AUG 14, 2017 • 6 MIN READ • SPRING

Conditional Beans Creation In Spring Boot

Spring Container is core to Spring Framework. It manages the entire lifecycle of Spring Beans. A bean is nothing but an instance of a class right?. You can declare a bean definition either using XML based configuration or Annotation based configuration, Annotation based is my personal favorite. But, have you came across a situation where you want to initialize bean based on a certain condition?

Like if my Java version is 7 and I want some beans to get created and if it's Java 8 than some other beans and not those that relate to Java 7. I know right now many of us add both the spring beans definition in our XML and java config file. And then, using some if...else logic to do the required thing, isn't it? But hey, we want to write programs that can be easily re-configured and doesn't require major code changes because in large application making a small change can trigger a big testing and build phase. And, you know how painful it is.

Conditional Beans Creation In Spring Boot

(since version 1.1.0). That means you don't have to write unnecessary if...else blocks to check the conditions and then use the right bean. In this post, I will walk you through different ways of conditionally creating spring beans.

There are various Annotation that are provided by Spring Boot each having a specific purpose. The below list shows all the different Conditional Annotation in Spring.

- ConditionalOnBean
- ConditionalOnClass
- ConditionalOnCloudPlatform
- ConditionalOnExpression
- ConditionalOnJava
- ConditionalOnJndi
- ConditionalOnMissingBean
- ConditionalOnMissingClass
- ConditionalOnNotWebApplication
- ConditionalOnProperty
- ConditionalOnResource
- ConditionalOnSingleCandidate
- ConditionalOnWebApplication

I won't be going through all the annotations in this post. However, after explaining you some of them you will get the idea of how to use others. Please look for the comments in the code blocks for explanations.

ConditionalOnBean

Conditional Beans Creation In Spring Boot

present in the Bean factory. The annotation when placed on a `@Bean` then the bean class is default to the return type of the bean definition.

```
@Configuration
public class ConditionalOnBeanConfig {

    @Bean
    public A beanA(){
        return new A();
    }

    @Bean
    @ConditionalOnBean(name="beanA")
    public B beanB(){
        return new B(); // it will initialize as beanA is present i
    }

    @Bean
    @ConditionalOnBean
    public C beanC(){
        return new C(); // will not get initialized as there is no
    }

    @Bean
    public SimpleInt beanAInt(){
        return new ASimpleInt();
    }

    @Bean
    @ConditionalOnBean
    public SimpleInt beanBInt(){
        return new BSimpleInt();
    }
}
```

Conditional Beans Creation In Spring Boot

Conditional that only matches when the specified class is available on the classpath. The attributes to the annotation are `name` and `value`. Use the name attribute in case if you want to specify the class name and you are not sure whether the classes will be available on the classpath and use value when the classes are available on the classpath.

```
@Configuration
public class ConditionOnClassConfig {

    @Bean
    @ConditionalOnClass(value={java.util.HashMap.class})
    public A beanA(){
        return new A(); // will get created as HashMap class is on
    }

    @Bean
    @ConditionalOnClass(name="com.sample.Dummy")
    public B beanB(){
        return new B(); // won't be created as Dummy class is not c
    }

    @Bean
    @ConditionalOnClass(value=com.sample.bean.conditional.model.ASi
    public C beanC(){
        return new C(); // ASimpleInt is on the classpath in the pr
                        //So, C's instance will be created.
    }

}
```

ConditionalOnMissingBean

Conditional that only matches when the specified bean is missing from the

Conditional Beans Creation In Spring Boot

configuration classes only.

```
@Configuration
public class ConditionalOnMissingBeanConfig {

    @Bean
    public A beanA(){
        return new A(); // will initialize as normal
    }

    @Bean
    @ConditionalOnMissingBean(name="beanA")
    public B beanB(){
        return new B(); // it will not initialize as
                        // beanA is present in the beanFactory.
    }

    @Bean
    @ConditionalOnMissingBean(name="beanD")
    public C beanC(){
        return new C(); // will get initialized as there is no
                        // bean with name beanD in BeanFactory.
    }

}
```

ConditionalOnMissingClass

Conditional that only matches when the specified Class is missing from the classpath.

```
@Configuration
public class ConditionOnMissingClassConfig {
```

Conditional Beans Creation In Spring Boot

```
    public A beanA(){
        return new A(); // will get created as Dummy class is missi
                        // from classpath.
    }

    @Bean
    @ConditionalOnMissingClass(value={"com.sample.bean.conditional.
    public B beanB(){
        return new B(); // won't be created as C is present in clas
    }

}
```

ConditionalOnWebApplication

Conditional that only matches when the application context is a web application context.

> *Note: Code is shown in next block.*

ConditionalOnNotWebApplication

Conditional that only matches when the application context is not a web application context.

```
@Configuration
public class ConditionalBeanNotWebConfig {

    @Bean
    @ConditionalOnWebApplication
    public A beanA(){
        return new A(); // will initiate if the context is
```

Conditional Beans Creation In Spring Boot

```
    }

    @Bean
    @ConditionalOnNotWebApplication
    public B beanB(){
        return new B(); // will initiate if the context is not
                        // WebApplicationContext.
    }
}
```

ConditionalOnResource

Conditional that only matches when the specified resource is available on the classpath.

```
@Configuration
public class ConditionalOnResourceConfig {

    @Bean
    @ConditionalOnResource(resources={"classpath:application.proper
    public A beanA(){
        return new A(); // will initiate as application.properties
                        // classpath.
    }

    @Bean
    @ConditionalOnResource(resources={"file:///e:/doc/data.txt"})
    public B beanB(){
        return new B(); // will not initialize as the file is not
                        // present in the given location.
    }
}
```

It checks whether the version matches the current JVM version the application is running. There are two properties to this annotation `value` and `range`. The `value` takes any one of the given `JavaVersion` enum as given below i.e. SIX, SEVEN, EIGHT, NINE.

```
@Configuration
public class ConditionalBeanJavaConfig {

    @Bean
    @ConditionalOnJava(value=JavaVersion.EIGHT)
    public A beanA(){
        return new A();
    }

    @Bean
    @ConditionalOnJava(value=JavaVersion.SEVEN,range=Range.OLDER_TH
    public B beanB(){
        return new B();
    }
}
```

In the above case, I am using JDK version 8. So, only `beanA` will get initialized while the `beanB` I am trying to check if the JDK version is below 7 using `range` property which initializes B.

ConditionalOnProperty

Conditional on property checks if the specified property has the specific value. It checks by default if the property is in the Environment and not equals to **false**.

```
@Configuration
public class ConditionalBeanPropertyConfig {
```

Conditional Beans Creation In Spring Boot

```
    @Bean
    @ConditionalOnProperty(name="test.property", havingValue="A")
    public A beanA(){
        return new A();
    }

    @Bean
    @ConditionalOnProperty(name="test.property", havingValue="B")
    public B beanB(){
        return new B();
    }

    @Bean
    @ConditionalOnProperty(name="test.property", havingValue="C",ma
    public C beanC(){
        return new C();
    }
}
```

The above example shows how to configure conditional on property annotation. The `havingValue` and `matchIfMissing` allows for additional customization. The having value checks if the given value equals to the value of the property and if it's true then initialize the bean. A `havingValue=""` will always returns **true**. `matchIfMissing` specifies that the condition should match if the property is missing.

JUnit Test:

I have wrote JUnits for each of the above example I have given. You can find all those in the given below github link[^1]. To make the post short I didn't include all those. But to show you how the tests works I have put one test case for ConditionBeanPropertyConfig Class as below.

```
@RunWith(SpringRunner.class)
```

Conditional Beans Creation In Spring Boot

```
public class ConditionalBeanPropertyConfigTest {

    @Autowired
    ApplicationContext ctx;

    @Test
    public void testBeanA() {
        A beanA = ctx.getBean(A.class);
        assertNotNull(beanA);
    }

    @Test(expected=NoSuchBeanDefinitionException.class)
    public void testBeanB() {
        B beanB = ctx.getBean(B.class);
        assertNotNull(beanB);
    }

    @Test(expected=NoSuchBeanDefinitionException.class)
    public void testBeanC() {
        C beanC = ctx.getBean(C.class);
        assertNotNull(beanC);
    }

}
```

The Spring Boot Test checks for initialization of All the three beans. But, as the properties file is configured with the value `A` only `testBeanA` will return the bean instance while the other two methods will return `NoSuchBeanDefinitionException`. If the property value is changed to `B` then, in that case, bean B will get initialized likewise in the case of C. However, if the property `test.property` is missing from the environment only the `C` instance will get created leaving A and B uninitialized.

Properties file for All above examples:

```
test.property=A
```

Test Output

I tried to explain most of the Annotation usage and for those that are missing you got the idea of how to use them. If you have any doubt don't bother to comment on the post I will get back to your queries as quickly as possible. Also, please subscribe to the newsletter to get the blog updates. If there's anything else you want to discuss with me use the Contact form. Please don't forget to share the post using the below-given links. Thanks!.

Links:

Download Code From Github

Other Entities Used

```
public class A {
    public String sayHello(){
        return "Hello From A";
    }
}

public class B {
    public String sayHello(){
        return "Hello From B";
    }
}

public class C {
    public String sayHello(){
        return "Hello From C";
    }
```

Conditional Beans Creation In Spring Boot

```
public interface SimpleInt {
    void sample();
}

public class ASimpleInt implements SimpleInt {
    @Override
    public void sample() {
        System.out.println("Hello From A implementation of SimpleIn
    }
}

public class BSimpleInt implements SimpleInt {
    @Override
    public void sample() {
        System.out.println("Hello from B implementation of SimpleIn
    }
}
```

Published by:

SHARE TWEET

You might also like...

JUL 28 Service Locator Using Spring Framework 4 min read

Conditional Beans Creation In Spring Boot

3 Comments iamninad Login

Recommend 2 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

murali • 2 years ago

good explanation

^ | v • Reply • Share ›

Abhijeet Kumar Sinha • 4 years ago

Can we also know which version of Boot added this feature. By the this is fantastic.

^ | v • Reply • Share ›

Ninad Ingole itsme ➚ Abhijeet Kumar Sinha • 4 years ago

Sure Will Add the version to the post

Powered by Ghost

◐ System theme