1.简介

在平时看源码或者很多配置类上面都会出现@Import注解,功能就是和Spring XML 里面 的一样. @Import注解是用来导入配置类或者一些需要前置加载的类.

在使用 Spring Boot 时，@Import 也是一个非常常见的注解，可以用来动态创建 Bean。

在 @Import 注解的属性中可以设置需要引入的类名，例如 @AutoConfigurationPackage 注解上的 @Import(AutoConfigurationPackages.Registrar.class)。根据该类的不同类型，Spring 容器针对 @Import 注解有以下四种处理方式：

1. 如果该类实现了 ImportSelector 接口，Spring 容器就会实例化该类，并且调用其 selectImports 方法；

2. 如果该类实现了 DeferredImportSelector 接口，则 Spring 容器也会实例化该类并调用其 selectImports方法。DeferredImportSelector 继承了 ImportSelector，区别在于 DeferredImportSelector 实例的 selectImports 方法调用时机晚于 ImportSelector 的实例，要等到 @Configuration 注解中相关的业务全部都处理完了才会调用；

3. 如果该类实现了 ImportBeanDefinitionRegistrar 接口，Spring 容器就会实例化该类，并且调用其 registerBeanDefinitions 方法；

4. 如果该类没有实现上述三种接口中的任何一个，Spring 容器就会直接实例化该类。

2.源码解析

2.1 导入配置的三种类型

@Import支持 三种方式

1.带有@Configuration的配置类(4.2 版本之前只可以导入配置类，4.2版本之后 也可以导入 普通类)

2.ImportSelector 的实现

3.ImportBeanDefinitionRegistrar 的实现

2.2 源码解释

```
/**
*功能类似XML 里面的 <import/> ,可以导入 @Configuration配置类，ImportSelector、
* ImportBeanDefinitionRegistrar 的实现,4.2 版本之后可以导入普通类(类似
AnnotationConfigApplicationContext#register
* )
* 可以在类级别声明或作为元注释声明
* 如需要引入XML或其他类型的文件，使用@ImportResource注解
*/@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
public @interface Import {

        /**
         * {@link Configuration}, {@link ImportSelector}, {@link
ImportBeanDefinitionRegistrar}
         * or regular component classes to import.
         */
        Class<?>[] value();

}
```

3、测试例子

3.1 导入普通类

1. 新建一个TestA

```
1  public class TestA {
2
3      public void fun(String str) {
4          System.out.println(str);
5      }
6
7      public void printName() {
8          System.out.println("类名 : " + Thread.currentThread().getStackTrace()[1].getClassName());
9      }
10 }
```

2. 新建一个ImportConfig,在类上面加上@Configuration，加上@Configuration是为了能让Spring 扫描到这个类，并且直接通过@Import引入TestA类

```
1  @Import({TestA.class})
2  @Configuration
3  public class ImportConfig {
4  }
```

复制

3.测试结果

TestA 是一个普通的类，现在可以被@Autowired注释然后调用，就直接说明已经被Spring 注入并管理了，普通的类都是需要先实例化

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = ApplicationMain.class)
public class ImportAnnotionTest {

    @Autowired
    TestA testA;

    @Test
    public void TestA() {
        testA.printName();
    }
}
```

打印:

```
类名 : com.test.importdemo.TestA
```

3.2 导入带有@Configuration的配置类

1. 新建TestB

```
@Configuration
public class TestB {
    public void fun(String str) {
        System.out.println(str);
    }

    public void printName() {
        System.out.println("类名 : " + Thread.currentThread().getStackTrace()[1].getClassName());
    }
}
```

2. 在ImportConfig.class里面直接引入TestB

```
@Import({TestA.class,TestB.class})
@Configuration
public class ImportConfig {
}
```

3.测试结果

TestB.class 的类上面已经有了@Configuration注解,本身就会配spring扫到并实例，@import引入带有@Configuration的配置文件，是需要先实例这个配置文件再进行相关操作

```
   @Autowired
    TestB testB;


    @Test
    public void TestB(){
        testB.printName();
    }
```

打印:

```
ImportAnnotionTest in 8.149 seconds (JVM running for 10.104)
类名 : com.test.importdemo.TestB
2019-01-31 14:12:05.737  INFO 23760 --- [       Thread-2]
```

3.3 通过ImportSelector 方式导入的类

1. 新建TestC.class

```
public class TestC {
    public void fun(String str) {
        System.out.println(str);
    }

    public void printName() {
        System.out.println("类名 : " + Thread.currentThread().getStackTrace()[1].getClassName());
    }
}
```

2.新建SelfImportSelector.class 实现ImportSelector 接口,注入TestC.class
//TODO ImportSelector 相关解释

```
public class SelfImportSelector implements ImportSelector {
    @Override
    public String[] selectImports(AnnotationMetadata importingClassMetadata) {
        return new String[]{"com.test.importdemo.TestC"};
    }
}
```

3.ImportConfig上面引入SelfImportSelector.class

```
@Import({TestA.class,TestB.class,SelfImportSelector.class})
@Configuration
public class ImportConfig {
}
```

4.测试结果

```
    @Autowired
    TestC testC;

    @Test
    public void TestC() {
        testC.printName();
    }
```

打印:

```
ImportAnnotionTest in 7.23 seconds (JVM running for 9.065)
类名 : com.test.importdemo.TestC
2019-01-31 14:23:15.330  INFO 1196 --- [
```

3.4 通过 ImportBeanDefinitionRegistrar 方式导入的类

1.新建TestD.class

```
public class TestD {
    public void fun(String str) {
        System.out.println(str);
    }

    public void printName() {
        System.out.println("类名 : " + Thread.currentThread().getStackTrace()[1].getClassName());
    }
}
```

2.新建SelfImportBeanDefinitionRegistrar.class,实现接口ImportBeanDefinitionRegistrar,注入TestD.class

```
public class SelfImportBeanDefinitionRegistrar implements ImportBeanDefinitionRegistrar {
    @Override
    public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata, BeanDefinitionRegistry regis
        RootBeanDefinition root = new RootBeanDefinition(TestD.class);
        registry.registerBeanDefinition("testD", root);
    }
}
```

3.ImportConfig类上加上导入SelfImportBeanDefinitionRegistrar.class

```
@Import({TestA.class,TestB.class,SelfImportSelector.class,
        SelfImportBeanDefinitionRegistrar.class})
@Configuration
public class ImportConfig {
}
```

4.测试结果

```
    @Autowired
    TestD testD;


    @Test
    public void TestD() {
        testD.printName();
    }
```

打印:

```
ImportAnnotionTest in 7.817 seconds (JVM running for 9.874)
类名 : com.test.importdemo.TestD
2019-01-31 14:30:05.781  INFO 23476 --- [
```

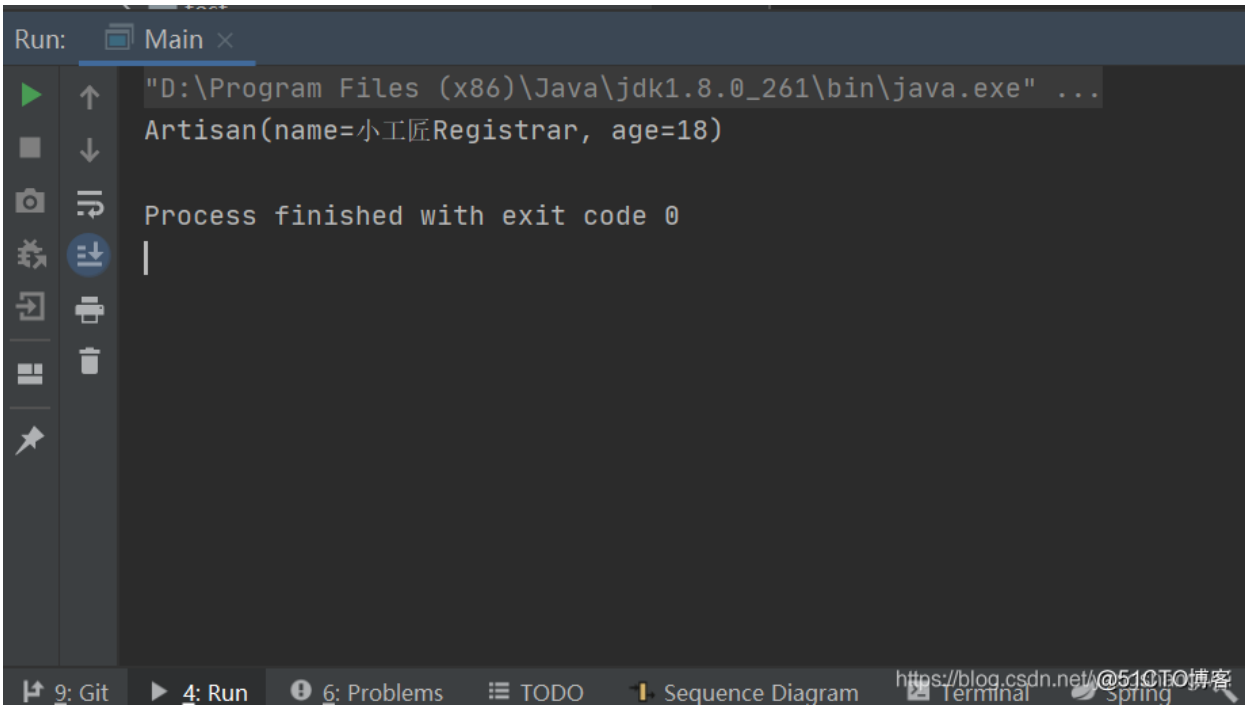
通过以上几种方法都是能成功注入Spring.

```
public class ArtisanRegistrar implements ImportBeanDefinitionRegistrar {
    @Override
    public void registerBeanDefinitions(AnnotationMetadata
importingClassMetadata,
        BeanDefinitionRegistry registry) {

        BeanDefinitionBuilder builder =
            BeanDefinitionBuilder.genericBeanDefinition(Artisan.class);
```

```
        builder.setScope(BeanDefinition.SCOPE_SINGLETON);
        builder.addPropertyValue("name", "小工匠Registrar");
        builder.addPropertyValue("age", "18");
        registry.registerBeanDefinition("artisan",
            builder.getBeanDefinition());
    }
}
```

测试



ImportBeanDefinitionRegistrar类似于ImportSelector用法，只不过这种用法能自定义化注册，往容器内注入一个BeanDefinition，然后BeanDeiniton在容器内转为一个实例bean。