Application frameworks

Spring Boot Configurations And Repositories

Overview

- Spring Boot Configurations
- REST Concepts
- Connecting MongoDB

Source : https://www.overviewdocs.com/

Spring Boot Configurations

- Property Files
- Adding a property file
- Changing server port
- Spring boot actuator

Store Application Configuration

- Application configurations
    - » DB Connection Strings
    - » External endpoints
    - » Messages

- Where can they be stored ?
    - » Utility Classes (Ex: DB connectors)
    - » External file
    - » External server

Property Files

- Most common file type for storing Java application configurations.
- **.properties** file is used.
- Stored in key value pairs.
- Loaded into instances of Properties class.
- Properties is a subclass of Hashtable.
- These can be used to write into and also to read from.
- Mostly used for reading only as those are being used to store configurations.

Property Files

- Where should the properties file be in maven projects ?

  `src -> main -> resources`

- Naming convention.
  - Any name can be used.
  - But has become a convention due to framework usage.
  - Default configuration name has become;
    - `application.properties`

Property Files with Spring Boot

- Picked automatically and no coding or additional configurations are needed.
- Pre-defined properties are listed under the .
    - Any name can be used.
    - But has become a convention due to framework usage.
    - Default configuration name has become;
        - `application.properties`

Spring Boot Properties

- Let’s try changing the server port.

```
server.port=8081
```

- All the pre-defined properties contain a default value.
- Should add new property in application.properties file to override any of those.
- Different property categories exist for different type of integrations.
    - Ex:

```
# spring-boot-data-mongo

spring.data.mongodb.database=my-app
```

Spring Boot Actuator

- Includes monitoring and managing support for the application.
- Supported features.

■**Endpoints**

/health for checking application health, /mappings.

■**Metrics**

Metrics for all the endpoints are automatically captured.

■**Audit**

A flexible audit framework.

■**Process Monitoring**

Spring Boot Actuator

- Maven dependency.

```
<dependency>
    <groupId>org.springframework.boot </groupId>
    <artifactId>spring-boot-starter-actuator </artifactId>
</dependency>
```

- Testing application health.

```
service-host/health
```

- Actuator runs in the port 8080 by default. But can be overridden by changing the property file.

```
management.port: 9001
```

REST Concepts

- HTTP Methods
- Defining resource paths
- Payload format
- Richardson Model
- HAL
- Validating Payloads

Design a REST API

- What are the resources and how the resource paths should be used ?
- What HTTP methods should be used ?
- How the payloads are defined and usage of Content Type ?
- What are the standards that should be followed ?

Defining Resource Paths

- Resource names in paths should be in plural.

  hostname/resourcename
  `comments-api.com/comments`

- Child resources should come after specific parent resources. (using path parameters)

  `assignments-api.com/courses/{courseId}/assignments`
  Ex: `assignments-api.com/courses/1234/assignments`

- Query string should be used only for querying purposes.

```
assignmentd-api.com/courses/1234/assignments
          ?filter=published::true&orderby=createdDate
```

Spring Boot Resource Paths

- Resource path should be defined in `@RequestMapping` annotation.

```
@RequestMapping(“/comments”)
public List<Comment> listComments() {...}
```

- Common paths for a resource can be defined for the whole controller.

```
@RestController
@RequestMapping("/comments")
public class CommentController {...}
```

Spring Boot Resource Paths

- `@PathVariable` is used to define path parameters.

```
@RequestMapping(value = "/{commentId}")
public Comment getComment(@PathVariable("commentId")
final String commentId)
```

- `@RequestParam` is used to define the query parameters.

```
public List<Comment> listComments(@RequestParam(value =
"commentType", required = true) final String
commentType)
```

HTTP Method usage in REST

- What is REST ?
- Why HTTP methods are needed for REST ?
- What are the HTTP methods available ?
- Which can be used for CRUD operations ?

Spring Boot Request Methods

- CRUD operations are the most commonly implemented REST methods in web services.
  - POST
  - PUT
  - GET
  - DELETE
  - HEAD

Spring Boot Request Methods

- Controllers methods can have different types of HTTP methods allowed.
- Default method being set is the GET method.
- `@RequestMapping` annotation can be used to override the default behavior.

```
@RequestMapping(method = RequestMethod.POST)
Public ReturnType createReturnType() {...}
```

Payload Formats

- Any type of payload format can be used in REST.
- Most commonly used are JSON and XML.
- JSON is preferred by most because of the support for javascript.
- Spring Boot supports JSON by default but can be configured to support XM easily.

Define Payloads in Spring Boot

- Spring Boot contains annotations to easily support handling payloads and defining content types for the endpoints.

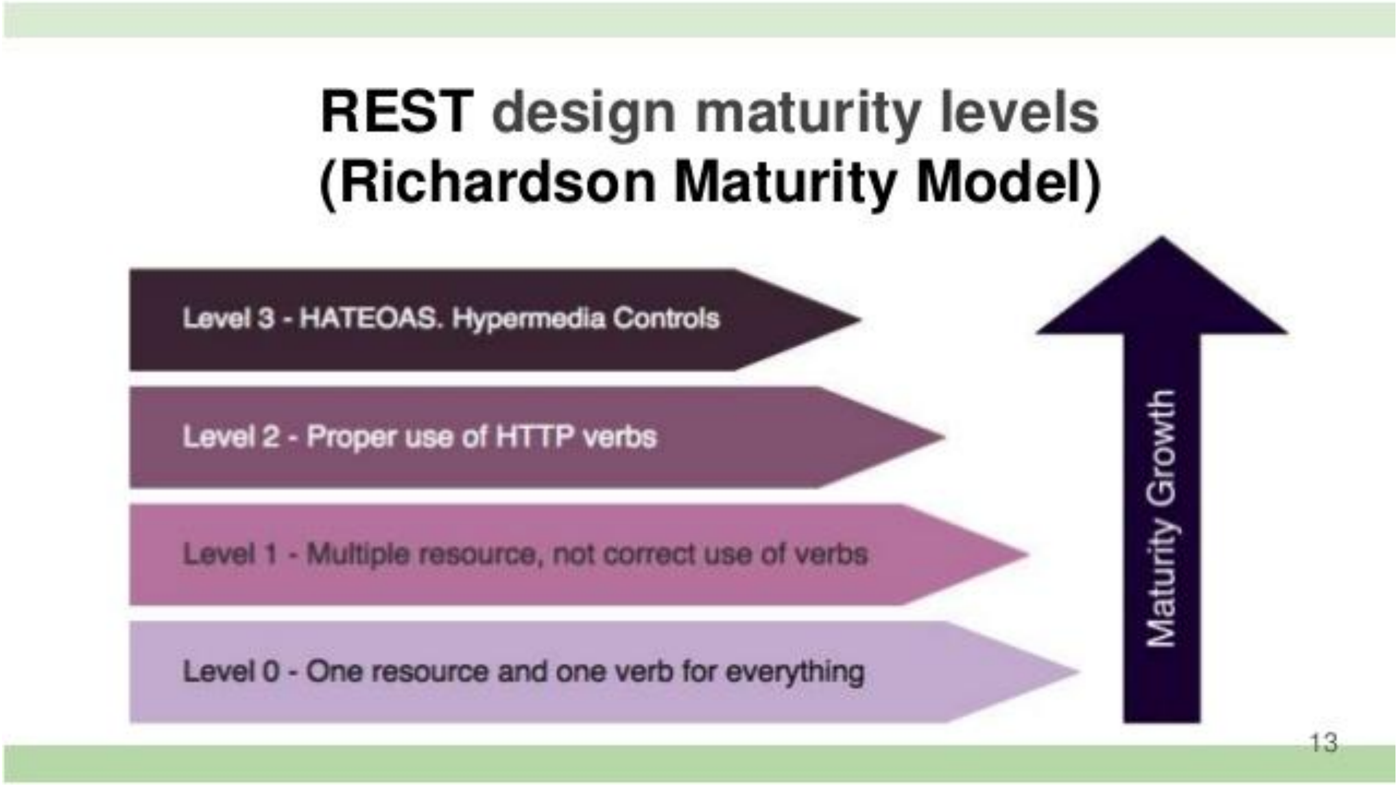
```
@RequestMapping(method = RequestMethod.POST, consumes =
"application/json")
public Comment createCommnet(@RequestBody final Comment
comment) {...}

@RequestMapping(value = "/{commentId}", produces =
MediaType.APPLICATION_JSON_VALUE)
public Comment getComment(@PathVariable("commentId")
final String commentId) {...}
```

Richardson Maturity Model

- REST services have no standards but a set of rules defined by community over time and not by any governing body.
- Richardson model can be used to identify the maturity of a REST service.
- These are concepts that you already know (not all) which is properly organized into levels.
- Most of the REST frameworks provides support to develop services up to the maturity model.
- Spring HATEOAS is one of them that supports the top end of the maturity model.

Richardson Maturity Model



REST design maturity levels
(Richardson Maturity Model)
Level 3 - HATEOAS. Hypermedia Controls
Level 2 - Proper use of HTTP verbs
Level 1 - Multiple resource, not correct use of verbs
Level 0 - One resource and one verb for everything
Maturity Growth
13

HAL - Hypertext Application Language

- A format to provide hyperlink between resources in APIs.
- Designed for building APIs in which client can move around resources with links being provided.
- An example as follows;

```
"_links": {
    "self": { "href": "/orders" },
    "next": { "href": "/orders?page=2" },
```

- Can be in both JSON and XML formats.

Spring HATEOAS

- Hypermedia As the Engine of Application State.
- Can be easily added using spring boot starter.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-hateoas</artifactId>
</dependency>
```

- Need to extend the models with `ResourceSupport` class.
- Adding implementation in controller is very easy.

```
Greeting greeting = new Greeting();
greeting.add(linkTo(methodOn(GreetingController.class).
greeting(name)).withSelfRel());
```

Spring HATEOAS

- Response sample.

```
{
  "content":"Hello, World!",
  "_links":{
    "self":{
      "href":"http://localhost:8080/greeting?name=World"
    }
  }
}
```

- Defined as **rel - relationship** and **href - complete URL**

Validating Entities

- Validating the payload is a common need in real world applications.
- Can use javax validations for basic validation needs.
- Hibernate validators are another alternative with additional features.
- Validation criteria should be defined in the models itself.
- Validation itself should be done at the controller level.

```
public Comment createComment(@Validated @RequestBody
final Comment comment)
```

Validating Entities

- Adding annotations for validation criteria.

```
import javax.validation.constraints.Min;
import org.hibernate.validator.constraints.NotEmpty;

@NotEmpty
private String message;
@Min(0)
private int age;
```

- All the error codes are handled by Spring boot by default.
- This can be overridden by overriding the Exception Handler Controller Advisors.

Mongo Connection

- Maven Dependencies
- Changes to model
- Repository concept
- MongoDB Configurations

How to add MongoDB dependencies

- Starter packages are defined for different types of storage types.

  Ex:
  –spring-boot-starter-data-mongodb
  –spring-boot-starter-data-jpa

- MongoDB starter;

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Models need to be changed ?

- Model classes need to be annotated using spring data annotations.

```
import
org.springframework.data.mongodb.core.mapping.Document;
import org.springframework.data.annotation.Id;

@Document(collection = "comment")
public class Comment implements Serializable {
    @Id
    private String commentId;
}
```

- Usage of serializable is optional.

Repository Concept

- Mongo repositories are used to create the repository layer for the mongoDB connections
- Implementing methods for saving, retrieving, deleting and listing takes a lot of effort.
- Adding new methods like findByAge, findByEmail and other filtering methods are also time consuming for the developers when it comes to advanced applications.
- So developers have to work more on technical difficulties rather than functional improvements.

Spring-Data Concepts

- Spring data frameworks provides or generates the implementation for your requirement.
- Generates implementation at runtime.
- Defining interface is the only thing to be done.
- In MongoDB, `MongoRepository` interface can be extended to create the repository interfaces.

Spring-Boot-Data-Mongo Usage

- Creating the repository interface.

```
public interface CommentRepository extends
MongoRepository<Comment, String> {...}
```

- Contains default methods needed for basic operations

```
    insert
    delete
    findOne
    findAll
    count
    exists
```

Spring-Boot-Data-Mongo Usage

- Custom method definition can be introduced by following the standard naming convention.

```
Iterable<Comment>
findBySectionIdAndCommentableIdAndDeleted(final
String sectionId, final String commentableId, final
boolean deleted);
```

```
Long countBySectionIdAndCommentableIdAndDeleted(final
String sectionId, final String commentableId, final
boolean deleted);
```

MongoDB Configurations

- Spring boot enables default configurations for any of the starters by default.
- The configurations can be overridden by adding properties to applicaiton.properties file.
- Spring boot has defined a set of properties for each of the integrations and can be found at the documentation of starter packages.

  Ex:

  ```
  #spring.data.mongodb.host=localhost
  #spring.data.mongodb.port=27017
  spring.data.mongodb.database=my-app
  ```

Misc.

- Java package usage
- Utilities and code reuse
- Implement Generic Code
- Usage of DTOs

Java Package Usage

- Readability is one of the key concerns in application development.
- Package structure should be easy understandable.
- Controllers, Services and Repositories should be included in their own packages.
- Interfaces and their implementation should be included in separate packages.

  Ex:

  –Service interfaces in service package.

  –Implementation in serviceImpl package.

Utilities and Code Reuse

- Common functionalities should be separated from the source and placed in utility (util) packages.
- These can include;
  - Technical functionality
  - Business functionality
- Features and models that are common for multiple services or multiple application components can be taken into separate modules packaged in jar files.

Implement Generic Code

- Always try to extract generic features that can be reused.
- Use Java Generics to implement those without binding them to specific types.
- Use Java reflections for more generic implementations that can work like frameworks.

Usage of DTOs (Data Transfer Objects)

- Used to carry data between processes.
- A pattern being used to define combined models or structures.
- Used when calling multiple remotes are expensive operations.
- Mostly used in aggregation layer services in the concept of Microservices.
- Commonly used in normal web services.
- Some use to show minimal version of a single entity also.

That's all Folks!
Any Questions?