Skillbox

Интегрирование Thymeleaf в Spring

Кирилл Кошаев

Java-разработчик в «Газпром информ»

Набор интеграций

Thymeleaf предлагает набор интеграций со Spring,
которые позволяют использовать его в Spring MVC приложениях
в качестве полнофункциональной замены для несколько архаичной
технологии JSP

- Мэппинг методов контроллера на соответствующие шаблоны.
- Создание форм, полностью интегрированных с бинами,
  которые поддерживают работу этих форм на стороне backend.
- Binding значений, обработка ошибок, валидация,
  conversion сервисы.
- Интернационализации контента при помощи message.property
  файлов под управлением Spring.
- Обработка шаблонов при помощи собственных
  механизмов Spring.

SpringStandardDialect

Thymeleaf предоставляет специализированный диалект,
который включает в себя все необходимые функции для правильной работы со Spring:

- возможность получать доступ к любому бину контекста при помощи синтаксиса ${@myBean.doSomething()};
- новые атрибуты для обработки форм: th:field, th:errors и th:errorclass;
- новая реализация th:object, которая позволяет передавать в форму объект и редактировать его свойства;
- служебные объекты #themes и #mvc.

Skillbox

Forms

С формой связаны так называемые командные объекты, задача которых заключается в поддержке формы. Эти бины моделируют поля формы и предоставляют методы получения и установки значений свойств

- Значения атрибутов th:object в тегах формы должны быть выражениями-переменными.
- Допустимо указывать только имя атрибута модели. Навигация по свойствам недопустима.
- Внутри тега <form> нельзя указать ещё один атрибут th:object.

```
...
<form action="#" th:action="@{/home}" th:object="${user}" method="post">
    ...
</form>
...
```

th:field

Атрибут: **th:field** очень важен для интеграции со Spring MVC, потому что он выполняет всю тяжёлую работу по привязке пользовательского ввода к свойству бина, поддерживающего форму

Элемент input с атрибутом th:field

```
...
<form action="#" th:action="@{/home}" th:object="${user}" method="post">
    <input type="text" th:field="*{email}" />
</form>
...
```

input элемент в примере ниже эквивалентен первому примеру

```
 ...
 <input type="text" id="email" name="email" th:value="*{email}" />
 ...
```

th:errors & #fields

Функция #fields.hasErrors (...) получает значение поля в качестве параметра и возвращает логическое значение, указывающее на наличие или отсутствие ошибки ввода в этом поле

```
...
<input type="text" th:field="*{email}"
th:class="${#fields.hasErrors('email')}? fieldError" />
...
```

Итерация по списку ошибок

```
...
<ul><li th:each="err : ${#fields.errors('email')}" th:text="${err}" /> </ul>
...
```

Атрибут th:errors строит список со всеми ошибками для указанного селектора

```
<input type="text" th:field="*{email}" />
<p th:if="${#fields.hasErrors('email')}" th:errors="*{email}">Incorrect
date</p>
```

th:errorclass

Пример с классом CSS стилей, который мы только что рассмотрели настолько распространен, что у Thymeleaf имеется специальный атрибут для выполнения точно такой же работы и называется **th:errorclass**

```
...
<input type="text" th:field="*{email}" class="small"
th:errorclass="fieldError" />
...
```

Если в поле ‘email’ нет ошибок, то при рендеринге разметка будет выглядеть так

```
<input type="text" id="email" name="email" value="workmail@gmail.com"
class="small fieldError" />
```

Fragment rendering

Фрагментарный рендеринг может быть достигнут с помощью использования спеков — объектов, реализующих интерфейс **org.thymeleaf.fragment.IFragmentSpec**. Наиболее распространённой из этих реализаций является **StandardDOMSelectorFragmentSpec**

Fragment rendering

Код ThymeleafView бина фрагмента

```
@Bean(name="content-part")
@Scope("prototype")
public ThymeleafView someViewBean() {
    ThymeleafView view = new ThymeleafView("index");
view.setMarkupSelector("content");
    return view;
 }
```

Метод контроллера, возвращающий content-part

```
@RequestMapping("/showContentPart")
 public String showContentPart() {
     ...
     return "content-part";
 }
```

Skillbox

Спасибо за внимание!