**Need for profiles**

Application development is complex process. You might have different environment for application development.

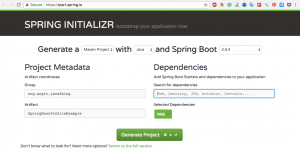
* Dev
* SIT
* QA
* Prod

We require different configuration based on the environment.

For example: Using a different database or different queues.

Let’s create a simple spring boot application.

**Step 1:** Go to <https://start.spring.io/> and create spring boot projects as per below screenshot.



**Step 2:** [Import the maven project in eclipse](https://java2blog.com/how-to-convert-existing-java-project-to/).

**step 3:** Create a package named "org.arpit.java2blog.controller"  
create a controller class named **"HelloWorldController.java"**

**WelcomeController.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | package org.arpit.java2blog.controller;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.RestController;    @RestController  public class WelcomeController {        @Value("${message}")      String message;        @GetMapping("/welcome")      public String getMessage()      {          return "Hi "+message;      }  } |

**Step 4:** Put “message=java2blog” in application.properties.

**application.properties**

|  |  |
| --- | --- |
| 1  2  3 | message= java2blog |

**Step 5:** Create a package named “org.arpit.java2blog”  
create a class named **“SpringBootHelloWorldApplication.java”**

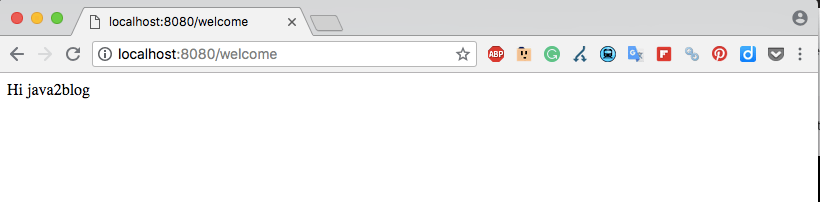
**SpringBootProfileExample.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package org.arpit.java2blog;  package org.arpit.java2blog;    import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;    @SpringBootApplication  public class SpringBootProfileExample {        public static void main(String[] args) {          SpringApplication.run(SpringBootProfileExample.class, args);      }  } |

**Step 6:** Run the application  
When you run above application, you will get below output:

…  
…  
2018-08-03 10:40:21.649 INFO 27143 — [ main] o.a.java2blog.SpringBootProfileExample : Started SpringBootProfileExample in 4.693 seconds (JVM running for 5.474)  
2018-08-03 10:40:36.761 INFO 27143 — [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring FrameworkServlet ‘dispatcherServlet’  
2018-08-03 10:40:36.762 INFO 27143 — [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet ‘dispatcherServlet’: initialization started  
2018-08-03 10:40:36.789 INFO 27143 — [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet ‘dispatcherServlet’: initialization completed in 27 ms

**Step 7:**Go to browser and browse url “http://localhost:8080/welcome”

  
As you can see, we are able to fetch “message” from application.properties.

**Profiles for environment specific properties**

Let’s configuration two profile- dev and prod for demonstration.  
You need to create .properties file with application-"$ProfileName".  
**For example:**  
For dev profile, you need to crate application-dev.properties, so whatever you put in application-dev.properties, it will override properties from application.properties.  
Suppose you need to start qa profile at port “8081”, you can put following entry in application in application-qa.properties

**application-qa.properties**

|  |  |
| --- | --- |
| 1  2  3 | server.port=8081 |

Let’s create two properties file for our application.  
**For dev profile**

**application-dev.properties**

|  |  |
| --- | --- |
| 1  2  3 | message = java2blog from dev |

**For prod profile**

**application-prod.properties**

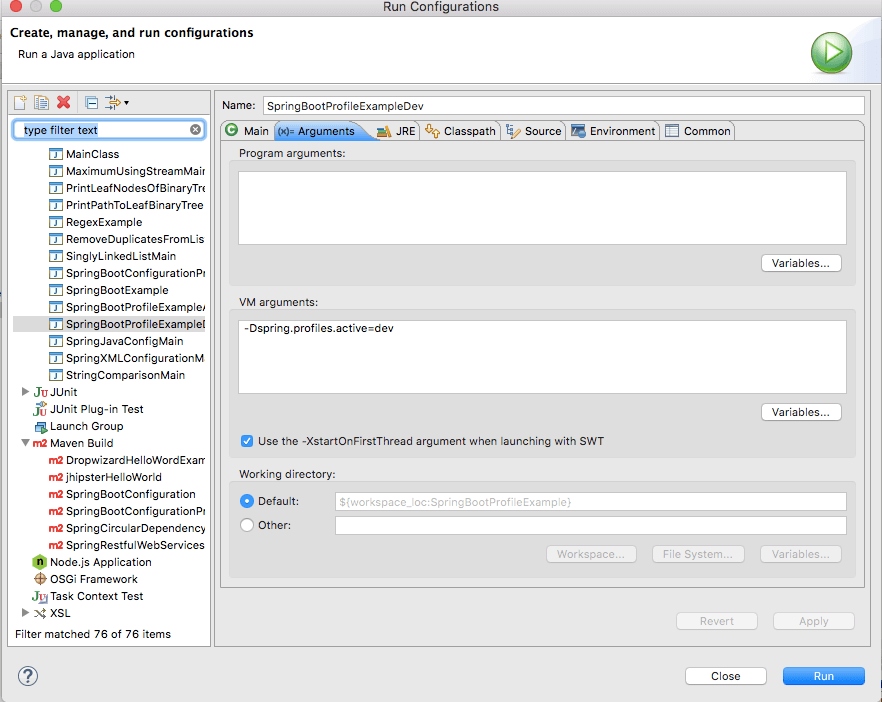
|  |  |
| --- | --- |
| 1  2  3 | message = java2blog from prod |

**Setting active profile**

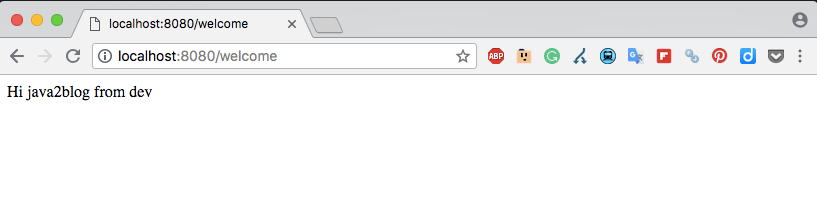
When you run spring boot application, you need to set active profile. There are multiple ways to set active profile.

1. By using -Dspring.profiles.active=dev in VM Arguments
2. By using spring.profiles.active=dev in application.properties

Let’s run the application with dev profile now. We will put spring.profiles.active=dev in VM arguments and click on run.



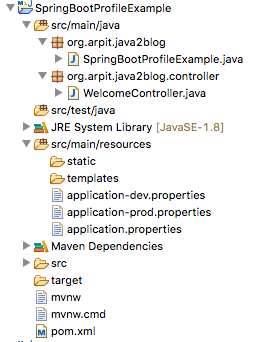
When you run the application and browse http://localhost:8080/welcome”.



**Spring boot @Profile annotation**

You can use [@Profile annotation](https://java2blog.com/spring-profile-annotation-example/) in java configuration to create profile based configuration.

**Project Structure**



# Spring Profile annotation example

Table of Contents [[hide](https://java2blog.com/spring-profile-annotation-example/)]

* [Project Structure](https://java2blog.com/spring-profile-annotation-example/#Project_Structure)
* [Source code](https://java2blog.com/spring-profile-annotation-example/#Source_code)
  + [Was this post helpful?](https://java2blog.com/spring-profile-annotation-example/#Was_this_post_helpful)

In this tutorial, we will see about Spring @Profile annotation. This tutorial is an extension to [Spring boot profile example](https://java2blog.com/spring-boot-profiles-example/).

Profile annotation represents named logical grouping  may be activated via

* By using ConfigurableEnvironment.setActiveProfiles(java.lang.String...)
* By setting the spring.profiles.active property as a [JVM](https://java2blog.com/java-virtual-machine-architecture/) system property, as an environment variable, or as a Servlet context parameter in web.xml for web applications.
* You can also use @ActiveProfiles< while doing integration testing

The @Profile annotation may be used in any of the following ways:

* As a type-level annotation on any class directly or indirectly annotated with [@Component](https://java2blog.com/spring-component-service-repository-and-controller-annotations/), including @Configuration classes
* As a meta-annotation, for the reason of composing custom stereotype annotations
* It can be used method-level annotation on any @Bean annotation

Let’s see with the help of an example.  
**Step 1:** Go to "<https://start.spring.io/>" and create spring boot projects as per below screenshot.

**Step 2:** Import the maven project in eclipse.

**step 3:** Create a package named "org.arpit.java2blog.model"  
create a model class named **"Customer.java"**

**Customer.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package org.arpit.java2blog.model;    public class Customer {          int id;          String name;          String email;            public Customer(int id, String name, String email) {              super();              this.id = id;              this.name = name;              this.email = email;          }            public int getId() {              return id;          }          public void setId(int id) {              this.id = id;          }          public String getName() {              return name;          }          public void setName(String name) {              this.name = name;          }          public String getEmail() {              return email;          }          public void setEmail(String email) {              this.email = email;          }    } |

**Step 4:**Create a package named “org.arpit.java2blog.config”  
create a config class named **“ConfigurationDatabase.java”**

**ConfigurationDatabase.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | package org.arpit.java2blog.config;    import org.arpit.java2blog.model.Customer;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.context.annotation.Profile;    @Configuration  public class ConfigurationDatabase {        @Profile("dev")      @Bean(name="customer")      public Customer getDevCustomer()      {          return new Customer(1,"John","John@gmail.com");      }        @Profile("prod")      @Bean(name="customer")      public Customer getProdCustomer()      {          return new Customer(1,"Martin","martin@gmail.com");      }    } |

**Step 5:** Create test class named “SpringProfileAnnotationDevTest” in src/test/java.

**SpringProfileAnnotationDevTest.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | package org.arpit.java2blog.SpringProfileAnnotation;    import static org.junit.Assert.assertEquals;  import static org.junit.Assert.assertNotNull;    import org.arpit.java2blog.config.ConfigurationDatabase;  import org.arpit.java2blog.model.Customer;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.context.ApplicationContext;  import org.springframework.test.context.ActiveProfiles;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringRunner;  import org.springframework.test.context.support.AnnotationConfigContextLoader;    @RunWith(SpringRunner.class)  @SpringBootTest  @ActiveProfiles(value="dev")  @ContextConfiguration(classes= {ConfigurationDatabase.class},loader=AnnotationConfigContextLoader.class)  public class SpringProfileAnnotationDevTest {        @Autowired      private ApplicationContext applicationContext;          @Test        public void testDevDataSource() {            Customer customer = (Customer)applicationContext.getBean("customer");            assertNotNull(customer);            assertEquals("John", customer.getName());        }    } |

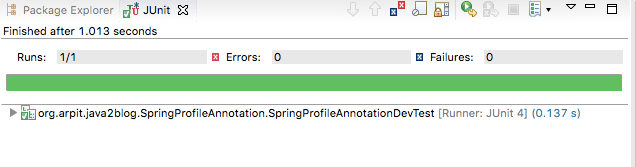
Here

**@SpringBootTest** annotation is used to run Spring boot test.

**@ContextConfiguration** provides class-level metadata to determine how to load and configure an ApplicationContext for integration tests.

**@ActiveProfiles** is used to provide name of the profile which will be activated while running integration tests.

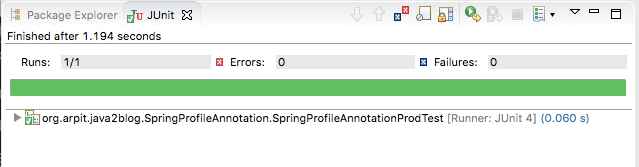
**Step 6:** Run the test  
When you run above test, you will get below output:



**Step 7:**Create another test class named “SpringProfileAnnotationProdTest” in src/test/java.  
create a class named **“SpringBootHelloWorldApplication.java”**

**SpringProfileAnnotationProdTest.java**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package org.arpit.java2blog.SpringProfileAnnotation;    import static org.junit.Assert.assertEquals;  import static org.junit.Assert.assertNotNull;    import org.arpit.java2blog.config.ConfigurationDatabase;  import org.arpit.java2blog.model.Customer;  import org.junit.Test;  import org.junit.runner.RunWith;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.boot.test.context.SpringBootTest;  import org.springframework.context.ApplicationContext;  import org.springframework.test.context.ActiveProfiles;  import org.springframework.test.context.ContextConfiguration;  import org.springframework.test.context.junit4.SpringRunner;  import org.springframework.test.context.support.AnnotationConfigContextLoader;    @RunWith(SpringRunner.class)  @SpringBootTest  @ActiveProfiles(value="prod")  @ContextConfiguration(classes= {ConfigurationDatabase.class},loader=AnnotationConfigContextLoader.class)  public class SpringProfileAnnotationProdTest {        @Autowired      private ApplicationContext applicationContext;          @Test        public void testDevDataSource() {            Customer customer = (Customer)applicationContext.getBean("customer");            assertNotNull(customer);            assertEquals("Martin", customer.getName());        }  } |

**Step 8:** Run the test  
When you run above test, you will get below output:  
  
As you can see, we are getting different bean based on active profile.

As we you can see we are getting name as “John” with @ActiveProfiles(value="dev") and “Martin in case of @ActiveProfiles(value="prod")

## Project Structure

