Java Enterprise Edition

ANUDIP FOUNDATION

Spring Data REST

Objective: • Reader Hierarchy • Writer Hierarchy	**Materials Required:** 1. Eclipse IDE/IntelliJ/STS 2. Notepad 3. Gradle/Maven 4. Jdk1.8 or later
Theory:150mins	**Practical:30mins**
Total Duration: 180 mins	

1. **Spring Data REST**
   - Introduction & Overview
   - Adding Spring Data REST to a Spring Boot Project
   - Configuring Spring Data REST
   - Repository resources, Default Status Codes, Http methods
   - Spring Data REST Associations
   - Define Query methods

Spring Data REST

Spring Data REST is itself a Spring MVC application and is designed in such a way that it should integrate with your existing Spring MVC applications with little effort. An existing (or future) layer of services can run alongside Spring Data REST with only minor additional work.

1.1. Adding Spring Data REST to a Spring Boot Project

The simplest way to get to started is to build a Spring Boot application because Spring Boot has a starter for Spring Data REST and uses auto-configuration. The following example shows how to use Gradle to include Spring Data Rest in a Spring Boot project:

Example 3. Spring Boot configuration with Gradle

```
dependencies {
  ...
  compile("org.springframework.boot:spring-boot-starter-data-rest")
  ...
}
```

The following example shows how to use Maven to include Spring Data Rest in a Spring Boot project:

Example 4. Spring Boot configuration with Maven

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

> You need not supply the version number if you use the Spring Boot Gradle plugin or the Spring Boot Maven plugin.

When you use Spring Boot, Spring Data REST gets configured automatically.

1.2. Adding Spring Data REST to a Gradle project

To add Spring Data REST to a Gradle-based project, add the spring-data-rest-webmvc artifact to your compile-time dependencies, as follows:

```
dependencies {
  … other project dependencies
  compile("org.springframework.data:spring-data-rest-webmvc:3.5.0")
}
```

1.3. Adding Spring Data REST to a Maven project

To add Spring Data REST to a Maven-based project, add the spring-data-rest-webmvc artifact to your compile-time dependencies, as follows:

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-rest-webmvc</artifactId>
  <version>3.5.0</version>
</dependency>
```

1.4. Configuring Spring Data REST

To install Spring Data REST alongside your existing Spring MVC application, you need to include the appropriate MVC configuration. Spring Data REST configuration is defined in a class called RepositoryRestMvcConfiguration and you can import that class into your application's configuration.

> This step is unnecessary if you use Spring Boot's auto-configuration. Spring Boot automatically enables Spring Data REST when you include **spring-boot-starter-data-rest** and, in your list of dependencies, your app is flagged with either @SpringBootApplication or @EnableAutoConfiguration.

To customize the configuration, register a RepositoryRestConfigurer and implement or override the configure…-methods relevant to your use case.
Make sure you also configure Spring Data repositories for the store you use. For details on that, see the reference documentation for the corresponding Spring Data module.

1.5. Basic Settings for Spring Data REST

This section covers the basic settings that you can manipulate when you configure a Spring Data REST application, including:

- Setting the Repository Detection Strategy
- Changing the Base URI
- Changing Other Spring Data REST Properties

1.5.1. Setting the Repository Detection Strategy

Spring Data REST uses a RepositoryDetectionStrategy to determine whether a repository is exported as a REST resource. The RepositoryDiscoveryStrategies enumeration includes the following values:

Table 1. Repository detection strategies

Name	Description
DEFAULT	Exposes all public repository interfaces but considers the exported flag of @(Repository)RestResource.
ALL	Exposes all repositories independently of type visibility and annotations.
ANNOTATED	Only repositories annotated with @(Repository)RestResource are exposed, unless their exported flag is set to false.
VISIBILITY	Only public repositories annotated are exposed.

1.5.2. Changing the Base URI

By default, Spring Data REST serves up REST resources at the root URI, '/'. There are multiple ways to change the base path.

With Spring Boot 1.2 and later versions, you can do change the base URI by setting a single property in application.properties, as follows:

```
spring.data.rest.basePath=/api
```

With Spring Boot 1.1 or earlier, or if you are not using Spring Boot, you can do the following:

```
@Configuration
class CustomRestMvcConfiguration {

  @Bean
  public RepositoryRestConfigurer repositoryRestConfigurer() {

    return new RepositoryRestConfigurer() {

      @Override
      public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config,
CorsRegistry cors) {
        config.setBasePath("/api");
      }
    };
  }
}
```

Alternatively, you can register a custom implementation of RepositoryRestConfigurer as a Spring bean and make sure it gets picked up by component scanning, as follows:

```
@Component
public class CustomizedRestMvcConfiguration extends RepositoryRestConfigurer {

  @Override
  public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config,
CorsRegistry cors) {
```

```
    config.setBasePath("/api");
  }
}
```

Both of the preceding approaches change the base path to /api.

1.5.3. Changing Other Spring Data REST Properties

You can alter the following properties:

Table 2. Spring Boot configurable properties

Property	Description
basePath	the root URI for Spring Data REST
defaultPageSize	change the default for the number of items served in a single page
maxPageSize	change the maximum number of items in a single page
pageParamName	change the name of the query parameter for selecting pages
limitParamName	change the name of the query parameter for the number of items to show in a page
sortParamName	change the name of the query parameter for sorting
defaultMediaType	change the default media type to use when none is specified
returnBodyOnCreate	change whether a body should be returned when creating a new entity
returnBodyOnUpdate	change whether a body should be returned when updating an entity

1.6. Starting the Application

At this point, you must also configure your key data store.

Spring Data REST officially supports:

- Spring Data JPA

- Spring Data MongoDB
- Spring Data Neo4j
- Spring Data GemFire
- Spring Data Cassandra

The following Getting Started guides can help you get up and running quickly:

- Spring Data JPA
- Spring Data MongoDB
- Spring Data Neo4j
- Spring Data GemFire

These linked guides introduce how to add dependencies for the related data store, configure domain objects, and define repositories.

You can run your application as either a Spring Boot app (with the links shown earlier) or configure it as a classic Spring MVC app.

> In general, Spring Data REST does not add functionality to a given data store. This means that, by definition, it should work with any Spring Data project that supports the repository programming model. The data stores listed above are the ones for which we have written integration tests to verify that Spring Data REST works with them.

From this point, you can customize Spring Data REST with various options.

2. Repository resources

2.1. Fundamentals

The core functionality of Spring Data REST is to export resources for Spring Data repositories. Thus, the core artifact to look at and potentially customize the way the exporting works is the repository interface. Consider the following repository interface:

```
public interface OrderRepository extends CrudRepository<Order, Long> { }
```

For this repository, Spring Data REST exposes a collection resource at /orders. The path is derived from the uncapitalized, pluralized, simple class name of the domain class being managed. It also exposes an item resource for each of the items managed by the repository under the URI template /orders/{id}. By default the HTTP methods to interact with these resources map to the according methods of CrudRepository. Read more on that in the sections on collection resources and item resources.

2.1.1. Repository methods exposure

Which HTTP resources are exposed for a certain repository is mostly driven by the structure of the repository. In other words, the resource exposure will follow which methods you have exposed on the repository. If you extend CrudRepository you usually expose all methods required to expose all HTTP resources we can register by default. Each of the resources listed below will define which of the

methods need to be present so that a particular HTTP method can be exposed for each of the resources. That means, that repositories that are not exposing those methods — either by not declaring them at all or explicitly using @RestResource(exported = false) — won't expose those HTTP methods on those resources.
For details on how to tweak the default method exposure or dedicated HTTP methods individually see

2.1.2. Default Status Codes

For the resources exposed, we use a set of default status codes:

- 200 OK: For plain GET requests.
- 201 Created: For POST and PUT requests that create new resources.
- 204 No Content: For PUT, PATCH, and DELETE requests when the configuration is set to not return response bodies for resource updates (RepositoryRestConfiguration.returnBodyOnUpdate). If the configuration value is set to include responses for PUT, 200 OK is returned for updates, and 201 Created is returned for resource created through PUT.

If the configuration values (RepositoryRestConfiguration.returnBodyOnUpdate and RepositoryRestConfiguration.returnBodyCreate) are explicitly set to null, the presence of the HTTP Accept header is used to determine the response code.

2.1.3. Resource Discoverability

A core principle of HATEOAS is that resources should be discoverable through the publication of links that point to the available resources. There are a few competing de-facto standards of how to represent links in JSON. By default, Spring Data REST uses HAL to render responses. HAL defines the links to be contained in a property of the returned document.

Resource discovery starts at the top level of the application. By issuing a request to the root URL under which the Spring Data REST application is deployed, the client can extract, from the returned JSON object, a set of links that represent the next level of resources that are available to the client.

For example, to discover what resources are available at the root of the application, issue an HTTP GET to the root URL, as follows:

```
curl -v http://localhost:8080/

< HTTP/1.1 200 OK
< Content-Type: application/hal+json

{ "_links" : {
   "orders" : {
     "href" : "http://localhost:8080/orders"
   },
   "profile" : {
     "href" : "http://localhost:8080/api/alps"
   }
  }
}
```

The property of the result document is an object that consists of keys representing the relation type, with nested link objects as specified in HAL.

For more details about the profile link, see Application-Level Profile Semantics (ALPS).

2.2. The Collection Resource

Spring Data REST exposes a collection resource named after the uncapitalized, pluralized version of the domain class the exported repository is handling. Both the name of the resource and the path can be customized by using @RepositoryRestResource on the repository interface.

2.2.1. Supported HTTP Methods

Collections resources support both GET and POST. All other HTTP methods cause a 405 Method Not Allowed.

GET

Returns all entities the repository servers through its findAll(…) method. If the repository is a paging repository we include the pagination links if necessary and additional page metadata.

Methods used for invocation

The following methods are used if present (decending order):

- findAll(Pageable)
- findAll(Sort)
- findAll()

For more information on the default exposure of methods, see Repository methods exposure.

Parameters

If the repository has pagination capabilities, the resource takes the following parameters:

- page: The page number to access (0 indexed, defaults to 0).
- size: The page size requested (defaults to 20).
- sort: A collection of sort directives in the format ($propertyname,)+[asc|desc]?.

Custom Status Codes

The GET method has only one custom status code:

- 405 Method Not Allowed: If the findAll(…) methods were not exported (through @RestResource(exported = false)) or are not present in the repository.

Supported Media Types

The GET method supports the following media types:

- application/hal+json
- application/json

Related Resources

The GET method supports a single link for discovering related resources:

- search: A search resource is exposed if the backing repository exposes query methods.

HEAD

The HEAD method returns whether the collection resource is available. It has no status codes, media types, or related resources.

Methods used for invocation

The following methods are used if present (decending order):

- findAll(Pageable)
- findAll(Sort)
- findAll()

For more information on the default exposure of methods, see Repository methods exposure.

POST

The POST method creates a new entity from the given request body.

Methods used for invocation

The following methods are used if present (decending order):

- save(…)

For more information on the default exposure of methods, see Repository methods exposure.

Custom Status Codes

The POST method has only one custom status code:

- 405 Method Not Allowed: If the save(…) methods were not exported (through @RestResource(exported = false)) or are not present in the repository at all.

Supported Media Types

The POST method supports the following media types:

- application/hal+json
- application/json

2.3. The Item Resource

Spring Data REST exposes a resource for individual collection items as sub-resources of the collection resource.

2.3.1. Supported HTTP Methods

Item resources generally support GET, PUT, PATCH, and DELETE, unless explicit configuration prevents that (see "The Association Resource" for details).

GET

The GET method returns a single entity.

Methods used for invocation

The following methods are used if present (decending order):

- findById(…)

For more information on the default exposure of methods, see Repository methods exposure.

Custom Status Codes

The GET method has only one custom status code:

- 405 Method Not Allowed: If the findOne(…) methods were not exported (through @RestResource(exported = false)) or are not present in the repository.

Supported Media Types

The GET method supports the following media types:

- application/hal+json
- application/json

Related Resources

For every association of the domain type, we expose links named after the association property. You can customize this behavior by using @RestResource on the property. The related resources are of the association resource type.

HEAD

The HEAD method returns whether the item resource is available. It has no status codes, media types, or related resources.

Methods used for invocation

The following methods are used if present (decending order):

- findById(…)

For more information on the default exposure of methods, see Repository methods exposure.

PUT

The PUT method replaces the state of the target resource with the supplied request body.

Methods used for invocation

The following methods are used if present (decending order):

- save(…)

For more information on the default exposure of methods, see Repository methods exposure.

Custom Status Codes

The PUT method has only one custom status code:

- 405 Method Not Allowed: If the save(…) methods were not exported (through @RestResource(exported = false)) or is not present in the repository at all.

Supported Media Types

The PUT method supports the following media types:

- application/hal+json
- application/json

PATCH

The PATCH method is similar to the PUT method but partially updates the resources state.

Methods used for invocation

The following methods are used if present (decending order):

- save(…)

For more information on the default exposure of methods, see Repository methods exposure.

Custom Status Codes

The PATCH method has only one custom status code:

- 405 Method Not Allowed: If the save(…) methods were not exported (through @RestResource(exported = false)) or are not present in the repository.

Supported Media Types

The PATCH method supports the following media types:

- application/hal+json
- application/json
- application/patch+json
- application/merge-patch+json

DELETE

The DELETE method deletes the resource exposed.

Methods used for invocation

The following methods are used if present (decending order):

- delete(T)
- delete(ID)
- delete(Iterable)

For more information on the default exposure of methods, see Repository methods exposure.

Custom Status Codes

The DELETE method has only one custom status code:

- 405 Method Not Allowed: If the delete(…) methods were not exported (through @RestResource(exported = false)) or are not present in the repository.

2.4. The Association Resource

Spring Data REST exposes sub-resources of every item resource for each of the associations the item resource has. The name and path of the resource defaults to the name of the association property and can be customized by using @RestResource on the association property.

2.4.1. Supported HTTP Methods

The association resource supports the following media types:

- GET
- PUT
- POST

- DELETE

GET
The GET method returns the state of the association resource.

Supported Media Types

The GET method supports the following media types:
- application/hal+json
- application/json

PUT
The PUT method binds the resource pointed to by the given URI(s) to the resource. This

Custom Status Codes

The PUT method has only one custom status code:
- 400 Bad Request: When multiple URIs were given for a to-one-association.

Supported Media Types

The PUT method supports only one media type:
- text/uri-list: URIs pointing to the resource to bind to the association.

POST
The POST method is supported only for collection associations. It adds a new element to the collection.

Supported Media Types

The POST method supports only one media type:
- text/uri-list: URIs pointing to the resource to add to the association.

DELETE
The DELETE method unbinds the association.

Custom Status Codes

The POST method has only one custom status code:
- 405 Method Not Allowed: When the association is non-optional.

2.5. The Search Resource

The search resource returns links for all query methods exposed by a repository. The path and name of the query method resources can be modified using @RestResource on the method declaration.

2.5.1. Supported HTTP Methods

As the search resource is a read-only resource, it supports only the GET method.
GET
The GET method returns a list of links pointing to the individual query method resources.

Supported Media Types

The GET method supports the following media types:
- application/hal+json
- application/json

Related Resources

For every query method declared in the repository, we expose a query method resource. If the resource supports pagination, the URI pointing to it is a URI template containing the pagination parameters.

HEAD
The HEAD method returns whether the search resource is available. A 404 return code indicates no query method resources are available.

2.6. The Query Method Resource

The query method resource runs the exposed query through an individual query method on the repository interface.

2.6.1. Supported HTTP Methods

As the search resource is a read-only resource, it supports GET only.
GET
The GET method returns the result of the query execution.

Parameters

If the query method has pagination capabilities (indicated in the URI template pointing to the resource) the resource takes the following parameters:

- page: The page number to access (0 indexed, defaults to 0).
- size: The page size requested (defaults to 20).
- sort: A collection of sort directives in the format ($propertyname,)+[asc|desc]?.

Supported Media Types

The GET method supports the following media types:
- application/hal+json
- application/json

HEAD
The HEAD method returns whether a query method resource is available.