Exception Handling and Advanced Caching

Exception handling is a very essential feature of any Java application. Every good open-source framework, such as Spring Boot, allows writing the exception handlers in such a way that we can separate them from our application code. The Spring framework also allows us to do so using the annotations @ControllerAdvice and @ExceptionHandler.

- The @ControllerAdvice annotation is used to define a class that will handle exceptions globally across all controllers. Its methods are annotated with @ExceptionHandler, @InitBinder, and @ModelAttribute annotations.
- The @ExceptionHandler annotation is used to handle specific exceptions. The annotated method is invoked when the specified exceptions are thrown from a @Controller. We can define these methods either in a @Controller class or in @ControllerAdvice class.

`Quick Reference`

```
@ControllerAdvice
public class GlobalExceptionHandler {

  @ExceptionHandler(ApplicationException.class)
  public ResponseEntity<String>
handleApplicationException(ApplicationException ex) {

    return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(ex.getMessage(
));
  }
}
```

Local Exception Handler vs. Global Exception Handler

A local exception handler is created with the *@ExceptionHandler* method inside a *@Controller* (or *@RestControllerAdvice*) class, it will handle the exceptions thrown from the handler methods within the controller class only.

`Quick Reference`

```
@ExceptionHandler(AppException.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
public String handleAppException(AppException ex) {

    return "errorPage";
}
```

A global exception handler is defined using the *@ControllerAdvice* (or *@RestControllerAdvice*) annotation on a class and *@ExceptionHandler* annotation on methods. The methods will be invoked when the specified exception is thrown from any of the *@Controller* (or *@RestController*) handler methods.

Quick Reference

```
@ControllerAdvice
public class GlobalExceptionHandler {

  @ExceptionHandler(NullPointerException.class)
  public ModelAndView handleNullPointerException(NullPointerException ex) {
    //...
  }

  @ExceptionHandler(RecordNotException.class)
  public ModelAndView handleException(RecordNotException ex) {
    //...
  }}
```

Spring @*ControllerAdvice*

If we want to centralize the exception-handling logic to one class that is capable of handling exceptions thrown from any handler class/controller class – then we can use `@ControllerAdvice` annotation.

By default, the methods in @ControllerAdvice apply globally to all controllers. We can create a class and add *@ControllerAdvice* annotation on top. Then add *@ExceptionHandler* methods for each type of specific exception class in it.

```
@ControllerAdvice
public class CustomExceptionHandler extends ResponseEntityExceptionHandler
{
  @ExceptionHandler(Exception.class)
  public final ResponseEntity<ErrorResponse> handleAllExceptions(Exception ex,
WebRequest request) {

    List<String> details = new ArrayList<>();
    details.add(ex.getLocalizedMessage());
    ErrorResponse error = new ErrorResponse(ApplicationConstants.SERVER_ERROR,
details);
    return new ResponseEntity<>(error, HttpStatus.INTERNAL_SERVER_ERROR);
  }
  @ExceptionHandler(RecordNotFoundException.class)

  public final ResponseEntity<ErrorResponse>

handleUserNotFoundException(RecordNotFoundException ex,

WebRequest request) {



    List<String> details = new ArrayList<>();

    details.add(ex.getLocalizedMessage());

    ErrorResponse error = new

ErrorResponse(ApplicationConstants.RECORD_NOT_FOUND, details);

    return new ResponseEntity<>(error, HttpStatus.NOT_FOUND);

  }}
```

ResponseEntityExceptionHandler

In the above example, we extended the exception handler class with ResponseEntityExceptionHandler. It is a convenient base class for @ControllerAdvice classes. It's designed specifically for handling exceptions in a RESTful API context where we need to return different types of responses based on the exception type.

ResponseEntityExceptionHandler provides exception handlers for internal Spring exceptions. If we don't extend it, then all the exceptions will be redirected to DefaultHandlerExceptionResolver which returns a ModelAndView object which is typically used in a traditional MVC setup for rendering views.

However, when building RESTful APIs, returning ModelAndView objects might not be suitable, as APIs typically require structured data formats like JSON or XML for error responses. That’s where ResponseEntityExceptionHandler comes in.

Implementing Redis Cache in Spring Boot Application

What is Redis Cache?

Redis Cache is nothing but a Cache Management feature offered by Redis. Redis is normally used as a cache to store repeatedly accessed data in memory so that the user can feel the better performance of the application. The Redis Cache offers various features like how long you want to keep data, and which data to remove first, and some other bright caching models.

What is the advantage of using Redis Cache in your application?

Like any other Caching Technique, Redis Cache also minimises the number of network calls made by your application, which in return improves performance of the application as a whole. One request from an application to the DB is similar to one network call.

How does the Redis Cache work in the Application?

When we perform a DB retrieve operation via an Application, the Redis Cache stores the result in its cache. Further, when we perform the same retrieve operation, it returns the result from the **cache itself and ignores the second call to the database**. Similarly, when we perform a DB update operation, the Redis Cache also updates the result in its cache. Needless to say, for delete operation also it deleted the data from the cache accordingly. In this way, there are no chances of getting incorrect data.

What is Redis Database?

Redis Database is an in-memory database that persists on disk. It means when we use Redis Database, we occupy a memory on the disk to use as a Database. The data model is key-value, but several kinds of values are supported such as Strings, Lists, Sets, Sorted Sets, Hashes, Streams, HyperLogLogs, Bitmaps etc.

What is Redis Server?

The full form of Redis is REmote DIctionary Server. When we use Redis in any form such as database, cache or Message Broker, we need to download a Redis Server in our system. People in the industry just call it Redis Server. Also we can use docker and pull the redis image rather than download.

Note : To pull using docker you have to install docker and run it on your local machine. To learn more about docker, visit Introduction to Docker.

What are the important annotations to enable Redis Cache in the Application?

Generally, there are four important annotations that we apply to implement the Redis Cache feature in our application. They are as below:

@EnableCaching

We apply this annotation at the main class (starter class) of our application in order to tell Spring Container that we need a Caching feature in our application.

@Cacheable

@Cacheable is used to fetch(retrieve) data from the DB to application and store in Redis Cache. We apply it on the methods that get (retrieve) data from DB.

@CachePut

We use @CachePut in order to update data in the Redis Cache while there is any update of data in DB. We apply it to the methods that make modifications in DB.

@CacheEvict

We use @CacheEvict in order to remove data in the Redis Cache while there is any removal of data in DB. We apply it on the methods that delete data from DB.

Conclusion

I hope that after going through all the theoretical concepts, you got better understanding about Redis Cache in a Spring Boot Application and how we can use annotation like the @ControllerAdvice and @ExceptionHandler to manage and handle exceptions in a spring boot application gracefully