Spring MVC Misc. Features

"Code with Passion!"

Topics

- Exception handling
- Locale handling
- @Value
- SpEL
- Static resource configuration
- Logging
- Debugging

Exception Handling

@ExceptionHandler

```
@Controller
public class SimpleController {

    // @RequestMapping methods omitted ...


    // Handles exceptions thrown within the @RequestMapping methods in this controller
    @ExceptionHandler(IOException.class)
    public ResponseEntity<String> handleIOException(IOException ex) {
        // prepare responseEntity
        return responseEntity;
    }

}
```

@ControllerAdvice

- You can also declare an @ExceptionHandler method within an @ControllerAdvice class in which case the exception handler handles exceptions across all controllers under the specified package

```
@Controller
@ControllerAdvice("com.jpassion.mvc")
public class SimpleController {

    // @RequestMapping methods omitted ...

    @ExceptionHandler(IOException.class)
    public ResponseEntity<String> handleIOException(IOException ex) {
        // prepare responseEntity
        return responseEntity;
    }

}
```

@ResponseStatus

- Marks a method or exception class with the status code and reason that should be returned
- The status code is applied to the HTTP response

```
@ResponseStatus(value=HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

    private Long resourceId;
    private Date date;

    public ResourceNotFoundException(Long resourceId) {
        this.resourceId = resourceId;
        date = new Date();
    }

    ...
}
```

What happens under the cover

- *HandlerExceptionResolver* interface
  - Implementations of this interface handles exceptions

- Behind the scenes, Spring MVC creates and configures three resolvers by default
  - *ExceptionHandlerExceptionResolver matches uncaught exceptions against for suitable @ExceptionHandler methods on both the handler (controller) and on any controller-advices.*
  - *ResponseStatusExceptionResolver looks for uncaught exceptions annotated by @ResponseStatus*
  - *DefaultHandlerExceptionResolver converts standard Spring exceptions and converts them to HTTP Status Codes*

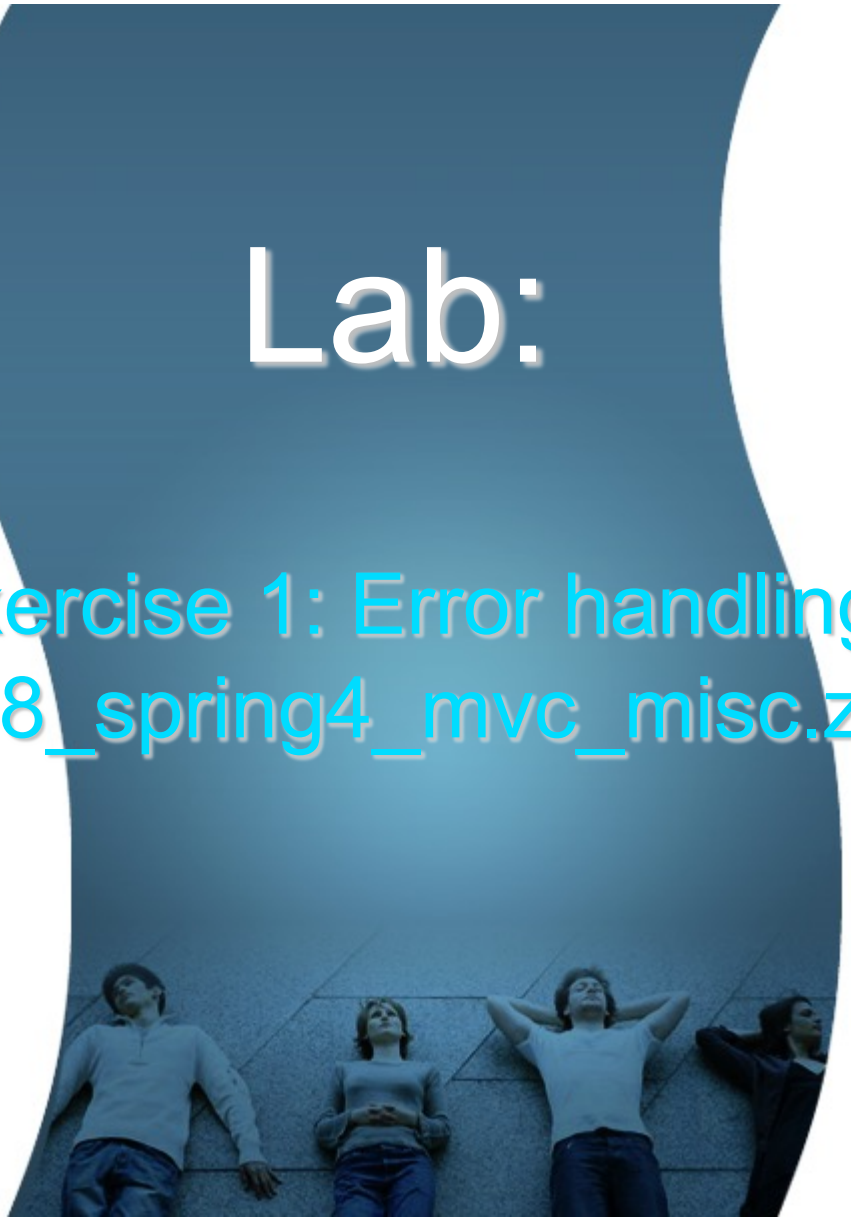
DefaultHandlerExceptionResolver

- This resolver handles certain standard Spring MVC exceptions by setting a specific HTTP response status code
  - *ConversionNotSupportedException* -> 500
  - *HttpMediaTypeNotAcceptableException* -> 406
  - *HttpMediaTypeNotSupportedException*-> 415
  - *NoSuchRequestHandlingMethodException* -> 404
  - *TypeMismatchException* -> 400 (Bad Request)
  - *MissingServletRequestParameterException* -> 400

Lab:

Exercise 1: Error handling

4948_spring4_mvc_misc.zip



Locale Handling

Locale Support in Spring MVC

- Configure *LocaleResolver*
- Provide an Interceptor for changing locale

LocaleResolver Interface

- *DispatcherServlet* enables you to automatically resolve messages using the client's locale
  - Through *LocaleResolver* object
  - When a request comes in, the *DispatcherServlet* looks for a locale resolver, and if it finds one, it tries to use it to set the locale
- Spring provides several *LocaleResolver* implementation classes
  - *AcceptHeaderLocaleResolver (default)*
  - *CookieLocaleResolver*
  - *SessionLocaleResolver*
- Using the *RequestContext.getLocale()* method, you can always retrieve the locale that was resolved by the locale resolver

AcceptHeaderLocaleResolver

- This locale resolver inspects the *accept-language* header in the request that was sent by the client (e.g., a web browser)
- Usually this header field contains the locale of the client's operating system
- If you do not configure your own LocaleResolver, Spring will use the AcceptHeaderLocaleResolver as a default

CookieLocaleResolver & SessionLocaleResolver

- *CookieLocaleResolver*
  - This locale resolver inspects a cookie that might exist on the client to see if a locale is specified. If so, it uses the specified locale.
  - Using the properties of this locale resolver, you can specify the name of the cookie as well as the maximum age

- *SessionLocaleResolver*
  - Use it if the application needs user sessions anyway, that is, when the HttpSession does not have to be created for the locale
  - The session may optionally contain an associated time zone attribute as well; alternatively, you may specify a default time zone.

SessionLocaleResolver Configuration

```
@Bean(name = "localeResolver")
public LocaleResolver getLocaleResolver() {
    SessionLocaleResolver localeResolver = new SessionLocaleResolver();
    localeResolver.setDefaultLocale(new Locale("de_DE"));
    return localeResolver;
}
```

Provide Interceptor for changing locale

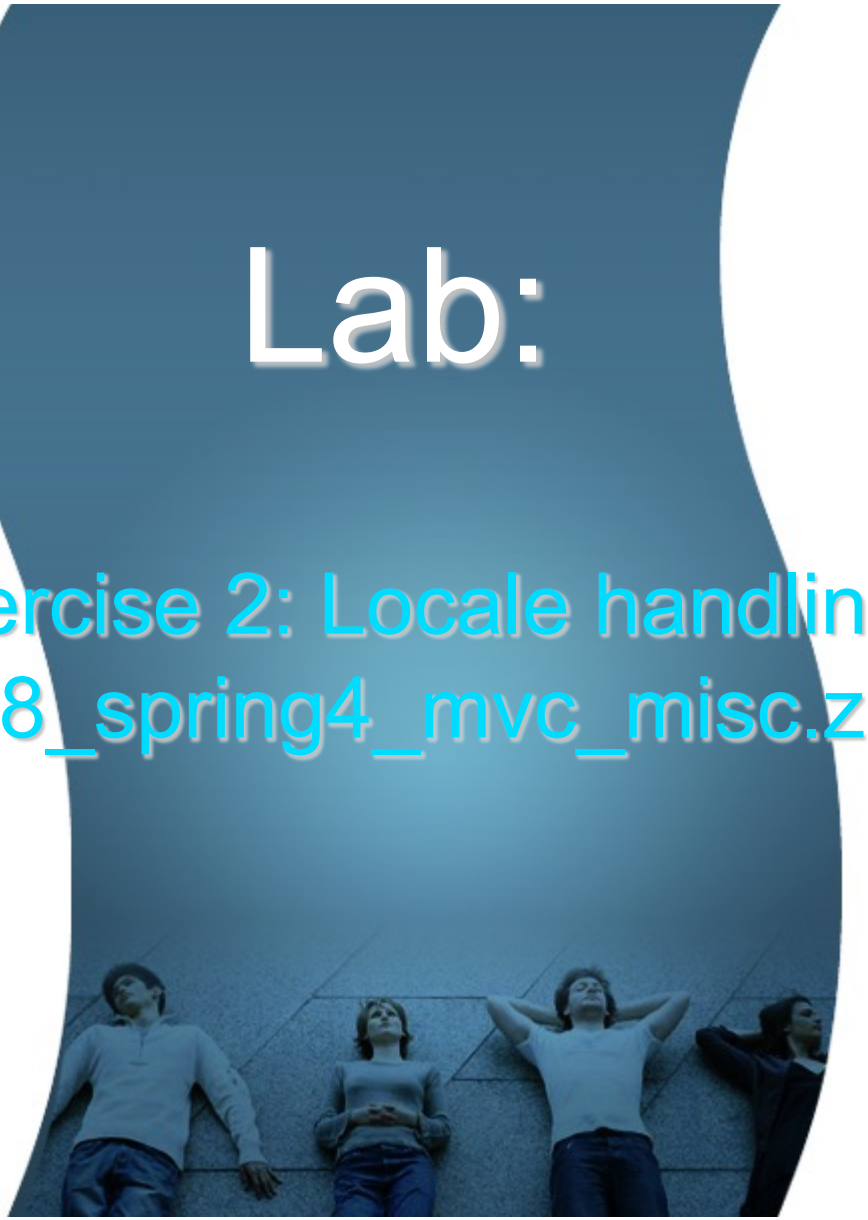
- You can enable runtime change of locales by configuring a Locale Interceptor implementation class
- Spring provides a Locale Interceptor implementation class
  - *org.springframework.web.servlet.i18n.LocaleChangeInterceptor*
- It will detect “change locale” request parameter and change locale accordingly
  - Default key is “*locale*”
  - For example, *?locale=de*

LocaleChangeInterceptor Configuration

```
// Define and add locale change interceptor to the interceptor registry
@Bean
public LocaleChangeInterceptor localeChangeInterceptor() {
    LocaleChangeInterceptor localeChangeInterceptor = new LocaleChangeInterceptor();
    localeChangeInterceptor.setParamName("locale");
    return localeChangeInterceptor;
}
```

Lab:

Exercise 2: Locale handling
4948_spring4_mvc_misc.zip



@Value

@Value

- The *@Value* annotation can be placed on fields, methods and method/constructor parameters to specify a default value or value from properties file

```
@Value("Live your life with Passion!")
private String somewords;
@Value("25")
private int someNumber;
@Value("${username}")    // Read value from system properties or a properties file
private String username;

@RequestMapping("/about")
public String courtReservation(Model model) {
    model.addAttribute("somewords", somewords);
    model.addAttribute("someNumber", someNumber);
    model.addAttribute("username", username);
    return "about";
}
```

Lab:

Exercise 3: @Value
4948_spring4_mvc_misc.zip

SpEL (Spring Expression Language)

SpEL

- Powerful expression language that supports querying and manipulating an object graph at runtime.
- The language syntax is similar to Unified EL (from JSP 2.1) but offers additional features, most notably
  - Basic string templating functionality
  - Method invocation
- SpEL expressions can be used with XML or annotation-based configuration metadata for defining BeanDefinitions
  - In both cases the syntax to define the expression is of the form #{ <expression string> }

Example #1: Usage of @Value with SpEL

```
@Component
public class Author {

    @Value("Sang Shin")
    private String name;

    // Bean reference in SpEL
    @Value("#{bookBean}")
    private Book book;

    @Value("Hello, #{bookBean.title}")
    private String bookTitle;

    // Method invocation in SpEL
    @Value("Sang says #{bookBean.getSomeInfo()}")
    private String bookInfo;

    …
}
```

Example #2: Usage of @Value with SpEL

```
@Controller
public class AboutController {

    // "messageSource" bean is declared in the configuration class
    // geMessage(<key>, <Array of arguments>, <locale>
    @Value("#{ messageSource.getMessage('welcome.title',null,'de')}")
    private String welcome;

    @Value("#{ messageSource.getMessage('admin.email',null,'de')}")
    private String email;

    @Value("#{ systemProperties['user.country'] }")
    private String defaultCountry;

    …
}
```

Lab:

Exercise 4: SpEL

4948_spring4_mvc_misc.zip

Static Resource Configuration

Static Resource Configuration

- Provides a convenient way to serve static resources from locations other than the web application root, including locations on the classpath
- The *CachePeriod* property may be used to set far future expiration header

```
@Bean
WebMvcConfigurerAdapter mvcViewConfigurer() {
    return new WebMvcConfigurerAdapter() {

        @Override
        public void addResourceHandlers(ResourceHandlerRegistry registry) {
            registry.addResourceHandler("/resources/**")
                    .addResourceLocations("/myresources/", "/myresources1/")
                    .setCachePeriod(3600);
        }
    };
}
```

Lab:

Exercise 5: Static Resource Configuration
4948_spring4_mvc_misc.zip

Logging

application.properties (with Spring Boot)

```
logging.level.org.springframework.web=DEBUG
logging.level.com.jpassion.mvc=DEBUG
```

Logging Levels

- DEBUG (most verbose)
- INFO
- WARN
- ERROR
- FATAL (least verbose)

Lab:

Exercise 6: Logging
4948_spring4_mvc_misc.zip

Debugging

Debugging

- Source-code level debugging is supported by all major IDE's
  - Setting breakpoints
  - Step by step source code-level debugging
  - Watching Variables
  - ..

Lab:

Exercise 7: Debugging
4948_spring4_mvc_misc.zip