Exception Handling in Spring MVC

Using @ExceptionHandler

You can add extra (@ExceptionHandler) methods to any controller to specifically handle exceptions thrown by request handling (@RequestMapping) methods in the same controller. Such methods can:

```
@Controller

public class ExceptionHandlingController {

  // @RequestHandler methods

     .. . . .

  // Exception handling methods

  // Convert a predefined exception to an HTTP Status code

  @ResponseStatus(value=HttpStatus.CONFLICT,reason="Data integrity violation")  /
/ 409

  @ExceptionHandler(DataIntegrityViolationException.class)

  public void conflict() {

    // Nothing to do

  }

  // Specify name of a specific view that will be used to display the error:

  @ExceptionHandler({SQLException.class,DataAccessException.class})

  public String databaseError() {

    // Nothing to do.  Returns the logical view name of an error page, passed

    // to the view-resolver(s) in usual way.
    // Note that the exception is NOT available to this view (it is not added

    // to the model) but see "Extending ExceptionHandlerExceptionResolver"

    // below.

    return "databaseError";

  }

  // Total control - setup a model and return the view name yourself. Or

  // consider subclassing ExceptionHandlerExceptionResolver (see below).

  @ExceptionHandler(Exception.class)

  public ModelAndView handleError(HttpServletRequest req, Exception ex) {

    logger.error("Request: " + req.getRequestURL() + " raised " + ex);

    ModelAndView mav = new ModelAndView();

    mav.addObject("exception", ex);

    mav.addObject("url", req.getRequestURL());

    mav.setViewName("error");

    return mav;

  }

}
```

Global Exception Handling

Using @ControllerAdvice Classes

A controller advice allows you to use exactly the same exception handling techniques but apply them across the whole application, not just to an individual controller.

```
@ControllerAdvice

class GlobalDefaultExceptionHandler {

  public static final String DEFAULT_ERROR_VIEW = "error";

  @ExceptionHandler(value = Exception.class)

  public ModelAndView

  defaultErrorHandler(HttpServletRequest req, Exception e) throws Exception {

    // If the exception is annotated with @ResponseStatus rethrow it and let

    // the framework handle it - like the OrderNotFoundException example

    // at the start of this post.

    // AnnotationUtils is a Spring Framework utility class.

    if (AnnotationUtils.findAnnotation

                (e.getClass(), ResponseStatus.class) != null)

      throw e;

    // Otherwise setup and send the user to a default error-view.

    ModelAndView mav = new ModelAndView();

    mav.addObject("exception", e);

    mav.addObject("url", req.getRequestURL());

    mav.setViewName(DEFAULT_ERROR_VIEW);

    return mav;

  }

}
```

SimpleMappingExceptionResolver

It provides options to:

- Map exception class names to view names - just specify the classname, no package needed.
- Specify a default (fallback) error page for any exception not handled anywhere else
- Log a message (this is not enabled by default).
- Set the name of the exception attribute to add to the Model so it can be used inside a View

Here is a typical configuration using XML:

```
<bean id="simpleMappingExceptionResolver" class=

     "org.springframework.web.servlet.handler.SimpleMappingExceptionResolver">

    <property name="exceptionMappings">

      <map>

         <entry key="DatabaseException" value="databaseError"/>

         <entry key="InvalidCreditCardException" value="creditCardError"/>

      </map>

    </property>

    <!-- See note below on how this interacts with Spring Boot -->

    <property name="defaultErrorView" value="error"/>

    <property name="exceptionAttribute" value="ex"/>

    <!-- Name of logger to use to log exceptions. Unset by default,

           so logging is disabled unless you set a value. -->

    <property name="warnLogCategory" value="example.MvcLogger"/>

  </bean>
```