Standard URL Syntax

The Thymeleaf standard dialects –called *Standard* and *SpringStandard*– offer a way to easily create URLs in your web applications so that they include any required *URL preparation* artifacts. This is done by means of the so-called *link expressions*, a type of *Thymeleaf Standard Expression*: **`@{...}`**

1. Absolute URLs

Absolute URLs allow you to create links to other servers. They start by specifying a protocol name (**`http://`** or **`https://`**)

```
<a th:href="@{http://www.thymeleaf/documentation.html}">
```

They are not modified at all (unless you have an *URL Rewriting* filter configured at your server and performing modifications at the **`HttpServletResponse.encodeUrl(...)`** method):

```
<a href="http://www.thymeleaf/documentation.html">
```

The most used type of URLs are *context-relative* ones. These are URLs which are supposed to be relative to the web application root once it is installed on the server. For example, if we deploy a **`myapp.war`** file into a Tomcat server, our application will probably be accessible as **`http://localhost:8080/myapp`**, and **`myapp`** will be the *context name*.

Context-relative URLs start with **`/`**:

```
<a th:href="@{/order/list}">
```

If our app is installed at **`http://localhost:8080/myapp`**, this URL will output:

```
<a href="/myapp/order/list">
```

3. Server-relative URLs

Server-relative URLs are very similar to *context-relative* URLs, except they do not assume you want your URL to be linking to a resource inside your application's context, and therefore allow you to link to a different context in the same server:

```
<a th:href="@{~/billing-app/showDetails.htm}">
```

The current application's context will be ignored, therefore although our application is deployed at **`http://localhost:8080/myapp`**, this URL will output:

```
<a href="/billing-app/showDetails.htm">
```

4. Protocol-relative URLs

Protocol-relative URLs are in fact absolute URLs which will keep the protocol (HTTP, HTTPS) being used for displaying the current page. They are typically used for including external resources like styles, scripts, etc.:

```
<script th:src="@{//scriptserver.example.net/myscript.js}">...</script>
```

…which will render unmodified (except for *URL rewriting*), like:

```
<script src="//scriptserver.example.net/myscript.js">...</script>
```

5. Adding parameters

How do we add parameters to the URLs we create with **`@{...}`** expressions? Simple:

```
<a th:href="@{/order/details(id=3)}">
```

Which would output as:

```
<a href="/order/details?id=3">
```

You can add several parameters, separating them with commas:

Which would output as:

```
<!-- Note ampersands (&) should be HTML-escaped in tag attributes... -->
<a href="/order/details?id=3&amp;action=show_all">
```

You can also include parameters in the form of *path variables* similarly to *normal* parameters but specifying a placeholder inside your URL's path:

```
<a th:href="@{/order/{id}/details(id=3,action='show_all')}">
```

Which would output as:

```
<a href="/order/3/details?action=show_all">
```

6. URL fragment identifiers

Fragment identifiers can be included in URLs, both with and without parameters. They will always be included at the URL base, so that:

```
<a th:href="@{/home#all_info(action='show')}">
```

…would output as:

7. URL rewriting

Thymeleaf allows you to configure *URL rewriting filters* in your application, and it does so by calling the **`response.encodeURL(...)`** method in the **`javax.servlet.http.HttpServletResponse`** class of the Servlet API for every URL generated from a Thymeleaf template.

This is the standard way of supporting URL rewriting operations in Java web applications, and allows URLs to:

- Automatically detect whether the user has cookies enabled or not, and add the **`;jsessionid=...`** fragment to the URL if not —or if it is the first request and cookie configuration is still unknown.
- Automatically apply proxy configuration to URLs when needed.
- Make use (if configured so) of different CDN (Content Delivery Network) setups, in order to link to content distributed among several servers.

A very common (and recommended) technology for URL Rewriting is URLRewriteFilter.

8. Only for th:href's?

Do not think URL **`@{...}`** expressions are only used in **`th:href`** attributes. They can, in fact, be used *anywhere* just like variable expressions (**`${...}`**) or message externalization / internationalization ones (**`#{...}`**).

For example, you could use them in forms…

```
<form th:action="@{/order/processOrder}">
```

```
<p th:text="#{orders.explanation('3', @{/order/details(id=3,action='show_all')})}">
```

9. Using expressions in URLs

What if we needed to write an URL expression like this:

```
<a th:href="@{/order/details(id=3,action='show_all')}">
```

…but neither `3` nor `'show_all'` could be literals, because we only know their value at run time?

No problem! Every URL parameter value is in fact an expression, so you can easily substitute your literals with any other expressions, including i18n, conditionals…:

```
<a th:href="@{/order/details(id=${order.id},action=(${user.admin} ? 'show_all' : 'show_public'))}">
```

What's more: an URL expression like:

```
<a th:href="@{/order/details(id=${order.id})}">
```

…is in fact a shortcut for:

```
<a th:href="@{'/order/details'(id=${order.id})}">
```

```
<a th:href="@{${detailsURL}(id=${order.id})}">
```

…or an externalized/internationalized text:

```
<a th:href="@{#{orders.details.localized_url}(id=${order.id})}">
```

…even complex expressions can be used, including conditionals, for example:

```
<a th:href="@{(${user.admin}? '/admin/home' : ${user.homeUrl})(id=${order.id})}">
```

Want it cleaner? Use `th:with` :

```
<a th:with="baseUrl=(${user.admin}? '/admin/home' : ${user.homeUrl})"
  th:href="@{${baseUrl}(id=${order.id})}">
```

…or…

```
<div th:with="baseUrl=(${user.admin}? '/admin/home' : ${user.homeUrl})">
  ...
  <a th:href="@{${baseUrl}(id=${order.id})}">...</a>
  ...
</div>
```