

- 1. Setting app properties at the command line
- 2. Specifying which properties file to use
- 3. Defining YAML properties files
- 4. Using Spring profiles
- 5. Spring Boot Actuator



1. Setting App Properties at the Command Line

- Recap of application properties
- Source of external configuration
- Setting properties at the command line

Recap of Application Properties

 A Spring Boot application can define properties in an application.properties file:

```
name=John Smith application.properties
```

• You can inject properties via @Value("\${propName}")

```
@Component
public class MyBean1 {

    @Value("${name}")
    private String name;
    ...
}
MyBean1.java
```



Source of External Configuration

- Spring Boot lets you define application properties in many places, such as:
 - Command-line arguments
 - Environment variable SPRING_APPLICATION_JSON
 - Operating system environment variables
 - Application properties outside your JAR
 - Application properties inside your JAR



Setting Properties at the Command Line

- If you define command-line args that start with --
 - Spring Boot converts them into application properties
- E.g., set the name property via a command-line arg:

```
--name="Mary Jones"
```

- Let's see an example in IntelliJ:
 - Edit configurations
 - Specify a command-line arg, as shown above
 - Run the configuration



2. Specifying which Properties File to Use

- Location of properties files
- Specifying a different properties file

Location of Properties Files

- SpringApplication looks in the following places to find properties files (highest priority first):
 - /config subdirectory of your Java app directory
 - Your Java app directory
 - /config package on classpath
 - Root package on classpath



Specifying a Different Properties File (1 of 2)

You can tell Spring to use a different properties file:

 Alternatively, you can set the SPRING_CONFIG_NAME environment variable



Specifying a Different Properties File (2 of 2)

 You can also use a command-line argument to specify which application properties file to use:

```
--spring.config.name=app2
```

- This enables you to specify a properties file as part of your overall CI/CD process
 - E.g. in a Jenkins build script



3. Defining YAML Properties Files

- Overview of YAML files
- Using YAML properties in beans technique 1
- Using YAML properties in beans technique 2

Overview of YAML Files

 Spring Boot supports YAML as an alternative format for defining application properties:

```
contact:
    tel: 555-111-2222
    email: contact@mydomain.com
    web: http://mydomain.com
    app3.yml
```

YAML is convenient for specifying hierarchical config data



Using YAML Properties in Beans - Technique 1

Here's one way to use YAML properties in a bean:

```
@Component
public class MyBean3a {

    @Value("${contact.tel}")
    private String tel;

    @Value("${contact.email}")
    private String email;

    @Value("${contact.web}")
    private String web;
    ...
}
MyBean3a.java
```



Using YAML Properties in Beans - Technique

Here's another way to use YAML properties in a bean:

```
@Component
@ConfigurationProperties(prefix="contact")
public class MyBean3b {

    private String tel;
    private String email;
    private String web;
    ...
    // Plus getters and setters - these are essential!
}
MyBean3b.java
```

You also need this dependency:



4. Using Spring Profiles

- Overview
- Defining profile-specific components
- Defining profile-specific properties
- Setting the active profile

Overview

- Spring profiles provide a way to segregate parts of your application configuration
 - So, configuration is only available in certain environments
- For example:
 - "development" profile
 - "production" profile



Defining Profile-Specific Components

You can annotate component classes with @Profile:

```
public interface MyBean4 {}
@Component
@Profile("development")
public class MyBean4Dev implements MyBean4 {
    @Override
    public String toString() { return "Hello from MyBean4Dev"; }
                                                                      MyBean4Dev.java
@Component
@Profile("production")
public class MyBean4Prod implements MyBean4 {
    @Override
    public String toString() { return "Hello from MyBean4Prod"; }
                                                                     MyBean4Prod.java
```

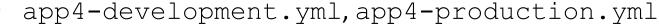


Defining Profile-Specific Properties

You can also define profile-specific properties:

```
apiserver:
  address: 192.168.1.100
                                                    Default values for properties
  port: 8080
spring:
  confia:
    activate:
                                           Properties for "development" profile
      on-profile: development
apiserver:
  address: 127.0.0.1
spring:
  confia:
    activate:
                                              Properties for "production" profile
      on-profile: production
apiserver:
  address: 192.168.1.120
                                                                                        app4.yml
```

- Alternatively, define profile-specific property files:
- app4-development.yr



Setting the Active Profile

- You must tell Spring what is the active profile
 - Set the spring.profiles.active property

To set the active profile via application properties:

spring.profiles.active=development

app4.properties

To set it at the command-line:

--spring.profiles.active=production



Annex: Spring Boot Actuator

- Overview of Spring Boot Actuator
- Enabling the Actuator
- Enabling Actuator endpoints
- Viewing mappings
- Health monitoring
- Gathering metrics
- Additional built-in actuator endpoints



Overview of Spring Boot Actuator

- Spring Boot Actuator is a sub-project of Spring Boot
 - Includes a number of additional features to help you monitor and manage your application when it's pushed to production
- You can manage and monitor your application using:
 - HTTP endpoints
 - JMX
 - Remote shell (SSH or Telnet)



Enabling the Actuator

 The simplest way to enable the Actuator is to add springboot-starter-actuator to your POM file



Enabling Actuator Endpoints

- From version 2 onwards, Spring Boot Actuator only has the following endpoints enabled by default:
 - /actuator/health
 - /actuator/info

 To enable other Spring Boot Actuator endpoints, set the following application property:

management.endpoints.web.exposure.include=*

application.properties



Viewing Mappings

 Spring Boot Actuator endpoints allow you to monitor and interact with your application, e.g. actuator/mappings

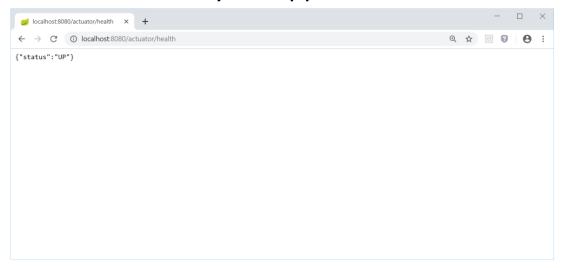
```
Jocalhost:8080/actuator/mapping X
              (i) localhost:8080/actuator/mappings
{"contexts":{"application":{"mappings":{"dispatcherServlets":{"dispatcherServlet":[{"handler":"ResourceHttpRequestHandler [class path
resource [META-INF/resources/], class path resource [resources/], class path resource [public/],
ServletContext resource [/], class path resource []]","predicate":"/**/favicon.ico","details":null},{"handler":"Actuator web endpoint
auditevents'", "predicate": "{GET /actuator/auditevents, produces [application/vnd.spring-boot.actuator.v2+json ||
application/ison]}"."details":{"handlerMethod":
{"className":"org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMycEndpointHandlerMapping.OperationHandler","name":"han
dle", "descriptor": "(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"}, "requestMappingConditions":
{"consumes":[], "headers":[], "methods":["GET"], "params":[], "patterns":["/actuator/auditevents"], "produces":
[{"mediaType":"application/vnd.spring-boot.actuator.v2+json","negated":false},{"mediaType":"application/json","negated":false}]}}},
{"handler":"Actuator web endpoint 'beans'","predicate":"{GET /actuator/beans, produces [application/vnd.spring-boot.actuator.v2+json
|| application/json]}", "details": { "handlerMethod":
{"className":"org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMapping.OperationHandler","name":"han
dle","descriptor":"(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"},"requestMappingConditions":
{"consumes":[], "headers":[], "methods":["GET"], "params":[], "patterns":["/actuator/beans"], "produces":
[{"mediaType":"application/vnd.spring-boot.actuator.v2+json","negated":false},{"mediaType":"application/json","negated":false}]}}},
{"handler": "Actuator web endpoint 'caches-cache'", "predicate": "{GET /actuator/caches/{cache}, produces [application/vnd.spring-
boot.actuator.v2+json || application/json]}","details":{"handlerMethod":
{"className":"org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMycEndpointHandlerMapping.OperationHandler"."name":"han
dle", "descriptor": "(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"}, "requestMappingConditions":
{"consumes":[],"headers":[],"methods":["GET"],"params":[],"patterns":["/actuator/caches/{cache}"],"produces":
[{"mediaType":"application/vnd.spring-boot.actuator.v2+json","negated":false},{"mediaType":"application/json","negated":false}]}}},
"handler": "Actuator web endpoint 'caches'", "predicate": "{GET /actuator/caches, produces [application/vnd.spring boot.actuator.v2+json
|| application/json]}","details":{"handlerMethod":
{"className":"org.springframework.boot.actuate.endpoint.web.servlet.AbstractWebMvcEndpointHandlerMapping.OperationHandler"."name":"han
dle", "descriptor": "(Ljavax/servlet/http/HttpServletRequest;Ljava/util/Map;)Ljava/lang/Object;"}, "requestMappingConditions":
```

- Note:
 - Depending on your Spring Boot version, you might need to allow access to the Actuator endpoints



Health Monitoring

 The actuator/health endpoint provides basic health information about your application





Gathering Metrics

 The actuator/metrics endpoint provides metrics that help you identify bottlenecks and optimize performance



• To get info for any of these metrics, append the metric name - e.g. actuator/metrics/jvm.memory.max



Additional Built-in Actuator Endpoints

- Here are some more built-in Actuator endpoints ...
 - /actuator/beans Lists all the beans in the app
 - /actuator/configprops Lists @ConfigurationProperties
 - /actuator/env Lists environment variables
 - /actuator/scheduledtasks Lists scheduled tasks in the app
 - /actuator/threaddump Performs a thread dump
- For full details of all the built-in Actuator endpoints, and info on how to define custom endpoints, see:
 - https://docs.spring.io/spring-boot/docs/current/reference/html/production-readyfeatures.html





- Setting app properties at the command line
- Specifying which properties file to use
- Defining YAML properties files
- Using Spring profiles
- Spring Boot Actuator

