

Spring Portlet MVC

Miroslav Ligas

Rastislav Papp

IBA CZ, s.r.o.

- @RequestParam
- Tvorba formulárov
 - Bindovanie prvkov formulára
 - @ActionMapping
 - @ModelAttribute
 - @InitBinder
- Validácia formulárov

- **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
 - 2) `Person person = new Person();`
 - 3) `model.addAttribute(ATTRIBUTE_PERSON, person);`
 - 4) V JSP použijeme na vytvorenie formulára a jeho položiek Springovské tagy
 - 5) `<form:form ... commandObject="<%=ATTRIBUTE_PERSON%>">`
 - 6) `<form:input path="firstName"/>`
 - 7) `<form:input path="lastName"/>`
 - 8) `</form:form>`
 - 9) - path určuje cestu k atribútu command objektu - je možné adresovať hierarchicky, napr. `path="address.city"`



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

```
@ActionMapping
```

```
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.:
- @PostMapping
- 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 `@ActionMapping` metóde naspať 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 políček. 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