First Edition

Thymeleaf with Spring Boot

by Michael Good

Preface

Thank you to everyone that participates in the Thymeleaf and Spring projects. Both technologies are such a pleasure to use and it is amazing that they are open source.

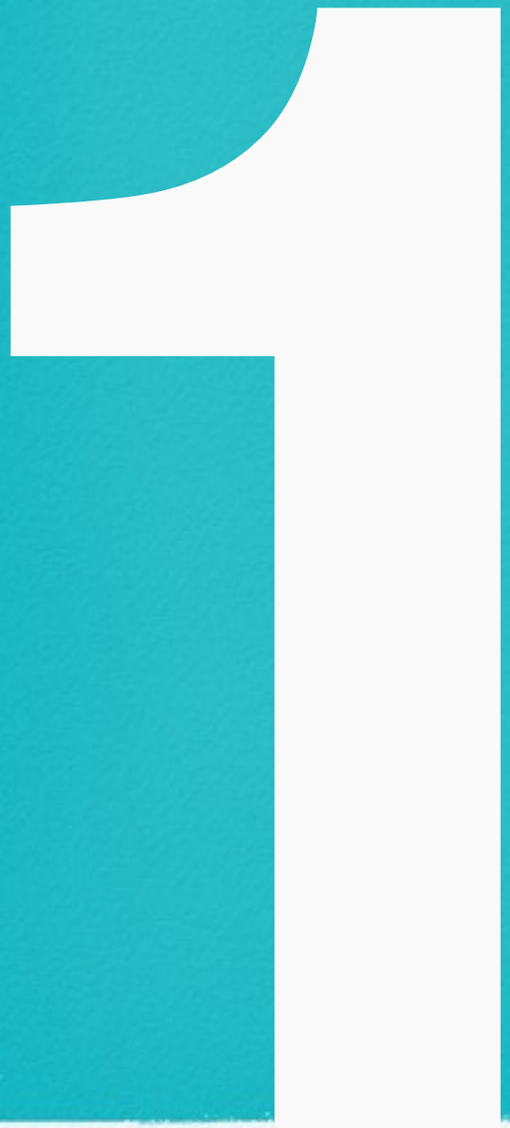
Thymeleaf's official site is at: http://www.thymeleaf.org/

Spring's official site is at: https://spring.io/

Thanks to my family for all their support in my life.

Lastly, thank you for reading this short book and I hope it is helpful. If you have any questions, please contact me at michael@michaelcgood.com

Getting Started



Here we will be reviewing the basics of getting Thymeleaf set up appropriately with a Spring Boot project.

Overview

Lorem Ipsum

1. **Overview of Thymeleaf**
2. **How to use this book**

What is Thymeleaf?

Thymeleaf is a Java library. It is an XML/XHTML/HTML5 template engine capable of applying a set of transformations to template files in order to display data produced by your applications.

Thymeleaf is most suited for serving XHTML/HTML5 in web applications, but it can process any XML file in web or standalone applications.

The primary goal of Thymeleaf is to provide an elegant and structured way of creating templates. In order to achieve this, it is based on XML tags and attributes that define the execution of predefined logic on the DOM (Document Object Model), instead of explicitly writing that logic as code inside the template.

Its architecture allows a fast processing of templates, relying on intelligent caching of parsed files in order to use the least possible amount of I/O operations during execution.

And last but not least, Thymeleaf has been designed from the beginning with XML and Web standards in mind, allowing you to create fully validating templates if that is a need for you.

How does Thymeleaf work with Spring?

The Model-View-Controller (MVC) software design pattern is a method for separating concerns within a software application. In principal, the application logic, or controller, is separated from the technology used to display information to the user, or the view layer. The model is a communications vehicle between the controller and view layers.

Within an application, the view layer may use one or more different technologies to render the view. Spring web-based applications support a variety of view options, often referred to as view templates. These technologies are described as "templates" because they provide a markup language to expose model attributes within the view during server-side rendering.

The following view template libraries, among others, are compatible with Spring:

- JSP/JSTL
- Thymeleaf
- Tiles
- Freemarker
- Velocity

A Brief Overview of Thymeleaf VS JSP

- Thymeleaf looks more HTML-ish than the JSP version – no strange tags, just some meaningful attributes.
- Variable expressions `(${...})` are Spring EL and execute on model attributes, asterisk expressions `(*{...})` execute on the form backing bean, hash expressions `(#{...})` are for internationalization and link expressions `(@{...})` rewrite URLs. (
- We are allowed to have prototype code in Thymeleaf.
- For changing styles when developing, working with JSP takes more complexity, effort and time than Thymeleaf because you have to deploy and start the whole application every time you change CSS. Think of how this difference would be even more noticeable if our development server was not local but remote, changes didn't involve only CSS but also adding and/or removing some HTML code, and we still hadn't implemented the required logic in our application to reach our desired page. This last point is especially important. What if our application was still being developed, the Java logic needed to show this or other previous pages wasn't working, and we had to new

styles to our customer? Or perhaps the customer wanted us to show new styles on-the-go?

- Thymeleaf has full HTML5 compatiblity

How to Use This Book

Just a small portion of this book is a reference manual and it is not as comprehensive as the Thymeleaf manual. The heart of the book is chapters 6 through 11, where you build real working applications that can hopefully help you with your own work. So for this section, it is best if you have at your computer with your IDE open. **If you have any problems following along or just have a question, please email me** at michael@michaelcgood.com and I will help you ASAP! It is my goal that this book is of some help to you.

application.properties

Goals

1. **Review various application.properties settings in regards to Thymeleaf**

Overview

Spring Boot applies it's typical convention over configuration approach to property files. This means that we can simply put an *"application.properties"* file in our *"src/main/resources"* directory, and it will be auto-detected. We can then inject any loaded properties from it as normal.

So, by using this file, we don't have to explicitly register a PropertySource, or even provide a path to a property file.

Common Thymeleaf Properties

```
# THYMELEAF (ThymeleafAutoConfiguration)
spring.thymeleaf.cache=true # Enable template cach-
ing.
spring.thymeleaf.check-template=true # Check that the
template exists before rendering it.
spring.thymeleaf.check-template-location=true # Check
that the templates location exists.
spring.thymeleaf.content-type=text/html # Content-
Type value.
spring.thymeleaf.enabled=true # Enable MVC Thymeleaf
view resolution.
spring.thymeleaf.encoding=UTF-8 # Template encoding.
```

```
spring.thymeleaf.excluded-view-names= # Comma-
separated list of view names that should be excluded
from resolution.
spring.thymeleaf.mode=HTML5 # Template mode to be ap-
plied to templates. See also StandardTemplateModeHan-
dlers.
spring.thymeleaf.prefix=classpath:/templates/ # Pre-
fix that gets prepended to view names when building a
URL.
spring.thymeleaf.suffix=.html # Suffix that gets ap-
pended to view names when building a URL.
spring.thymeleaf.template-resolver-order= # Order of
the template resolver in the chain.
spring.thymeleaf.view-names= # Comma-separated list
of view names that can be resolved.
```

Basic Content

2

Here we cover how to connect a controller to a Thymeleaf template and the basic properties used in a template.

Hello World

Topics

1. Model
2. ModelAndView
3. Thymeleaf

1. Hello World with Model

To use Model, you can return the name of the template as a String:

```
    @GetMapping("helloworldM")
    public String helloworldM(Model model){
        // add some attributes using model

        return "helloM";
    }
```

2. Hello World with ModelAndView

Here we assign a template name to ModelAndView this way, but it can be done with other methods like *setView* or *setViewName*:

```
  @GetMapping("helloworldMV")
    public ModelAndView helloWorldMV(){
        ModelAndView modelAndView = new ModelAnd-
View("helloMV");

        return modelAndView;

    }
```

Objects can be added to to ModelAndView, sort of like how attributes are added to Model.

3. Thymeleaf

A very basic Thymeleaf template we could use to say “Hello World”:

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

<head>

</head>
<body>
<h1> Hello world with ModelAndView</h1>


</body>
</html>
```

4. Adding Attributes to Model

Let’s add some attributes to our Model from step 1.

First I make a entity:

```
@Entity
public class Planet {


    @GeneratedValue(strategy = GenerationType.AUTO)
    @Id
    private Long id;
    private String name;
// basic getters and setters
```

Make a simple repository:

```
@Repository
public interface PlanetDAO extends JpaReposito-
ry<Planet, Long> {

    Planet findByname(String name);

}
```

Now we can do this:

```
    @GetMapping("helloworldM")
    public String helloworldM(Model model){
        // add neptune here to demonstrate adding it
as attribute
        Planet Neptune = new Planet();
        Neptune.setName("neptune");
        planetDAO.save(Neptune);
        model.addAttribute("neptune", Neptune);

        // call repository and get list for attribute
        addPlanets();
        Iterable<Planet> planetList =
planetDAO.findAll();
        model.addAttribute("planetList", planetList);
        // favorite planets
        List<Planet> favoritePlanets = new Ar-
rayList<>();

favoritePlanets.add(planetDAO.findByname("earth"));

favoritePlanets.add(planetDAO.findByname("mars"));
```

```
        model.addAttribute("favoritePlanets", favor-
itePlanets);

        return "helloM";
    }
```

5. Attributes In Thymeleaf

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

<head>
<!-- CSS INCLUDE -->
<link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/css/bootstrap.min.css"

integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7
on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous"></link>
<!-- EOF CSS INCLUDE -->
</head>
<body>
    <h1>Hello world with Model</h1>

    <h2>th:text example</h2>
    <span th:text="${neptune.name}">here is text </
span>

    <h2>th:each example</h2>
    <div class="planetList"
th:unless="${#lists.isEmpty(planetList)}">
        <table class="table datatable">
            <thead>
                <tr>
                    <th>No.</th>
                    <th>planet</th>

                </tr>
            </thead>
            <tbody>
                <tr th:each="planetList : ${planet-
List}">
                    <td
th:text="${planetList.id}">Text ...</td>
                    <td
th:text="${planetList.name}">Text ...</td>
                </tr>
            </tbody>
        </table>
    </div>
    <br />
    <h2>th:if example</h2>
    <table class="table datatable"
        th:if="${#lists.size(favoritePlanets)}>1">
        <thead>
            <tr>
                <th>No.</th>
                <th>planet</th>

            </tr>
        </thead>
        <tbody>
            <tr th:each="favoritePlanets : ${favor-
itePlanets}">
                <td
th:text="${favoritePlanets.id}">Text ...</td>
                <td
th:text="${favoritePlanets.name}">Text ...</td>
            </tr>
        </tbody>
    </table>
```

```
</body>
</html>
```

Basic Concepts

We Are Reviewing:

1. **xmlns:th**
2. **th:text**
3. **th:each**
4. **th:unless**
5. **th:if**

xmlns:th

This is the thymeleaf namespace being declared for `th:*` attributes

th:text

The `th:text` attribute evaluates its value expression and sets the result of this evaluation as the body of the tag it is in, substituting the current content between the tags with what the expression evaluates to.

th:each

`th:each` is used for iteration. These objects that are considered iterable by a th:each attribute

- `java.util.List` objects
- Any object implementing `java.util.Iterable`
- Any object implementing `java.util.Map`. When iterating maps, iter variables will be of class `java.util.Map.Entry`.
- Any array
- Any other object will be treated as if it were a single-valued list containing the object itself.

th:if

This is used if you need a fragment of your template only to appear in the result if a certain condition is met.

Note that the `th:if` attribute will not only evaluate *boolean* conditions. It will evaluate the specified expression as `true` following these rules:

- If value is not null:
  - If value is a boolean and is `true`.
  - If value is a number and is non-zero
  - If value is a character and is non-zero
  - If value is a String and is not “false”, “off” or “no”
  - If value is not a boolean, a number, a character or a String.

th:unless

This is the negative counterpart of `th:if` .

Forms

Here we cover how forms work in Thymeleaf and the typical elements of a form like radio buttons and checkboxes.

Command Object

Command object is the name Spring MVC gives to form-backing beans, this is, to objects that model a form's fields and provide getter and setter methods that will be used by the framework for establishing and obtaining the values input by the user at the browser side.

Thymeleaf requires you to specify the command object by using a **th:object** attribute in your **<form>** tag:

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

This is consistent with other uses of **th:object,** but in fact this specific scenario adds some limitations in order to correctly integrate with Spring MVC's infrastructure:

- Values for **th:object** attributes in form tags must be variable expressions (**${...}**) specifying only the name of a model attribute, without property navigation. This means that an expression like **${storeGuide}** is valid, but **${storeGuide.data}** would not be.
- Once inside the **<form>** tag, no other **th:object** attribute can be specified. This is consistent with the fact that HTML forms cannot be nested.

Inputs

Let's see now how to add an input to our form:

```
<input type="text" th:field="*{dateAcquired}" />
```

As you can see, we are introducing a new attribute here: **th:field.** This is a very important feature for Spring MVC integration because it does all the heavy work of binding your input with a property in the form-backing bean. You can see it as an equivalent of the path attribute in a tag from Spring MVC's JSP tag library.

The **th:field** attribute behaves differently depending on whether it is attached to an **<input>**, **<select>** or **<textarea>** tag (and also depending on the specific type of **<input>** tag). In this case (**input[type=text]**), the above line of code is similar to:

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

…but in fact it is a little bit more than that, because **th:field** will also apply the registered Spring Conversion Service, including the **DateFormatter** we saw before (even if the field expression is not double-bracketed). Thanks to this, the date will be shown correctly formatted.

Values for **th:field** attributes must be selection expressions (***{...}**), which makes sense given the fact that they will be evaluated on the form-backing bean and not on the context variables (or model attributes in Spring MVC jargon).

Contrary to the ones in **th:object**, these expressions can include property navigation (in fact any expression allowed for the path attribute of a **<form:input>** JSP tag will be allowed here).

Note that **th:field** also understands the new types of **<input>** element introduced by HTML5 like **<input type="datetime" ... />**, **<input type="color" ... />**, etc., effectively adding complete HTML5 support to Spring MVC.

Checkbox Fields

th:field also allows us to define checkbox inputs. Let's see an example from our HTML page:

```
<div>
  <label th:for="${#ids.next('covered')}"
th:text="#{storeGuide.covered}">Covered</label>
  <input type="checkbox" th:field="*{covered}" />
</div>
```

Note there's some fine stuff here besides the checkbox itself, like an externalized label and also the use of the **#ids.next('covered')** function for obtaining the value that will be applied to the id attribute of the checkbox input.

Why do we need this dynamic generation of an id attribute for this field? Because checkboxes are potentially multi-valued, and thus their id values will always be suffixed a sequence number (by internally using the **#ids.seq(...)** function) in order to ensure that each of the checkbox inputs for the same property has a different id value.

We can see this more easily if we look at such a multi-valued checkbox field:

```
<ul>
  <li th:each="feat : ${allFeatures}">
    <input type="checkbox" th:field="*{features}"
th:value="${feat}" />
    <label th:for="${#ids.prev('features')}"
           th:text="#{${'storeGuide.feature.' +
feat}}">Heating</label>
  </li>
</ul>
```

Note that we've added a **th:value** attribute this time, because the features field is not a boolean like covered was, but instead is an array of values.

Let's see the HTML output generated by this code:

```
<ul>
  <li>
    <input id="features1" name="features" type="chec-
kbox" value="features-one" />
    <input name="_features" type="hidden" value="on"
/>
    <label for="features1">Features 1</label>
  </li>
  <li>
    <input id="features2" name="features" type="chec-
kbox" value="features-two" />
    <input name="_features" type="hidden" value="on"
/>
    <label for="features2">Features 2</label>
  </li>
  <li>
    <input id="features3" name="features" type="chec-
kbox" value="features-three" />
    <input name="_features" type="hidden" value="on"
/>
    <label for="features3">Features 3</label>
  </li>
</ul>
```

We can see here how a sequence suffix is added to each input's id attribute, and how the `#ids.prev(...)` function allows us to retrieve the last sequence value generated for a specific input id.

Don't worry about those hidden inputs with `name="_features"`: they are automatically added in order to avoid problems with browsers not sending unchecked checkbox values to the server upon form submission.

Also note that if our features property contained some selected values in our form-backing bean, `th:field` would have taken care of that and would have added a `checked="checked"` attribute to the corresponding input tags.

Radio Buttons

Radio button fields are specified in a similar way to non-boolean (multi-valued) checkboxes —except that they are not multivalued, of course:

```
<ul>
  <li th:each="ty : ${allTypes}">
    <input type="radio" th:field="*{type}"
th:value="${ty}" />
    <label th:for="${#ids.prev('type')}"
th:text="#{${'storeGuide.type.' + ty}}">Wireframe</
label>
  </li>
</ul>
```

Dropdown/List Selectors

Select fields have two parts: the `<select>` tag and its nested `<option>` tags. When creating this kind of field, only the `<select>` tag has to include a `th:field` attribute, but the `th:value` attributes in the nested `<option>` tags will be very important because they will provide the means of knowing which is the currently selected option (in a similar way to non-boolean checkboxes and radio buttons).

Let's re-build the type field as a dropdown select:

```
<select th:field="*{type}">
  <option th:each="type : ${allTypes}"
          th:value="${type}"
          th:text="#{${'storeGuide.type.' +
type}}">Wireframe</option>
</select>
```

At this point, understanding this piece of code is quite easy. Just notice how attribute precedence allows us to set the `th:each` attribute in the `<option>` tag itself.

Fragments

4

Learn how to use fragments, reusable pieces of code.

We will often want to include in our templates fragments from other templates. Common uses for this are footers, headers, menus…

In order to do this, Thymeleaf needs us to define the fragments available for inclusion, which we can do by using the **`th:fragment`** attribute.

Now let's say we want to add a standard copyright footer to all our grocery pages, and for that we define a **`/WEB-INF/templates/footer.html`** file containing this code:

```
<!DOCTYPE html SYSTEM
"http://www.thymeleaf.org/dtd/xhtml1-strict-thymeleaf-4.dtd">

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

  <body>

    <div th:fragment="copy">
      &copy; 2018 Some fictional company
    </div>

  </body>

</html>
```

The code above defines a fragment called `copy` that we can easily include in our home page using one of the `th:include` or `th:replace` attributes:

```
<body>

  ...

  <div th:include="footer :: copy"></div>

</body>
```

The syntax for both these inclusion attributes is quite straightforward. There are three different formats:

- **`"templatename::domselector"`** or the equivalent **`templatename::[domselector]`** Includes the fragment resulting from executing the specified DOM Selector on the template named **`templatename`**.
  - Note that **`domselector`** can be a mere fragment name, so you could specify something as simple as **`templatename::fragmentname`** like in the **`footer :: copy`** above.

DOM Selector syntax is similar to XPath expressions and CSS selectors, see the Appendix C for more info on this syntax.

- **`"templatename"`** Includes the complete template named **`templatename`**.

Note that the template name you use in **`th:include`**/**`th:replace`** tags will have to be resolvable by the Template Resolver currently being used by the Template Engine.

- **`::domselector"`** or **`"this::domselector"`** Includes a fragment from the same template.

Both **`templatename`** and **`domselector`** in the above examples can be fully-featured expressions (even conditionals!) like:

```
<div th:include="footer :: (${user.isAdmin}?
#{footer.admin} : #{footer.normaluser})"></div>
```

Fragments can include any `th:* attributes`. These attributes will be evaluated once the fragment is included into the target template (the one with the `th:include`/`th:replace` attribute), and they will be able to reference any context variables defined in this target template.

A big advantage of this approach to fragments is that you can write your fragments' code in pages that are perfectly displayable by a browser, with a complete and even validating XHTML structure, while still retaining the ability to make Thymeleaf include them into other templates.

URLs

Here we cover the basics of URLs in Thymeleaf.

5

Absolute URLs

Completely written URLs like **http://www.thymeleaf.org**

Page-Relative URLs

Page-relative, like **shop/login.html**

Context-Relative URLs

URL based on the current context,a URL would be like **/item?id=5** (context name in server will be added automatically)

Server-Relatve URLs

Relative to server's address, a URL would be like **~/account/viewInvoice** (allows calling URLs in another context (= application) in the same server.

th:href

So, `th:href` is used for URLs and it is an attribute modifier attribute: once processed, it will compute the link URL to be used and set the href attribute of the <a> tag to this URL.

```
<!-- Will produce
'http://localhost:8080/some/order/details?orderId=3'
(plus rewriting) -->
<a href="details.html"

th:href="@{http://localhost:8080/some/order/details(o
rderId=${o.id})}">view</a>

<!-- Will produce '/some/order/details?orderId=3'
(plus rewriting) -->
<a href="details.html"
th:href="@{/order/details(orderId=${o.id})}">view</a>

<!-- Will produce '/some/order/3/details' (plus re-
writing) -->
<a href="details.html"
th:href="@{/order/{orderId}/details(orderId=${o.id})}
">view</a>
```

PagingAndSorting Example

For this tutorial, I will demonstrate how to display a list of a business' clients in Thymeleaf with pagination.

For this tutorial, I will demonstrate how to display a list of a business' clients in Thymeleaf with pagination.

View and Download the code from Github

1 – Project Dependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.michaelcgood</groupId>
    <artifactId>michaelcgood-pagingandsorting</
artifactId>
    <version>0.0.1</version>
    <packaging>jar</packaging>

    <name>PagingAndSortingRepositoryExample</name>
    <description>Michael C  Good -
PagingAndSortingRepository</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</
artifactId>
        <version>1.5.6.RELEASE</version>
        <relativePath /> <!-- lookup parent from reposi-
tory -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.buil
d.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.
reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</ar
tifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</
artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</art
ifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-web</
artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupI
d>
                <artifactId>spring-boot-maven-plugin</art
ifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

For this tutorial, I will demonstrate how to display a list of a business' clients in Thymeleaf with pagination.

View and Download the code from Github

1 – Project Structure

We have a normal Maven project structure.

2 – Project Dependencies

Besides the normal Spring dependencies, we add Thymeleaf and hsqldb because we are using an embedded database.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.michaelcgood</groupId>
    <artifactId>michaelcgood-pagingandsorting</
artifactId>
    <version>0.0.1</version>
    <packaging>jar</packaging>

    <name>PagingAndSortingRepositoryExample</name>
    <description>Michael C  Good -
PagingAndSortingRepository</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</
artifactId>
        <version>1.5.6.RELEASE</version>
        <relativePath /> <!-- lookup parent from reposi-
tory -->
    </parent>

    <properties>
```

```
        <project.build.sourceEncoding>UTF-8</project.buil
d.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.
reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</ar
tifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</
artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</art
ifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</
artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupI
d>
                <artifactId>spring-boot-maven-plugin</art
ifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

3 – Models

We define the following fields for a client:

- a unique identifier
- name of the client
- an address of the client
- the amount owed on the current invoice

The getters and setters are quickly generated in Spring Tool Suite.
The @Entity annotation is needed for registering this model to @SpringBootApplication.

ClientModel.java

```
package com.michaelcgood.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class ClientModel {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public Integer getCurrentInvoice() {
        return currentInvoice;
    }
    public void setCurrentInvoice(Integer currentInvoice)
{
        this.currentInvoice = currentInvoice;
    }
    private String name;
    private String address;
    private Integer currentInvoice;

}
```

The PagerModel is just a POJO (Plain Old Java Object), unlike the ClientModel. There are no imports, hence no annotations. This PagerModel is purely just used for helping with the pagination on our webpage. Revisit this model once you read the Thymeleaf template and see the demo pictures. The PagerModel makes more sense when you think about it in context.

PagerModel.java

```
package com.michaelcgood.model;

public class PagerModel {
    private int buttonsToShow = 5;

    private int startPage;

    private int endPage;

    public PagerModel(int totalPages, int currentPage,
int buttonsToShow) {

        setButtonsToShow(buttonsToShow);
```

```
        int halfPagesToShow = getButtonsToShow() / 2;

        if (totalPages <= getButtonsToShow()) {
            setStartPage(1);
            setEndPage(totalPages);

        } else if (currentPage - halfPagesToShow <= 0) {
            setStartPage(1);
            setEndPage(getButtonsToShow());

        } else if (currentPage + halfPagesToShow == totalPages) {
            setStartPage(currentPage - halfPagesToShow);
            setEndPage(totalPages);

        } else if (currentPage + halfPagesToShow > totalPages) {
            setStartPage(totalPages - getButtonsToShow() + 1);
            setEndPage(totalPages);

        } else {
            setStartPage(currentPage - halfPagesToShow);
            setEndPage(currentPage + halfPagesToShow);
        }

    }

    public int getButtonsToShow() {
        return buttonsToShow;
    }

    public void setButtonsToShow(int buttonsToShow) {
        if (buttonsToShow % 2 != 0) {
            this.buttonsToShow = buttonsToShow;
        } else {
            throw new IllegalArgumentException("Must be an odd value!");
        }
    }

    public int getStartPage() {
        return startPage;
    }

    public void setStartPage(int startPage) {
        this.startPage = startPage;
    }

    public int getEndPage() {
        return endPage;
    }

    public void setEndPage(int endPage) {
        this.endPage = endPage;
    }

    @Override
    public String toString() {
        return "Pager [startPage=" + startPage + ", endPage=" + endPage + "]";
    }

}
```

4 – Repository

The PagingAndSortingRepository is an extension of the CrudRepository. The only difference is that it allows you to do pagination of entities. Notice that we annotate the interface with @Repository to make it visible to @SpringBootApplication.

ClientRepository.java

```
package com.michaelcgood.dao;

import
org.springframework.data.repository.PagingAndSortingRepos
itory;
import org.springframework.stereotype.Repository;

import com.michaelcgood.model.ClientModel;

@Repository
public interface ClientRepository extends PagingAndSortin-
gRepository<ClientModel,Long> {

}
```

5 – Controller

We define some variables in the beginning of the class. We only want to show 3 page buttons at time. The initial page is the first page of results, the initial amount of items on the page is 5, and the user has the ability to have either 5 or 10 results per page.

We add some example values to our repository with the addtorepository() method, which is defined further below in this class.

With the addtorepository method(), we add several “clients” to our repository, and many of them are hat companies because I ran out of ideas.

ModelAndView is used here rather than Model. ModelAndView is used instead because it is a container for both a ModelMap and a view object. It allows the controller to return both as a single value. This is desired for what we are doing.

ClientController.java

```
package com.michaelcgood.controller;

import java.util.Optional;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;
import com.michaelcgood.model.PagerModel;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;

import com.michaelcgood.dao.ClientRepository;
import com.michaelcgood.model.ClientModel;

@Controller
public class ClientController {
```

```
    private static final int BUTTONS_TO_SHOW = 3;
    private static final int INITIAL_PAGE = 0;
    private static final int INITIAL_PAGE_SIZE = 5;
    private static final int[] PAGE_SIZES = { 5, 10};
    @Autowired
    ClientRepository clientrepository;

    @GetMapping("/")
    public ModelAndView homepage(@RequestParam("page-
Size") Optional<Integer> pageSize,
            @RequestParam("page") Optional<Integer>
page){

        if(clientrepository.count()!=0){
            ;//pass
        }else{
            addtorepository();
        }

        ModelAndView modelAndView = new ModelAndView("in-
dex");
        //
        // Evaluate page size. If requested parameter is
null, return initial
        // page size
        int evalPageSize =
pageSize.orElse(INITIAL_PAGE_SIZE);
        // Evaluate page. If requested parameter is null
or less than 0 (to
        // prevent exception), return initial size. Other-
wise, return value of
        // param. decreased by 1.
        int evalPage = (page.orElse(0) < 1) ? INITI-
AL_PAGE : page.get() - 1;
        // print repo
        System.out.println("here is client repo " +
clientrepository.findAll());
        Page<ClientModel> clientlist =
clientrepository.findAll(new PageRequest(evalPage, eval-
PageSize));
        System.out.println("client list get total pages"
+ clientlist.getTotalPages() + "client list get number "
+ clientlist.getNumber());
        PagerModel pager = new
PagerModel(clientlist.getTotalPages(),clientlist.getNumbe
r(),BUTTONS_TO_SHOW);
        // add clientmodel
        modelAndView.addObject("clientlist",clientlist);
        // evaluate page size
        modelAndView.addObject("selectedPageSize", eval-
PageSize);
        // add page sizes
        modelAndView.addObject("pageSizes", PAGE_SIZES);
        // add pager
        modelAndView.addObject("pager", pager);
        return modelAndView;

    }

public void addtorepository(){

        //below we are adding clients to our repository
for the sake of this example
                ClientModel widget = new ClientModel();
                widget.setAddress("123 Fake Street");
                widget.setCurrentInvoice(10000);
```

```
        widget.setName("Widget Inc");

        clientrepository.save(widget);

        //next client
        ClientModel foo = new ClientModel();
        foo.setAddress("456 Attorney Drive");
        foo.setCurrentInvoice(20000);
        foo.setName("Foo LLP");

        clientrepository.save(foo);

        //next client
        ClientModel bar = new ClientModel();
        bar.setAddress("111 Bar Street");
        bar.setCurrentInvoice(30000);
        bar.setName("Bar and Food");
        clientrepository.save(bar);

        //next client
        ClientModel dog = new ClientModel();
        dog.setAddress("222 Dog Drive");
        dog.setCurrentInvoice(40000);
        dog.setName("Dog Food and Accessories");
        clientrepository.save(dog);

        //next client
        ClientModel cat = new ClientModel();
        cat.setAddress("333 Cat Court");
        cat.setCurrentInvoice(50000);
        cat.setName("Cat Food");
        clientrepository.save(cat);

        //next client
```

```
        ClientModel hat = new ClientModel();
        hat.setAddress("444 Hat Drive");
        hat.setCurrentInvoice(60000);
        hat.setName("The Hat Shop");
        clientrepository.save(hat);

        //next client
        ClientModel hatB = new ClientModel();
        hatB.setAddress("445 Hat Drive");
        hatB.setCurrentInvoice(60000);
        hatB.setName("The Hat Shop B");
        clientrepository.save(hatB);

        //next client
        ClientModel hatC = new ClientModel();
        hatC.setAddress("446 Hat Drive");
        hatC.setCurrentInvoice(60000);
        hatC.setName("The Hat Shop C");
        clientrepository.save(hatC);

        //next client
        ClientModel hatD = new ClientModel();
        hatD.setAddress("446 Hat Drive");
        hatD.setCurrentInvoice(60000);
        hatD.setName("The Hat Shop D");
        clientrepository.save(hatD);

        //next client
        ClientModel hatE = new ClientModel();
        hatE.setAddress("447 Hat Drive");
        hatE.setCurrentInvoice(60000);
        hatE.setName("The Hat Shop E");
        clientrepository.save(hatE);
```

```
                //next client
                ClientModel hatF = new ClientModel();
                hatF.setAddress("448 Hat Drive");
                hatF.setCurrentInvoice(60000);
                hatF.setName("The Hat Shop F");
                clientrepository.save(hatF);

    }
}
```

6 – Thymeleaf Template

In Thymeleaf template, the two most important things to note are:

- Thymeleaf Standard Dialect
- Javascript

Like in a CrudRepository, we iterate through the PagingAndSortingRepository with th:each=”clientlist : ${clientlist}”. Except instead of each item in the repository being an Iterable, the item is a Page.

With select class=”form-control pagination” id=”pageSizeSelect”, we are allowing the user to pick their page size of either 5 or 10. We defined these values in our Controller.

Next is the code that allows the user to browse the various pages. This is where our PagerModel comes in to use.

The changePageAndSize() function is the JavaScript function that will update the page size when the user changes it.

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

<head>
<!-- CSS INCLUDE -->
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7
on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous"></link>

<!-- EOF CSS INCLUDE -->
<style>
.pagination-centered {
    text-align: center;
}

.disabled {
    pointer-events: none;
    opacity: 0.5;
}

.pointer-disabled {
    pointer-events: none;
}
</style>

</head>
<body>
```

```
    <!-- START PAGE CONTAINER -->
    <div class="container-fluid">
        <!-- START PAGE SIDEBAR -->
        <!-- commented out     <div
th:replace="fragments/header :: header"> </div> -->
        <!-- END PAGE SIDEBAR -->
        <!-- PAGE TITLE -->
        <div class="page-title">
            <h2>
                <span class="fa fa-arrow-circle-o-
left"></span> Client Viewer
            </h2>
        </div>
        <!-- END PAGE TITLE -->
        <div class="row">
            <table class="table datatable">
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Address</th>
                        <th>Load</th>
                    </tr>
                </thead>
                <tbody>
                    <tr th:each="clientlist : ${cli-
entlist}">
                        <td
th:text="${clientlist.name}">Text ...</td>
                        <td
th:text="${clientlist.address}">Text ...</td>
                        <td><button type="button"
                                class="btn btn-primary
btn-condensed">
                                    <i class="glyphicon
glyphicon-folder-open"></i>
                                </button></td>
                    </tr>
                </tbody>
            </table>
            <div class="row">
                <div class="form-group col-md-1">
                    <select class="form-control pagina-
tion" id="pageSizeSelect">
                        <option th:each="pageSize :
${pageSizes}" th:text="${pageSize}"
                            th:value="${pageSize}"
                            th:selected="${pageSize} ==
${selectedPageSize}"></option>
                    </select>
                </div>
                <div th:if="${clientlist.totalPages !=
1}"
                    class="form-group col-md-11
pagination-centered">
                    <ul class="pagination">
                        <li th:class="${clientlist.number
== 0} ? disabled"><a
                            class="pageLink"
                            th:href="@{/(pageSize=${selec
tedPageSize}, page=1)}">&laquo;</a>
                        </li>
                        <li th:class="${clientlist.number
== 0} ? disabled"><a
                            class="pageLink"
                            th:href="@{/(pageSize=${selec
tedPageSize}, page=${clientlist.number})}">&larr;</a>
                        </li>
```

```
                        <li
                            th:class="${clientlist.number
== (page - 1)} ? 'active pointer-disabled'"
                            th:each="page :
${#numbers.sequence(pager.startPage, pager.endPage)}">
                            <a class="pageLink"
                            th:href="@{/(pageSize=${selec
tedPageSize}, page=${page})}"
                            th:text="${page}"></a>
                        </li>
                        <li
                            th:class="${clientlist.number
+ 1 == clientlist.totalPages} ? disabled">
                            <a class="pageLink"
                            th:href="@{/(pageSize=${selec
tedPageSize}, page=${clientlist.number + 2})}">&rarr;</a>
                        </li>
                        <li
                            th:class="${clientlist.number
+ 1 == clientlist.totalPages} ? disabled">
                            <a class="pageLink"
                            th:href="@{/(pageSize=${selec
tedPageSize},
page=${clientlist.totalPages})}">&raquo;</a>
                        </li>
                    </ul>
                </div>
            </div>
        </div>
        <!-- END PAGE CONTENT -->
        <!-- END PAGE CONTAINER -->
    </div>
        <script
  src="https://code.jquery.com/jquery-1.11.1.min.js"
  integrity="sha256-VAvG3sHdS5LqTT+5A/aeq/bZGa/Uj04xKxY8K
M/w9EE="
  crossorigin="anonymous"></script>

    <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.
3.7/js/bootstrap.min.js"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZx
UPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
        crossorigin="anonymous"></script>
    <script th:inline="javascript">
        /*<![CDATA[*/
        $(document).ready(function() {
    changePageAndSize();
});

function changePageAndSize() {
    $('#pageSizeSelect').change(function(evt) {
        window.location.replace("/?pageSize=" +
this.value + "&page=1");
    });
}
        /*]]>*/
    </script>

</body>
</html>
```

7 – Configuration

The below properties can be changed based on your preferences but were what I wanted for my environment.

application.properties

```
#==================================
# = Thymeleaf configurations
#==================================
spring.thymeleaf.check-template-location=true
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.thymeleaf.content-type=text/html
spring.thymeleaf.cache=false
server.contextPath=/
```

8 – Demo

Visit *localhost:8080* and see how you can change page size and page number :-).

Validation in Thymeleaf Example

Here we build an example application that validates input.

Overview

Important topics we will be discussing are dealing with null values, empty strings, and validation of input so we do not enter invalid data into our database.

In dealing with null values, we touch on use of *java.util.Optional* which was introduced in Java 1.8.

0 – Spring Boot + Thymeleaf Example Form Validation Application

We are building a web application for a university that allows potential students to request information on their programs.

View and Download the code from Github

1 – Project Dependencies

Besides our typical Spring Boot dependencies, we are using an embedded HSQLDB database and nekohtml for LEGACYHTML5 mode.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.michaelcgood</groupId>
    <artifactId>michaelcgood-validation-thymeleaf</
artifactId>
    <version>0.0.1</version>
    <packaging>jar</packaging>

    <name>michaelcgood-validation-thymeleaf</name>
    <description>Michael C  Good - Validation in Thyme-
leaf Example Application</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</
artifactId>
        <version>1.5.7.RELEASE</version>
        <relativePath/> <!-- lookup parent from reposi-
tory -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.buil
d.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.
reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</art
ifactId>
```

```
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</ar
tifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</
artifactId>
        </dependency>

        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</
artifactId>
            <scope>test</scope>
        </dependency>
        <!-- legacy html allow -->
        <dependency>
            <groupId>net.sourceforge.nekohtml</groupId>
            <artifactId>nekohtml</artifactId>
            <version>1.9.21</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupI
d>
                <artifactId>spring-boot-maven-plugin</art
ifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

3 – Model

In our model we define:

- An autogenerated id field
- A name field that cannot be null
- That the name must be between 2 and 40 characters
- An email field that is validated by the *@Email* annotation
- A boolean field “openhouse” that allows a potential student to indicate if she wants to attend an open house
- A boolean field “subscribe” for subscribing to email updates
- A comments field that is optional, so there is no minimum character requirement but there is a maximum character requirement

```
package com.michaelcgood.model;

import javax.persistence.Entity;
```

```java
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import org.hibernate.validator.constraints.Email;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @NotNull
    @Size(min=2, max=40)
    private String name;
    @NotNull
    @Email
    private String email;
    private Boolean openhouse;
    private Boolean subscribe;
     @Size(min=0, max=300)
    private String  comments;

    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Boolean getOpenhouse() {
        return openhouse;
    }
    public void setOpenhouse(Boolean openhouse) {
        this.openhouse = openhouse;
    }
    public Boolean getSubscribe() {
        return subscribe;
    }
    public void setSubscribe(Boolean subscribe) {
        this.subscribe = subscribe;
    }
    public String getComments() {
        return comments;
    }
    public void setComments(String comments) {
        this.comments = comments;
    }

}
```

4 – Repository

We define a repository.

```
package com.michaelcgood.dao;

import
org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.michaelcgood.model.Student;

@Repository
public interface StudentRepository extends JpaReposito-
ry<Student,Long> {

}
```

5 – Controller

We register *StringTrimmerEditor* to convert empty Strings to null values automatically.

When a user sends a POST request, we want to receive the value of that Student object, so we use *@ModelAttribute* to do just that.

To ensure that the user is sending values that are valid, we use the appropriately named *@Valid* annotation next.

BindingResult must follow next, or else the user is given an error page when submitting invalid data instead of remaining on the form page.

We use *if…else* to control what happens when a user submits a form. If the user submits invalid data, the user will remain on the current page and nothing more will occur on the server side. Otherwise, the application will consume the user's data and the user can proceed.

At this point, it is kind of redundant to check if the student's name is null, but we do. Then, we call the method *checkNullString*, which is defined below, to see if the comment field is an empty String or null.

```
package com.michaelcgood.controller;

import java.util.Optional;

import javax.validation.Valid;

import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.beans.propertyeditors.StringTrimmerEd
itor;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.WebDataBinder;
import
org.springframework.web.bind.annotation.GetMapping;
```

```
import
org.springframework.web.bind.annotation.InitBinder;
import
org.springframework.web.bind.annotation.ModelAttribute;
import
org.springframework.web.bind.annotation.PostMapping;
import com.michaelcgood.dao.StudentRepository;
import com.michaelcgood.model.Student;

@Controller
public class StudentController {
    @InitBinder
    public void initBinder(WebDataBinder binder) {
        binder.registerCustomEditor(String.class, new
StringTrimmerEditor(true));
    }
    public String finalString = null;
    @Autowired
    private StudentRepository studentRepository;
    @PostMapping(value="/")
    public String addAStudent(@ModelAttribute @Valid Stu-
dent newStudent, BindingResult bindingResult, Model
model){
        if (bindingResult.hasErrors()) {
            System.out.println("BINDING RESULT ERROR");
            return "index";
        } else {
            model.addAttribute("student", newStudent);

            if (newStudent.getName() != null) {
                try {
                    // check for comments and if not pre-
sent set to 'none'
                    String comments =
checkNullString(newStudent.getComments());
                    if (comments != "None") {
                        System.out.println("nothing
changes");
                    } else {
                        newStudent.setComments(comments);
                    }
                } catch (Exception e) {

                    System.out.println(e);

                }
                studentRepository.save(newStudent);
                System.out.println("new student added: "
+ newStudent);
            }

            return "thanks";
        }
    }

    @GetMapping(value="thanks")
    public String thankYou(@ModelAttribute Student newStu-
dent, Model model){
        model.addAttribute("student",newStudent);

        return "thanks";
    }

    @GetMapping(value="/")
    public String viewTheForm(Model model){
        Student newStudent = new Student();
        model.addAttribute("student",newStudent);
```

```
        return "index";
    }

    public String checkNullString(String str){
        String endString = null;
        if(str == null || str.isEmpty()){
            System.out.println("yes it is empty");
            str = null;
            Optional<String> opt =
Optional.ofNullable(str);
            endString = opt.orElse("None");
            System.out.println("endString : " + end-
String);
        }
        else{
            ; //do nothing
        }


        return endString;

    }
}
```

Optional.ofNullable(str); means that the String will become the data type Optional, but the String may be a null value.

endString = opt.orElse("None"); sets the String value to "None" if the variable *opt* is null.

6 – Thymeleaf Templates

As you saw in our Controller's mapping above, there are two pages. The *index.html* is our main page that has the form for potential University students.

Our main object is Student, so of course our *th:object* refers to that. Our model's fields respectively go into *th:field*.

We wrap our form's inputs inside a table for formatting purposes.

Below each table cell (td) we have a conditional statement like this one: *[...]*
th:if="${#fields.hasErrors('name')}" th:errors="{name}"*
[...]

The above conditional statement means if the user inputs data into that field that doesn't match the requirement we put for that field in our Student model and then submits the form, show the input requirements when the user is returned to this page.

index.html

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

<head>
```

```
<!-- CSS INCLUDE -->
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7
on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous"></link>

<!-- EOF CSS INCLUDE -->
</head>
<body>

    <!-- START PAGE CONTAINER -->
    <div class="container-fluid">
        <!-- PAGE TITLE -->
        <div class="page-title">
            <h2>
                <span class="fa fa-arrow-circle-o-
left"></span> Request University
                Info
            </h2>
        </div>
        <!-- END PAGE TITLE -->
        <div class="column">
            <form action="#" th:action="@{/}"
th:object="${student}"
                method="post">
                <table>
                    <tr>
                        <td>Name:</td>
                        <td><input type="text"
th:field="*{name}"></input></td>
                        <td
th:if="${#fields.hasErrors('name')}"
th:errors="*{name}">Name
                            Error</td>
                    </tr>
                    <tr>
                        <td>Email:</td>
                        <td><input type="text"
th:field="*{email}"></input></td>
                        <td
th:if="${#fields.hasErrors('email')}"
th:errors="*{email}">Email
                            Error</td>
                    </tr>
                    <tr>
                        <td>Comments:</td>
                        <td><input type="text"
th:field="*{comments}"></input></td>
                    </tr>
                    <tr>
                        <td>Open House:</td>
                        <td><input type="checkbox"
th:field="*{openhouse}"></input></td>

                    </tr>
                    <tr>
                        <td>Subscribe to updates:</td>
                        <td><input type="checkbox"
th:field="*{subscribe}"></input></td>

                    </tr>
                    <tr>
                        <td>
```

```
                            <button type="submit"
class="btn btn-primary">Submit</button>
                        </td>
                    </tr>
                </table>
            </form>

        </div>
        <!-- END PAGE CONTENT -->
        <!-- END PAGE CONTAINER -->
    </div>
    <script
src="https://code.jquery.com/jquery-1.11.1.min.js"
        integrity="sha256-VAvG3sHdS5LqTT+5A/aeq/bZGa/Uj04
xKxY8KM/w9EE="
        crossorigin="anonymous"></script>


    <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.
3.7/js/bootstrap.min.js"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZx
UPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
        crossorigin="anonymous"></script>


</body>
</html>
```

Here we have the page that a user sees when they have successfully completed the form. We use *th:text* to show the user the text he or she input for that field.

thanks.html

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

<head>
<!-- CSS INCLUDE -->
<link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7
/css/bootstrap.min.css"
    integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7
on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous"></link>

<!-- EOF CSS INCLUDE -->


</head>
<body>

    <!-- START PAGE CONTAINER -->
    <div class="container-fluid">

        <!-- PAGE TITLE -->
        <div class="page-title">
            <h2>
                <span class="fa fa-arrow-circle-o-
left"></span> Thank you
            </h2>
        </div>
        <!-- END PAGE TITLE -->
        <div class="column">
            <table class="table datatable">
```

```
                <thead>
                    <tr>
                        <th>Name</th>
                        <th>Email</th>
                        <th>Open House</th>
                        <th>Subscribe</th>
                        <th>Comments</th>
                    </tr>
                </thead>
                <tbody>
                    <tr th:each="student : ${student}">
                        <td
th:text="${student.name}">Text ...</td>
                        <td
th:text="${student.email}">Text ...</td>
                        <td
th:text="${student.openhouse}">Text ...</td>
                        <td
th:text="${student.subscribe}">Text ...</td>
                        <td
th:text="${student.comments}">Text ...</td>
                    </tr>
                </tbody>
            </table>
        </div>
        </div>
        <!-- END PAGE CONTAINER -->
    </div>
        <script
  src="https://code.jquery.com/jquery-1.11.1.min.js"
  integrity="sha256-VAvG3sHdS5LqTT+5A/aeq/bZGa/Uj04xKxY8K
M/w9EE="
  crossorigin="anonymous"></script>

    <script
        src="https://maxcdn.bootstrapcdn.com/bootstrap/3.
3.7/js/bootstrap.min.js"
        integrity="sha384-Tc5IQib027qvyjSMfHjOMaLkfuWVxZx
UPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
        crossorigin="anonymous"></script>

</body>
</html>
```

7 – Configuration

Using Spring Boot Starter and including Thymeleaf dependencies, you will automatically have a templates location of */templates/*, and Thymeleaf just works out of the box. So most of these settings aren't needed.

The one setting to take note of is LEGACYHTM5 which is provided by nekohtml. This allows us to use more casual HTML5 tags if we want to. Otherwise, Thymeleaf will be very strict and may not parse your HTML. For instance, if you do not close an *input* tag, Thymeleaf will not parse your HTML.

application.properties

```
#===================================
# = Thymeleaf configurations
#===================================
spring.thymeleaf.check-template-location=true
spring.thymeleaf.prefix=classpath:/templates/
```

```
spring.thymeleaf.suffix=.html
spring.thymeleaf.content-type=text/html
spring.thymeleaf.cache=false
spring.thymeleaf.mode=LEGACYHTML5

server.contextPath=/
```

8 – Demo

Visit *localhost:8080* to get to the homepage.

I input invalid data into the name field and email field.

Now I put valid data in all fields, but do not provide a comment. It is not required to provide a comment. In our controller, we made all empty Strings null values. If the user did not provide a comment, the String value is made “None”.

9 – Conclusion

Wrap up

This demo application demonstrated how to valid user input in a Thymeleaf form.
In my opinion, Spring and Thymeleaf work well with *javax.validation.constraints* for validating user input.
The source code is on Github

Notes

Java 8’s Optional was sort of forced into this application for demonstration purposes, and I want to note it works more organically when using *@RequestParam* as shown in my PagingAndSortingRepository tutorial.

However, if you were not using Thymeleaf, you could have possibly made our not required fields *Optional*. Here Vlad Mihalcea discusses the best way to map Optional entity attribute with JPA and Hibernate.

AJAX with CKEditor Example

Here we build an application that uses CKEditor and in the process do AJAX with Thymeleaf and Spring Boot.

1. Overview

In this article, we will cover **how to use CKEditor with Spring Boot**. In this tutorial, we will be importing an XML document with numerous data, program the ability to load a set of data to the CKEditor instance with a GET request, and do a POST request to save the CKEditor's data.

Technologies we will be using include MongoDB, Thymeleaf, and Spring Batch.

The full source code for this tutorial is available on Github.

2. What is CKEditor?

CKEditor is a **browser-based What-You-See-Is-What-You-Get (WYSIWYG) content editor**. CKEditor aims to bring to a web interface common word processor features found in desktop editing applications like Microsoft Word and OpenOffice.

CKEditor has numerous features for end users in regards to the user interface, inserting content, authoring content, and more.

There are different versions of CKEditor, but for this tutorial we are using CKEditor 4. To see a demo,
visit: https://ckeditor.com/ckeditor-4/

3. The XML Document

As mentioned, we are uploading an XML document in this application. The XML data will be inserted into the database and used for the rest of the tutorial.

```
<?xml version="1.0"?>
<Music
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
id="MUS-1" style="1.1">
<status date="2017-11-07">draft</status>
 <title xmlns:xhtml="http://www.w3.org/1999/xhtml" >Guide
to Music I Like - No Specific Genre</title>
  <description xmlns:xhtml="http://www.w3.org/1999/xhtml"
>This guide presents a catalog of music that can be found
on Spotify.
    <html:br xmlns:html="http://www.w3.org/1999/xhtml"/>
    <html:br xmlns:html="http://www.w3.org/1999/xhtml"/>
    This is a very small sample of music found on Spotify
and is no way to be considered comprehensive.
    </description>
    <songs>
    <song>
    <artist>
    Run the Jewels
    </artist>
    <song-title>Legend Has It</song-title>
    </song>
    <song>
    <artist>
    Kendrick Lamar
    </artist>
    <song-title>ELEMENT.</song-title>
```

```
    </song>
    <song>
    <artist>
    Weird Al Yankovic
    </artist>
    <song-title>NOW That's What I Call Polka!</song-
title>
    </song>
    <song>
    <artist>
    Eiffel 65
    </artist>
    <song-title>Blue (Da Ba Dee) - DJ Ponte Ice Pop
Radio</song-title>
    </song>
    <song>
    <artist>
    YTCracker
    </artist>
    <song-title>Hacker Music</song-title>
    </song>
    <song>
    <artist>
    MAN WITH A MISSION
    </artist>
    <song-title>
    Raise Your Flag
    </song-title>
    </song>
    <song>
    <artist>
    GZA, Method Man
    </artist>
    <song-title>
    Shadowboxin'
    </song-title>
    </song>
    </songs>
</Music>
```

4. Model

For the above XML code, we can model a Song like this:

```
public class SongModel {
    @Id
    private String id;
    @Indexed
    private String artist;
    @Indexed
    private String songTitle;
    @Indexed
    private Boolean updated;

   // standard getters and setters
```

For our application, we will be differentiating between an un-modified song and a song that has been modified in CKEditor with a separate Model and Repository.

Let's now define what an Updated Song is:

```
public class UpdatedSong {
```

```
    @Id
    private String id;
    @Indexed
    private String artist;
    @Indexed
    private String songTitle;
    @Indexed
    private String html;
    @Indexed
    private String sid;

   // standard getters and setters
```

5. File Upload and Processing

As this article's focus is on CKEditor and AJAX, we aren't going to go into detail on the file upload and processing with Spring Batch. We have reviewed this process in depth in these prior posts, however:

- Spring Batch CSV Processing
- Converting XML to JSON + Raw Use in MongoDB + Spring Batch

6. Setting Data to CKEditor – GET Request

It is our goal to retrieve the data for an individual song and display that data in CKEditor. There are two issues to tackle: retrieving the data for an individual song and displaying it in CKEditor.

6.1 Client Side Code

In *view.html,* we **use a table in Thymeleaf to iterate** through each *Song* in the Song repository. To be able to retrieve the data from the server for a specific song, we pass in the Song's *id* to a function.

Here's the snippet of code that is responsible for calling the function that retrieves the data from the server and subsequently **sets the data to the CKEditor instance**:

```
<table class="table datatable">
<thead>
<tr>
<th>Artist</th>
<th>Song Title</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr th:each="songList : ${songList}">
<td th:text="${songList.artist}">Text ...</td>
<td th:text="${songList.songTitle}">Text ...</td>
<td><button th:onclick="|getSong('${songList.id}')|"
id="button" class="btn btn-primary btn-condensed">
<i class="glyphicon glyphicon-folder-open"></i>
</button></td>
</tr>
</tbody>
</table>
```

As we can see, the *id* of *Song* is essential for us to be able to retrieve the data.

In the *getSong* function, we use a deferred promise to ensure that *data* is set after the GET request:

```
function getSong(song) {
    $.ajax({
        url : "/api/show/?sid=" + song,
        type : 'GET',
        dataType : 'text'
    }).then(function(data) {
        var length = data.length-2;
        var datacut = data.slice(9,length);
        CKEDITOR.instances.content.setData(datacut);

    });

    $("#form").attr("action", "/api/save/?sid=" + song);

};
```

6.2 Server Side Code

getSong accepts a parameter named *sid*, which stands for Song id. *sid* is also a path variable in the *@GetMapping*. We treat *sid* as a String because this is the *id* of the Song from MongoDB.

We check if the Song has been modified and if so we retrieve the associated *UpdatedSong* entity. If not, we treat the Song differently. Ultimately, we return a simple POJO with a String for data named *ResponseModel*, however:

```
@GetMapping(value={"/show/","/show/{sid}"})
public ResponseEntity<?> getSong(@RequestParam String
sid, Model model){
    ResponseModel response = new ResponseModel();
    System.out.println("SID :::::" + sid);
    ArrayList<String> musicText = new ArrayList<S-
tring>();
    if(sid!=null){
        String sidString = sid;
        SongModel songModel = songDAO.findOne(sidString);
        System.out.println("get status of boolean during
get ::::::" + songModel.getUpdated());
        if(songModel.getUpdated()==false ){

            musicText.add(songModel.getArtist());
            musicText.add(songModel.getSongTitle());
            String filterText =
format.changeJsonToHTML(musicText);
            response.setData(filterText);

        } else if(songModel.getUpdated()==true){
            UpdatedSong updated =
updatedDAO.findBysid(sidString);
            String text = updated.getHtml();
            System.out.println("getting the updated text
::::::::" + text);
            response.setData(text);
        }

    }
```

```
    model.addAttribute("response", response);

    return ResponseEntity.ok(response);
}
```

ResponseModel is a very simple POJO as mentioned:

```
public class ResponseModel {
    private String data;

    public ResponseModel(){

    }

    public ResponseModel(String data){
            this.data = data;
    }

    public String getData() {
            return data;
    }

    public void setData(String data) {
            this.data = data;
    }
}
```

7. Saving CKEditor Data – POST Request

POSTing the data isn't much of a challenge; however, ensuring that the data is handled appropriately can be.

7.1 Client Side Code

As the CKEditor instance is a textarea within a form, we can trigger a function on a form submit:

```
$(document)
.ready(
function() {

// SUBMIT FORM
$("#form").submit(function(event) {
// Prevent the form from submitting via the browser.
event.preventDefault();
ajaxPost();
});
```

ajaxPost() retrieves the current data in the CKEditor and sets it to the variable *formData*:

```
function ajaxPost() {

// PREPARE FORM DATA
var formData = CKEDITOR.instances.content
.getData();

// DO POST
$
.ajax({
type : "POST",
contentType : "text/html",
url : $("#form").attr("action"),
data : formData,
dataType : 'text',
```

```
success : function(result) {

$("#postResultDiv")
.html(
"

"
+ "Post Successfully! "
+ "

");

console.log(result);
},
error : function(e) {
alert("Error!")
console.log("ERROR: ", e);
}
});

}

})
```

It is important to note:

- *contentType* is *"text/html"*
- *dataType* is *"text"*

Having the incorrect contentType or dataType can lead to errors or malformed data.

7.2 Server Side Code

We stated in our *contentType* for the POST request that the mediatype is *"text/html"*. We need to specify in our mapping that this will be consumed. Therefore, we add *consumes = MediaType.TEXT_HTML_VALUE* with our *@PostMapping*.

Areas for us to note include:

- *@RequestBody String body* is responsible for setting the variable *body* to the content of our request
- We once again return *ResponseModel*, the simple POJO described earlier that contains our data
- We treat a previously modified *SongModel* different than one that has not been modified before

Also, like the GET request, the *sid* allows us to deal with the correct Song:

```
@PostMapping(value={"/save/","/save/[sid]"}, consumes =
MediaType.TEXT_HTML_VALUE)
   public @ResponseBody ResponseModel saveSong( @Request-
Body String body, @RequestParam String sid){
       ResponseModel response = new ResponseModel();
       response.setData(body);
       SongModel oldSong = songDAO.findOne(sid);
       String songTitle = oldSong.getSongTitle();
       String artistName = oldSong.getArtist();
       if(oldSong.getUpdated() == false){
           UpdatedSong updatedSong = new UpdatedSong();
           updatedSong.setArtist(artistName);
           updatedSong.setSongTitle(songTitle);
           updatedSong.setHtml(body);
           updatedSong.setSid(sid);
```

```
            oldSong.setUpdated(true);
            songDAO.save(oldSong);
            updatedDAO.insert(updatedSong);
            System.out.println("get status of boolean dur-
ing post :::::" + oldSong.getUpdated());
        }else{
            UpdatedSong currentSong =
updatedDAO.findBysid(sid);
            currentSong.setHtml(body);
            updatedDAO.save(currentSong);
        }

        return response;
    }
```

8. Demo

We visit *localhost:8080*:

We upload the provided *music-example.xml* file:

We click “Load” for a song:

We add content and click “Save”:

If you return to the saved content, you may see “\n”for line breaks. For the time being, discussing this is out of the scope for the tutorial.

9. Conclusion

In this tutorial, we covered how to load data using a GET request with the object’s id, set the data to the CKEditor instance, and save the CKEditor’s data back to the database with a POST request. There’s extra code, such as using two different entities for the data (an original and a modified version), that isn’t necessary, but hopefully is instructive.

The full code can be found on Github.

Redis with Spring Boot Example

Here we build an application that uses Redis, Thymeleaf and Spring Boot.

1. Overview

In this article, we will review the basics of how to use **Redis with Spring Boot** through the **Spring Data Redis** library.

We will build an application that demonstrates how to perform **CRUD operations Redis** through a web interface. The full source code for this project is available on Github.

2. What is Redis?

Redis is an open source, in-memory key-value data store, used as a database, cache and message broker. In terms of implementation, Key Value stores represent one of the largest and oldest members in the NoSQL space. Redis supports data structures such as strings, hashes, lists, sets, and sorted sets with range queries.

The Spring Data Redis framework makes it easy to write Spring applications that use the Redis key value store by providing an abstraction to the data store.

3. Setting Up A Redis Server

The server is available for free at http://redis.io/download.

If you use a Mac, you can install it with homebrew:

```
brew install redis
```

Then start the server:

```
mikes-MacBook-Air:~ mike$ redis-server
10699:C 23 Nov 08:35:58.306 # oO0OoO0OoO0Oo Redis is
starting oO0OoO0OoO0Oo
10699:C 23 Nov 08:35:58.307 # Redis version=4.0.2,
bits=64, commit=00000000, modified=0, pid=10699, just
started
10699:C 23 Nov 08:35:58.307 # Warning: no config file
specified, using the default config. In order to specify
a config file use redis-server /path/to/redis.conf
10699:M 23 Nov 08:35:58.309 * Increased maximum number of
open files to 10032 (it was originally set to 256).
                _.-``__
''-._
      _.-``    `.  `_.  ''-._           Redis 4.0.2
(00000000/0) 64 bit
  .-`` .-```.  ```\/    _.,_
''-._
 (    '      ,       .-`  | `,    )     Running in standa-
lone mode
 |`-._`-...-` __...-.``-._|'` _.-'|     Port: 6379
 |    `-._   `._    /     _.-'    |     PID: 10699
  `-._    `-._  `-./  _.-'
_.-'
 |`-._`-._    `-.__.-'
_.-'_.-'|
 |    `-._`-._        _.-'_.-'    |
http://redis.io
```

```
  `-._    `-._`-.__.-'_.-'
_.-'
 |`-._`-._    `-.__.-'
_.-'_.-'|
 |    `-._`-._        _.-'_.-'
|
  `-._    `-._`-.__.-'_.-'
_.-'
      `-._    `-.__.-'
_.-'
          `-._
_.-'
              `-.__.-'


10699:M 23 Nov 08:35:58.312 # Server initialized
10699:M 23 Nov 08:35:58.312 * Ready to accept connections
```

4. Maven Dependencies

Let's declare the necessary dependencies in our *pom.xml* for the example application we are building:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</
artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</
artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

5. Redis Configuration

We need to connect our application with the Redis server. To establish this connection, we are using Jedis, a Redis client implementation.

5.1 Config

Let's start with the configuration bean definitions:

```
@Bean
JedisConnectionFactory jedisConnectionFactory() {
    return new JedisConnectionFactory();
}

@Bean
public RedisTemplate<String, Object> redisTemplate() {
    final RedisTemplate<String, Object> template = new Re-
disTemplate<String, Object>();
    template.setConnectionFactory(jedisConnectionFactory(
));
    template.setValueSerializer(new
GenericToStringSerializer<Object>(Object.class));
    return template;
}
```

The *JedisConnectionFactory* is made into a bean so we can create a *RedisTemplate* to query data.

5.2 Message Publisher

Following the principles of SOLID, we create a *MessagePublisher* interface:

```
public interface MessagePublisher {

    void publish(final String message);
}
```

We implement the *MessagePublisher* interface to use the high-level RedisTemplate to publish the message since the RedisTemplate allows arbitrary objects to be passed in as messages:

```
@Service
public class MessagePublisherImpl implements MessagePub-
lisher {

    @Autowired
    private RedisTemplate<String, Object> redisTemplate;
    @Autowired
    private ChannelTopic topic;

    public MessagePublisherImpl() {
    }

    public MessagePublisherImpl(final RedisTemplate<S-
tring, Object> redisTemplate, final ChannelTopic topic) {
        this.redisTemplate = redisTemplate;
        this.topic = topic;
    }

    public void publish(final String message) {
        redisTemplate.convertAndSend(topic.getTopic(),
message);
    }

}
```

We also define this as a bean in *RedisConfig*:

```
@Bean
MessagePublisher redisPublisher() {
    return new MessagePublisherImpl(redisTemplate(),
topic());
}
```

6. RedisRepository

Now that we have configured the application to interact with the Redis server, we are going to prepare the application to take example data.

6.1 Model

For this example, we defining a *Movie* model with two fields:

```
private String id;
private String name;
//standard getters and setters
```

6.2 Repository interface

Unlike other Spring Data projects, **Spring Data Redis does offer any features to build on top of the other Spring Data interfaces**. This is odd for us who have experience with the other Spring Data projects.

Often there is no need to write an implementation of a repository interface with Spring Data projects. We simply just interact with the interface. Spring Data JPA provides numerous repository interfaces that can be extended to get features such as CRUD operations, derived queries, and paging.

So, unfortunately, **we need to write our own interface** and then define the methods:

```
public interface RedisRepository {

    Map<Object, Object> findAllMovies();

    void add(Movie movie);

    void delete(String id);

    Movie findMovie(String id);

}
```

6.3 Repository implementation

Our implementation class uses the *redisTemplate* defined in our configuration class *RedisConfig*.

We use the *HashOperations* template that Spring Data Redis offers:

```
@Repository
public class RedisRepositoryImpl implements RedisReposi-
tory {
    private static final String KEY = "Movie";

    private RedisTemplate<String, Object> redisTemplate;
    private HashOperations hashOperations;

    @Autowired
    public RedisRepositoryImpl(RedisTemplate<String, Ob-
ject> redisTemplate){
        this.redisTemplate = redisTemplate;
    }

    @PostConstruct
    private void init(){
        hashOperations = redisTemplate.opsForHash();
    }

    public void add(final Movie movie) {
        hashOperations.put(KEY, movie.getId(),
movie.getName());
    }

    public void delete(final String id) {
        hashOperations.delete(KEY, id);
    }
```

```
    public Movie findMovie(final String id){
        return (Movie) hashOperations.get(KEY, id);
    }

    public Map<Object, Object> findAllMovies(){
        return hashOperations.entries(KEY);
    }
}
```

Let's take note of the *init()* method. In this method, we use a function named *opsForHash(),* which returns the operations performed on hash values bound to the given key. We then use the *hashOps*, which was defined in init(), for all our CRUD operations.

7. Web interface

In this section, we will review adding Redis CRUD operations capabilities to a web interface.

7.1 Add A Movie

We want to be able to add a Movie in our web page. The Key is the is the Movie *id* and the Value is the actual object. However, we will later address this so only the Movie name is shown as the value.

So, let's add a form to a HTML document and assign appropriate names and ids :

```
<form id="addForm">
<div class="form-group">
                    <label for="keyInput">Movie ID
(key)</label>
                    <input name="keyInput" id="keyInput"
class="form-control"/>
                </div>
<div class="form-group">
                    <label for="valueInput">Movie Name
(field of Movie object value)</label>
                    <input name="valueInput" id="valueI-
nput" class="form-control"/>
                </div>
                <button class="btn btn-default"
id="addButton">Add</button>
            </form>
```

Now we use JavaScript to persist the values on form submission:

```
$(document).ready(function() {
    var keyInput = $('#keyInput'),
        valueInput = $('#valueInput');

    refreshTable();
    $('#addForm').on('submit', function(event) {
        var data = {
            key: keyInput.val(),
            value: valueInput.val()
        };

        $.post('/add', data, function() {
            refreshTable();
            keyInput.val('');
            valueInput.val('');
```

```
            keyInput.focus();
        });
        event.preventDefault();
    });

    keyInput.focus();
});
```

We assign the *@RequestMapping* value for the POST request, request the key and value, create a *Movie* object, and save it to the repository:

```
@RequestMapping(value = "/add", method =
RequestMethod.POST)
public ResponseEntity<String> add(
    @RequestParam String key,
    @RequestParam String value) {

    Movie movie = new Movie(key, value);

    redisRepository.add(movie);
    return new ResponseEntity<>(HttpStatus.OK);
}
```

7.2 Viewing the content

Once a *Movie* object is added, we refresh the table to display an updated table. In our JavaScript code block for section 7.1, we called a JavaScript function called *refreshTable()*. This function performs a GET request to retrieve the current data in the repository:

```
function refreshTable() {
    $.get('/values', function(data) {
        var attr,
            mainTable = $('#mainTable tbody');
        mainTable.empty();
        for (attr in data) {
            if (data.hasOwnProperty(attr)) {
                mainTable.append(row(attr, data[attr]));
            }
        }
    });}
```

The GET request is processed by a method named *findAll()* that retrieves all the *Movie* objects stored in the repository and then converts the datatype from *Map<Object, Object>* to *Map<String, String>*:

```
@RequestMapping("/values")
public @ResponseBody Map<String, String> findAll() {
    Map<Object, Object> aa =
redisRepository.findAllMovies();
    Map<String, String> map = new HashMap<String,
String>();
    for(Map.Entry<Object, Object> entry : aa.entrySet()){
        String key = (String) entry.getKey();
        map.put(key, aa.get(key).toString());
    }
    return map;
}
```

7.3 Delete a Movie

We write Javascript to do a POST request to */delete*, refresh the table, and set keyboard focus to key input:

```
function deleteKey(key) {
```

```
    $.post('/delete', {key: key}, function() {
        refreshTable();
        $('#keyInput').focus();
    });
}
```

We request the key and delete the object in the *redisRepository* based on this key:

```
@RequestMapping(value = "/delete", method =
RequestMethod.POST)
public ResponseEntity<String> delete(@RequestParam String
key) {
    redisRepository.delete(key);
    return new ResponseEntity<>(HttpStatus.OK);
}
```

8. Demo

Visit *localhost:8080* and provide values for Movie Id and and Movie Name. You will see it is added on the right side of the page and then you can delete the entrees too.

9. Conclusion

In this tutorial, we introduced Spring Data Redis and one way of connecting it to a web application to perform CRUD operations.

The source code for the example application is on Github.

HTML to Microsoft Excel Example

10

Here we build an application that has a Thymeleaf interface, takes input, and converts HTML to rich text for Microsoft Excel.

1. Overview

In this tutorial, we will be building an application that takes HTML as an input and creates a Microsoft Excel Workbook with a **RichText representation of the HTML** that was provided. To generate the Microsoft Excel Workbook, we will be using **Apache POI**. To analyze the HTML, we will be using Jericho.

The full source code for this tutorial is available on Github.

2. What is Jericho?

Jericho is a java library that allows analysis and manipulation of parts of an HTML document, including server-side tags, while reproducing verbatim any unrecognized or invalid HTML. It also provides high-level HTML form manipulation functions. It is an open source library released under the following licenses: Eclipse Public License (EPL), GNU Lesser General Public License (LGPL), and Apache License.

I found Jericho to be very easy to use for achieving my goal of converting HTML to RichText.

3. pom.xml

Here are the required dependencies for the application we are building. Please take note that for this application we have to use **Java 9**. This is because of a java.util.regex appendReplacement method we use that has only been available since Java 9.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.9.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository
-->
</parent>

<properties>
    <project.build.sourceEncoding>UTF-8</project.build.so
urceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.repo
rting.outputEncoding>
    <java.version>9</java.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</
artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</
artifactId>
    </dependency>

    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
```

```
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <!--
https://mvnrepository.com/artifact/org.apache.commons/com
mons-lang3 -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.7</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.batch</groupId>
        <artifactId>spring-batch-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi</artifactId>
        <version>3.15</version>
    </dependency>

    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>3.15</version>
    </dependency>
    <!--
https://mvnrepository.com/artifact/net.htmlparser.jericho
/jericho-html -->
    <dependency>
        <groupId>net.htmlparser.jericho</groupId>
        <artifactId>jericho-html</artifactId>
        <version>3.4</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-configuration-processor</
artifactId>
        <optional>true</optional>
    </dependency>
    <!-- legacy html allow -->
    <dependency>
        <groupId>net.sourceforge.nekohtml</groupId>
        <artifactId>nekohtml</artifactId>
    </dependency>
</dependencies>
```

Web Page: Thymeleaf

We use Thymeleaf to create a basic webpage that has a form with a textarea. The source code for Thymeleaf page is available here on GitHub. This textarea could be replaced with a RichText Editor if we like, such as CKEditor. We just must be mindful to make the *data* for AJAX correct, using an appropriate *setData* method. There is a previous tutorial about CKeditor titled AJAX with CKEditor in Spring Boot.

Controller

In our controller, we Autowire *JobLauncher* and a Spring Batch job we are going to create called *GenerateExcel*. Autowiring these two classes allow us to run the Spring Batch Job *GenerateExcel* on demand when a POST request is sent to *"/export"*.

Another thing to note is that to ensure that the Spring Batch job will run more than once we include unique parameters with this code: *addLong("uniqueness", System.nanoTime()).toJobParameters()*. An error may occur if we do not include unique parameters because only unique *JobInstances* may be created and executed, and Spring Batch has no way of distinguishing between the first and second *JobInstance* otherwise.

```
@Controller
public class WebController {

    private String currentContent;

    @Autowired
    JobLauncher jobLauncher;

    @Autowired
    GenerateExcel exceljob;

    @GetMapping("/")
    public ModelAndView getHome() {
        ModelAndView modelAndView = new ModelAndView("in-
dex");
        return modelAndView;
    }


    @PostMapping("/export")
    public String postTheFile(@RequestBody String body,
RedirectAttributes redirectAttributes, Model model)
        throws IOException, JobExecutionAlreadyRunningEx-
ception, JobRestartException, JobInstanceAlreadyComplete-
Exception, JobParametersInvalidException {


        setCurrentContent(body);

        Job job = exceljob.ExcelGenerator();
        jobLauncher.run(job, new
JobParametersBuilder().addLong("uniqueness",
System.nanoTime()).toJobParameters()
                );

        return "redirect:/";
    }

    //standard getters and setters

}
```

6. Batch Job

In Step1 of our Batch job, we call the *getCurrentContent()* method to get the content that was passed into the Thymeleaf form, create a new *XSSFWorkbook*, specify an arbitrary Micro-

soft Excel Sheet tab name, and then pass all three variables into the *createWorksheet* method that we will be making in the next step of our tutorial :

```
@Configuration
@EnableBatchProcessing
@Lazy
public class GenerateExcel {

    List<String> docIds = new ArrayList<String>();

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Autowired
    WebController webcontroller;

    @Autowired
    CreateWorksheet createexcel;

    @Bean
    public Step step1() {
        return stepBuilderFactory.get("step1")
            .tasklet(new Tasklet() {
                @Override
                public RepeatStatus execute(StepContribu-
tion stepContribution, ChunkContext chunkContext) throws
Exception, JSONException {

                    String content =
webcontroller.getCurrentContent();

                    System.out.println("content is ::" +
content);
                    Workbook wb = new XSSFWorkbook();
                    String tabName = "some";
                    createexcel.createWorkSheet(wb, con-
tent, tabName);

                    return RepeatStatus.FINISHED;
                }
            })
            .build();
    }

    @Bean
    public Job ExcelGenerator() {
        return jobBuilderFactory.get("ExcelGenerator")
            .start(step1())
            .build();

    }
}
```

We have covered Spring Batch in other tutorials such as Converting XML to JSON + Spring Batch and Spring Batch CSV Processing.

7. Excel Creation Service

We use a variety of classes to create our Microsoft Excel file. Order matters when dealing with converting HTML to RichText, so this will be a focus.

7.1 RichTextDetails

A class with two parameters: a String that will have our contents that will become RichText and a font map.

```
public class RichTextDetails {
    private String richText;
    private Map<Integer, Font> fontMap;
    //standard getters and setters
    @Override
    public int hashCode() {

        // The goal is to have a more efficient hashcode
than standard one.
        return richText.hashCode();
    }
```

7.2 RichTextInfo

A POJO that will keep track of the location of the RichText and what not:

```
public class RichTextInfo {
    private int startIndex;
    private int endIndex;
    private STYLES fontStyle;
    private String fontValue;
    // standard getters and setters, and the like
```

7.3 Styles

A enum that contains HTML tags that we want to process. We can add to this as necessary:

```
public enum STYLES {
    BOLD("b"),
    EM("em"),
    STRONG("strong"),
    COLOR("color"),
    UNDERLINE("u"),
    SPAN("span"),
    ITALLICS("i"),
    UNKNOWN("unknown"),
    PRE("pre");
    // standard getters and setters
```

7.4 TagInfo

A POJO to keep track of tag info:

```
public class TagInfo {
    private String tagName;
    private String style;
    private int tagType;
```

```
    // standard getters and setters
```

7.5 HTML to RichText

This is not a small class, so let's break it down by method.

Essentially, we are surrounding any arbitrary HTML with a *div* tag, so we know what we are looking for. Then we look for all elements within the *div* tag, add each to an *ArrayList* of *RichTextDetails* , and then pass the whole *ArrayList* to the *mergeTextDetails* method. *mergeTextDetails* returns *RichtextString*, which is what we need to set a cell value:

```
public RichTextString fromHtmlToCellValue(String html,
Workbook workBook){
    Config.IsHTMLEmptyElementTagRecognised = true;

    Matcher m = HEAVY_REGEX.matcher(html);
    String replacedhtml =  m.replaceAll("");
    StringBuilder sb = new StringBuilder();
    sb.insert(0, "<div>");
    sb.append(replacedhtml);
    sb.append("</div>");
    String newhtml = sb.toString();
    Source source = new Source(newhtml);
    List<RichTextDetails> cellValues = new Ar-
rayList<RichTextDetails>();
    for(Element el : source.getAllElements("div")){
        cellValues.add(createCellValue(el.toString(),
workBook));
    }
    RichTextString cellValue = mergeTextDetails(cellVal-
ues);


    return cellValue;
}
```

As we saw above, we pass an ArrayList of *RichTextDetails* in this method. Jericho has a setting that takes boolean value to recognize empty tag elements such as `<br/>` : *Config.IsHTMLEmptyElementTagRecognised*. This can be important when dealing with online rich text editors, so we set this to true. Because we need to keep track of the order of the elements, we use a *LinkedHashMap* instead of a *HashMap*.

```
private static RichTextString mergeTextDetails(L-
ist<RichTextDetails> cellValues) {
    Config.IsHTMLEmptyElementTagRecognised = true;
    StringBuilder textBuffer = new StringBuilder();
    Map<Integer, Font> mergedMap = new LinkedHashMap<Inte-
ger, Font>(550, .95f);
    int currentIndex = 0;
    for (RichTextDetails richTextDetail : cellValues) {
        //textBuffer.append(BULLET_CHARACTER + " ");
        currentIndex = textBuffer.length();
        for (Entry<Integer, Font> entry :
richTextDetail.getFontMap()
                .entrySet()) {
            mergedMap.put(entry.getKey() + currentIndex,
entry.getValue());
        }
        textBuffer.append(richTextDetail.getRichText())
```

```
            .append(NEW_LINE);
    }

    RichTextString richText = new
XSSFRichTextString(textBuffer.toString());
    for (int i = 0; i < textBuffer.length(); i++) {
        Font currentFont = mergedMap.get(i);
        if (currentFont != null) {
            richText.applyFont(i, i + 1, currentFont);
        }
    }
    return richText;
}
```

As mentioned above, we are using Java 9 in order to use *StringBuilder* with the *java.util.regex.Matcher.appendReplacement*. Why? Well that's because *StringBuffer* slower than *StringBuilder* for operations. *StringBuffer* functions are synchronized for thread safety and thus slower.

We are using *Deque* instead of *Stack* because a more complete and consistent set of LIFO stack operations is provided by the *Deque* interface:

```
static RichTextDetails createCellValue(String html, Work-
book workBook) {
    Config.IsHTMLEmptyElementTagRecognised  = true;
    Source source = new Source(html);
    Map<String, TagInfo> tagMap = new LinkedHashMap<S-
tring, TagInfo>(550, .95f);
    for (Element e : source.getChildElements()) {
        getInfo(e, tagMap);
    }

    StringBuilder sbPatt = new StringBuilder();
    sbPatt.append("(").append(StringUtils.join(tagMap.key
Set(), "|")).append(")");
    String patternString = sbPatt.toString();
    Pattern pattern = Pattern.compile(patternString);
    Matcher matcher = pattern.matcher(html);

    StringBuilder textBuffer = new StringBuilder();
    List<RichTextInfo> textInfos = new ArrayList<RichTex-
tInfo>();
    ArrayDeque<RichTextInfo> richTextBuffer = new ArrayDe-
que<RichTextInfo>();
    while (matcher.find()) {
        matcher.appendReplacement(textBuffer, "");
        TagInfo currentTag =
tagMap.get(matcher.group(1));
        if (START_TAG == currentTag.getTagType()) {
            richTextBuffer.push(getRichTextInfo(currentTa
g, textBuffer.length(), workBook));
        } else {
            if (!richTextBuffer.isEmpty()) {
                RichTextInfo info = richTextBuffer.pop();
                if (info != null) {
                    info.setEndIndex(textBuffer.length())
;
                    textInfos.add(info);
                }
            }
```

```
        }
    }
    matcher.appendTail(textBuffer);
    Map<Integer, Font> fontMap = buildFontMap(textInfos,
workBook);

    return new RichTextDetails(textBuffer.toString(),
fontMap);
}
```

We can see where *RichTextInfo* comes in to use here:

```
private static Map<Integer, Font> buildFontMap(L-
ist<RichTextInfo> textInfos, Workbook workBook) {
    Map<Integer, Font> fontMap = new LinkedHashMap<Inte-
ger, Font>(550, .95f);

    for (RichTextInfo richTextInfo : textInfos) {
        if (richTextInfo.isValid()) {
            for (int i = richTextInfo.getStartIndex(); i
< richTextInfo.getEndIndex(); i++) {
                fontMap.put(i, mergeFont(fontMap.get(i),
richTextInfo.getFontStyle(), richTextInfo.getFontValue(),
workBook));
            }
        }
    }

    return fontMap;
}
```

Where we use *STYLES* enum:

```
private static Font mergeFont(Font font, STYLES fontStyle, String fontValue,
Workbook workBook) {
  if (font == null) {
    font = workBook.createFont();
  }

  switch (fontStyle) {
  case BOLD:
  case EM:
  case STRONG:
    font.setBoldweight(Font.BOLDWEIGHT_BOLD);
    break;
  case UNDERLINE:
    font.setUnderline(Font.U_SINGLE);
    break;
  case ITALLICS:
    font.setItalic(true);
    break;
  case PRE:
    font.setFontName("Courier New");
  case COLOR:
    if (!isEmpty(fontValue)) {

      font.setColor(IndexedColors.BLACK.getIndex());
    }
    break;
  default:
    break;
  }

  return font;
}
```

We are making use of the *TagInfo* class to track the current tag:

```
private static RichTextInfo getRichTextInfo(TagInfo cur-
rentTag, int startIndex, Workbook workBook) {
```

```
    RichTextInfo info = null;
    switch (STYLES.fromValue(currentTag.getTagName())) {
    case SPAN:
        if (!isEmpty(currentTag.getStyle())) {
            for (String style : currentTag.getStyle()
                .split(";")) {
                String[] styleDetails = style.split(":");
                if (styleDetails != null &&
styleDetails.length > 1) {
                    if
("COLOR".equalsIgnoreCase(styleDetails[0].trim())) {
                        info = new RichTextInfo(startIn-
dex, -1, STYLES.COLOR, styleDetails[1]);
                    }
                }
            }
        }
        break;
    default:
        info = new RichTextInfo(startIndex, -1,
STYLES.fromValue(currentTag.getTagName()));
        break;
    }
    return info;
}
```

We process the HTML tags:

```
private static void getInfo(Element e, Map<String, TagIn-
fo> tagMap) {
    tagMap.put(e.getStartTag()
        .toString(),
        new TagInfo(e.getStartTag()
            .getName(), e.getAttributeValue("style"),
START_TAG));
    if (e.getChildElements()
        .size() > 0) {
        List<Element> children = e.getChildElements();
        for (Element child : children) {
            getInfo(child, tagMap);
        }
    }
    if (e.getEndTag() != null) {
        tagMap.put(e.getEndTag()
            .toString(),
            new TagInfo(e.getEndTag()
                .getName(), END_TAG));
    } else {
        // Handling self closing tags
        tagMap.put(e.getStartTag()
            .toString(),
            new TagInfo(e.getStartTag()
                .getName(), END_TAG));
    }
}
```

7.6 Create Worksheet

Using *StringBuilder*, I create a String that is going to written to *FileOutPutStream*. In a real application this should be user defined. I appended my folder path and filename on two different lines. Please change the file path to your own.

sheet.createRow(0) creates a row on the very first line and *dataRow.createCell(0)* creates a cell in column A of the row.

```
public void createWorkSheet(Workbook wb, String content,
String tabName) {
        StringBuilder sbFileName = new StringBuilder();
        sbFileName.append("/Users/mike/javaSTS/michaelcgo
od-apache-poi-richtext/");
        sbFileName.append("myfile.xlsx");
        String fileMacTest = sbFileName.toString();
        try {
            this.fileOut = new FileOutputStream(fileMacT-
est);
        } catch (FileNotFoundException ex) {
            Logger.getLogger(CreateWorksheet.class.getNam
e())
                    .log(Level.SEVERE, null, ex);
        }

        Sheet sheet = wb.createSheet(tabName); // Create
new sheet w/ Tab name

        sheet.setZoom(85); // Set sheet zoom: 85%


        // content rich text
        RichTextString contentRich = null;
        if (content != null) {
            contentRich =
htmlToExcel.fromHtmlToCellValue(content, wb);
        }


        // begin insertion of values into cells
        Row dataRow = sheet.createRow(0);
        Cell A = dataRow.createCell(0); // Row Number
        A.setCellValue(contentRich);
        sheet.autoSizeColumn(0);


        try {
            ////////////////////////////////
            // Write the output to a file
            wb.write(fileOut);
            fileOut.close();
        } catch (IOException ex) {
            Logger.getLogger(CreateWorksheet.class.getNam
e())
                    .log(Level.SEVERE, null, ex);
        }
    }
```

8. Demo

We visit *localhost:8080*.

We input some text with some HTML:

We open up our excel file and see the RichText we created:

9. Conclusion

We can see it is not trivial to convert HTML to Apache POI's *RichTextString* class; however, for business applications converting HTML to *RichTextString* can be essential because readability is important in Microsoft Excel files. There's likely room to improve upon the performance of the application we build, but we covered the foundation of building such an application .

The full source code is available on [Github]

Conclusion

11

Thanks for reading. Let's cover what's next.

Thanks for reading. I plan to make further revisions and additions to this book as time goes on. If you have the current book, then you will be getting any future versions free of charge.

For now, to stay up-to-date on Spring Boot, Thymeleaf, and Java related topics in general:

- First, I am shamelessly promoting myself. Check out my blog at www.michaelcgood.com and my Twitter at https://twitter.com/michaelcgood
- The official Spring blog: https://spring.io/blog
- The official Thymeleaf Twitter account: https://twitter.com/thymeleaf
- The official Thymeleaf site: http://www.thymeleaf.org/
- Baeldung - a good website for Java tutorials http://www.baeldung.com/
- DZone - Java has many educational articles: https://dzone.com/java-jdk-development-tutorials-tools-news

Lastly, once again if you have any questions or just want to contact me, send me an email at michael@michaelcgood.com .