Spring MVC2 02 [타임리프 - 기본 기능2]

스프링 MVC 2편 - 백엔드 웹 개발 활용 기술

김영한

2022.10.10

타임리프의 기본 기능들을 추가로 더 알아보는것.

유틸리티 객체 & 날짜

여러 유틸리티 객체, 매뉴얼이 자세히 나와있다. 문자 날짜 다양한 유틸리티 객체들,uri등

종류

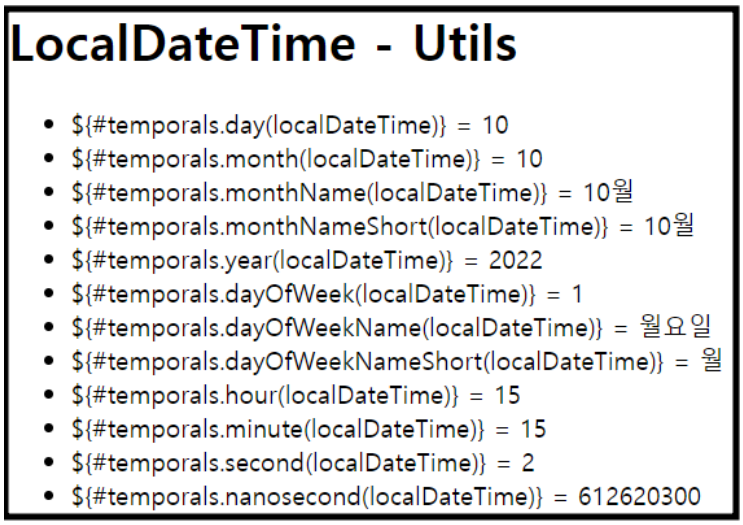
https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#expression-utility-objects

예시

https://www.thymeleaf.org/doc/tutorials/3.0/usingthymeleaf.html#appendix-b-expression-utility-objects

자바 8 날짜 사용

model.addAttribute("localDateTime", LocalDateTime.now()); – 출력 방법



URL 링크

@{...} 문법으로 사용한다

param1 = data1, param2 = data2 일때

```
<li><a th:href="@{/hello}">basic url</a></li>
→ /hello
<li><a th:href="@{/hello(param1=${param1}, param2=${param2})}">hello query
param</a></li>
→ /hello?param1=data1&parma2=data2
<li><a th:href="@{/hello/{param1}/{param2}(param1=${param1},
param2=${param2})}">path variable</a></li>
→ /hello/data1/data2
<li><a th:href="@{/hello/{param1}(param1=${param1}, param2=${param2})}">path
variable + query parameter</a></li>
→ /hello/data1?param2=data2
```

단순한 **URL** @{/hello} : /hello

쿼리 파라미터 @{/hello(param1=${param1}, param2=${param2})}
/hello?param1=data1&m2=data2 () 에 있는 부분은 쿼리 파라미터로 처리된다.

경로 변수 @{/hello/{param1}/{param2}(param1=${param1}, param2=${param2})}
/hello/data1/data2 URL 경로상에 변수가 있으면 () 부분은 경로 변수로 처리된다.

경로 변수 **+** 쿼리 파라미터 @{/hello/{param1}(param1=${param1}, param2=${param2})}
/hello/data1?param2=data2 경로 변수와 쿼리 파라미터를 함께 사용할 수 있다.

리터럴

-소스 코드상에 고정된 값을 말한다.

타임리프에서는 문자 리터럴은 항상 **'** (작은 따옴표 **)** 로 감싸야 한다**.**

** 이런 식으로 사용해야 한다.**

공백없이 쭉 이어진다면 작은 따옴표를 생략가능하다.

주의!

** ⇒** 가능

** ⇒** 오류

```
<li>"hello world!" = <span th:text="'hello world!'"></span></li>
→ 작은 따옴표를 없애면 오류가 발생한다.
<li>'hello' + ' world!' = <span th:text="'hello' + ' world!'"></span></li>
<li>'hello world!' = <span th:text="'hello world!'"></span></li>
<li>'hello ' + ${data} = <span th:text="'hello ' + ${data}"></span></li>
<li>리터럴 대체 |hello ${data}| = <span th:text="|hello ${data}|"></span></li>
→ 리터럴 대체를 사용하게되면 편리하게 사용가능하다.
```

연산

operation

```
model.addAttribute("nullData", null);
model.addAttribute("data", "Spring!");
```

```
<li>Elvis 연산자
 <ul>
   <li>${data}?: '데이터가 없습니다.' = <span th:text="${data}?:
'데이터가없습니다.'"></span></li>
   <li>${nullData}?: '데이터가 없습니다.' = <span th:text="${nullData}?:'데이터가
없습니다.'"></span></li>
 </ul>
</li>
<li>No-Operation
 <ul>
   <li>${data}?: _ = <span th:text="${data}?: _">데이터가 없습니다.</span></li>
   <li>${nullData}?: _ = <span th:text="${nullData}?:
_">데이터가없습니다.</span></li>
 </ul>
```

```
</li>
```

→ _ (언더바) **no-operation** 연산을 작동하지 않는 것.

- Elvis 연산자
  - ${data}?: '데이터가 없습니다.' = Spring!
  - ${nullData}?: '데이터가 없습니다.' = 데이터가 없습니다.
- No-Operation
  - ${data}?: _ = Spring!
  - ${nullData}?: _ = 데이터가없습니다.

→결과

속성값 변경

th:* 속성을 지정하면 타임리프는 기존 속성을 th:* 로 지정한 속성으로 대체한다. 기존 속성이 없다면 새로 만든다.

<input type="text" name="mock" th:name="userA" />

타임리프 렌더링 후 <input type="text" name="userA" />

속성 값 추가

th:attrappend : 속성 값의 뒤에 값을 추가한다.

th:attrprepend : 속성 값의 앞에 값을 추가한다.

th:classappend : class 속성에 자연스럽게 추가한다.

예시 →

```
<input type="text" class="text" th:attrappend="class=' large'" />
```

```
<input type="text" class="text large" /> -> 렌더링시
```

속성 값 -checked

```
- checked x <input type="checkbox" name="active" th:checked="false" /><br/>
- checked=false <input type="checkbox" name="active" checked="false" /><br/>
```

HTML에서는 checked 속성이 존재하기만 해도 체크가 되어버린다.

그러므로 th:checked =”false” 시 렌더링 하면 checked 속성을 없애버린다.

```
<input type="checkbox" name="active" checked="false" th:checked="false"/>
```

이렇게 설정시 렌더링하면

<input type="checkbox" name="active" /> ⇒ checked 가 사라진다.

반복

```
List<User> list = new ArrayList<>();
list.add(new User("UserA", 10));
list.add(new User("UserB", 20));
list.add(new User("UserC", 30));

model.addAttribute("users", list);
```

th:each 문법을 사용한다.

```
<tr th:each="user : ${users}">
   <td th:text="${user.username}">username</td>
   <td th:text="${user.age}">0</td>
</tr>
```

```
<tr th:each="user, userStat : ${users}">
```

루프의 state를 불러올수있다. 이렇게 사용시

```
index = <span th:text="${userStat.index}"></span>
count = <span th:text="${userStat.count}"></span>
size = <span th:text="${userStat.size}"></span>
```

이런식으로 반복의 상태를 가져올 수 있다.

index, count , size ,even , odd : 홀수, 짝수 여부**(boolean), first , last :**처음, 마지막 여부**(boolean), current :** 현재 객체 확인 가능하다**.**

두번째 파라미터 이름 : 지정한 변수명 + (Stat) 으로 생략가능하니 파라미터를 생략해도 사용하다.

조건부 평가

if, unless

조건이 충족되어야 태그가 렌더링시 존재한다.

```
<span th:text="'미성년자'" th:if="${user.age lt 20}"></span>
```

만약 나이가 20이상이 되면 span태그가 사라지게 된다.

unless는 if(!)

switch ,cass

```
<td th:switch="${user.age}">
   <span th:case="10">10살</span>
   <span th:case="20">20살</span>
   <span th:case="*">기타</span>
</td>
```

조건에 맞는 태그만 렌더링 된다.

주석

HTML 주석 , 타임리프 주석, 렌더링시 보이는 주석

```
<h1>1. 표준 HTML 주석</h1>
<!--
<span th:text="${data}">html data</span>
-->
<h1>2. 타임리프 파서 주석</h1> → 주로 사용하게 된다
<!--/* [[${data}]] */-->
<!--/*-->
<span th:text="${data}">html data</span>
<!--*/-->
<h1>3. 타임리프 프로토타입 주석</h1>
<!--/*/
<span th:text="${data}">html data</span>
/*/-->
```

타임리프 프로토 타입 주석

-만약 서버 사이드에서 렌더링 된다면 정상 렌더링되어서 표출된다.

블록

<th:block> 타임리프의 자체 태그 **!**

```
<th:block th:each="user : ${users}">
   <div>
       사용자 이름1 <span th:text="${user.username}"></span>
       사용자 나이1 <span th:text="${user.age}"></span>
   </div>
   <div>
       요약 <span th:text="${user.username} + ' / ' + ${user.age}"></span>
   </div>
</th:block>
```

여러 태그가 포함된 블록단위로 속성을 정할 수 있다**.**

렌더링시 **<th:block >** 표시는 사라진다**.**