Difference btwn yml and properties file

| **YAML(.yml)** | **.properties** |
| --- | --- |
| Spec can be found [here](https://yaml.org/spec/) | It doesn’t really actually have a spec. The closest thing it has to a spec is actually the javadoc. |
| Human Readable (both do quite well in human readability) | Human Readable |
| Supports key/val, basically map, List and scalar types (int, string etc.) | Supports key/val, but doesn’t support values beyond the string |
| Its usage is quite prevalent in many languages like Python, Ruby, and Java | It is primarily used in java |
| Hierarchical Structure | Non-Hierarchical Structure |
| Spring Framework doesn’t support @PropertySources with .yml files | supports @PropertySources with .properties file |
| If you are using spring profiles, you can have multiple profiles in one single .yml file | Each profile need one separate .properties file |
| While retrieving the values from .yml file we get the value as whatever the respective type (int, string etc.) is in the configuration | While in case of the .properties files we get strings regardless of what the actual value type is in the configuration |

Use the **@Value** annotation to access the properties which is defined in the application – properties file.

**How do you Add, Filter to an application?**

There are three methods to add filter to Spring Boot application:

* By implementing Filter interface.
* Using FilterRegistrationBean.
* Using MVC controller.

Once you create a filter in Spring Boot and annotated with @Component annotation,it will automatically identified by Spring Boot.In some cases you many want to apple the **Spring Boot filter** based on a certain URL pattern and not for every request.

To run the filter for URL pattern, we can use FilterRegistrationBean. Don’t use @Component annotation for the filter class but register the filter using a FilterRegistrationBean.

Register the custom Filter using FilterRegistrationBean as a bean in @configuration file.

The OncePerRequestFilter filter base class that aims to guarantee a single execution per request dispatch, on any servlet container.You also have the option to configure it based on the URL pattern.

If you like to use the Servlet 3.0 annotations, we have the option to use the [@WebFilter annotation](https://www.javadevjournal.com/spring-boot/spring-boot-servletcomponentscan-annotation/) for the ***filter in Spring Boot***. This is how we can define a filter using @WebFilter annotation

|  |  |
| --- | --- |
| **Level** | **Description** |
| ALL | All levels including custom levels. |
| DEBUG | Designates fine-grained informational events that are most useful to debug an application. |
| INFO | Designates informational messages that highlight the progress of the application at coarse-grained level. |
| WARN | Designates potentially harmful situations. |
| ERROR | Designates error events that might still allow the application to continue running. |
| FATAL | Designates very severe error events that will presumably lead the application to abort. |
| OFF | The highest possible rank and is intended to turn off logging. |
| TRACE | Designates finer-grained informational events than the DEBUG |

Consider a scenario, where there are no stockDetails in the DB and still, whenever you hit the GET method you get 200(OK) even though the resource is not found which is not expected. Instead of 200, you should get 404 error.  
So to handle this, you need to create an exception, in the above scenario “StockNotFoundException”.

GetMapping("/stocks/{number}")

public Stock retriveStock(@PathVariable int number)

{

Stock stock = service.findOne(number);

if(Stock ==null)

//runtime exception

throw new StockNotFoundException("number: "+ number);

return stock;

}

Now, create a Constructor from [Superclass](https://www.mygreatlearning.com/blog/java-super-keyword-and-wrapper-class/?gl_blog_id=25325).

Right-click on the file -> Go to Source ->And generate constuctors from superclass-> and check the RuntimeException(String)-> and generate.

And add an annotation called ***@ResponseStatus*** which will give you 404 (not found) error.

package com.greatlearning;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class StockNotFoundException extends RuntimeException

{

public StockNotFoundException(String message)

{

super(message);

}

}

For **handling generic Exception** and RunTimeException, we can have 2 exception handlers.

@ControllerAdvice public class ApplicationExceptionHandler { @ExceptionHandler(Exception.class) public ResponseEntity handleGenericException(Exception e){ ShopException shopException = new ShopException(100, “Items are not found”); return ResponseEntity.status(HttpStatus.INTERNAL\_SERVER\_ERROR) .body(shopException); }

1. Spring boot application execution will start from the main() method
2. The main() method internally call SpringApplication.run() method
3. SpringApplication.run() method performs bootstrapping for our spring boot application
4. Starts StopWatch to identify the time taken to bootstrap the spring boot application
5. Prepares environment to run our spring boot application (dev, prod, qa, uat)
6. Print banner ( Spring Boot Logo prints on console)
7. Start the IOC container ( ApplicationContext) based on the classpath ( default, Web servlet/ Reactive)
8. Refresh context
9. Trigger Runners (ApplicationRunner or CommandLineRunner)
10. Return ApplicationContext reference ( Spring IOC)

springbootTest-Integration Test

1. @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)
2. @AutoConfigureMockMvc
4. @Autowired
5. private MockMvc mockMvc;

@WebMvcTest to test web layer or [@DataJpaTest](https://howtodoinjava.com/spring-boot2/testing/datajpatest-annotation/) to test persistence later.

Add **spring-boot-starter-test** in pom

@SpringBootTest(classes = SpringBootDemoApplication.**class**, webEnvironment = WebEnvironment.RANDOM\_PORT)

**public** **class** EmployeeControllerIntegrationTests

{

@LocalServerPort

**private** **int** port;

@Autowired

**private** TestRestTemplate restTemplate;

@Sql({ "schema.sql", "data.sql" })

@Test

**public** **void** testAllEmployees()

{

assertTrue(

**this**.restTemplate

.getForObject("http://localhost:" + port + "/employees", Employees.**class**)

.getEmployeeList().size() == 3);

}

Multi maven module project

Depency vs dependency management