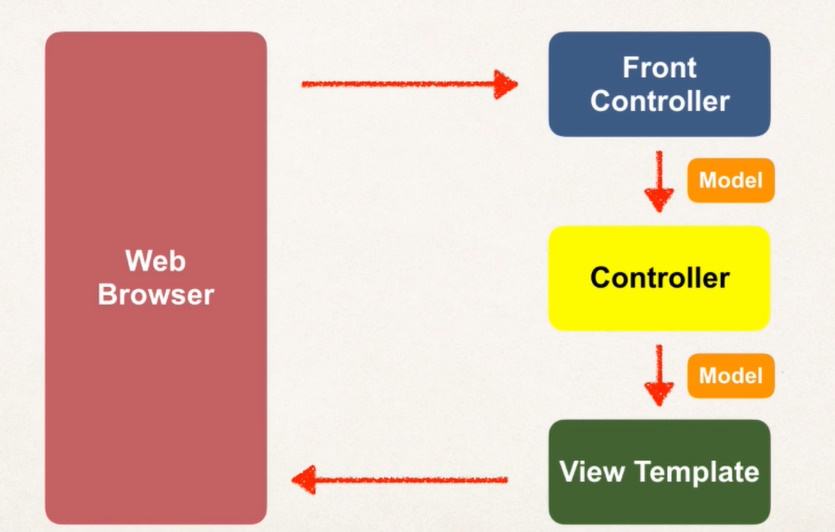
**Ce este Spring MVC**

* Spring MVC este un Framework pentru a face web aplicatii
* E bazat pe Model-View-Controller design pattern.
* Functioneaza pe principiile la Spring Core Framework(IoC, DI)

**Model View Controller**

****

1. Intai avem o cerere venita de la browser
2. Aceasta va fi receptionata de Spring MVC Front Controller
3. Cererea e procesata de un Cod Controller pe care noi il cream.
4. Cream un model si il intoarcem inapoi la Front Controller si Front Controller il trimite la View Template, ceva gen ca HTML page
5. La final, View Template e trimis la browser

**Front Controller**

E cunoscut ca DispatcherServlet. El primeste cererile venite de la user prin UI components si le trimite catre obiecte sau iteme corespunzatoare.

**Model Objects**

* Obiecte care contin date despre cerere si despre modul in care se proceseaza.
* Cand Controllerul pastreaza si recupereaza datele prin backend systems ca:
* Database, web service
* Poate fi si Spring bean

Modelul va fi cel care va pastra aceste date.Datele pot fi orice java object sau colectie.

* Deci object model este ca un bagaj care pastreaza datele ce trebuie trimise de la o parte a frameworkului la alta. **Model e creat doar de Controller si e trimis la View. View nu poate crea Model.**

**View Template**

* Este JSP(Java server pages) sau JSTL(JSP Standart Tag Library) actual page
* JSP sau JSTL pot citi aceste modele de date si sa le afiseze. De ex, daca avem o lista de studenti, JSP poate crea un tabel ca sa o afiseze. Sau daca cineva cumpara un bilet, View Template va afisa o pagina HTML,sau o JSP page, care sa spuna ca totul s-a facut cu succes.

**Controller**

* Contine Codul creat de developeri.
* Contine bussines logic creat de noi:
* Primeste cererea interceptata de Front Controller
* Se ocupa de cererile venite din partea userului
* Pastreaza sau recupereaza datele
* Pune datele intr-un model.
* Trimite raspuns prin Model object, care stocheaza datele de raspuns, catre cel mai apropiat View Template

**Componentele unei Spring MVC aplicatii**

* Un set de pagini web pentru a schita UI componente
* Spring beanuri
* Spring configuration(XML, Java code etc.)

**Configurarea Spring MVC**

**Partea1: Adaugam configuratii in in WEB-INF/web.xml**

1. Configuram Spring MVCDispatcher Servlet
2. Setam URL mappings la Spring MVCDispatcher Servlet

**Partea2: Adaugam configuratii fisierului WEB-INF/spring-mvc-demo-servlet.xml**

1. Adaugam suport pentru Spring component scanning
2. Adaugam suport pentru conversie, formatare si validare
3. Configuram Spring MVC View Resolver

**Partea1:**

**Step1:**

<servlet>  
 <servlet-name>dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <load-on-startup>1</load-on-startup>  
</servlet>

Aceasta parte adauga intrarea pentru Spring DispatcherServlet

Apoi aici adaugam ce e in verde, ca sa aratam unde situat fisierul SpringConfiguration file.

<servlet>  
 <servlet-name>dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>web/WEB-INF/applicationContext.xml</param-value>  
 </init-param>  
  
 <load-on-startup>1</load-on-startup>  
  
</servlet>

**Step2:**

Partea asta de cod spunem ca vrem ca fiecare URL pattern care vine vrem sa il dam sau sa il trecem lui DispatcherServlet. Adica toate web requests vor fi administrate de Dispatcher Servlet.

<servlet-mapping>  
 <servlet-name>dispatcher</servlet-name>  
 <url-pattern>/</url-pattern>  
</servlet-mapping>

De asta si se scoate \*form si se pune doar /, ca toate url requests vor fi administrate de dispatcher servlet, dar putem avea si alte pattern.

**Step3:** in config file adaugam tot ce facem ca deobicei cu beanurile, si aici avem nevoie de component-scan.

**Step4**: tot in config file adaugam si

<mvc:annotation-driven />

Dar pentru asta, avem nevoie de

xmlns:mvc="http://www.springframework.org/schema/mvc"

annotation-driven adauga suport pentru form conversie, formatare si validare.

Dupa asta, adaugam un bean pentru MVC View Resolver

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="WEB-INF/view"/>  
 <property name="suffix" value=".jsp"/>  
</bean>

Acest bean face ca orice nume de view returnat sa aiba in fata adauga prefixul WEB-INF/view si suffixul .jsp. view este un folder unde pastram vederile.

Trebuie de creat un folder view in WEB-INF unde sa cream un fisier .jsp cu numele paginii returnat de metoda.

**Configurarea Spring MVC fara XML**

**File: DemoAppConfig.java**

package com.luv2code.springdemo.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages="com.luv2code.springdemo")

public class DemoAppConfig {

// define a bean for ViewResolver

@Bean

public ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/view/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

}

---

**File: MySpringMvcDispatcherServletInitializer.java**

package com.luv2code.springdemo.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class MySpringMvcDispatcherServletInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

// TODO Auto-generated method stub

return null;

}

@Override

protected Class<?>[] getServletConfigClasses() {

return new Class[] { DemoAppConfig.class };

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

**Developping Spring Controllers and Views**

1. Cream Controller class
2. Definim Controller method
3. Adaugam Request Mapping la Controller method
4. Returnam View Name
5. Dezvoltam View page

**Step1:**

Cream o clasa si adaugam anotatia @Controller care mosteneste de la @Component

@Controller spune ca acesta este un Spring MVC Controller. El este un special @Component care suporta web MVC

Cand Spring face component scanning, el scaneaza si @Controller, asa cum el mosteneste de la @Component. Pentru controller se creaza de asemenea automat un bean.

**Step2:**

Cream o metoda cu orice nume. Poate returna orice si avea oricati parametri formali.

**Step3:**

Adaugam anotatia @RequestMapping(“/”) la metoda precedenta. Aceasta va mapa/schita anumite tipuri de web request la aceasta metoda. Intre (“ “) punem pathul pe care vrem sa il schitam anume. De exemplu, daca punem “/”, asta inseamna ca daca cineva viziteaza pagina, se va intampla ceea ce e specificat in metoda, sau se spune ca orice http request va avea drept raspuns ce returneaza metoda. Adica anotatia data leaga anumite tipuri de request mapate deja de aceasta metoda, care va oferi raspunsuri la ele.

**Step4:**

In metoda returnam numele la view.

@Controller  
class HomeController{  
  
 @RequestMapping("/")  
 public String showPage(){  
 return "main-page";  
 }  
}

Multumita la beanul declarat in configapp, Spring va returna prefix+main-page+sufix: WEB-INF/view/main-page.jsp

**Stept4:**

Cream o pagina HTML. Nu cream fisierul cu extensia .html, ci .jsp

**Separarea fisierelor**

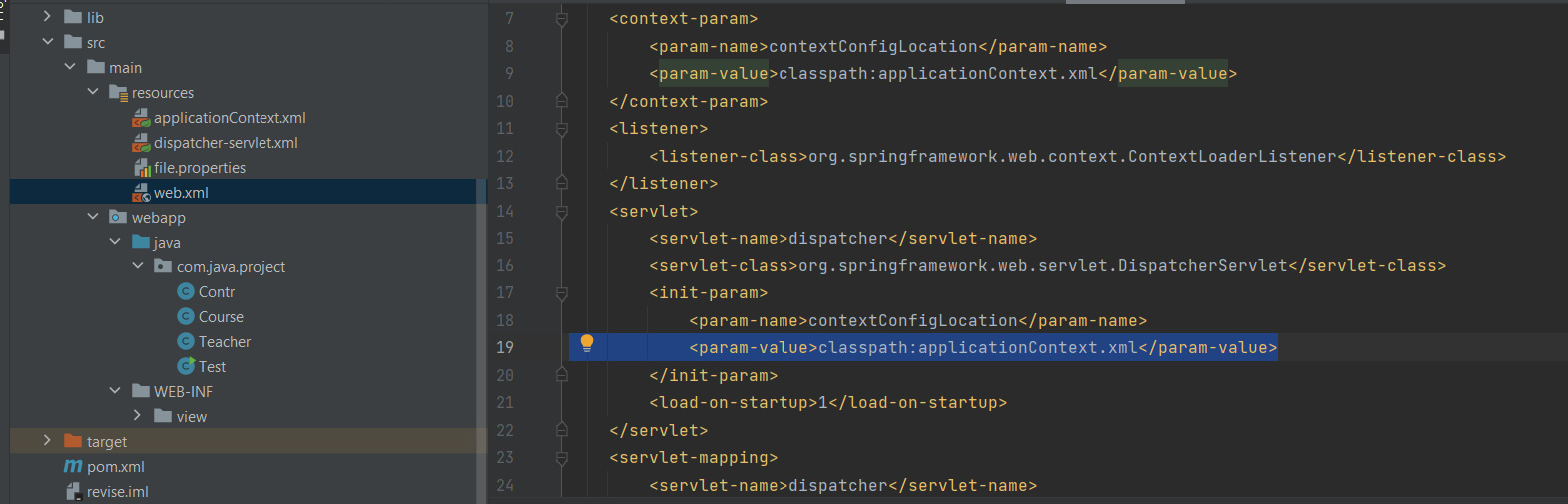
* De multe ori putem avea nevoie sa punem applicationContext.xml in resources. Acum avem 2 variante:

1. Punem si web.xml si dispatcher-servlet.xml acolo, si modificam locatiile lor din setari, sa nu fie WEB-INF
2. web.xml si dispatcher-servlet.xml le lasam in WEB-INF.

Oriunde nu le punem, mereu ne referim la orice fisier din resources prin classpath:

<param-value>**classpath:**applicationContext.xml</param-value>

Chiar daca toate aceste fisiere se gasesc in resources, oricum folosim classpth: ca Spring sa nu le caute in webapp.



Daca web.xml si celalalt .xml sunt in WEB-INF, si context e in resources, to folosim classpath pentru a ne referi la context.

* **classpath inseamna nu doar package la clase(java aici), ci si folderul resources**
* **Dar, acum, vedem ca view nu este impreuna cu ele, deci cand cream bean pentru InternalResourceViewer, nu vom pune la adresa la pagin jsp si classpath:, ci pur si simplu /WEB-INF/view/, caci in webapp mereu se cauta totul automat.**
* **Spring cauta aceste fisiere si in resources chiar!**
* **Nu se pun paginile jsp in resources! Nu ne vom putea referi la ele, asa cum mereu se vor cauta in webapp, si asta din cauza ca Tomcat are drept folder webapp, si el poate cauta doar acolo paginile jsp, in classpath pur si simplu nu are acces.**
* **Spring daca nu gaseste web.xml in webapp, nu il va cauta in classpath. Va trebui noi sa setam din setari noua adresa pentru web.xml, daca ea nu s-a setat automat cand am mutat fisierul.**