



CS544 EA

Integration

Spring Boot: Code & Config

Code & Config

- Make a class with a **main()** your **@Configuration**
 - Spring boot prefers Java Config
 - Spring boot requires a `main()` method
 - Starts your Spring Boot application

```
@Configuration  
@ComponentScan  
@EnableAutoConfiguration  
@EnableWebSecurity  
public class Application {
```

Spring Boot annotation to have it look through your JARs and configure it self based on what it finds

```
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```

`SpringApplication.run()` is a Spring Boot method that requires the name of your primary configuration class and starts the application

@SpringBootApplication

- @SpringBootApplication is a **composite** of:
 - @Configuration
 - @ComponentScan
 - @EnableAutoConfiguration

Since all Spring Boot applications generally need all 3, might as well create a single composite

```
@SpringBootApplication
@EnableWebSecurity
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Properties or YAML

- Auto-configuration can't do everything
 - **Needs certain values** such as DB user / pass
- These values can be stored in:
 - application.properties
 - application.yml

Inside your resources dir

YAML is a superset of JSON

Examples

```
spring.datasource.url = jdbc:mysql://localhost/cs544
spring.datasource.username = root
spring.datasource.password = root
```

application.properties

```
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
spring.jpa.hibernate.ddl-auto = create-drop
```

```
spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.jsp
```

```
logging.level.root=DEBUG
```

```
spring:
  datasource:
    url: jdbc:mysql://localhost/cs544
    username: root
    password: root
  jpa:
    properties:
      hibernate.dialect: org.hibernate.dialect.MySQL5Dialect
    hibernate:
      ddl-auto: create-drop
      show-sql: true
  mvc:
    view:
      prefix: /WEB-INF/views
      suffix: .jsp
  logging:
    level:
      root: DEBUG
```

application.yml

Although it's longer for this small application you can see how it's useful for large applications

Additional Configuration

- Autoconfig takes care of most things
 - But if needed **you can add / override** config
- Can be in main or in extra config files:
 - @Import to include @Configuration files
 - @ImportResource can include XML config files

Profiles & External Configuration

- Spring Boot has **profile support**
 - Parts of your internal (in-project) configuration are active only in certain environments (dev / test / production / ...)
- External config can be picked up in many ways
 - To indicate which environment you are in
 - Or to overwrite internal config values

Profiles

- Multiple profiles can be active, indicated by:

```
spring.profiles.active=dev,mysql
```

- Then includes profile specific properties files:

```
application-dev.properties  
application-mysql.properties
```

- And activates configuration annotated for it

```
@Configuration  
@Profile("production")  
public class ProductionConfiguration {
```


External Config Options

Order of precedence

- 1) Devtools global settings properties on your home directory (`~/spring-boot-devtools.properties` when devtools is active).
- 2) `@TestPropertySource` annotations on your tests.
- 3) properties attribute on your tests. Available on `@SpringBootTest` and the test annotations for testing a particular slice of your application.
- 4) Command line arguments.
- 5) Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
- 6) `ServletConfig` init parameters.
- 7) `ServletContext` init parameters.
- 8) JNDI attributes from `java:comp/env`.
- 9) Java System properties (`System.getProperties()`).
- 10) OS environment variables.
- 11) A `RandomValuePropertySource` that has properties only in `random.*`.
- 12) Profile-specific application properties outside of your packaged jar (`application-{profile}.properties` and YAML variants).
- 13) Profile-specific application properties packaged inside your jar (`application-{profile}.properties` and YAML variants).
- 14) Application properties outside of your packaged jar (`application.properties` and YAML variants).
- 15) Application properties packaged inside your jar (`application.properties` and YAML variants).
- 16) `@PropertySource` annotations on your `@Configuration` classes.
- 17) Default properties (specified by setting `SpringApplication.setDefaultProperties`).

Command Line Specifying Profile(s)

- You can start a Spring Boot application as:

```
java -jar app.jar --spring-profiles-active=prod
```

- Or you can set a environment variable

```
SET SPRING_PROFILES_ACTIVE=prod
```

```
java -jar app.jar
```