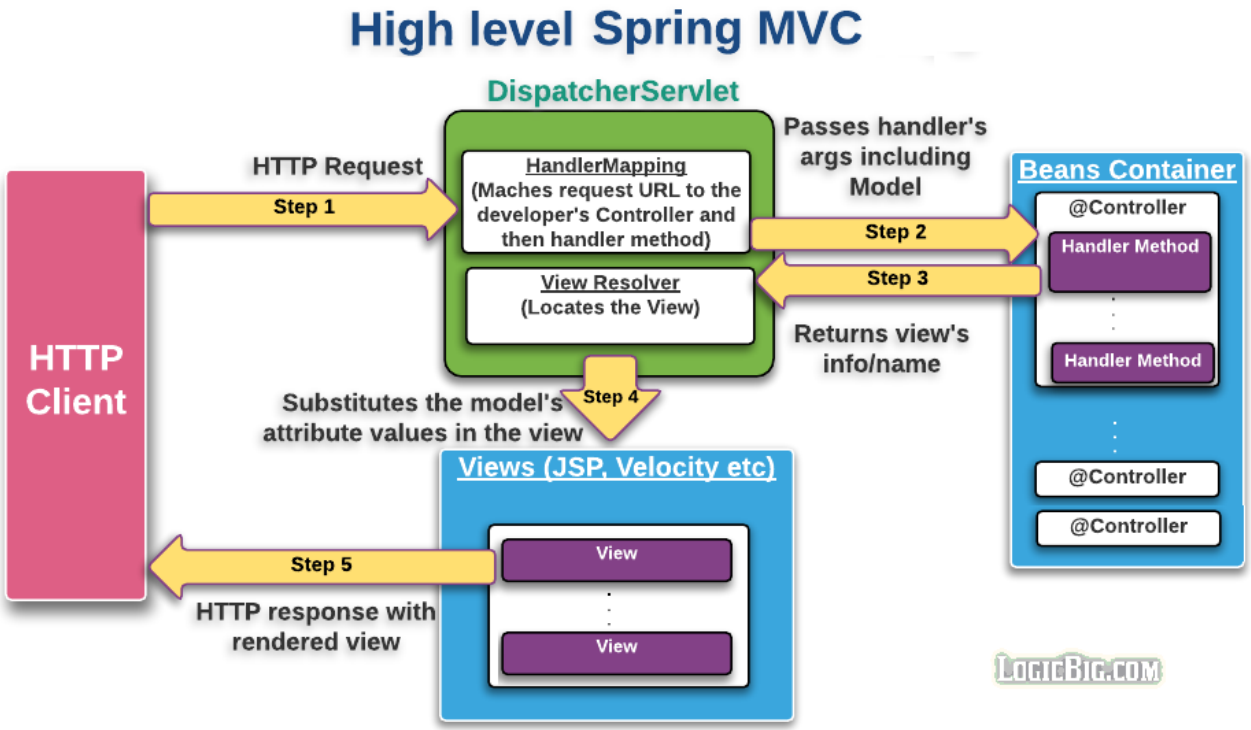
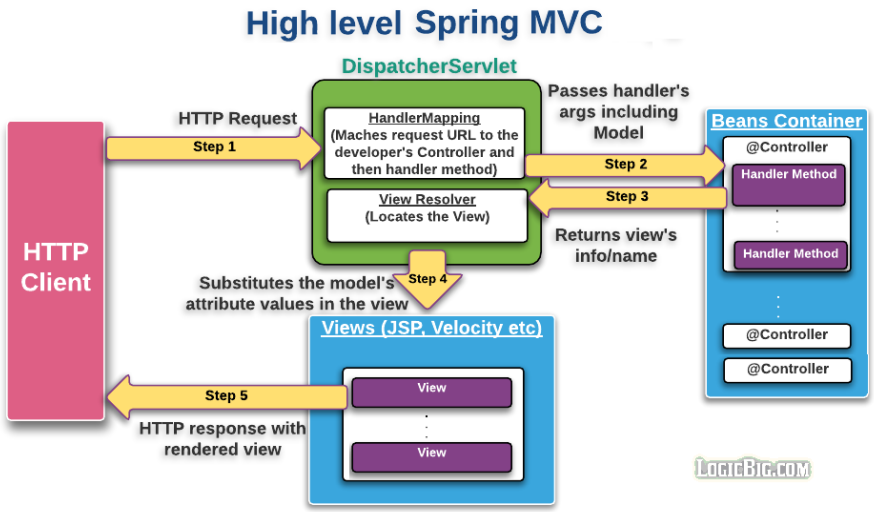
3 SPRING MVC

Spring MVC

- Spring MVC ist ein Modul des Spring Frameworks,
  - welches ein Model View Controller Framework darstellt.
  - https://de.wikipedia.org/wiki/Model_View_Controller

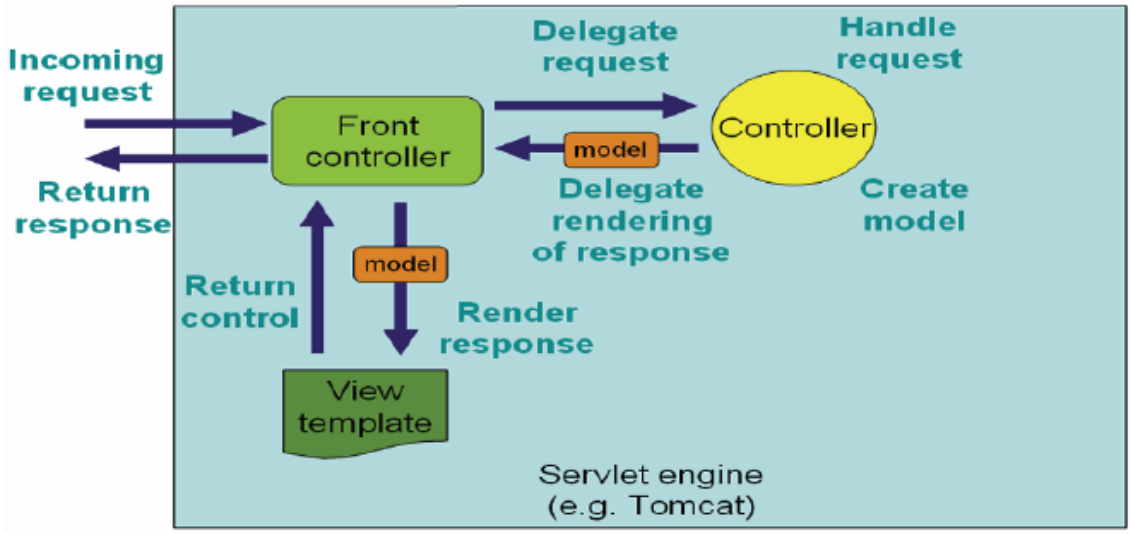


Spring MVC



- **Nutzung Variante 1**
  - serverseitig dynamisch HTML Seiten rendern (produzieren) und diese ausliefern.
- **Nutzung Variante 2**
  - REST-Implementierung produziert JSON oder XML Daten und liefert diese aus (kein HTML Rendern serverseitig)

Der Request



HTTP-Request -->

Frontcontroller (Dispacher-Servlet)

technischer Einstiegspunkt

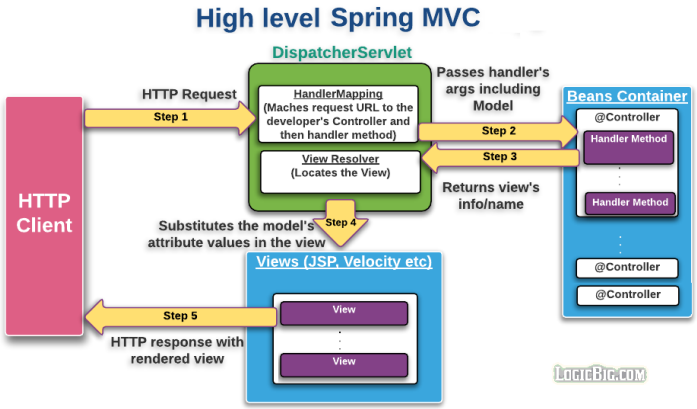
--> Controller bzw. Restcontroller

fachliche Einstiegspunkt

--> Service

--> Repository

@Controller



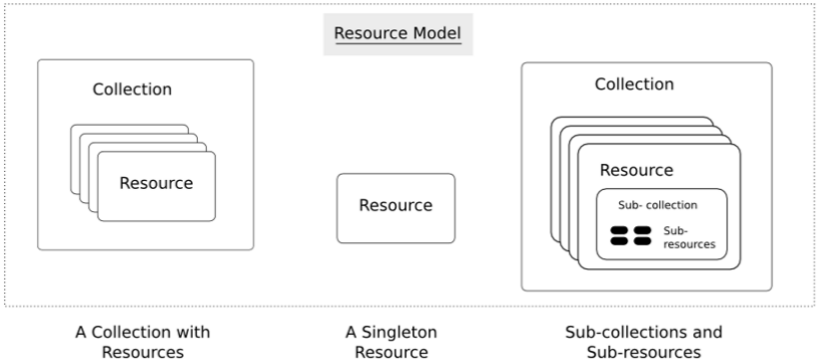
- **@Controller**
- Damit lassen sich non-Restcontroller erstellen.
- Hierbei würden die Methoden der annotierten Klasse nicht direkt an den Client ausliefern,
  - sondern zunächst an die Viewtechnologie von Spring MVC delegieren. (JSP,Thymeleaf)
  - Dabei wird eine HTML-Seite samt fachlichen Inhalt serverseitig erzeugt und an den Client ausgeliefert.

REST Architekturstil

- **Representational State Transfer**
- Der als REST bezeichnete Architekturansatz beschreibt,
  - wie verteilte Systeme miteinander kommunizieren können.

REST Architekturstil

- Das grundlegende Konzept in jeder RESTful-API ist die **Ressource.**
- **Was ist eine Ressource?**
- Eine Ressource ist ein Objekt
  - mit einem Typ,
  - zugehörigen Daten,
  - Beziehungen zu anderen Ressourcen
  - und einem Satz von Methoden, die auf ihr operieren.
  - Sie ähnelt einer Objektinstanz in einer objektorientierten Programmiersprache



REST Architekturstil

- In Ressource werden nur einige wenige Standardmethoden definiert
  - (entsprechend den Standard-HTTP-Methoden GET, POST, PUT und DELETE)
- https://tools.ietf.org/html/rfc2616

- **GET** - fordert Daten vom Server an
- **POST** - übermittelt Daten an den Server
- **PUT/PATCH** - ändern bestehende Daten auf dem Server
- **DELETE** - löscht bestehende Daten auf dem Server

REST Architekturstil

- **Wie komme ich zu einer Ressource?**

- URI und HTTP-Methode
  - führt zur Ressource und deren Funktionalität

REST Architekturstil

- Der Vorteil von REST liegt darin, dass im WWW bereits ein Großteil der für REST nötigen Infrastruktur vorhanden ist
  - (z. B. Web- und Application-Server,
  - HTTP-fähige Clients,
  - HTML- und XML-Parser, Sicherheitsmechanismen) vorhanden ist,
  - und viele Web-Dienste per se REST-konform sind.
- Eine Ressource kann dabei über verschiedene Medientypen dargestellt werden, auch *Repräsentation der Ressource* genannt.

@RestController

- **Wie bekommen wir das REST-Konzept mit Spring umgesetzt?**

- **@RestController**
  - ist ein Verwandter von **@Controller** bzw. **@Component**
  - erlaubt POJOS zu annotieren und damit Rest-Ressourcen zu bauen.
  - Darin werden HTTP-Methoden auf Javamethoden gemappt.

@Restcontroller


```
@RestController
@RequestMapping(path = "/persons", produces =
{MediaType.APPLICATION_JSON_UTF8_VALUE,MediaType.APPLICATION_XM
L_VALUE})
public class PersonRestController {
…..
```

- Klassen die mit **@Restcontroller** annotiert sind, enthalten **Pfad** und **HTTP-Methoden-mappings**.

```
@PutMapping(path = "books", consumes =
MediaType.APPLICATION_JSON_VALUE)
        public void updateBook(@RequestBody Book book) {
                booksService.updateBook(book);
        }
```

Pfadinformation

- Pfadinformation ergibt sich additiv.

```
@RestController
@RequestMapping(path = "/persons", produces =
{MediaType.APPLICATION_JSON_UTF8_VALUE,MediaType.APPLICATION_XML_VALUE})
public class PersonRestController {
…..

@PostMapping(path = "/{personId}/addresses", consumes = MediaType.APPLICATION_JSON_UTF8_VALUE)
public ResponseEntity<List<AddressResource>> addAddress(@PathVariable("personId") UUID personIdentifier,
@RequestBody AddressResource addressResource) {
……
```

Aufruf:

localhost:8080/persons/4711/addresses

- Dadurch werden Sie bei einem Request von Spring MVC auswählbar,
  - um Requests innerhalb bestimmter Methoden entgegenzunehmen.

Datenformat der Kommunikation

- Das Datenformat der Kommunikation ist oft JSON und/ oder XML.
- Die Pfade einer Restresource können dynamische Anteile enthalten ({isbn}),
  - die dann auf der Serverseite ausgelesen werden können mittels @PathVariable("isbn")

```
@GetMapping(path = "books/{isbn}", produces =
MediaType.APPLICATION_JSON_VALUE)
public Book findBook(@PathVariable("isbn") String isbn) {
        return booksService.findBookByIsbn(isbn);
}
```

Statuscodes

- Ein Restaufruf sollte HTTP-konforme Statuscodes zurückliefern.
  - https://tools.ietf.org/html/rfc2616

```
@PostMapping(path = "/{personId}/addresses", consumes =
MediaType.APPLICATION_JSON_UTF8_VALUE)
public ResponseEntity<List<AddressResource>> addAddress(@PathVariable("personId") UUID
personIdentifier, @RequestBody AddressResource addressResource) {
……
return ResponseEntity.ok(personService.save(person).getAddresses()          .stream()
.map(a -> new ModelMapper().map(a, AddressResource.class))
.collect(Collectors.toList()));
```

- Die Klasse **ResponseEntity** erlaubt das verpacken von Statuscodes samt Fachdaten

Exceptions


```
@ExceptionHandler(DataIntegrityViolationException.class)
public ResponseEntity<String> handleIntegrityViolations(DataIntegrityViolationException ex) {
        LOGGER.error("Integrity violation: {}", ex.getMessage());
        return ResponseEntity.badRequest().body("Submitted data is not valid");
}
@ExceptionHandler(Exception.class)
public ResponseEntity<String> handleInternalErrors(Exception ex) {
        LOGGER.error("General error: {}", ex.getMessage());
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
        .body("Submitted data is not valid");
 }
```

- **@Exceptionhandler** markiert eine Methode als Handler (Abarbeiter), im Falle einer Exception bestimmten Typs.

Exceptions

- Was macht manwenn mehrere Controllerklassen mit gleichen Handlern vorkommen?
- Codewiederholung?

- Baue eine neue Klasse
  - mit **@ControllerAdvice** annotieren
  - darin die globalen Handlermethoden implementieren
  - diese mit **@Exceptionhandler** annotieren

- https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc