Spring Annotations

Sistemas Distribuídos 2022/23

Sources

- https://www.javatpoint.com/spring-boot-annotations
- https://stackoverflow.com/questions/15745140/are-spring-objects-thread-safe
- https://stackoverflow.com/questions/38487099/is-spring-transactional-thread-safe
- http://mvpjava.com/component-vs-service-spring/
- https://www.logicbig.com/tutorials/spring-framework.html
- https://stackoverflow.com/questions/15965735/is-a-spring-data-jpa-repository-thread-safe-aka-is-simplejp
- https://www.tutorialspoint.com/difference-between-bean-and-component-annotation-in-spring

Core Spring Framework Annotations

Core Spring Framework Annotations

- @Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

```
@Configuration
class AppConfiguration {

   @Bean
   public BeanExample beanExample()
   {
       return new BeanExample ();
   }
}
```

Core Spring Framework Annotations

- @Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions

```
@Configuration
public class Vehicle
{
    @Bean
    Vehicle engine() {
        return new Vehicle();
    }
}
```

Core Spring Framework Annotations

- @Required: It applies to the bean setter method. It indicates that the annotated bean must be populated at configuration time with the required property, else it throws an exception BeanInitilizationException

```
public class Machine
{
    private Integer cost;

    @Required
    public void setCost(Integer cost) {
        this.cost = cost;
    }

    public Integer getCost() {
        return cost;
    }
}
```

Core Spring Framework Annotations

- @Autowired: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

```
@Component
public class Customer
{
    private Person person;

    @Autowired
    public Customer(Person person) {
        this.person=person;
    }
}
```

Core Spring Framework Annotations

- @ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

```
@ComponentScan(basePackages = "com.javatpoint")
@Configuration
public class ScanComponent
{
// ...
}
```

Spring Framework Stereotype Annotations

Spring Framework Stereotype Annotations

- @Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath scan. The Spring Framework pick it up and configure it in the application context as a Spring Bean.

```
@Component
public class Student
{
.......
}
```

Bean vs Component

Sr. No.	Key	@Bean	@Component
1	Auto detection	It is used to explicitly declare a single bean, rather than letting Spring do it automatically.	If any class is annotated with @Component it will be automatically detect by using classpath scan.
2	Spring Container	Bean can be created even class is outside the spring container	We can't create bean if class is outside spring container
3	Class/Method Level Annotation	It is a method level annotation	It is a class level annotation
4	@Configuration	It works only when class is also annotated with @Configuration	It works without @Configuration annotation
5	Use Case	We should use @bean, if you want specific implementation based on dynamic condition.	We can't write specific implementation based on dynamic condition

Spring Framework Stereotype Annotations

- @Repository: It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.

```
@Repository
public class TestRepository
{
    public void delete() {
        //persistence code
    }
}
```

Spring Framework Stereotype Annotations

- @Service: It is also used at class level. It tells the Spring that class contains the business logic.

```
package com.javatpoint;

@Service
public class TestService
{
   public void service1() {
      //business code
   }
}
```

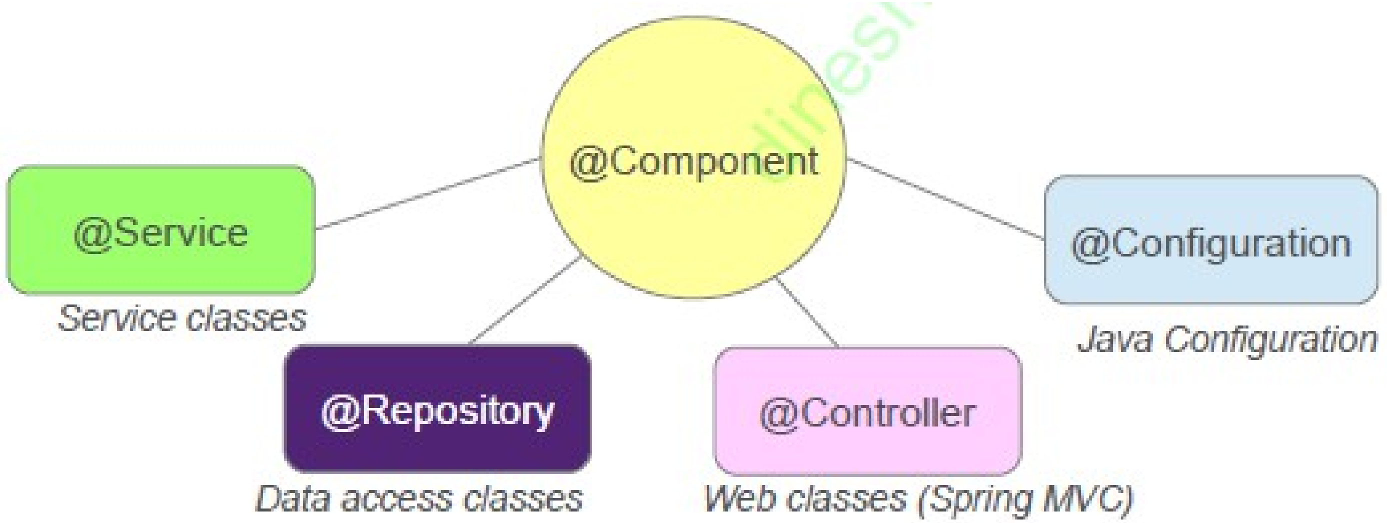
Spring Framework Stereotype Annotations

- @Controller: The @Controller is a class-level annotation. It is a specialization of @Component. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with @RequestMapping annotation.

```
@Controller
@RequestMapping("books")
public class BooksController
{
   @RequestMapping(value = "/{name}", method = RequestMethod.GET)
   public Employee getBooksByName() {
      return booksTemplate;
   }
}
```

Why use @Service in Spring?

- Intent!
- Your @Service layer classes should have your DAO @Repository Spring beans dependency injected into them thereby achieving a clean separation of concern.
- These service layer methods should not have any explicit data access code (like an SQL query) and should only throw business level exceptions back to the caller – not an SQLException for example.



Spring Boot Annotations

Spring Boot Annotations

- @EnableAutoConfiguration
  - Auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e., @SpringBootApplication.
- @SpringBootApplication
  - It is a combination of three annotations @EnableAutoConfiguration, @ComponentScan, and @Configuration.

Spring MVC and REST Annotations

Spring MVC and REST Annotations

- @RequestMapping: It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.

```
@Controller
public class BooksController
{
   @RequestMapping("/computer-science/books")
   public String getAllBooks(Model model) {
      //application code
      return "bookList";
   }
}
```

Spring MVC and REST Annotations

- **@GetMapping:** It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using @RequestMapping(method = RequestMethod.GET)
- **@PostMapping**
- **@PutMapping**
- **@DeleteMapping**
- **@PatchMapping**
- **@RequestBody:** It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with @RequestBody, the Spring framework binds the incoming HTTP request body to that parameter.
- **@ResponseBody:** It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

@RequestBody

```
@RestController
public class BookController {
    @Autowired
    private BookService bookService;

    @PostMapping(value = "book", consumes = { MediaType.APPLICATION_JSON_VALUE })
    public ResponseEntity<Book> addBook(@RequestBody Book book, UriComponentsBuilder builder) {
        bookService.addBook(book);
        HttpHeaders headers = new HttpHeaders();
        headers.setLocation(builder.path("/book/
{id}").buildAndExpand(book.getBookId()).toUri());
        return new ResponseEntity<Book>(book, headers, HttpStatus.CREATED);
    }

    @PostMapping(value = "send", consumes = { MediaType.APPLICATION_OCTET_STREAM_VALUE })
    public String userData(@RequestBody byte[] data) {
        bookService.userData(data);
        return "Data sent.";
    }
}
```

@ResponseBody

```
@Controller
public class BookController {
    @Autowired
    private BookService bookService;

    @GetMapping(value = "bookABC", produces = { MediaType.APPLICATION_JSON_VALUE })
    public @ResponseBody List<Book> getBooksABC() {
        List<Book> list = bookService.getAllBooks();
        return list;
    }

    @GetMapping(value = "bookXYZ", produces = { MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    @ResponseStatus(HttpStatus.OK)
    public List<Book> getBooksXYZ() {
        List<Book> list = bookService.getAllBooks();
        return list;
    }
}
```

Spring MVC and REST Annotations

- **@PathVariable**: It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple @PathVariable in a method.
- **@RequestParam**: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.
- **@RequestHeader**: It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method
- **@RestController**: It can be considered as a combination of @Controller and @ResponseBody annotations. The @RestController annotation is itself annotated with the @ResponseBody annotation. It eliminates the need for annotating each method with @ResponseBody.

Are Beans Thread Safe?

- No.
  - But the scope of the bean may ensure that sharing never occurs
- Spring has different bean scopes (e.g., Prototype, Singleton, etc.) but all these scopes enforce is *when* the bean is created. For example, a "prototype" scoped bean will be created each time this bean is "injected", whereas a "singleton" scoped bean will be created once and shared within the application context. There are other scopes, but they just define a time span (e.g., a "scope") of *when* a new instance will be created.
- The above has little, if anything to do with being thread safe, since if several threads have access to a bean (no matter the scope), it will only depend on the *design* of that bean to be or not to be "thread safe".
- The reason I said "little, if anything" is because it might depend on the problem you are trying to solve. For example, if you are concerned whether 2 or more HTTP requests may create a problem for the same bean, there is a "request" scope that will create a new instance of a bean for each HTTP request, hence you can "think" of a particular bean as being "safe" in the context of multiple HTTP requests. But it is still not truly thread safe *by*

How to Make/Design a Thread Safe "Object"?

- Here are a few examples:
- Design your beans **immutable**: for example, have no setters and only use constructor arguments to create a bean. There are other ways, such as Builder pattern, etc..
- Design your beans **stateless**: for example, a bean that *does* something can be just a function (or several). This bean in most cases can and *should* be stateless, which means it does not have any state, it only *does* things with function arguments you provide each time (on each invocation)
- Design your beans persistent: which is a special case of “immutable” but has some very nice properties. Usually is used in functional programming, where Spring (at least yet) not as useful as in imperative world, but I have used them with Scala/Spring projects.
- Design your beans with locks [last resort]: I would recommend against this unless you are working on a *lower-level* library. The reason is we (humans) are not good thinking in terms of locks. Just the way we are raised and

Is a Spring Data (JPA) Repository thread-safe?

- Generally, yes. It's assuming a managed EntityManager which we'll either obtain from Spring's factory classes (in case you're using Spring as container) or as a CDI managed bean (declared through an @Producer method).
- What about @Transactional?
  - You need to set your transaction isolation level to protect from dirty reads from the database, not worry about thread safety. The database takes care of the "dirty read" aspect of this question - not Spring's threading model.

- https://stackoverflow.com/questions/15965735/is-a-spring-data-jpa-repository-thread-safe-aka-is-simplejp
- https://stackoverflow.com/questions/38487099/is-spring-transactional-thread-safe