

## Spring Portlet MVC

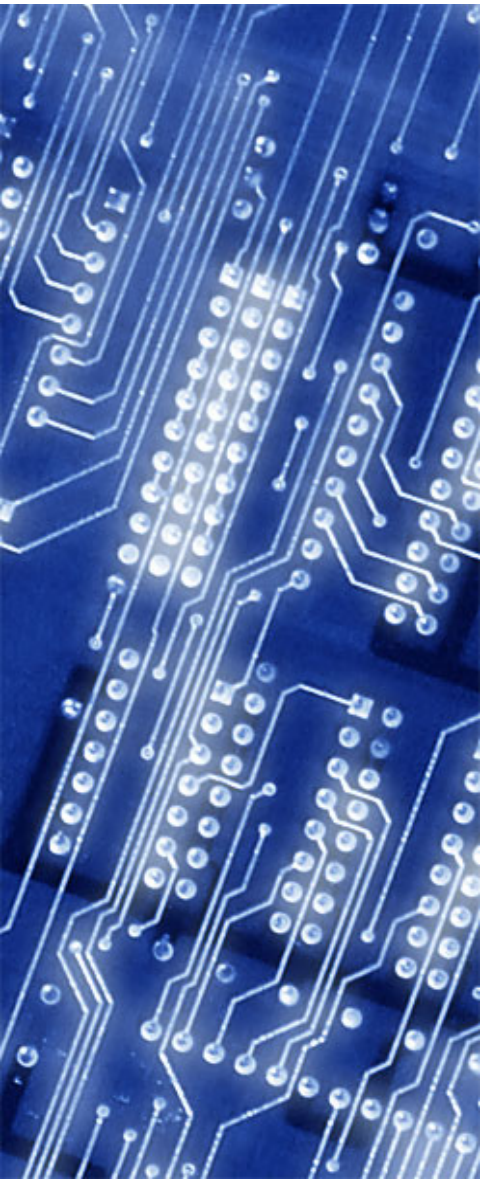
Miroslav Ligas

Rastislav Papp

Tomáš Polešovský

IBA CZ, s.r.o.

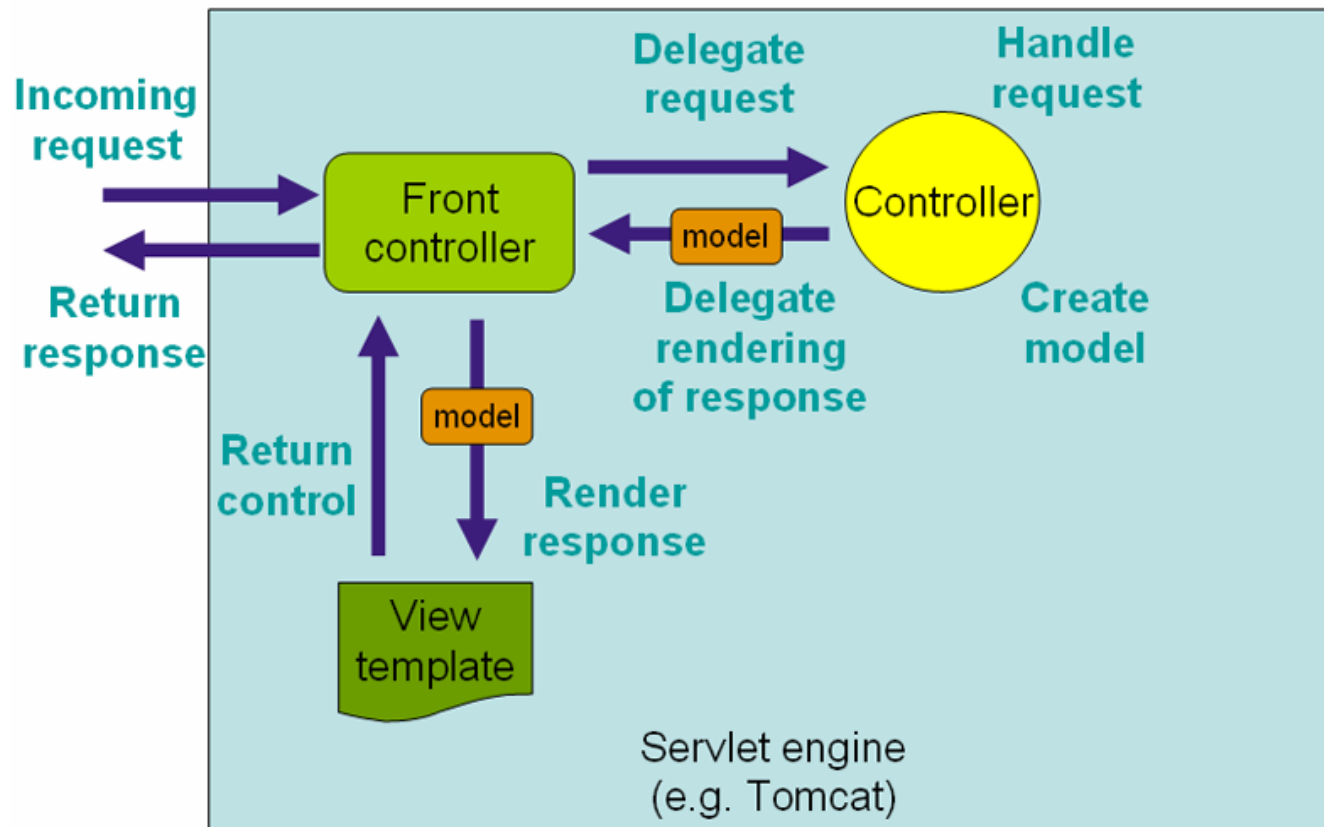
- Frameworky na tvorbu portletov
- Spring Core - IoC Container
- Spring Portlet MVC
  - Konfigurácia aplikácie pre použitie frameworku
  - @Controller vs. Portlet
  - Anotácie
  - Lokalizácia



- V drvinej väčšine vychádzajú z prostredia servletov a fungujú ako most medzi portletmi a servletmi (portlet-bridge)
- Snažia se napasovať svoj spôsob myslenia na portlety - nepodporujú vývoj portletov natívne
- Request-based (push) frameworky se ťažko vysporiadávajú s viacerými fázami (render, action, resource, event)
- Spring Portlet MVC podporuje vývoj portletov natívne, nie pomocou portlet-bridge
- Jeden z najrozšírenejších frameworkov.

- Vychádza z Spring Web MVC
- Ak viete používať Web MVC, viete aj Portlet MVC
- Zmeny reflektujú portletové prostredie s dôrazom na anotácie
  
- Controller
  - jednoduchý POJO objekt
  - Definujeme anotáciou `@Controller` a `@RequestMapping`
  - Zastupuje funkciu portletu
- Model
  - Slúži na predanie dát medzi Controllerom a View
- View
  - Prezentačná technológia, môže byť ľubovoľná
  - Zväčša JSP + JSTL
  - Velocity, Freemarker, Tiles, ...









- Aplikácia beží v Springovom IoC kontajneri
- Jednotlivé objekty (beany) aplikácie sú vytvárané Springom
- Beany, ktoré majú závislosti na iných beanoch, sa nestarajú o ich vytváranie, len deklarujú že ich potrebujú, a Spring im ich nastaví automaticky



- Vo web.xml je nutné deklarovať Springovský listener a Servlet a môžeme uviesť lokáciu hlavného konfiguračného súboru - applicationContext.xml (ale nie je to nutné, defaultne sa hľadá na ceste WEB-INF/applicationContext.xml)
- Samotné portlety v portlet.xml:  

```
<portlet-class>  
    org.springframework.web.portlet.DispatcherPortlet  
</portlet-class>
```
- Jednotlivé konfiguračné súbory portletov deklarujeme tiež v portlet.xml, alebo sa hľadajú na ceste WEB-INF/<názov\_portletu>-portlet.xml



- Anotácia @Controller
- Anotácia @RequestMapping - VIEW, EDIT, ... + ďalšie parametre
- Metódy @XXXMapping - podľa názvu portletového módu ktorého požiadavky obsluhujú:
  - @RequestMapping - doView
  - @PostMapping - processAction
  - @GetMapping - processEvent
  - @ResourceMapping - serveResource
- Signatúra metód nieje predpísaná, programátor si ju volí sám
- @InitBinder, @RequestParam, @ModelAttribute, ...





- Podpora lokalizácie
- Zadefinovať bean s názvom 'messageSource' (konkrétne ukážky v príklade) v konfiguračnom súbore portletu
- V JSP môžeme na vytiahnutie hlášky z ResourceBundle použiť tag zo springovského taglibu:

```
<spring:message code="key-from-resource-bundle"/>
```

- Ten sa nahradí hodnotou kľúča

- Veľmi užitočná anotácia :-)

```
@RequestMapping(...)
public String renderSomePage(RenderRequest request) {
    String pageNumberStr = request
        .getParameter(PARAM_PAGE_NUMBER);
    Long pageNumber;
    if (pageNumberStr == null) {
        pageNumberStr = DEFAULT_PAGE_NUMBER;
    }
    pageNumber = Long.parseLong(pageNumberStr);
    ...
}
```

**VS.**

```
@RequestMapping(...)
public String renderSomePage(RenderRequest request,
    @RequestParam(value = PARAM_PAGE_NUMBER,
        required = false, defaultValue = DEFAULT_PAGE_NUMBER)
    Long pageNumber) {
    ...
}
```



- Formulár zväčša predstavuje nejaký doménový objekt, jednotlivé položky formulára sú atribúty tohto objektu
- Pri odoslaní prídu v requeste parametre, ktoré majú názvy podľa názvov jednotlivých inputov formulára
- Keď chceme daný doménový objekt vytvoriť, musíme vytiahnuť parametre z requestu a nasetovať ich do tohto objektu
- Zložité, zdĺhavé, zbytočné

- Na vytiahnutie parametrov z requestu môžeme použiť `@RequestParam` - stále príliš zdĺhavé
- Použijeme Springovú podporu pre tvorbu formulárov
- Jednotlivé položky formulára sa mapujú priamo do objektu, ktorý si vyberieme a tento objekt nám Spring injectuje do metód (zväčša `@ActionMapping`)

Ako na to:

1) Daný objekt vložíme v render fáze do Modelu

```
Person person = new Person();
model.addAttribute(ATTRIBUTE_PERSON, person);
```

2) V JSP použijeme na vytvorenie formulára a jeho položiek Springovské tagy

```
<form:form ... commandObject="<%=ATTRIBUTE_PERSON%>">
    <form:input path="firstName"/>
    <form:input path="lastName"/>
</form:form>
```

- path určuje cestu k atribútu command objektu - je možné adresovať hierarchicky, napr. `path="address.city"`





3) V `@RequestMapping` metóde vytiahneme z modelu daný objekt, ktorý už bude mať nasetované všetky položky, ktoré sme naň namapovali

```
@RequestMapping
public void actionSave(@ModelAttribute(ATTR_PERSON) Person p){
    ...
}
```

- Môžeme mapovať aj iné typy ako String
- Jednoduché typy konvertuje automaticky Spring
  - číselné typy, boolean, enumy
- Na zložitejšie typy je potrebné Springu určiť akým spôsobom ich má konvertovať na String a zo Stringu na daný typ - toto sa určuje pomocou anotácie `@InitBinder`
- Môžeme mapovať aj zoznamy objektov



- Oanotujeme ňou metódu, v ktorej vytvoríme editory na položky formulára ktoré Spring nevie konvertovať automaticky

@InitBinder

```
public void initBinder(WebDataBinder binder) {
    binder.registerCustomEditor(Date.class, "dateOfBirth",
        new CustomDateEditor(...));
    binder.registerCustomEditor(Workplace.class, "workplace",
        new WorkplacePropertyEditor());
}

public class WorkplaceEditor extends PropertyEditorSupport {
    @Override
    public String getAsText(Object obj) {
        // konverzia z objektu na String
    }
    @Override
    public void setAsText(String text) {
        // konverzia zo Stringu na objekt
        setValue(obj);
    }
}
```



- Deje sa v 2 krokoch - bindovanie a validácia. Najprv sa Spring pokúsi namapovať (nabindovať) všetky položky formulára na dané atribúty objektu a následne takto namapovaný objekt zvalidujeme našim Validator-om
- Chyby bindovania sa ukladajú do triedy BindingResult, ktorú musíme zadať ako atribút metódy v ktorej používame @ModelAttribute, a to priamo za tento atribút, tj.:

@ActionMapping

```
public void actionSavePerson(@ModelAttribute(ATTR_PERSON)
    Person p, BindingResult result) {
    myValidator.validate(p, result);
    if (!result.hasErrors()) {
        //no validation errors
    } else {
        //handle errors
    }
}
```

- Validátor (myValidator na predch. slide) je trieda, kt. implementuje rozhranie Validator, a nachádzajú sa v nej naše custom validácie, napr. či je zadaný String e-mail, či je vek užívateľa väčší ako 18 rokov, a podobne

```
public class PersonValidator implements Validator {
    @Override
    boolean supports(Class<?> clazz) {
        return clazz.isAssignableFrom(Person.class);
    }
    @Override
    void validate(Object target, Errors errors) {
        Person p = (Person) target;
        ValidationUtils.rejectIfEmptyOrWhiteSpace(errors,
            "name", "msg-err-field-required");
        if (p.getAge() < 18) {
            errors.rejectValue("age", "msg-err-too-young");
        }
    }
}
```

- Stringy, kt. zadávame reject\*() metódam sú názov atribútu a kód chybovej správy, ktorý pre ňu máme v resource bundli.



- Ak sa v BindingResult nachádzajú chyby, zvyčajne presmerujeme užívateľa v @RequestMapping metóde naspäť na stránku s formulárom a zobrazíme chyby
- Validačné chyby jednotlivých polí vypisujeme nasledovne:  

```
<form:input path="firstName"/>
<form:errors path="firstName"/>
```
- Tag `<form:errors path="somePath"/>` vypíše všetky chyby, ktoré pri validácii daného políčka nastali (môže ich byť viac). Pri použití `path="*"` sa vypíšu všetky chyby všetkých políčok. Pomocou atribútu `element` môžeme určiť v akom elemente sa bude hláška chyby nachádzať (`span`, `div`), a atribútom `cssClass` pridáme tomuto elementu css triedu.
- Chybám, ktoré vzniknú pri bindovaní nastavíme text chybových hlášok tak, že si zistíme kód danej chyby a dáme ho do resource bundlu. Napr. pre chybu kt. vznikne tak, že zadáme chybné dátum je kód hlášky `typeMismatch.java.util.Date`

- Neprepisovať si ModelAttribute v render metóde.  
BindingResult je viazaný na atribút v modely. Ak sa tento model prepíše strácame BindingResult.
- Nevkladať v action fázy objekty do modelu. Namiesto modelu využiť `response.setRenderParameter()`
- Pri úspešnom prevedení akcie treba ModelAttribute vymazať z modelu. Inak sa bude formulár prdvypĺňovať starými hodnotami.
- Využívať logovanie a debugger