1. Goals

The simple goal of this lesson is to explore the new alternative to Classpath scanning for beans, that Spring 5 introduces.

2. Lesson Notes

2.1. Why a Classpath Scanning Alternative?

Spring 5 introduces a new feature - which can generate a static list candidate components/beans. The container then uses this list for bootstrapping the context of the application.

The goal here is to reduce application startup time.

2.2. What Does Classpath Scanning Look Like?

First, let's run the app and see what the typical classpath scanning mechanism looks like, by analyzing the log output of the startup process.

The primary class doing all the logging here is *ClassPathBeanDefinitionScanner*.

And here's what a typical logging entry related to the scanning of a bean looks like:

- Identified candidate component class: file [C:\opt\git\courses\private\restwithspring\m12-lesson4-start\um-webapp\target\classes\com\baeldung\um\service\impl\PrivilegeServiceImpl.class]

Notice how the component is identified (by scanning the classpath).

2.3. What Does the New Static Mechanism Look Like?

To enable support for this new indexing, let's first define the Maven dependency (in both modules):

HTML

```html
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-indexer</artifactId>
</dependency>
```

This dependency is actually only required at compile time - as it uses annotation processing for generating candidates components.

2.4. Compile and Run

Now, let's build the parent, and then open and make sure the list of components/beans has been correctly generated:

target/classes/META-INF/spring.components

Notice how this file does exist in both modules, and how all annotated beans are listed here.

Now, with the new mechanism active, let's run the application and have a look at what a typical log entry would look like:

ClassPathBeanDefinitionScanner

- *Using candidate component class from index: com.baeldung.um.service.impl.PrivilegeServiceImpl*

Notice that the component is identified from the index.

2.5. How It Works

Using annotation processing, Spring generates the list of beans - by scanning classes annotated with any version of *@Compenent*.

The indexer also adds any class or interface that:

- has a type-level annotation from the *javax* package
- this includes JPA (*@Entity* and related)
- also CDI (*@Named*, *@ManagedBean*)
- and Servlet annotations (i.e. *@WebFilter*)

We could also add the new *@Indexed*annotation on any other annotation - to instructs the scanner to index a source file that contains that annotation

Now, when starting up the application and loading the context - Spring will use these generated files to bootstrap everything.

2.6. Fallback

Finally, let's talk about disabling the mechanism and falling back to the standard Classpath scanning we had before.

We can do that by setting the **spring.index.ignore** to **true**:

- as a system property or
- or in the *spring.properties* file at the root of the classpath

Let's use the first option - as a system property - in the VM arguments of the run config:

-Dspring.index.ignore=true

3. Conclusion

And we're done.

We now have everything we need set up to take advantage of the new bootstrapping functionality.