Summary

1. Spring IOC

IOC: Inversion of Control

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

Advantages:

1. Decoupling the execution of a task from its implementation
2. Greater modularity
3. Easier to switch implementations

Achieved by:

1. Strategy/Service Locator/Factory design pattern and DI

DI: Dependency Injection

Dependency injection is a pattern we can use to implement IoC, where the control being inverted is setting an object's dependencies. dependency injection is a design pattern in which an object or function receives other objects or functions that it depends on.

Dependency Injection Approaches

1. constructor based

The container will invoke a constructor with arguments each representing a dependency we want to set. Spring resolves each argument primarily by type, followed by name of the attribute, and index for disambiguation.

@Autowired on a constructor.

1. setter based

The container will call setter methods of our class after invoking a no-argument constructor or no-argument static factory method to instantiate the bean.

@Autowired on a setter.

1. field based

The container will inject the dependency by assigning the autowired object to instance variable

@Autowired on an instance variable.

Bean Scope

Spring framework provides 6 bean scopes, they are:

1. singleton

The container creates a single instance of a bean and all request for that bean name will return the same one. It’s cached and all modification will be reflected in all references to that bean.

1. Prototype

The container will return a different instance when it’s requested.

for web-aware scope:

1. Request : @RequestScope : instance beans when a request comes
2. session
3. application
4. websocket

Bean Lifecycle

the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed.

Bean lifecycle is managed by the spring container

Container started->Bean init -> DI -> Custom init() -> utility -> Custom destroy()

*Source:*

*https://www.baeldung.com/spring-bean-scopes*

*https://www.geeksforgeeks.org/bean-life-cycle-in-java-spring/*

2. Spring AOP

Spring AOP:

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns. It does this by adding additional behavior to existing code without modifying the code itself.

Aspect:

An aspect is a modularization of a concern that cuts across multiple classes. Unified logging can be an example of such cross-cutting concern.

Advice

An *Advice* is an action taken by an aspect at a particular *Joinpoint*. Different types of advice include *“around,” “before,”* and *“after.”*

In Spring, an *Advice* is modelled as an interceptor, maintaining a chain of interceptors around the *Joinpoint*.

JoinPoint

A Joinpoint is a point during the execution of a program, such as the execution of a method or the handling of an exception.

PointCut

A Pointcut is a predicate that helps match an Advice to be applied by an Aspect at a particular JoinPoint.

We often associate the Advice with a Pointcut expression, and it runs at any Joinpoint matched by the Pointcut.

Diagram

Description automatically generated

*Sources:*

[*https://howtodoinjava.com/spring-aop-tutorial/*](https://howtodoinjava.com/spring-aop-tutorial/)

[*https://www.baeldung.com/spring-aop*](https://www.baeldung.com/spring-aop)

@Transactional

**Propagation:**

The **propagation** attribute indicates if any component or service will participate in the transaction and how it will behave if the calling component or service already has or does not have an existing transaction. The default value is **REQUIRED** which means it will use the existing transaction created by the caller or it will create a new transaction if it doesn’t exist.

Other available settings are:

**SUPPORTS**: uses the existing transaction of the caller, never creates a new transaction.

**NOT\_SUPPORTED**: runs without transaction and never creates a new transaction. Caller’s existing transaction will be suspended.

**REQUIRES\_NEW**: always creates a new transaction. Caller’s existing transaction will be suspended.

**NEVER**: never creates a new transaction and **throws an exception (IllegalTransactionStateException)** if the caller has an existing transaction.

**MANDATORY**: always use the caller’s existing transaction. **Throws an exception** if the caller does not have an existing transaction.

**rollBackFor:**

The default **rollback** behavior without explicit declaration will rollback on throwed **runtime unchecked** exceptions. (The checked exceptions does not trigger a rollback)

**ReadOnly**: A boolean flag that can be set to true if the transaction is effectively read-only. Defaults to **false**

**Note:**

When there’re write attempts in the transaction, it won’t cause any failure. It’s just used to give a hint to system to optimize the transaction.

**Isolation:** indicates the transaction level of the transactional method.

5 options provided:

Default: isolation level in transaction method is as same as that in the database

The rest 4 isolation levels are:

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation Level** | **Dirty Read** | **Nonrepeatable Read** | **Phantom Read** |
| READ UNCOMMITTED | Permitted | Permitted | Permitted |
| READ COMMITTED | -- | Permitted | Permitted |
| REPEATABLE READ | -- | -- | Permitted |
| SERIALIZABLE | -- | -- | -- |

3, Spring MVC

The Spring Web model-view-controller (MVC) framework is designed around a DispatcherServlet that dispatches requests to handlers, with configurable handler mappings, view resolution, locale, and theme resolution as well as support for uploading files.

MVC: model view controller

1. The **Model** encapsulates the application data and in general they will consist of POJO.
2. The **View** is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
3. The **Controller** is responsible for processing user requests and building an appropriate model and passes it to the view for rendering.

Spring MVC workflow

Request -> DispatcherServlet ----dispatch---> controllers – build model and return view -> view resolver checked by DispatcherServlet -> response as a view

Diagram

Description automatically generated

*Figure Source: https://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview.htht*

*Sources:*

*https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html*

4. Spring Boot

Spring Boot is an open source Java-based framework used to create a micro Service.

Spring boot advantages

1. Easy to configure java beans and avoid complex XML configuration in Spring
2. Easy dependency management
3. Offering annotation based spring application
4. Included embedded servlet container and easy to deploy

Spring boot Starter

Help dependency management. Starter POMs are a set of convenient dependency descriptors.

We can find any available starters in *spring-boot-starter-parent*

Auto Configuration

Spring Boot auto-configuration attempts to automatically configure your Spring application based on the jar dependencies that you have added.

We can also customize the configuration in properties files

Rest API Design

http method: CRUD operations

* create: post
* read: get
* update: post
* delete: delete

idempotent: get, put, delete

safe: get

cacheable: get

cache control

http status code:

* 1xx information
* 2xx success
* 3xx redirect
* 4xx client side error
* 5xx server side error

200 ok

201 created

202 accepted

204 no content

400 bad request

401 unauthorized

403 forbidden

404 not found

405 method not allowed

500 internal server error

Spring Restful API

get /api/user/{uid}

get /api/user?pageNo=2&rows=10&orderBy=salary

post /api/user

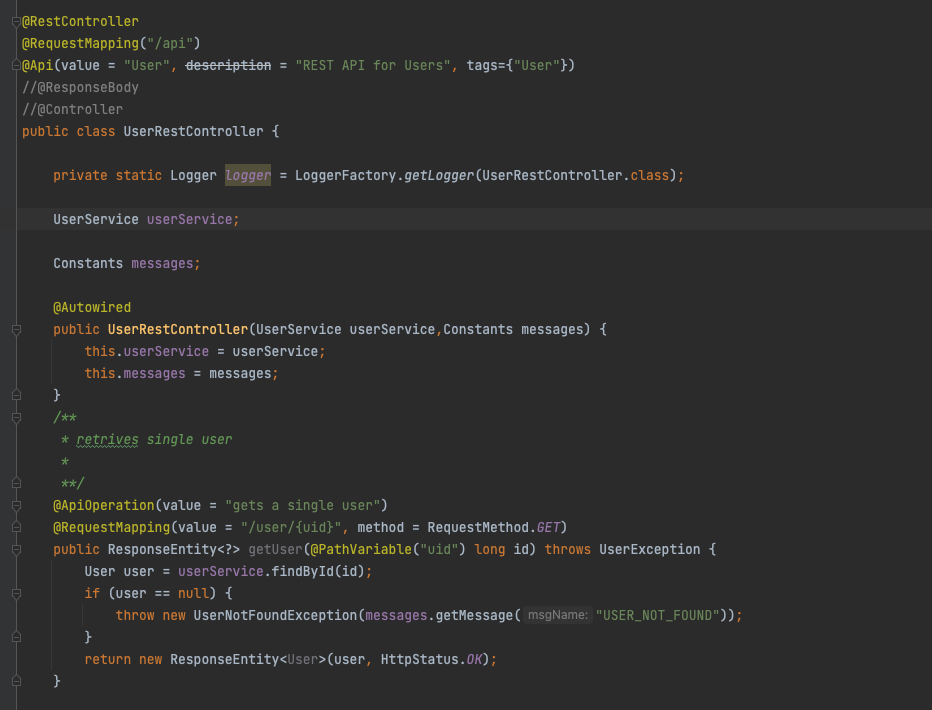
put /api//user/{id}

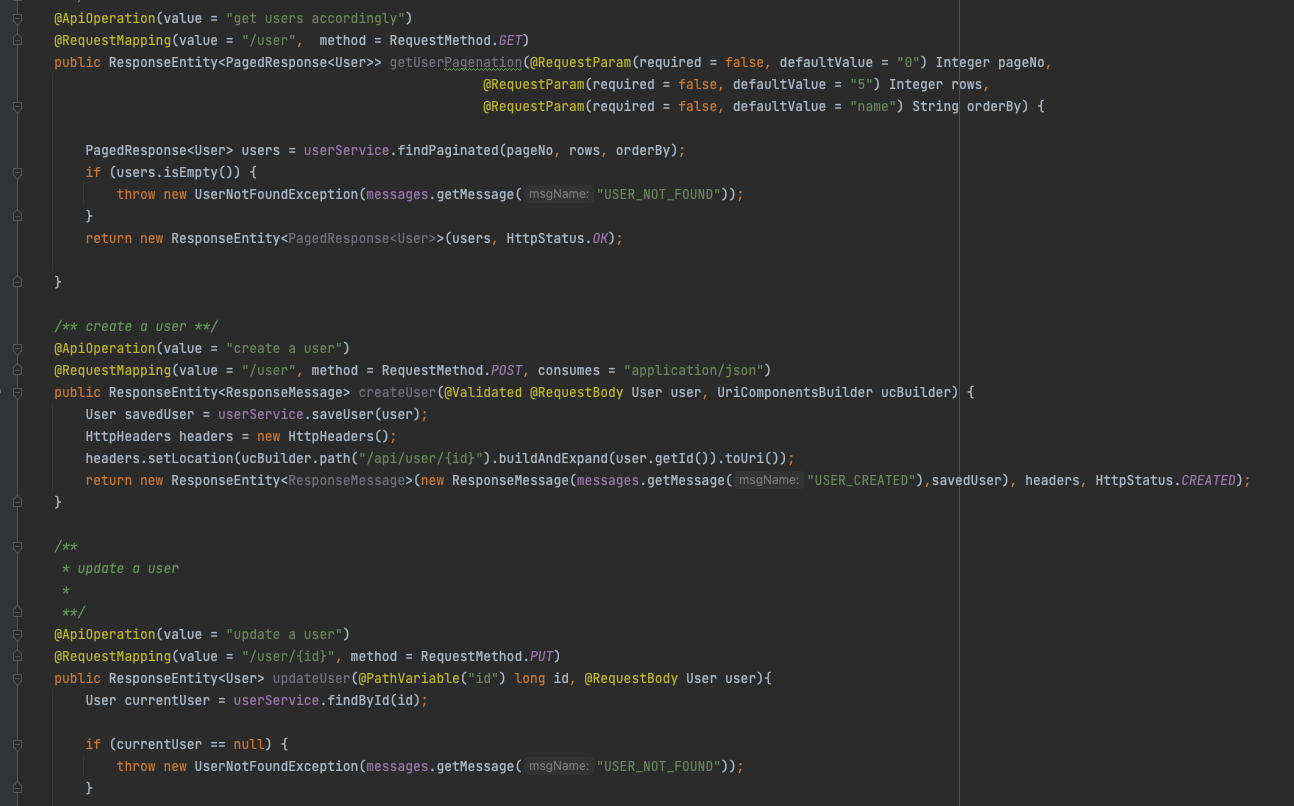
delete /api//user/{id}

annotation:

* RequestMapping, GetMapping, PutMapping, PostMapping, DeleteMapping
* RequestParam, PathVariable
* RequestBody (json -> java object), ResponseBody (java object -> json)
* Controller/RestController, Service, Repository
  + Controller + ResponseBody = RestController

<https://github.com/Arthur-Shuahua-Zhang/RestDemo>





5. Exception Handling Process

Diagram

Description automatically generated

When an exception is thrown from a controller, first check if exists a corresponded @ExceptionHandler in customization and @ControllerAdvice, if yes handle the exception and return a response.

If no, check if the exception is annotated with @responseStatus, if yes handle it by ResponseStatusExceptionResolver class. If no, handle it by DefaultHandlerExceptionResovler

6. Validation

All related dependencies are in *spring-boot-starter-validation.*

We can add any constraints easily when defining entities in spring boot

The code is like:

@NotNull, @Max, @Min, @Pattern(regex), @Email, @NotBlank and so on.

7. Swagger

All related dependencies are in *<io.springfox, springfox-swagger2>*

Swagger2 is an open-source project used to generate the REST API documents for RESTful web services. It provides a user interface to access our RESTful web services via the web browser.