

Thymeleaf

# Hello World

- Hello World Thymeleaf šablona vypadá následovně: (například `src/main/resources/templates/index.html`)

```
<!DOCTYPE html>

<html xmlns:th="http://www.thymeleaf.org"
      xmlns="http://www.w3.org/1999/xhtml">

  <body>

  </body>

</html>
```

- Fyzicky se jedná o XHTML soubor! (tzn. musí být validní!!!!)
- Thymeleaf šablona je VIEW v MODEL-VIEW-CONTROLLER.
- Plná dokumentace je zde:
  - <http://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>

# Controller I.

- K tomu, abyste mohli Thymeleaf stránku zobrazit, potřebujete Controller:

```
@Controller
```

```
public class IndexController {
```

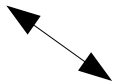
```
    @RequestMapping("/")
```

```
    public String index() {
```

```
        return "index";
```

```
    }
```

```
}
```



Vrací thymeleaf šablonu  
src/main/resources/templates/index.html

# Controller II.

- Controller z předcházející stránky jenom staticky servíruje stránku klientovi. Jak do něj poslat nějaká data? Prostřednictvím modelu:

```
@RequestMapping("/")  
  
public String index(Model model) {  
    model.addAttribute("hello", "cokoli ... toto je Object");  
    return "index";  
}
```

- V Thymeleaf šabloně se pak tyto data zpřístupní například tímto způsobem:

```
<th:block th:text="${hello}"></th:block>
```

Poznámka: pomocí `#{ }` je možné získat i 18n texty z překladových souborů

# Expression Language I.

- EL není programovací jazyk! V drtivé většině případů jenom zobrazuje data získaná z modelu. Nicméně je docela dost flexibilní:
  - Uvnitř můžete kombinovat texty, čísla, boolean (true, false), null:
- Umí i aritmetické operace:
- Boolean výrazy: and, or, !, not
- Výrazy pro porovnání: >, <, >=, <=, ==, != (gt, lt, ge, le, eq, ne)
- Nejpokročilejší je podmínkové větvení:

(if) ? (then)

(if) ? (then) : (else)

(value) ?: (defaultValue)

```
<th:block th:text="${option eq 0 ? 'display text for option 0'}"></th:block>
```

# Expression Language II.

- EL také umí pracovat s polem: `${pole[0]}`
- Nebo s mapou: `${mapa['klic']}`
- Můžete také volat metody objektů z modelu (i s parametry):  
`${person.dopocitejHodnoty('vstup')}`  
... ale tady bych byl hodně opatrný, tato metoda se zavolá při každém jejím použití.
- Také můžete přímo volat metody servisní vrstvy (u toho bych byl opět velice opatrný):  
`${beans.testService.compute()}`

# th:\*

- Existuje pouze jeden tag: `<th:block>`, ale je možné používat celou řadu `th:` atributů.
- `th:text` jsme již viděli, nahradí vnitřek tagu nějakým textem. Nemusí být aplikován pouze na tag `<th:block>`, ale na jakýkoli tag! (jako `<p>`, `<div>`, `<a>` apod.).
- Existují další zajímavé `th:` atributy:
- `th:id` vloží výsledek EL výrazu do atributu `id` (obdobně například `th:span`, `th:href` apod.)
- `th:if`: Pokud EL výraz vrátí `true`, pak se interpretuje vnitřek tagu, který obsahuje tento atribut.

# th:each

- th:each použijete pro sekvenční průchod polem nebo kolekcí:

```
<table>
  <tr th:each="item : ${items}">
    <td>
      <th:block th:text="${item.name}"></th:block>
    </td>
  </tr>
</table>
```

Atribut přidáný v controlleru  
pod tímto názvem do modelu

Atribut ve třídě Item



# Formulář I.

- Spring umožňuje bindování formuláře na objekt:
- Controller:

```
@ModelAttribute public Item construct() { return new Item(); }
```

```
@RequestMapping(value = "/form", method = RequestMethod.GET)
```

```
public String form() { return "item-form"; }
```

```
@RequestMapping(value = "/form", method = RequestMethod.POST)
```

```
public String save(@ModelAttribute Item item) {
```

```
    itemService.save(item);
```

```
    return "redirect:/items";
```

```
}
```

- Na další stránce je to samé jiným způsobem.

# Formulář II.

- Nebo Controller:

```
@RequestMapping(value = "/form", method = RequestMethod.GET)
```

```
public String form(Model model) {  
    model.addAttribute("item", new Item());  
    return "item-form";  
}
```

```
@RequestMapping(value = "/form", method = RequestMethod.POST)
```

```
public String save(@ModelAttribute Item item) {  
    itemService.save(item);  
    return "redirect:/items";  
}
```

# Formulář III.

- item-form.html:

Absolutní URL, nepovinné,  
může zde být také relativní URL: /items/form

```
<form th:action="@{/items/form}" th:object="${item}" method="post">
```

```
<div class="form-group">
```

```
<label>name:</label>
```

```
<input type="text" th:field="*{name}" class="form-control" />
```

```
</div>
```

```
<div class="form-group">
```

```
<label>price:</label>
```

```
<input type="text" th:field="*{price}" class="form-control" />
```

```
</div>
```

```
<input type="submit" class="btn btn-primary" />
```

```
</form>
```

Název atributu v modelu

Atribut třídy Item

Poznámka: kdybyste nepoužili th:object, pak by  
místo th:field="\*{name}" bylo th:field="\${item.name}"

# Formulář & Bean Validation I.

- V Controlleru se pouze změní metoda namapovaná na POST požadavek, zbytek je stejný:

```
@RequestMapping(value = "/form", method = RequestMethod.POST)
```

Vynutí validaci

```
public String save(@ModelAttribute @Valid Item item, BindingResult bindingResult,  
                   RedirectAttributes redirectAttributes) {
```

```
    if(bindingResult.hasErrors()) {
```

```
        return "item-form";
```

```
    }
```

```
    itemService.save(item);
```

```
    redirectAttributes.addFlashAttribute("success", true);
```

```
    return "redirect:/items";
```

```
}
```

Něco pěkného navíc :-)

Výsledek validace se uloží sem.  
Tento parametr MUSÍ následovat  
za parametrem s anotací @Valid!!!

V případě chyby se vrátíme na stránku s formulářem

# Formulář & Bean Validation II.

- Ve třídě, jejíž objekt je ve formuláři jsou anotace jako @Size:

```
public class Item {  
    @Size(min = 1, message = "Name cannot be empty!")  
    private String name;  
  
    @Min(value = 1, message = "Price cannot be empty!")  
    private double price;  
}
```

# Formulář & Bean Validation III.

- Ve formuláři se pro výpis chybových hlášek použije `th:errors`:

```
<div class="alert alert-success" th:if="${success eq true}">saved!</div>
<form th:action="@{/items/form}" th:object="${item}" method="post" class="form">
    <input type="hidden" th:field="*{id}" />
    <div class="form-group">
        <label>name:</label>
        <input type="text" th:field="*{name}" class="form-control" />
        <th:block th:errors="*{name}"></th:block>
    </div>
    <div class="form-group">
        <label>price:</label>
        <input type="text" th:field="*{price}" class="form-control" />
        <th:block th:errors="*{price}"></th:block>
    </div>
    <input type="submit" class="btn btn-primary" />
</form>
```

Něco pěkného navíc :-)

Pro editaci záznamu

Sem se vypíše chyba validace

# Fragmenty stránky I.


- Můžete mít více html souborů, které je možné navzájem includovat (podobná funkcionality jako `<jsp:include>` u JSP):

templates/item-form.html:

```
<head>  
  <th:block th:include="layout/header :: meta"></th:block>  
</head>
```

templates/layout/header.html:

```
<th:block xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">  
  <th:block th:fragment="meta">  
    <meta name="viewport" content="width=device-width, initial-scale=1" />  
  </th:block>  
</th:block>
```



- Poznámka: Kdybyste měli hodně stránek, pak je lepší použít Apache Tiles

# Fragmenty stránky II.

- Fragmenty mohou mít parametry:

```
<th:block th:fragment="nav(shortTitle, title)">  
  
</th:block>
```

- Použití:

```
<th:block th:include="layout/header :: nav('item-form', 'Item Form')"></th:block>
```