

Properties Externalized Configurations

Configurations

✓ Spring Boot lets externalize your configuration so that you can work with the same application code in different environments. You can use a variety of external configuration sources, include Java properties files, YAML files, environment variables, and command-line arguments.

✓ By default, Spring Boot look for the configurations or properties inside application.properties/yaml present in the classpath location. But we can have other property files as well and make SpringBoot to read from them.

Config/Properties Preferences

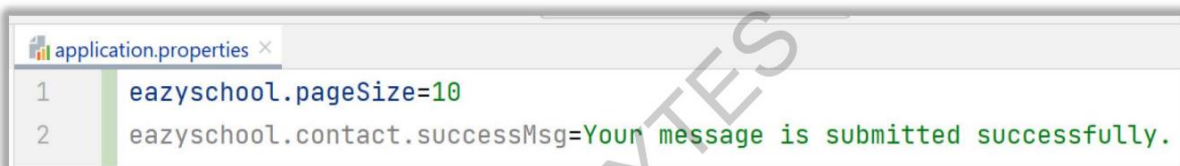
✓ Spring Boot uses a very particular order that is designed to allow sensible overriding of values. Properties are considered in the following order (with values from lower items overriding earlier ones):

- Properties present inside files like application.properties
- OS Environmental variables
- Java System properties (System.getProperties())
- JNDI attributes from java:comp/env.

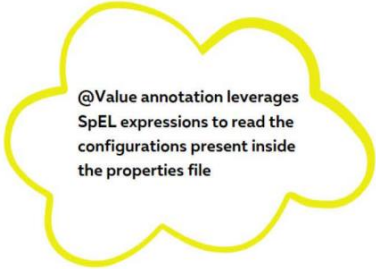
- ServletContext init parameters.
- ServletConfig init parameters.
- Command line arguments.

Reading properties with @Value

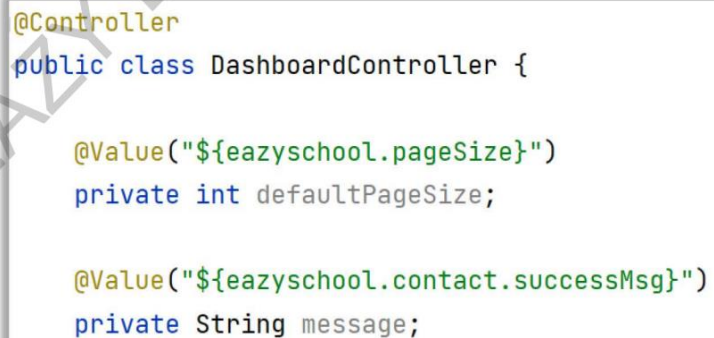
- We can read the properties/configurations, defined inside a properties file with the help of @Value annotation like shown below,



```
application.properties x
1 eazyschool.pageSize=10
2 eazyschool.contact.successMsg=Your message is submitted successfully.
```



@Value annotation leverages SpEL expressions to read the configurations present inside the properties file



```
@Controller
public class DashboardController {

    @Value("${eazyschool.pageSize}")
    private int defaultPageSize;

    @Value("${eazyschool.contact.successMsg}")
    private String message;
```

Reading properties using Environment

- Along with @Value, we can read the properties/configurations loaded with the help of Environment bean as well which is created by Spring framework. Apart from user defined properties, using Environment we can read any environment specific properties as well.

```
application.properties ×
1 eazyschool.pageSize=10
2 eazyschool.contact.successMsg=Your message is submitted successfully.
```

```
@Controller
public class DashboardController {

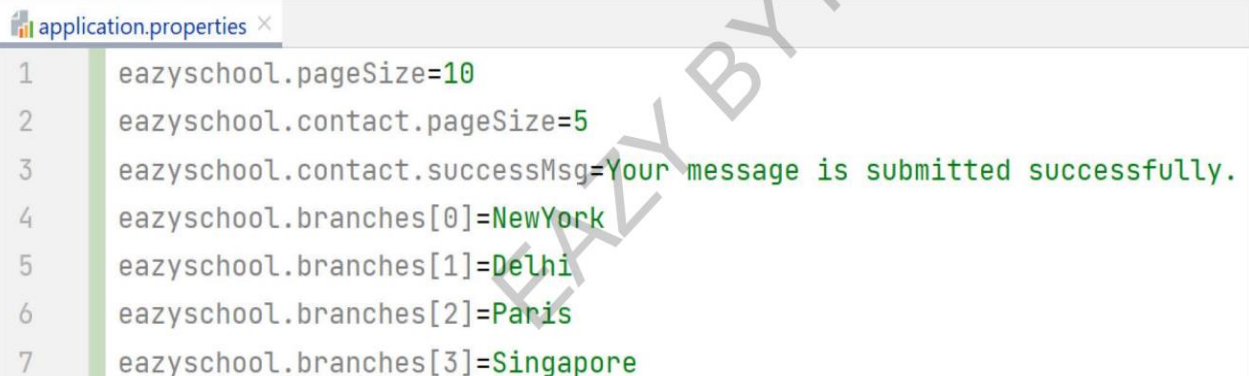
    @Autowired
    private Environment environment;

    private void logProperties() {
        log.info(environment.getProperty("eazyschool.pageSize"));
        log.info(environment.getProperty("eazyschool.contact.successMsg"));
        log.info(environment.getProperty("JAVA_HOME"));
    }
}
```

Reading properties with @ConfigurationProperties

- SpringBoot allow to load all the properties which are logically together into a java bean. For the same we can use `@ConfigurationProperties` annotation on top of a java bean by providing the prefix value. We need to make sure to use the names inside bean and properties file.
- Please follow below steps to read the properties using `@ConfigurationProperties`,

Step 1: We need to maintain the properties like below which have same prefix like 'eazyschool'



```
1 eazyschool.pageSize=10
2 eazyschool.contact.pageSize=5
3 eazyschool.contact.successMsg=Your message is submitted successfully.
4 eazyschool.branches[0]=NewYork
5 eazyschool.branches[1]=Delhi
6 eazyschool.branches[2]=Paris
7 eazyschool.branches[3]=Singapore
```

Step 2: Create a bean like below with all the required details,

```
@Component("eazySchoolProps")
@Data
@PropertySource("classpath:some.properties")
@ConfigurationProperties(prefix = "eazyschool")
@Validated
public class EazySchoolProps {

    @Min(value=5, message="must be between 5 and 25")
    @Max(value=25, message="must be between 5 and 25")
    private int pageSize;
    private Map<String, String> contact;
    private List<String> branches;
}
```

...
@PropertySource – can be used to mention the property file name if we are using something other than application.properties

...
@ConfigurationProperties – can be used to mention the prefix value that needs to be considered while loading the properties into a given bean

...
@Validated – can be used if we want to perform validations on the properties based on the validation mentioned on the field

Step 3: Finally we can inject the bean which we created in the previous step and start reading the properties from it using java style like shown below,

```
@Service
public class ContactService {

    @Autowired
    EazySchoolProps eazySchoolProps;

    public Page<Contact> findMsgsWithOpenStatus(int pageNum, String sortField, String sortDir){
        int pageSize = eazySchoolProps.getPageSize();
        if(null!=eazySchoolProps.getContact() && null!=eazySchoolProps.getContact().get("pageSize")){
            pageSize = Integer.parseInt(eazySchoolProps.getContact().get("pageSize").trim());
        }
    }
}
```

Reading properties with @PropertySource

- Sometimes we maintain properties inside files which has name is not equal to application.properties. In those scenarios, if we try to use the @Value, it will not work.
- First we need to communicate to SpringBoot about the property files with the below steps,

We need to create a class with @Configuration and @PropertySource annotation like mentioned below. Here we need to mention the property file name. Post these changes, we can refer the properties present inside the config.properties using @Value annotation or Environment bean. ignoreResourceNotFound = true will not throw an exception in case the file is missing.

```
@Configuration
@PropertySource(value = "classpath:config.properties", ignoreResourceNotFound = true)
public class AppConfig {

}
```

We can configure multiple property files as well using @PropertySources annotation like mentioned here,

```
@Configuration
@PropertySources({
    @PropertySource(value = "classpath:config.properties", ignoreResourceNotFound = true),
    @PropertySource("classpath:server.properties")
})
public class AppConfig {

}
```

Profiles for grouping Configurations

Profiles

✓ Spring provides a great tool for grouping configuration properties into so-called profiles(dev, uat, prod) allowing us to activate a bunch of configurations based on the active profile.

✓ Profiles are perfect for setting up our application for different environments, but they're also being used in another use cases like Bean creation based on a profile etc.

✓ So basically a profile can influence the application properties loaded and beans which are loaded into the Spring context.

Configuring Profiles

✓ The default profile is always active. Spring Boot loads all properties in application.properties into the default profile.

✓ We can create another profiles by creating property files like below,

application_prod.properties --> for prod profile

application_uat.properties --> for uat profile

✓ We can activate a specific profile using spring.profiles.active property like below,

spring.profiles.active=prod

QUICK TIP

Do you know there are many ways to activate a profile. Below are the most commonly used.

- ✓ By mentioning `spring.profiles.active=prod` inside the properties files.
- ✓ Using environment variables like below,

```
export SPRING_PROFILES_ACTIVE=prod
java -jar myApp-0.0.1-SNAPSHOT.jar
```
- ✓ Using Java System property,

```
java "-Dspring-boot.run.profiles=prod" -jar myApp-0.0.1-SNAPSHOT.jar
mvn spring-boot:run "-Dspring-boot.run.profiles=prod"
```
- ✓ Activating a profile programmatically invoking the method `setAdditionalProfiles("prod")` inside `SpringApplication` class.
- ✓ Using `@ActiveProfiles` while doing testing,

```
@SpringBootTest
@ActiveProfiles({"uat"})
```

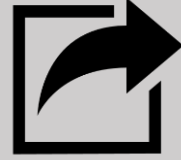


Conditional Bean creation using Profiles

- With the help of Profiles we can create the Bean conditionally. Below is an example where we can create different beans based on the active profile.

```
@Component
@Profile("!prod")
public class EazySchoolNonProdUsernamePwdAuthenticationProvider
    implements AuthenticationProvider
{
```

```
@Component
@Profile("prod")
public class EazySchoolUsernamePwdAuthenticationProvider
    implements AuthenticationProvider
{
```

http://www.instagram.com/_sarthak_.2