoda ce mapeaza acelasi request, doar ca acest request nu apartine aceluiasi parinte. Daca vrem sa folsim Request /form din prima clasa, vom scrie la final de adresa de pagina doar **/form** , insa daca vom vrea sa il accesam pe cel din a 2 clasa, vom scrie **/hello/form.** Asta e doar un mod diferit de a face referire la aceleasi pagini HTML, dar cu diferite request, ca sa evitam ambiguitatile cand exista clase ce au metode ce primesc acelasi request, fara a avea vreun parinte.

Totodata, daca de exemplu din /form vrem sa mai trimitem un reuqest, ca /hei, requestul va fi /form/hei, si deci posibil sa avem o clasa speciala pentru requestul /form. Insa, daca forma e deja trimisa din clasa cu RequestMapping, ea nu va avea la action si hello/, ci doar form

**Hello nu e un folder, ci doar o alta request ce se va ocupa de sub request cu nume ce mai apar deja pe altundeva.**