

How to use fragments in Thymeleaf

#Thymeleaf • Februar 08, 2020

A fragment in Thymeleaf is a small piece of code that can be included in other templates. It is a common practice in web development to create reusable, small components like the header, footer, navigation menu and other parts of a website that are repeated used on multiple pages.

In this article, you'll learn how to create and use fragments in Thymeleaf templates. If you are new to Thymeleaf, take a look at the [introductory tutorial](#) to learn how to use Thymeleaf with Spring Boot.

Note: For more complex Thymeleaf layouts and reusable templates, take a look at [Thymeleaf Layout Dialect](#) tutorial.

Defining & Referencing Fragments

To define a Thymeleaf fragment, you need to use the `th:fragment` attribute. You can define fragments either in separate files (recommended) or in a common file. It is also a good practice to place all fragments in a dedicated folder called `fragments` inside the templates directory (`src/main/resources/templates/`).

Let us say you want to define a reusable footer component to add copyright information to all of your web pages, so you just create a `footer.html` file with the following content:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
  <footer th:fragment="footer">
    <p>&copy; 2020 attacomsian.com</p>
  </footer>
</body>
</html>
```



Bring your team together with Slack, the collaboration hub for work.

 ads via Carbon

Share Article

Share this article with your friends & followers!

Tweet

Share

Share

Recent Articles

[How to get all query string parameters as an object in JavaScript](#)

[How to get query string parameters in JavaScript](#)


[How to add and update query string parameters using URLSearchParams](#)

[How to convert an object into query string parameters in JavaScript](#)

[How to check if an image exists on the server in JavaScript](#)

[How to edit an XML file with Node.js](#)

#1 CLOUD HOSTING



DigitalOcean
The simplest cloud platform for developers & teams. Start with a \$100 free credit. [Try now](#)

The above code defined a fragment called `footer` that you can easily include to your homepage by using one of the `th:insert` or `th:replace` attributes:

```
<body>
...
<div th:insert="fragments/footer :: footer"></div>
</body>
```

You can also define more than one fragment in a single HTML document as shown in the blow file called `components.html` :

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
    <header th:fragment="header">
        <h1>Welcome to My Blog</h1>
    </header>

    <nav th:fragment="menu">
        <a th:href="@{/}">Homepage</a>
        <a th:href="@{/about}">About Me</a>
        <a th:href="@{/blog}">Blog</a>
        <a th:href="@{/contact}">Contact</a>
    </nav>

    <footer th:fragment="footer">
        <p>&copy; 2020 attacomsian.com</p>
    </footer>
</body>
</html>
```

th:insert & th:replace Attributes

Thymeleaf provides multiple ways to include a fragment into a template:

- `th:insert` — Inserts the fragment content inside the host tag
- `th:replace` &mdahs; Replaces the host tag with the specified fragment content
- `th:include` — Similar to `th:insert` but it only inserts the contents of the specified fragment (depreciated since 3.0)

The following example shows how you can include the `header` fragment from `components.html` using the above three approaches:

Become a Better Web Developer 🙋

Join the weekly newsletter to improve your coding skills quickly, with easy to follow tutorials and protips every week.

Enter your email address

Subscribe

No spam, ever. You can unsubscribe anytime.

```
<body>

<div th:insert="fragments/components :: header"></div>

<div th:replace="fragments/components :: header"></div>

<div th:include="fragments/components :: header"></div>

</body>
```

The rendered HTML document will look like the following:

```
<body>

    <div>
        <header>
            <h1>Welcome to My Blog</h1>
        </header>
    </div>

    <header>
        <h1>Welcome to My Blog</h1>
    </header>

    <div>
        <h1>Welcome to My Blog</h1>
    </div>

</body>
```

Including with DOM Selectors

In Thymeleaf, you don't need to explicitly use `th:fragment` attribute to define fragments. They can be included in another template by using just DOM selectors like class name, element ID, or tag name similar to do what we do in [JavaScript](#).

Here is an example:

```
<div th:insert="fragments/components :: div.title"></div>
```

The above example will include a `<div>` element that has the `.title` CSS class from the `components.html` file.

Parametrized Fragments

Thymeleaf fragments defined with `th:fragment` attribute can specify arguments, just like methods. To use a parameterized fragment, you need to define it as a function call with a declared list of parameters:

```
<div th:fragment="name(firstName, lastName)">
  <p>
    Hey! I'm <span th:text="${firstName + ' ' + lastName}"></span>!
  </p>
</div>
```

Now you can use one of the following syntaxes to include the above fragment using `th:replace` or `th:insert` attribute:

```
<!--Option 1-->
<div th:replace="fragments/components :: name('John', 'Doe')"></div>

<!--Option 2-->
<div th:replace="fragments/components :: name(firstName='John', lastName='Doe')"></div>
```

Since the second option uses named parameters, the order of the parameters is no longer important:

```
<div th:replace="fragments/components :: name(lastName='Doe', firstName='John')"></div>
```

Thymeleaf parameterized fragments are very useful as it allows us to reuse one small piece of code in many different ways.

Fragment Inclusion Expressions

Thymeleaf fragment inclusion syntax fully supports [conditional expressions](#) to dynamically load different fragments. In `templatename :: selector` format, both `templatename` and `selector` can be fully-featured expressions.

For example, in the below code snippet, we want to include different fragments depending on a condition. If the authenticated user is an administrator, we will show a different footer; otherwise, the default one:

```
<div th:replace="fragments/footer :: ${user.admin} ? 'footer-admin' : 'footer'"></div>
```

The `footer.html` file looks slightly different, as we have to define two footers now:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<body>
    <!--Default Footer-->
    <footer th:fragment="footer">
        <p>&copy; 2020 Default Footer</p>
    </footer>

    <!--Admin Footer-->
    <footer th:fragment="admin-footer">
        <p>&copy; 2020 Admin Footer</p>
    </footer>

</body>
</html>
```

If the fragments, you want to conditionally include, are defined in separate files, you have to use the fragment expression syntax introduced in version 3.0:

```
<div th:replace="${user.admin} ? ~{fragments/footer :: footer} :
~{fragments/components :: footer}"></div>
```

Flexible Layouts

Fragment expressions are useful not only for defining fragments that use numbers, strings, bean objects as parameters, but also fragments of markup. This allows us to create fragments that can be enhanced with the markup coming from the calling templates, thus providing a very flexible layout technique.

Let us say that you want to create a base `<head>` section with a default list of styles and scripts, but also want to have a possibility to extend the list with other resources if required.

Here is how the base header looks like:

↓ [base.html](#)

```
<head th:fragment="baseHead(title, links)">

    <title th:replace="${title}">Atta | Founder. Developer. Blogger.</title>

    <!-- default styles and scripts -->
    <link rel="shortcut icon" th:href="@{/images/favicon.ico}">
    <link rel="stylesheet" th:href="@{/css/bootstrap.min.css}">
    <script th:src="@{/js/jquery.min.js}"></script>

    <!-- placeholder for additional links -->
    <th:block th:replace="${links}" />

</head>
```

Now if you want to include additional links to the default header, just call the `baseHead` fragment like below in your template:

```
<head th:replace="fragments/base :: baseHead(~{::title}, ~{::link})">

    <title>Introduction to Thymeleaf Fragments | Atta Blog</title>

    <link rel="stylesheet" th:href="@{/css/font-awesome.min.css}">
    <link rel="stylesheet" th:href="@{/css/jquery-ui.css}">

</head>
```

In the above example, `::title` and `::link` are pointing to the title and link tags in the calling template. The resultant header will use the actual `<title>` and `<link>` tags from the calling template as the values of the `title` and `links` parameters as shown in the below-rendered HTML:

```
<head>

    <title>Introduction to Thymeleaf Fragments | Atta Blog</title>

    <!-- default styles and scripts -->
    <link rel="shortcut icon" href="/images/favicon.ico">
    <link rel="stylesheet" href="/css/bootstrap.min.css">
    <script src="/js/jquery.min.js"></script>

    <!-- placeholder for additional links -->
    <link rel="stylesheet" href="/css/font-awesome.min.css">
    <link rel="stylesheet" href="/css/jquery-ui.css">

</head>
```

Layout Inheritance

Thymeleaf fragments can also be used to create a simple layout. For example, you can create a single file called `layout.html` that has the following structure:

↓ `layout.html`

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org" th:fragment="layout(title, content)">
<head>
  <meta charset="UTF-8">
  <title th:replace="${title}">Layout Title</title>
</head>
<body>

  <h1>Layout Main Heading</h1>

  <section th:replace="${content}">
    <p>Layout contents</p>
  </section>

  <footer>
    <p>&copy; 2020 Layout footer</p>
  </footer>

</body>
</html>
```

As you can see above, we have declared a fragment called `layout` with `title` and `content` as parameters. Both these parameters will be replaced with the calling template tags by using the fragment expressions as shown below:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      th:replace="layout :: layout(~{::title}, ~{::section})">
<head>
  <title>Welcome to My Site</title>
</head>
<body>

  <section>
    <p>This is just an extra text.</p>
    <a th:href="@{/contact-us}">Contact Us</a>
  </section>

</body>
</html>
```

In rendered HTML document, the `<html>` tag will be replaced by the `layout` fragment. But in the `layout` fragment, `title` and `content` will be replaced by `<title>` and `<section>` blocks respectively.

Here is the final HTML output:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Welcome to My Site</title>
</head>
<body>

  <h1>Layout Main Heading</h1>

  <section>
    <p>This is just an extra text.</p>
    <a href="/contact-us">Contact Us</a>
  </section>

  <footer>
    <p>&copy; 2020 Layout footer</p>
  </footer>

</body>
</html>
```

Conclusion

That's all folks. In this article, we looked at how to create and reuse different components of a web page with the help of Thymeleaf fragments, a powerful feature that makes the template management easier and flexible.

We also discussed different formats available to include fragments in Thymeleaf templates, and how to work with parameterized fragments. In the end, we presented an example to create simple layouts using fragment expression syntax.

To learn more about Thymeleaf layouts, take a look at the [official documentation](#) that presents more in-depth examples.

Read Next: [Working with Thymeleaf Layout Dialect in Spring Boot](#)

👉 Like this article? Follow [@attacomsian](#) on Twitter. You can also follow me on [LinkedIn](#) and [DEV](#). Subscribe to [RSS Feed](#).

☺ If you enjoy reading my articles and want to support me to continue creating free tutorials, ☕ [Buy me a coffee](#) (cost \$5) .

Need help to launch a new product? I am available for contract work.
[Hire me](#) to accomplish your business goals with engineering and design.