Softwareentwicklung (SW)

Übertragung von Daten von der View zum Controller und umgekehrt

Prof. Dr. Alixandre Santana
alixandre.santana@oth-regensburg.de

Wintersemester
2024/2025

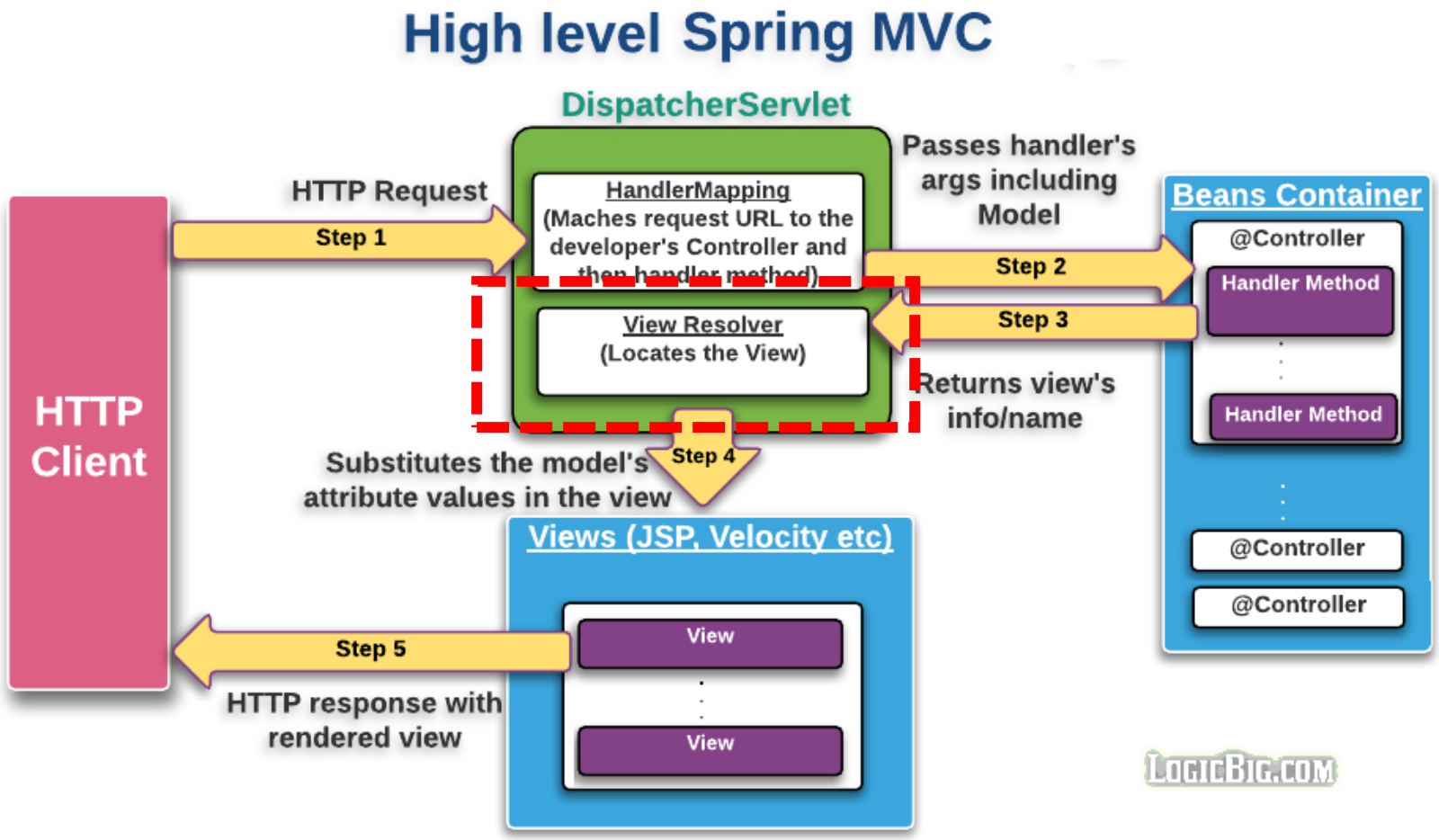
Lernziele

- Das viewResolver-Konzept zu verstehen
- Das Konzept von Taglib in Thymeleaf zu verstehen
- Die Annotations für Request Parameters und -attribute anzuwenden
- Die Annotation "@ModelAttribute" zu beschreiben und anzuwenden

Agenda

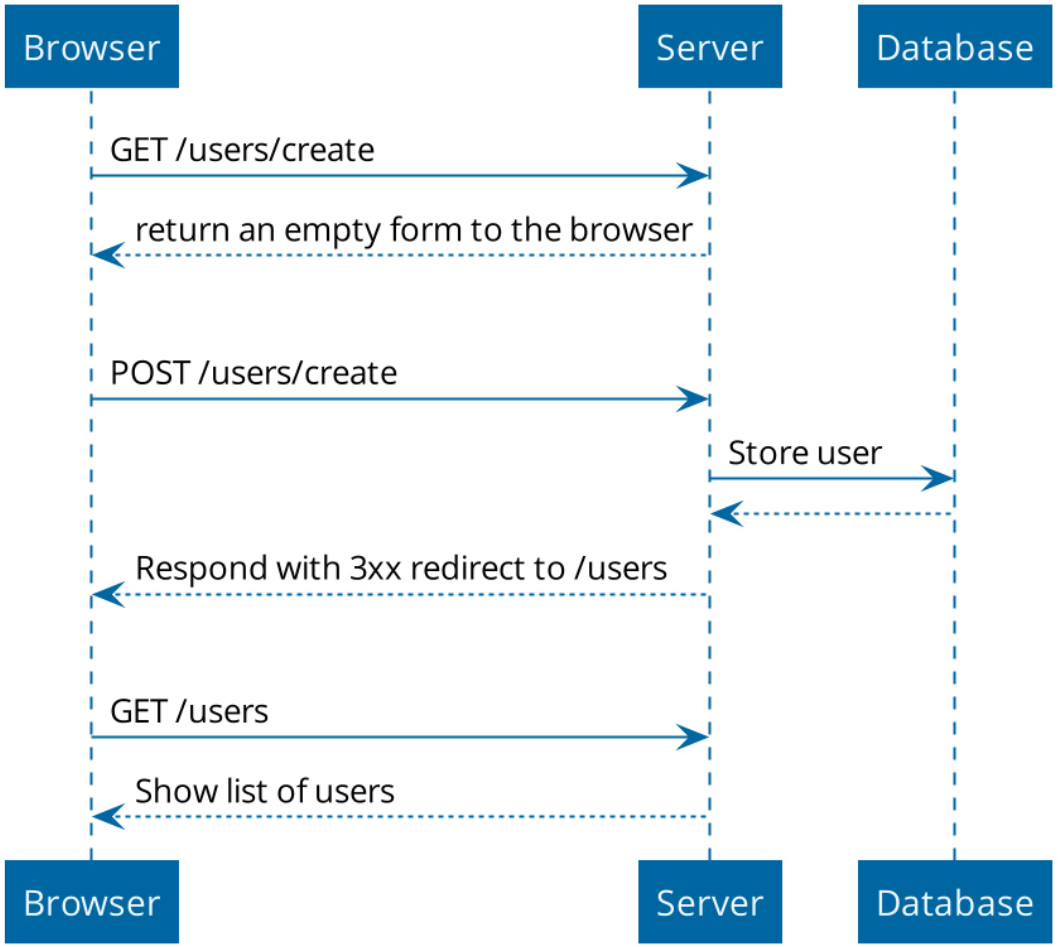
1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen Model and ModelAndView
5. Taglibs für Forms
6. @ModelAttribute

1. MVC Review



https://www.logicbig.com/tutorials/spring-framework/spring-web-mvc/spring-mvc-intro.html

1. MVC Review



https://www.wimdeblauwe.com/blog/2021/05/23/form-handling-with-thymeleaf/

1. Festlegen des View Resolvers

Mit “Pure JSPs” als Views...

```
@Bean
public ViewResolver viewResolver() {
    InternalResourceViewResolver viewResolver =
                    new InternalResourceViewResolver();

    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");
    return viewResolver;
}
```

ODER

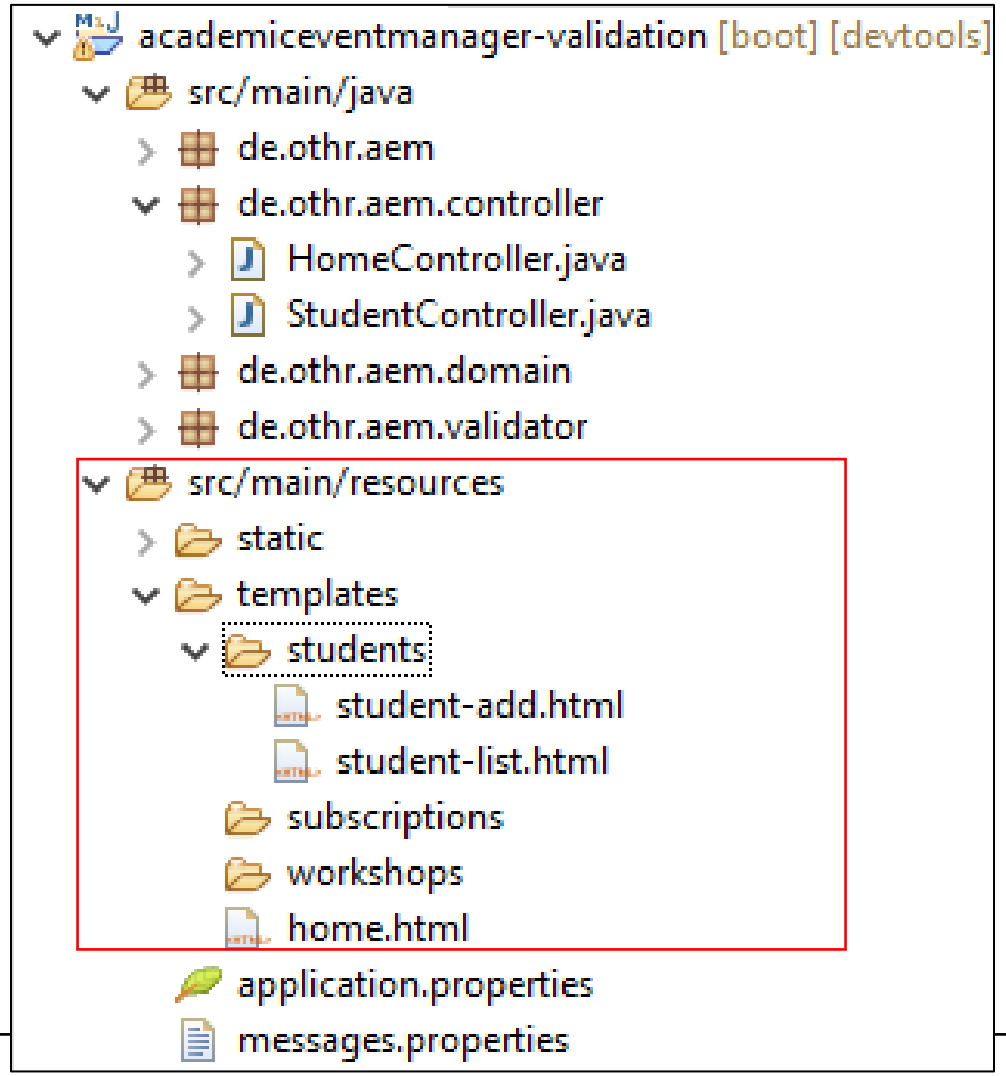
application.properties

```
1 spring.mvc.view.prefix: /WEB-INF/views/
2 spring.mvc.view.suffix: .jsp
3
```

1. Festlegen des View Resolvers

Aber mit Thymeleaf...

- Wir definieren nicht weder das ViewResolver noch das “view.prefix”
- Es wurd schon von Thymeleaf definiert
- Man hat schon die folgende Struktur:



Agenda

1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen Model and ModelAndView
5. Taglibs für Forms
6. @ModelAttribute

2. JavaServer Pages Standard Tag Library (JSTL)

- "The Java Server Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates core functionality common to many JSP applications."

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
```

- Ebenso bietet Spring MVC eine eigene Tag-Bibliothek, um Spring JSP-Ansichten einfach und effektiv zu entwickeln.
- Diese Tags bieten viele nützliche allgemeine Funktionen wie Formularbindung, Fehlerauswertung und Nachrichtenausgabe und mehr, wenn wir mit Spring MVC arbeiten.

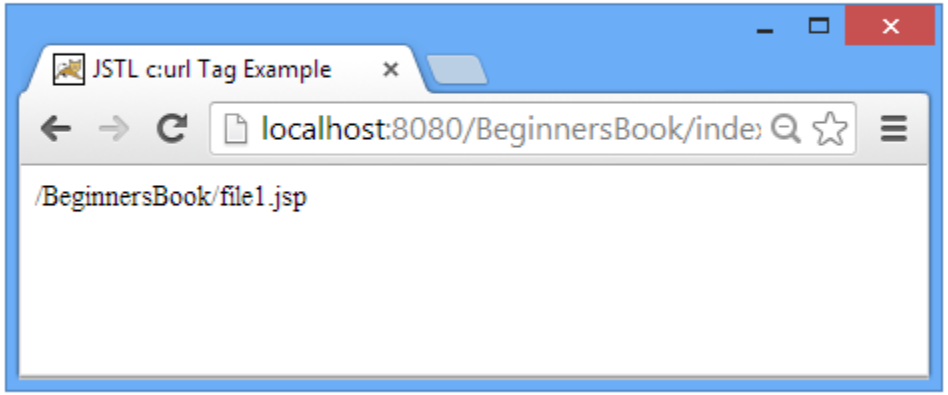
```
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

Sehen Sie: https://www.thymeleaf.org/doc/articles/thvsjsp.html

2. Beispile mit <c:url>

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html> <head> <title>JSTL c:url Tag Example</title>
</head>
<body> <c:url value="/file1.jsp"/>
</body>
</html>
```

```
/BeginnersBook/file1.jsp
```



Sehen Sie: https://beginnersbook.com/2013/11/jstl-curl-core-tag/

2. Taglib Thymeleaf

- zum Erstellen von Formularen, die vollständig in Ihre Formular-basierten Beans und Ergebnisbindungen integriert sind, einschließlich der Verwendung von Konvertierungsdiensten und Validierungsfehlerbehandlung.

```
<form th:action="@{/students/add/” th:object="${student}" method="POST">

<input type="text" class="form-control" id="email" placeholder="Email"
th:field="*{email}" />

...
```

- Access values from the request paramerers

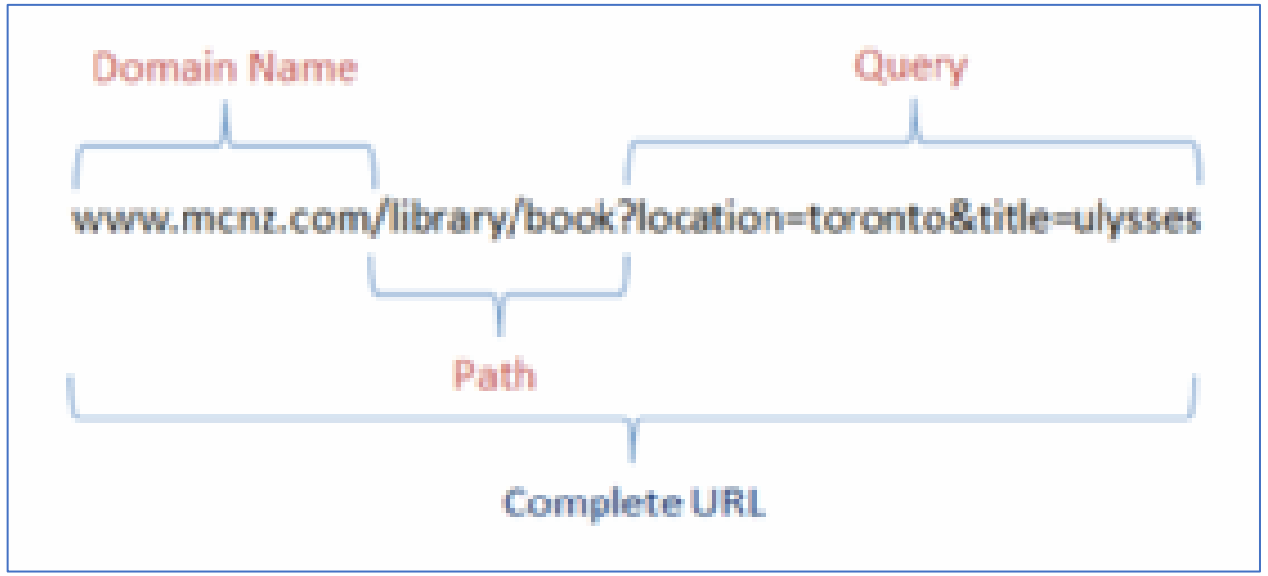
```
<span th:text=" ${myparam}" />
```

Agenda

1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen Model and ModelAndView
5. Taglibs für Forms
6. @ModelAttribute

3.1 @RequestParam

- **Spring *@RequestParam* annotation maps request parameters to the arguments of a request handler method in a Spring Controller**.
- Wir können diese Annotation für Methodenparameter verwenden, um Request-parameters anhand ihrer Namen zu binden.



3.1 @RequestParam

- Diese „Annotation“ enthält drei Attribute:
  - ***name*** **attribute** gibt den Namen des Request Parameters an und wir können es weglassen, wenn der Methodenparameter und die Request parameters denselben Namen haben.
  - ***required*** **attribute** stellt dar, ob der Request Parameter obligatorisch ist, was standardmäßig true ist.
  - ***default*** **attribute** kann einen Fallback-Wert eines Request Parameters angeben, der nur verwendet wird, wenn der Request Parameter nicht vorhanden ist.

3.1 Nutzung von @RequestParam - Controller

```
12 @Controller
13 public class ContactController {
14
15     @RequestMapping("/contact")
16     public String contact(
17             @RequestParam String name,
18             @RequestParam(name="sname") String surname,
19             @RequestParam String gender,
20             @RequestParam String birthdate,
21             @RequestParam (required = true) String email,
22             @RequestParam String telephone,
23             @RequestParam String city,
24             @RequestParam String state,
25             @RequestParam String message,
26             @RequestParam (defaultValue = "0") String newsletter
27             ) {
28
29                 System.out.println("name sent ="+ name+".....");
30                 return "contact-sent";
31     }
32
```

3.1. Nutzung von @RequestParam - View

```
<form th:action="@{/}">
        <input type="text" th:name=“login"/>
        <input type="submit"/>
</form>
```

3.1 Lesen von String-Parameter mit mehreren Werten

```
public String multiValueParams(@RequestParam List<String> myId) {
  return "id: " + myId;
}
```

```
<input type="checkbox" value="55" id="myId">
<input type="checkbox" value="65" id="myId">
<input type="checkbox" value="75" id="myId">
<input type="checkbox" value="85" id="myId">
<input type="checkbox" value="95" id="myId">
```

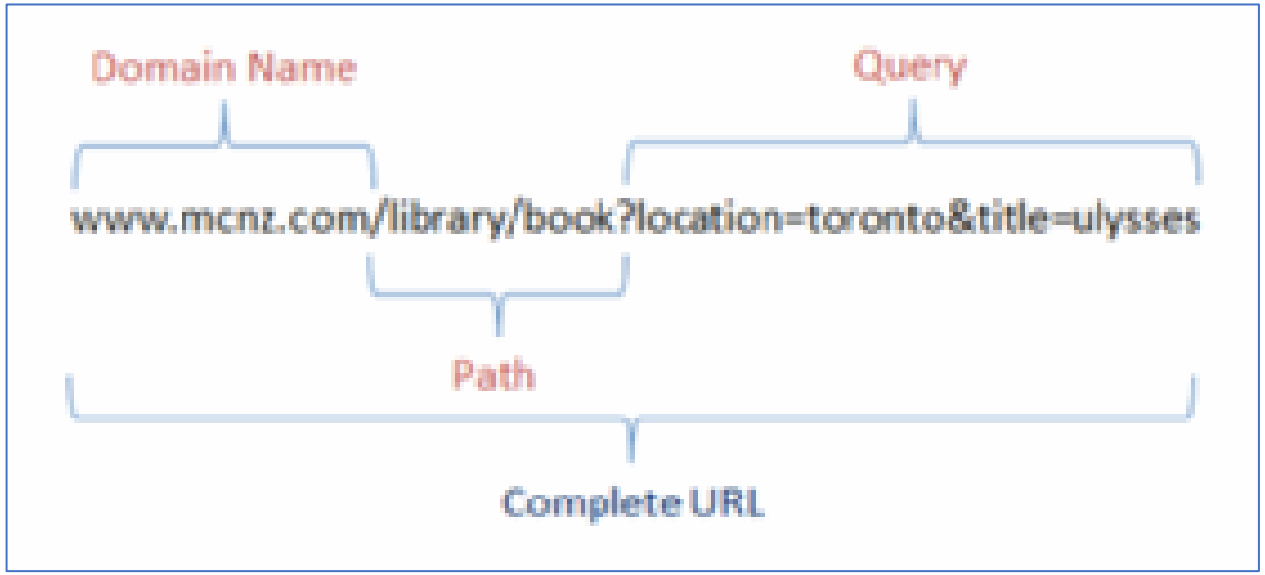
HTML Form

3.1 Lesen von String-Parameter als Java-Map

- Mit der Spring @RequestParam-Annotation können wir alle Abfrageparameter der Anfrage als HashMap sammeln. Das ist nützlich, wenn wir alle Abfrageparameter und ihre Werte zusammen lesen möchten..

```
@GetMapping("/data9")
 public String mappedParams(@RequestParam Map<String, String>   dataQuery) {
        return dataQuery.toString();
}
```

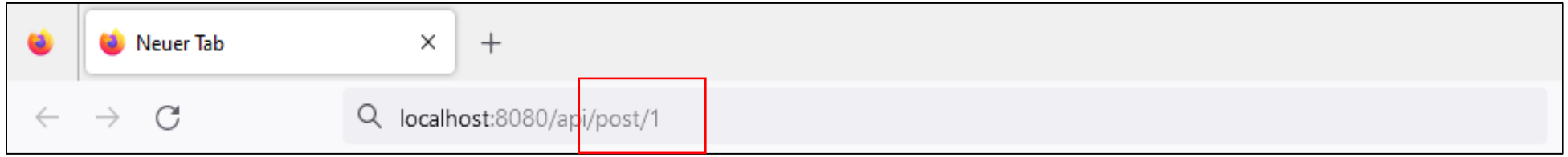
3.2 @PathVariable



www.mcnz.com/library/toronto/book/ulysses

https://www.baeldung.com/spring-thymeleaf-path-variables
https://www.theserverside.com/opinion/RESTful-parameters-antipattern-considerations-for-queries-paths

3.2 @PathVariable




```
@GetMapping("/api/post/{id}")
public String getPostById(@PathVariable String id)
{
    return id;
}
```

PostController

https://www.baeldung.com/spring-thymeleaf-path-variables
https://www.theserverside.com/opinion/RESTful-parameters-antipattern-considerations-for-queries-paths

Agenda

1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen “Model” und “ModelAndView”
5. Taglibs für Forms
6. @ModelAttribute

4. The Model in Spring MVC

- Ein Modell im Kontext Model View Controller (MVC) ist:
  - (1) die Darstellung der Daten, die an den Controller gesendet werden,
  - (2) der Daten, die in einer Ansicht bearbeitet werden,
  - oder (3) die Darstellung der domänenspezifischen Entitäten, die auf der Geschäftsschicht ausgeführt werden.

4. Model-Klasse von Spring

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.springframework.ui

Interface Model

All Known Subinterfaces:

RedirectAttributes

All Known Implementing Classes:

BindingAwareConcurrentModel, BindingAwareModelMap, ConcurrentModel, ExtendedModelMap, RedirectAttributesModelMap

---

```
public interface Model
```

Interface that defines a holder for model attributes.

Primarily designed for adding attributes to the model.

Allows for accessing the overall model as a `java.util.Map`.

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html

4. Verwendung der Model-Klasse

```
@Controller
public class MyMvcController {
    @RequestMapping(value = "/my-uri-path")
    public String prepareView(Model model) {
       model.addAttribute("msg", "msg-value");
        .....
    }
}
```

```
<%@ page language="java"
    contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html>
<body>
Message : ${msg}
</body>
</html>
```

```
...
<span th:text="${msg}"></span>
...
```

4.2 Klasse ModelAndView von Spring

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Spring Framework

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

org.springframework.web.servlet

Class ModelAndView

java.lang.Object
org.springframework.web.servlet.ModelAndView

```
public class ModelAndView
extends Object
```

Holder for both Model and View in the web MVC framework. Note that these are entirely distinct. This class merely holds both to make it possible for a controller to return both model and view in a single return value.

Represents a model and view returned by a handler, to be resolved by a DispatcherServlet. The view can take the form of a String view name which will need to be resolved by a ViewResolver object; alternatively a View object can be specified directly. The model is a Map, allowing the use of multiple objects keyed by name.

4.2 Verwendung der ModelAndView-Klasse

```
@GetMapping("/helloworld")

public ModelAndView passParametersWithModelAndView() {

ModelAndView modelAndView = new ModelAndView("view/helloworld");
modelAndView.addObject("message", “Hello World!");
return modelAndView;

}
```

Agenda

1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen Model and ModelAndView
5. Taglibs für Forms (Thymeleaf)
6. @ModelAttribute

5. Taglibs für Forms

```
<form th:action="@{/requestparam/process}" method="POST">
<div class="form-row">
<div class="form-group col-md-6">
<label for="login">Login </label>
<input type="text" class="form-control" id="login" placeholder="Login"
autofocus="autofocus" th:name="login" />
</div>
</div>
<div class="form-row">
<div class="form-group col-md-6">
<label for="pass">Password </label>
<input type="password" class="form-control" id="pass" placeholder="****"
th:name="pass" />
</div>
</div>
<input type="hidden" id="id" th:name="id"/>
<button type="submit" class="btn btn-primary btn-sm">Go!</button>
</form>
```

https://www.codejava.net/frameworks/spring-boot/spring-boot-thymeleaf-form-handling-tutorial
https://education.launchcode.org/java-web-development/chapters/spring-model-validation/thymeleaf-form-tools.html

5. Taglibs für Forms

```
<form th:action="@{/students/add/process” th:object="${student}" method="POST">
  <div class="form-row">
    <div class="form-group col-md-6">
        <label for="name">Name </label>
        <input type="text" class="form-control" id="name" placeholder="Name"
        autofocus="autofocus" th:field="*{name}"
        th:classappend="${#fields.hasErrors('name')} ? 'is-invalid'" />
        <div class="invalid-feedback">
        <span th:errors="*{name}"></span>
        </div>
    </div>
  </div>
</form>
```

https://www.codejava.net/frameworks/spring-boot/spring-boot-thymeleaf-form-handling-tutorial
https://education.launchcode.org/java-web-development/chapters/spring-model-validation/thymeleaf-form-tools.html
https://www.baeldung.com/spring-thymeleaf-error-messages

Agenda

1. Wiederholung von der Spring MVC Architecture
2. “Taglib” Begriff
3. @RequestParam, @PathVariable
4. Klassen Model and ModelAndView
5. Taglibs für Forms
6. @ModelAttribute

6. Binding forms to the Bean in the Controller

- Hier helfen uns die Tags der Spring-Tag-Bibliothek, die Werte des HTML-Tag-Elements an ein Formular-Backing-Bean im Modell zu binden.
- Später kann der Controller das Formular-Backing-Bean mithilfe der Annotation @ModelAttribute aus dem Modell abrufen

6. Annotation @ModelAttribute

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP Spring Framework

PREV CLASS NEXT CLASS FRAMES NO FRAMES ALL CLASSES

SUMMARY: FIELD | REQUIRED | OPTIONAL DETAIL: FIELD | ELEMENT

org.springframework.web.bind.annotation

Annotation Type ModelAttribute

```
@Target(value={PARAMETER,METHOD})
 @Retention(value=RUNTIME)
 @Documented
public @interface ModelAttribute
```

Annotation that binds a method parameter or method return value to a named model attribute, exposed to a web view. Supported for controller classes with `@RequestMapping` methods.

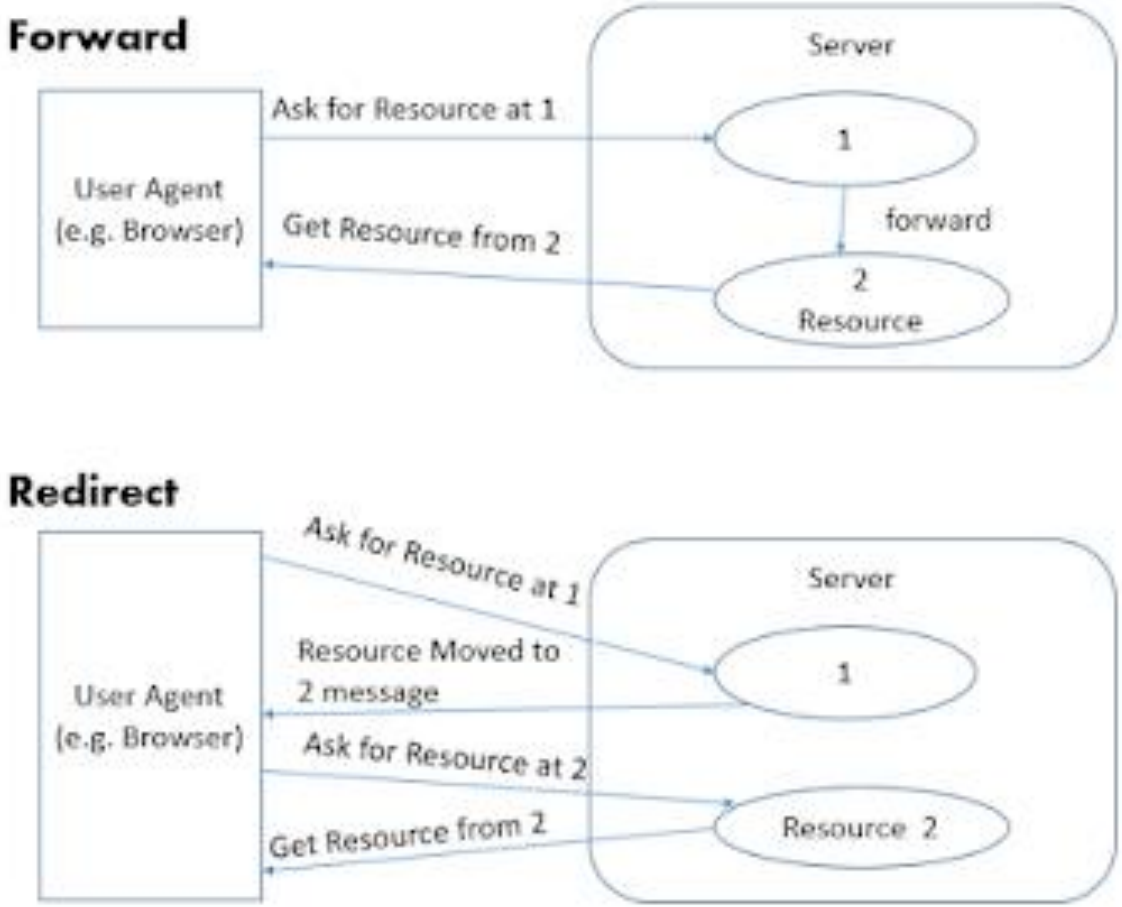
6. Verwendung von ModelAttribute

```
@RequestMapping(value = "/students/add/process")
public String addStudent(@ModelAttribute Student studentRequest){

    return "/students/student-add";
}
```

```
<form th:action="@{/students/add/process" th:object="${student}" method="POST">
  <div class="form-row">
     <div class="form-group col-md-6">
          <label for="name">Name </label>
          <input type="text" class="form-control" id="name" placeholder="Name"
          autofocus="autofocus" th:field="*{name}"
          th:classappend="${#fields.hasErrors('name')} ? 'is-invalid'" />
          <div class="invalid-feedback">
          <span th:errors="*{name}"></span>
          </div>
     </div>
  </div>
</form>
```

Extra: forward x redirect



https://www.baeldung.com/servlet-redirect-forward
https://www.java67.com/2016/03/6-difference-between-forward-and-sendredirect-in-Servlet-JSP.html

Extra: forward x redirect

Simply put, forwarded requests still carry this value, but redirected requests don't.

Forward:
- The request will be further processed on the server side
- The client isn't impacted by forward, URL in a browser stays the same
- Request and response objects will remain the same object after forwarding. Request-scope objects will be still available

Redirect:
- The request is redirected to a different resource
- The client will see the URL change after the redirect
- A new request is created

https://www.baeldung.com/servlet-redirect-forward
https://www.java67.com/2016/03/6-difference-between-forward-and-sendredirect-in-Servlet-JSP.html

Extra: forward Action

```
@RequestMapping(value = "/forward")
public String showForwardForm (Model model) {
        return "forward";
}
@RequestMapping(value = "/forward/process")
public String ProcessForwardForm (@RequestParam String name) {
    System.out.println("name in forward/process is .."+ name);
    return "forward:/forwardprocess2";
}
@RequestMapping(value = "/forwardprocess2")
public String ProcessForwardForm2 (@RequestParam String name) {
    System.out.println("name in forwardprocess2 is .."+ name);
    return "forward-processed";
}
```

Sehen Sie: https://www.baeldung.com/spring-redirect-and-forward

Extra: redirect Action

```
@RequestMapping(value = "/redirect")
public String showREdirectForm (Model model) {
    return "redirect";
}
@RequestMapping(value = "/redirect/process")
public String ProcessRedirectForm (@RequestParam String name) {
    System.out.println("name in redirect/process is .."+ name);
    return "redirect:/redirectprocess2";
}
@RequestMapping(value = "/redirectprocess2")
public String ProcessRedirectForm2 (@RequestParam(required = false) String name) {
    System.out.println("name in redirectprocess2 is .."+ name);
    return "redirect-processed";
}
```

Sehen Sie: https://www.baeldung.com/spring-redirect-and-forward

Referenzen

- https://docs.spring.io/spring-framework/docs/current/javadoc-api/
- https://www.baeldung.com/spring-mvc-model-model-map-model-view
- https://www.baeldung.com/spring-mvc-and-the-modelattribute-annotation
- https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html