

Spring Java Config

Craig Kelly

November 11, 2015

XML Config for us today

- Fairly standard XML file detailing the beans we need
- Generally fine, but any “dynamic” action is limited to factory beans and the like

```
<bean id="mainBean" class="demo.com.fedex.devframework.BigBean">  
  <property name="neededBean" ref="otherBean" />  
</bean>
```

```
<bean id="otherBean" class="demo.com.fedex.devframework.SubBean">  
  <property name="name" value="A" />  
</bean>
```

Autowiring is Cool, too

- Annotated classes are “auto-wired” on context startup
- Much of the configuration is inferred by scanning for annotated classes

Want More Info?

See Raja's presentations on Spring Annotations!

Java Config is an alternative

- Replace the XML file with a Java class
- Get all the power of executing custom Java code at context startup
- Compatible with existing XML config and Autowiring

```
@Configuration
public class JavaAppContext {
    @Bean
    public BigBean mainBean() {
        BigBean bean = new BigBean();
        bean.setNeededBean(childBean());
        return bean;
    }

    @Bean
    public SubBean childBean1() {
        SubBean bean = new SubBean();
        bean.setName("A");
        return bean;
    }
}
```

Basic Annotations

- @Configuration is used to annotate a class and specify that it can be used for configuring a context
- @Bean is used to annotate a method indicating that the method creates a bean

Build Alert

Using these annotations means that you WILL need cglib

Some Restrictions

- @Configuration classes must be non-final
- @Configuration classes must be non-local (may not be declared within a method)
- @Configuration classes must have a default/no-arg constructor and
- @Configuration classes may not use @Autowired constructor parameters.
- Any nested configuration classes in a @Configuration class must be static

Loading the Context

If you can load a context with XML, then there is an alternative for Java Config. Take the simplest example:

```
ApplicationContext ctx;  
ctx = new ClassPathXmlApplicationContext("applicationContext.xml");  
ctx.getBean("mainBean", SomeMainClass.class).doStuff();
```

There is an equivalent for Java Config. Let's assume that the Java class that contains our context configuration is named `JavaAppContext`.

```
ApplicationContext ctx;  
ctx = new AnnotationConfigApplicationContext(JavaAppContext.class);  
ctx.getBean("mainBean", SomeMainClass.class).doStuff();
```

Spring Magic for Java Config, Part 1

Consider this example. There are only two instances in the context: the `catHat` bean, and the `someThing` bean. The `catHat` bean has two references to the **same** instance of `someThing`.

```
<bean id="someThing" class="demo.com.fedex.Thing" />

<bean id="catHat" class="demo.com.fedex.CatInTheHat">
  <property name="thingOne" ref="someThing" />
  <property name="thingTwo" ref="someThing" />
</bean>
```


Spring Magic for Java Config, Part 2

Here's the same example done with Java Config. `something` appears to be called twice, so you might expect the `catHat` bean to have two different objects. Luckily, the Spring annotation handles this so our Java code “behaves” like the XML context. *

```
@Configuration
public class JavaAppContext {
    @Bean
    public Thing something() {
        return new Thing();
    }

    @Bean
    public CatInTheHat catHat() {
        CatInTheHat bean = new CatInTheHat();
        bean.setThingOne(something());
        bean.setThingTwo(something());
        return bean;
    }
}
```

*Yes, I realize that this is also a radical reinterpretation of a classic piece of literature.

Sample Project

A sample project is available demonstrating all of the above. There is even a unit test demonstrating that an XML context and a Java Config context produce the same results. The sample project will be available in the same directory that you found this presentation.

Build Alert

It's a Maven project!