1 @SpringBootApplication 注解的应用
2 @SpringBootTest 注解的应用
3 @SpringBootApplication 和 @SpringBootTest 的区别
4 @ComponentScan(包含了两个filter) 解析
    4.1 TypeExcludeFilter 解析
    4.2 AutoConfigurationExcludeFilter 解析
5 @EnableAutoConfiguration 注解解析
6 @...Test 注解
参考

1 @SpringBootApplication 注解的应用

- 一般情况我们使用 @SpringBootApplication 注解来启动 SpringBoot 项目
- 它其实只相当于 @Configuration、@EnableAutoConfiguration、@ComponentScan(包含了两个filter)

```
@SpringBootApplication
public class FrameworkUnitRealTestApp {
    public static void main(String[] args) {
        SpringApplication.run(FrameworkUnitRealTestApp.class, args);
    }
}
```

2 @SpringBootTest 注解的应用

- 一般情况我们使用 @SpringBootTest 和 @RunWith(SpringRunner.class) 注解来启动 SpringBoot **测试**项目

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class FrameworkUnitRealTestApp {
    @Test
    public void test() {}
}
```

3 @SpringBootApplication 和 @SpringBootTest 的区别

- 这两个注解的区别的核心在于两个注解：@EnableAutoConfiguration、@ComponentScan(包含了两个filter)
- @EnableAutoConfiguration 启动了所有的自动配置类
- @ComponentScan(包含了两个filter)：在扫描阶段过滤掉 @TestComponent 等专属于测试的类和过滤掉**被@Configuration 注解的自动配置类**（使得自动配置类不会在扫描阶段就被注册 beanDefinition，因为 自动配置类的优先级应该是最低的）
- 可以看出 @SpringBootTest 并没有启用任何自动配置类，所以就不需要加 AutoConfigurationExcludeFilter 了
- springboot 通过引入 @Test** 注解来在 **测试环境下** 引入不同的自动配置类！

4 @ComponentScan(包含了两个filter) 解析

- 详细的代码如下：添加了 TypeExcludeFilter 和 AutoConfigurationExcludeFilter 两个 excludeFilter
- 作用：扫描包的时候过滤掉被这两个 Filter **匹配**的类！

```
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
        @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
```

4.1 TypeExcludeFilter 解析

- 主要移除测试相关的类

```
public class TypeExcludeFilter implements TypeFilter, BeanFactoryAware {
   @Override
   public boolean match(MetadataReader metadataReader,
         MetadataReaderFactory metadataReaderFactory) throws IOException {
      if (this.beanFactory instanceof ListableBeanFactory
            && getClass() == TypeExcludeFilter.class) {
         Collection<TypeExcludeFilter> delegates = ((ListableBeanFactory)
this.beanFactory)
               .getBeansOfType(TypeExcludeFilter.class).values();
         for (TypeExcludeFilter delegate : delegates) {
            if (delegate.match(metadataReader, metadataReaderFactory)) {
               return true;
            }
         }
      }
      return false;
   }
}
//delegate.match 走这个类的 match 方法
class TestTypeExcludeFilter extends TypeExcludeFilter {
    private static final String[] CLASS_ANNOTATIONS = { "org.junit.runner.RunWith",
            "org.junit.jupiter.api.extension.ExtendWith" };
    private static final String[] METHOD_ANNOTATIONS = { "org.junit.Test",
            "org.junit.platform.commons.annotation.Testable" };

    @Override
    public boolean match(MetadataReader metadataReader,
            MetadataReaderFactory metadataReaderFactory) throws IOException {
        //是否被 @TestComponent 及其父注解注释
        if (isTestConfiguration(metadataReader)) {return true;}
        //类上或类中方法上有没有 CLASS_ANNOTATIONS、METHOD_ANNOTATIONS 中的注解
        if (isTestClass(metadataReader)) {return true;}
        String enclosing = metadataReader.getClassMetadata().getEnclosingClassName();
        if (enclosing != null) {
            //递归内部类、父类
            if (match(metadataReaderFactory.getMetadataReader(enclosing),
                    metadataReaderFactory)) {
                return true;
            }
        }
```

```
            return false;
        }
    }
```

4.2 AutoConfigurationExcludeFilter 解析

- 主要移除被 @Configuration 修饰的 **自动配置类**

```
public class AutoConfigurationExcludeFilter implements TypeFilter, BeanClassLoaderAware
{
    @Override
    public boolean match(MetadataReader metadataReader,
            MetadataReaderFactory metadataReaderFactory) throws IOException {
        //如果被 @Configuration 注解，并且是 自动配置类就返回 true，即匹配成功
        //注：被 @Component 等注解并不匹配
        return isConfiguration(metadataReader) && isAutoConfiguration(metadataReader);
    }
}
```

5 @EnableAutoConfiguration 注解解析

- 作用：启用自动配置类

```
@AutoConfigurationPackage
//启用 AutoConfigurationImportSelector 配置类：扫描得到所有自动配置类
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
   String ENABLED_OVERRIDE_PROPERTY = "spring.boot.enableautoconfiguration";
   //定义不启用的 自动配置类
   Class<?>[] exclude() default {};
   //同上
   String[] excludeName() default {};
}
//这个注解主要是向容器中注册 AutoConfigurationPackages.Registrar 类用来存储自动配置包
@Import(AutoConfigurationPackages.Registrar.class)
public @interface AutoConfigurationPackage {}
//关键：这个类继承了 DeferredImportSelector 接口，所以是到最后才解析的！！
public class AutoConfigurationImportSelector implements DeferredImportSelector{
    @Override
    public String[] selectImports(AnnotationMetadata annotationMetadata) {
        if (!isEnabled(annotationMetadata)) {
            return NO_IMPORTS;
        }
        AutoConfigurationMetadata autoConfigurationMetadata =
AutoConfigurationMetadataLoader
            .loadMetadata(this.beanClassLoader);
        AutoConfigurationEntry autoConfigurationEntry = getAutoConfigurationEntry(
            autoConfigurationMetadata, annotationMetadata);
        return StringUtils.toStringArray(autoConfigurationEntry.getConfigurations());
    }
}
```

6 @...Test 注解

- Spring Boot 中文文档 对每个 @...Test 注解导入的自动配置类做了详细的说明

参考

spring-boot-2.1.3.RELEASE Spring Boot 中文文档