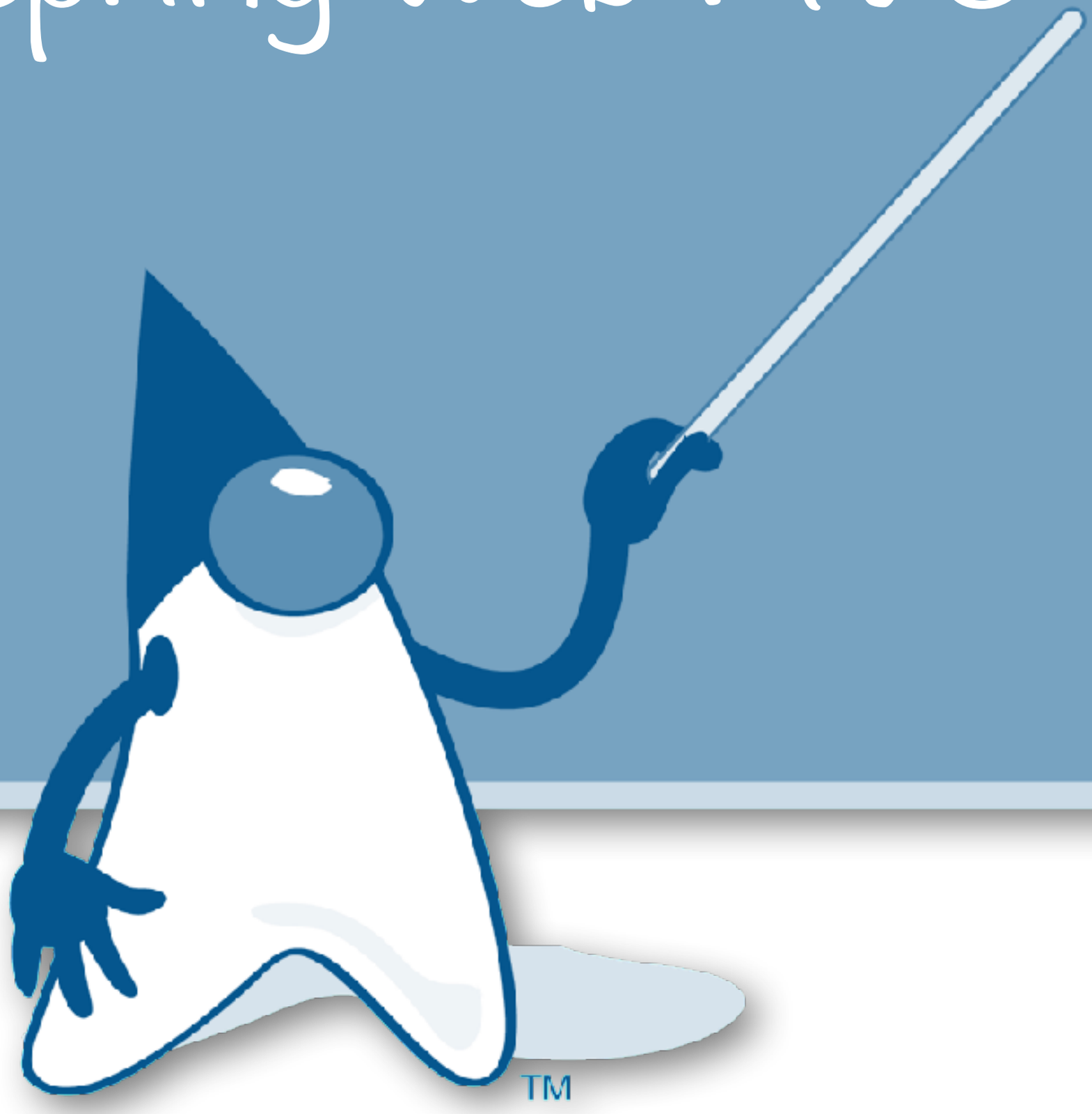
Info Support
Solid Innovator

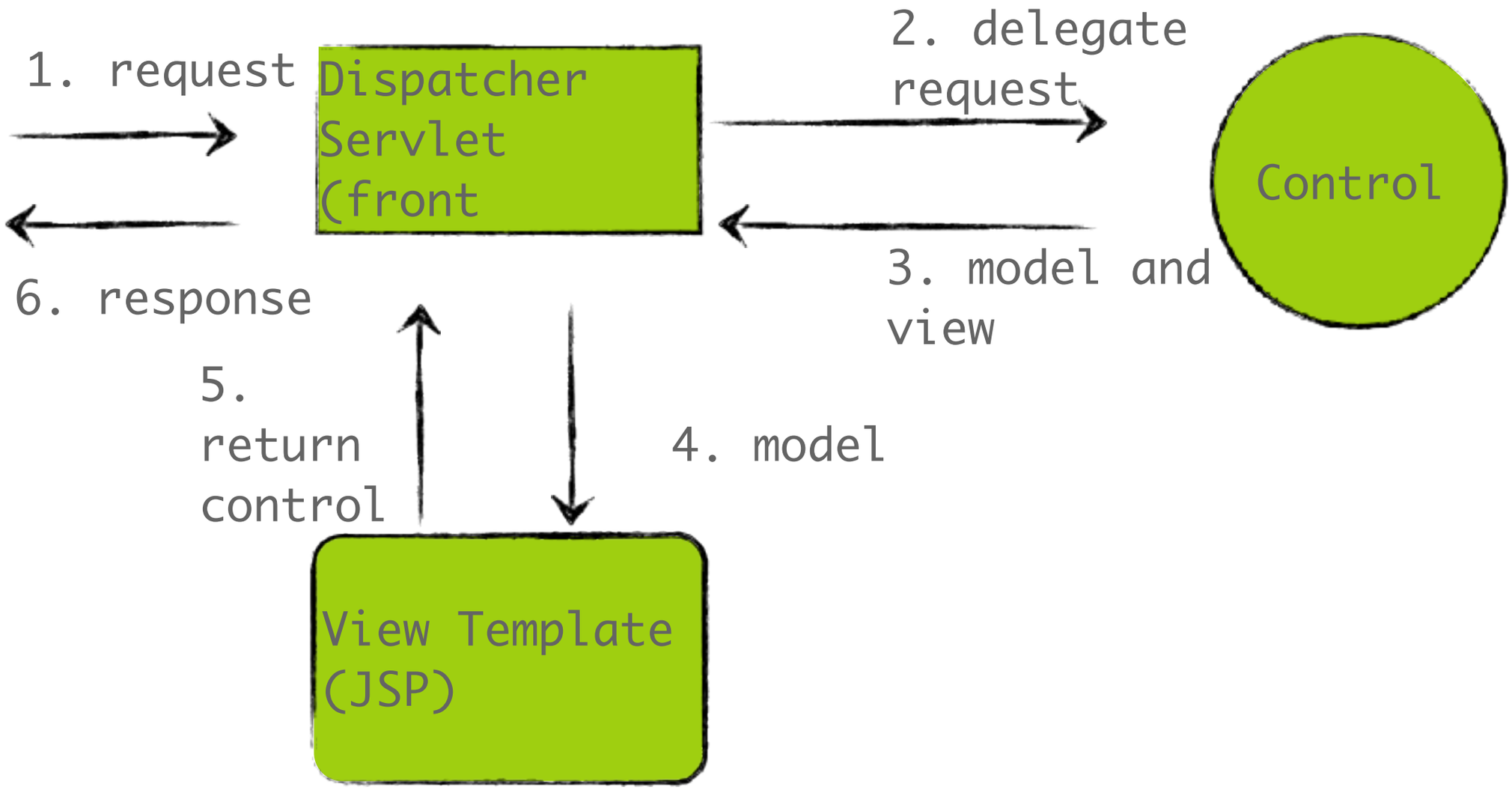
Spring Web MVC



TM

infoSupport
Solid Innovator

Handling requests



Configuring Web MVC

Add the Dispatcher Servlet to web.xml

Needs a WEB-INF/spring-servlet.xml config file

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/spring/*</url-pattern>
</servlet-mapping>
```

Additional configuration files

web.xml

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml
        /WEB-INF/applicationContext-security.xml
    </param-value>
</context-param>

<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

Implementing controllers

Controller stereotype

Handle url: /spring/example

Name of the view template

The model is a Map

```
@Controller
public class ExampleController {
    @RequestMapping("/example")
    public ModelAndView example() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("hello");
        mav.addObject("message", "Hello world!");
        return mav;
    }
}
```

Thymeleaf page

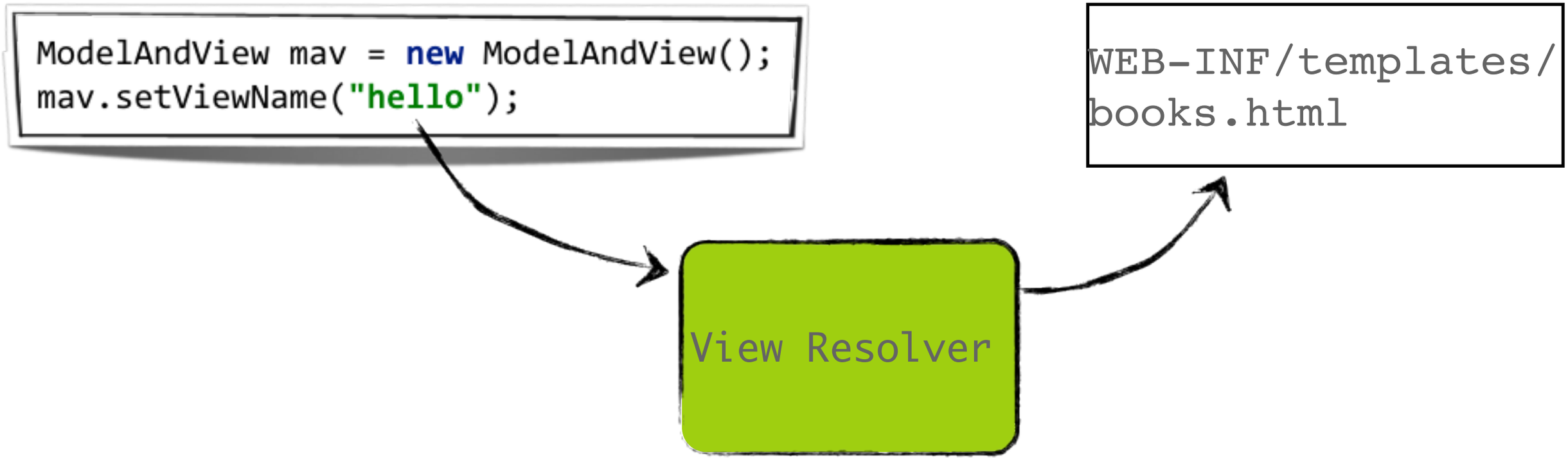
- Use Expression Language to render model values
- Use thymeleaf and Spring tags to make pages dynamic

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<body>
 <table >
    <tr>
        <th>Id</th><th>Title</th>
    </tr>
    <tr th:each="book : ${books}">
        <td th:text="${book.id}">1</td>
        <td th:text="${book.title}">Java 10</td>
    </tr>
  </table>
</body>
</html>
```

View resolvers

- View names are resolved to views using view resolvers

```
ModelAndView mav = new ModelAndView();
mav.setViewName("hello");
```



Most useful View resolvers

View resolver	Explanation
`InternalResourceViewResolver`	Servlet and JSP view resolver. View names are prefixed and suffixed to generate file name
`ResourceBundleViewResolver`	Views are defined in views.properties. Supports multiple kinds of views working together (e.g. JSP and Excel)
`ContentNegotiatingViewResolver`	Resolve views based on the *accept* header in the HTTP request. Useful for RESTful web services

There are more view resolvers,
but are less likely to be used

Resolving Thymeleaf views

```
@Bean
public SpringResourceTemplateResolver templateResolver(){

    SpringResourceTemplateResolver templateResolver =
                        new SpringResourceTemplateResolver();
    templateResolver.setApplicationContext(this.applicationContext);
    templateResolver.setPrefix("/WEB-INF/templates/");
    templateResolver.setSuffix(".html");

    templateResolver.setTemplateMode(TemplateMode.HTML);

    return templateResolver;
}
```

Resolving Thymeleaf views cont.

```java
@Bean
public SpringTemplateEngine templateEngine(){
    // SpringTemplateEngine automatically applies SpringStandardDialect and
    // enables Spring's own MessageSource message resolution mechanisms.
    SpringTemplateEngine templateEngine = new SpringTemplateEngine();
    templateEngine.setTemplateResolver(templateResolver());

    templateEngine.setEnableSpringELCompiler(true);
    return templateEngine;
}

@Bean
public ThymeleafViewResolver viewResolver(){
    ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
    viewResolver.setTemplateEngine(templateEngine());
    return viewResolver;
}
```

Creating PDF documents

```
public class MyPdfView extends AbstractPdfView{
    @Override
    protected void buildPdfDocument(
            Map<String, Object> model,
            com.lowagie.text.Document document,
            com.lowagie.text.pdf.PdfWriter pdfWriter,
            HttpServletRequest httpServletRequest,
            HttpServletResponse httpServletResponse) throws Exception {
        document.add(new Paragraph((String)model.get("message")));
    }
}
```

Resolving PDF views

```
<bean id="pdfViewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="order" value="1"/>
    <property name="basename" value="views"/>
</bean>
```

views.properties

```
mypdf.(class)=dvdstore.views.MyPdfView
```

controller

```
ModelAndView mav = new ModelAndView();
mav.setViewName("mypdf");
```

URI Templates

- URLs can contain variable data
  - /products/{productId}
  - /products/{category}/all
- Use parameter in request handling

```
@RequestMapping("products/{productId}")
public ModelAndView productDetails(@PathVariable int productId) {
    ModelMap model = new ModelMap();
    model.addAttribute("productId", productId);
    return new ModelAndView("product", model);
}
```

Accessing request parameters

- /search?query=Spring

```
@RequestMapping("search")
public void search(@RequestParam(required = true) String query) {
    System.out.println("Query: " + query);
}
```

Accessing request headers

Request Headers

Name	Value
Accept	text/html

```
@RequestMapping("/example")
public ModelAndView example(@RequestHeader String accept) {
    System.out.println("Accept header: " + accept);
```

Accessing request cookies

```java
@RequestMapping("/example")
public ModelAndView example(@CookieValue String userId) {
    System.out.println("Cookie value: " + userId);
```

Request method arguments

Argument Type	Explanation
`@RequestBody`	The body of the request mapped by a HttpMessageConverter (for RESTful web services)
`HttpServletRequest / HttpServletResponse`	Plain Servlet API request and response
`HttpSession`	The Servlet API session object (think about thread safety)
`Locale`	The current request's locale
`InputStream / Reader`	Raw inputstream to access the request's content
`OutputStream / Write`	Raw outputstream to write response content
`Map / Model / ModelMap`	Implicit model that's exposed to the web view
`Command objects`	Command objects available in the model
`Errors / BindingResult`	Validation results for the preceding command object
`SessionStatus`	Status handle for marking form processing complete, triggering session cleanup

Request method return types

Return types	Explanation
`ModelAndView`	Combination of explicit model and view object
`Model / Map`	View name is resolved using the `RequestToViewNameTranslator`
`View`	Model resolved using implicit model command objects
`String`	View name. Model is resolved using implicit model command objects
`Void`	Model resolved using implicit model command objects. View name is resolved using the `RequestToViewNameTranslator`
`@ResponseBody`	Object converted using a HttpMessageConverter. Useful for RESTful web services

Showing a form

Handle only GET requests

```
@RequestMapping(value = "books/edit", method = RequestMethod.GET)
public String editBook(Book book) {
    return "books/edit";
}
```

Create implicit 'book' command object

HTML Form

Form fields will be saved to

```
<form  th:object="${book}" th:action="@{/book}" method="post">

    <input type="hidden" th:field="*{id}"/>

    <input type="text" class="form-control" th:field="*{title}"/>

    <button type="submit">Submit</button>

</form>
```

Property on

Submitting a form

All properties will be set on

```
@RequestMapping(value = "books/edit", method = RequestMethod.POST)
public String saveBook(Book book) {
    bookCatalog.saveBook(book);
    return "redirect:/spring/books";
}
```

Redirect to overview page

Bean Validation

- Bean Validation API (JSR-303)
- Define field constraints on Java classes
- Integrates with other frameworks
  - e.g. JSF 2.0, Spring 3, JPA 2

Validation Example

- @Null
- @NotNull
- @AssertTrue
- @AssertFalse
- @Min
- @Max
- @DecimalMin
- @DecimalMax
- @Size
- @Digits
- @Past
- @Future
- @Pattern

```
@Entity
public class Employee {
    @Id
    private Long id;

    @NotNull @Size(min = 2, max = 20)
    private String name;

    @Past
    private Date birthDate;

    @Min(value = 1000)
    private double salary;
```

Triggering validation

Trigger bean validation

```
@RequestMapping(value = "employees/edit", method = RequestMethod.POST)
public String saveBook(@Valid Employee employee,
                       BindingResult result) {
    if (result.hasErrors()) {
        return "employee/addForm";
    } else {
        //Save employee
        System.out.println("Employee: " + employee.getFirstname());
        return "redirect:/spring/employees";
    }
}
```

Contains error information

Rendering validation errors

Check if there are errors

```
<form  th:action="@{/}" th:object="${book}" method="post">

      <ul th:if="${#fields.hasErrors('*')}">
        <li th:each="err : ${#fields.errors('*')}" th:text="${err}">Input is incorrect</li>
      </ul>
        <table>
            <tr>
                <td>Title:</td>
                <td><input type="text" th:field="*{title}" /></td>
                <td th:if="${#fields.hasErrors('title')}" th:errors="*{title}">Title Error</td>
            </tr>
            <tr>
                <td><button type="submit">Submit</button></td>
            </tr>
        </table>
</form>
```

Render error for property

Custom bean validation invocation

```
validator.validate(Object instance, Class<?>...
groups)
```

```
ValidatorFactory factory = Validation.buildDefaultValidatorFactory();
Validator validator = factory.getValidator();

Set<ConstraintViolation<Employee>> violations =
        validator.validate(emp1, Communicating.class);

for (ConstraintViolation<Employee> violation : violations) {
    System.out.println("Error: " + violation.getMessage());
}
```

Creating constraints

- Create annotation
- Implement validator class

```
@ElementCollection
@NonEmptyCollection
private Set<String> emailAddresses;
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.FIELD})
@Constraint(validatedBy = NonEmptyCollectionValidator.class)
public @interface NonEmptyCollection {
    String message() default "Collection may not be empty";
    Class<?>[] groups() default {};
    Class<? extends Payload>[] payload() default {};
}
```

Creating constraints

```
public class NonEmptyCollectionValidator implements
        ConstraintValidator<NonEmptyCollection, Collection<?>> {
    @Override
    public void initialize(NonEmptyCollection nonEmptyCollection) {

    }

    @Override
    public boolean isValid(Collection<?> objects,
                           ConstraintValidatorContext
                                   constraintValidatorContext) {
        return objects != null && objects.size() > 0;
    }
}
```

Validation Groups

- Define different sets of constraints for different situations
- Trigger validation only for a certain constraints

```
@ElementCollection
@NonEmptyCollection(groups = Communicating.class)
private Set<String> emailAddresses;
```

```
public interface Communicating {
}
```

```
validator.validate(emp1, Communicating.class);
```

ModelAttribute

Method to produce model

```
@ModelAttribute("roles")
public List<String> listRoles() {
    return Arrays.asList("Developer", "Sales", "Management");
}
```

Handling exceptions

Map exceptions to custom error pages

```
<bean class="...SimpleMappingExceptionResolver">
    <property name="defaultErrorView" value="error"/>
    <property name="exceptionMappings">
        <util:map>
            <entry key="dvdstore.controllers.MyCustomException"
                   value="customerror"/>
        </util:map>
    </property>
</bean>
```

Handling exceptions

Handle exceptions from code

```
@Component
public class CustomErrorResolver
        implements HandlerExceptionResolver {
    @Override
    public ModelAndView resolveException(HttpServletRequest request,
                                         HttpServletResponse response,
                                         Object handler,
                                         Exception ex) {
        if (ex instanceof MyCustomException) {
            return new ModelAndView("customerror");
        } else {
            return new ModelAndView("error");
        }
    }
}
```

Interceptors

- Intercept requests before handling
- Similar to Servlet Filters but within the Spring context
- Implement HandlerInterceptor interface
  - or extend HandlerInterceptorAdaptor convenience class

Interceptor example

```
public class TimeBasedAccessInterceptor extends
        HandlerInterceptorAdapter {

    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        //implement intercepter logic here
        if (1 > 2) {
            return true;
        } else {
            response.setStatus(403);
            return false;
        }

    }
}
```

Configuring interceptors

```
<mvc:interceptors>
    <bean class="...TimeBasedAccessInterceptor"/>

    <bean class="...ThemeChangeInterceptor">
        <property name="paramName" value="theme"/>
    </bean>

    <bean class="...LocaleChangeInterceptor">
        <property name="paramName" value="lang"/>
    </bean>
</mvc:interceptors>
```

Intermezzo Thymeleaf

- Thymeleaf is a Java library
- An XML/XHTML/HTML5 template engine
- Transforms template files to display data / text produced by applications
- Most of the processors of the Standard Dialect are attribute processors
  - Allows browsers to correctly display HTML5 template files before processing because they will simply ignore the additional attributes

Thymeleaf details

```
<input type="text" name="title" value="James Carrot"
th:value="“${book.title}” />
```

- Allows to (optionally) specify a value attribute, here “James Carrot”, that will be displayed when the page is directly opened by browser
- It is substituted by the value resulting from the evaluation of ${book.title} during Thymeleaf processing of the template when browser request this page from the web application

Thymeleaf Syntax: ${}

```
<input type="text" name="title" th:value="“${book.title}” />
```

- The ${reference to attribute on request}
- Here an attribute with name book
- The attribute refers to Book instance with member title
- th:value is runtime evaluated and assigned to the value attribute

Thymeleaf Syntax: ${}

```
<input type="text" name=“title" th:value=“${book.title}” />
```

- ${x} will return a variable stored into the Thymeleaf context or request object
- ${param.x} will return request param x
- ${session.x} will return session param x
- ${application.x} returns ServletContext param x

Thymeleaf Syntax: th:each iteration

```
<tr th:each="book : ${books}">
      <td th:text="${book.id}">1</td>
      <td th:text="${book.title}">Java 10</td>
</tr>
```

- ${books} will return a reference to a request attribute books representing a List<Book>
- book is an ad hoc name for individual elements of type Book
- Note the resemblance with the enhanced for loop in Java

Thymeleaf Syntax: th:object

```
<form  th:object="${book}" th:action="@{/book}" method="post">

  <input type="hidden" th:field="*{id}"/>

  <label>Title:</label> <input type=“text"  th:field="*{title}"/>
  <button type="submit">Submit</button>

</form>
```

- th:object=“{book}” -> inside the childelements the * operator can be used to refer to fields of book
- i.e. : th:field=“*{title}” -> refers title attribute of book
- th:field -> is translated into id and name attribute

Thymeleaf Syntax: th:object

```
<form  th:object="${book}" th:action="@{/book}" method="post">

  <input type="hidden" th:field="*{id}"/>

  <label>Title:</label> <input type=“text"  th:field="*{title}"/>
  <button type="submit">Submit</button>

</form>
```

- becomes after processing of the Thymeleaf template engine:

```
<form action="/web-mvc/book" method="post">

  <input type="hidden" id="id" name="id" value="19"/>

  <label >Title:</label> <input type="text" id="title" name="title" value="Java 9"/>
  <button type="submit" >Submit</button>

</form>
```

Thymeleaf Syntax: @{path}

```
<a href="books.html" th:href="“@{/books}">BookList</a>
```

- The @{path}
- @ refers to context root -> the name of the web app
- th:href is runtime evaluated by the template engine and the value is assigned to the href attribute

```
<a href="/web-mvc/books">BookList</a>
```

Thymeleaf Syntax: #{home.welcome}

```
<p th:text="#{home.welcome}">Welcome to the library</p>
```

- Use #{key} to externalise and internationalise text
- The #{internationalised message}
- home.welcome refers to key in properties file
- default language: -> home.properties
  - home_en.properties -> english
  - home_nl.properties -> dutch
  - home_fr.properties -> french

Thymeleaf Syntax: #{home.welcome}

home.properties

```
home.welcome=Welcome to the library
logo=Royal Library
date.format=MMMM dd'',''yyyy
```

home_nl.properties

```
home.welcome=Welkom in de bibliotheek
logo=De koningklukke bibliotheek
date.format=MMMM dd'',''yyyy
```

Thymeleaf Syntax: #{home.welcome}

home.properties

```
home.welcome=Welcome to the library
logo=Royal Library
date.format=MMMM dd'',''yyyy
```

home_nl.properties

```
home.welcome=Welkom in de bibliotheek
logo=De bibliotheek van de koning
date.format=MMMM dd'',''yyyy
```

Syntax: Simple expressions

- Variable Expressions: `${...}`
- Selection Variable Expressions: `*{...}`
- Message Expressions: `#{...}`
- Link URL Expressions: `@{...}`

Syntax: Literals

- Text literals: `'one text'`, `'Another one!'`,...
- Number literals: `0`, `34`, `3.0`, `12.3`,...
- Boolean literals: `true`, `false`
- Null literal: `null`
- Literal tokens: `one`, `sometext`, `main`,...

Syntax: Operations

- Text operations:
  - String concatenation: `+`
  - Literal substitutions: `|The name is ${name}|`
- Arithmetic operations:
  - Binary operators: `+`, `-`, `*`, `/`, `%`
  - Minus sign (unary operator): `-`

Syntax: Operations

- Boolean operations:
  - Binary operators: `and` , `or`
  - Boolean negation (unary operator): `!` , `not`
- Comparisons and equality:
  - Comparators: `>` , `<` , `>=` , `<=` (`gt` , `lt` , `ge` , `le`)
  - Equality operators: `==` , `!=` (`eq` , `ne`)

Syntax: Operations

- Conditional operators:
  - If-then: `(if) ? (then)`
  - If-then-else: `(if) ? (then) : (else)`
  - Default: `(value) ?: (defaultvalue)`

Combining all these features

```
'Book is of age category ' +
(${book.isAdult()} ? 'Adult' : (${user.type} ?: 'Unknown'))
```

Syntax: Expression utility methods

- `#dates` : utility methods for `java.util.Date` objects: formatting, component extraction, etc.
- `#calendars` : analogous to `#dates` , but for `java.util.Calendar` objects.
- `#numbers` : utility methods for formatting numeric objects.
- `#strings` : utility methods for `String` objects: contains, startsWith, prepending/appending, etc.
- `#objects` : utility methods for objects in general.
- `#bools` : utility methods for boolean evaluation.
- `#arrays` : utility methods for arrays.
- `#lists` : utility methods for lists.
- `#sets` : utility methods for sets.
- `#maps` : utility methods for maps.
- `#aggregates` : utility methods for creating aggregates on arrays or collections.

Expression utility methods, examples

```
<td th:text="${book.onSale}? #{true} : #{false}">yes</td>
<td>
  <span th:text="“${#lists.size(book.reviews)}">2</span> reviews
  <a href="reviews.html"
     th:href="@{/book/review(id=${book.id})}"
     th:unless="${#lists.isEmpty(book.reviews)}">view</a>
</td>
```

```
<td>no</td>
<td>
  <span>2</span> comment/s
  <a href="/web-mvc/book/reviews?id=2“>view</a>
  <a href="/gtvg/product/comments?prodId=2">view</a>
</td>
```