配置类@Configuration

01、概述

配置类：在springboot中被@Configuration或者@SpringBootConfiguration标注的类称之为配置类。

02、作用&目的

在配置类可以定义很多@Bean的方法，可以让这些@Bean修饰的方式让spring框架加载到ioc容器中去。

03、那为什么会存在配置

- 方便你覆盖底层的配置类
- 让你去扩展的bean的一种机制。

04、一个springboot项目中的加载的bean有那些呢？

- 程序员自己编写的开发的bean ,比如加了：@Service，@Mapper，@Controller，@RestController，@Component，@Repository的类，都会加载到ioc容器中。

> 如果一个类加了注解就能够加载到*ioc*容器中去吗？
>
> 不能，需要*@ComponentScan*扫包才行。刚好*springboot*的注解是一个复合注解其中就包含了*@ComponentScan*注解，然后*springbooot*启动类启动会去扫包把这些加了注解的*bean*全部加*ioc*容器中

- Starter提供配置配置类+@Bean也会加载到ioc容器中。

05、思考为什么会存在配置类？

- 它其实就一种额外扩展和加载bean的一种机制。
- 可以方便进行额外的扩展和引用第三方的bean。如果没有这种机制，那么所以的中间件比如：mybatis,redis，jpa，kafka等这些初始化并且放入到ioc容器中必须全部要自己取编写。
- springboot为了解放程序开发人员，所以提供starter机制，而这些个starter机制里面的原理全部都是：配置类+@Bean
- 如果官方提供的满足不了，当然你可以自己取定义配置类和@Bean方法进行加载到ioc容器中，进行额外扩展。

06、定义一个配置类

1：准备一个接口：

```
package com.kuangstudy.service;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/24 22:13
 */
public interface IUserService {

    public void sayHello();
}

```

2：准备一个实现类

```
package com.kuangstudy.service;
```

```
2
3  import org.springframework.stereotype.Service;
4
5  /**
6   * @author 飞哥
7   * @Title: 学相伴出品
8   * @Description: 飞哥B站地址：
   https://space.bilibili.com/490711252
9   * 记得关注和三连哦！
10  * @Description: 我们有一个学习网站：https://www.kuangstudy.com
11  * @date 2021/12/24 22:14
12  */
13 public class UserServiceImpl implements IUserService {
14
15     @Override
16     public void sayHello() {
17         System.out.println("nihao");
18     }
19 }
20
```

3：给bean定义配置将其初始化

```
1  package com.kuangstudy.config;
2
3  import com.kuangstudy.service.IUserService;
4  import com.kuangstudy.service.UserServiceImpl;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.context.annotation.Configuration;
7
8  /**
9   * @author 飞哥
10  * @Title: 学相伴出品
11  * @Description: 飞哥B站地址：
   https://space.bilibili.com/490711252
12  * 记得关注和三连哦！
13  * @Description: 我们有一个学习网站：https://www.kuangstudy.com
14  * @date 2021/12/24 22:15
```

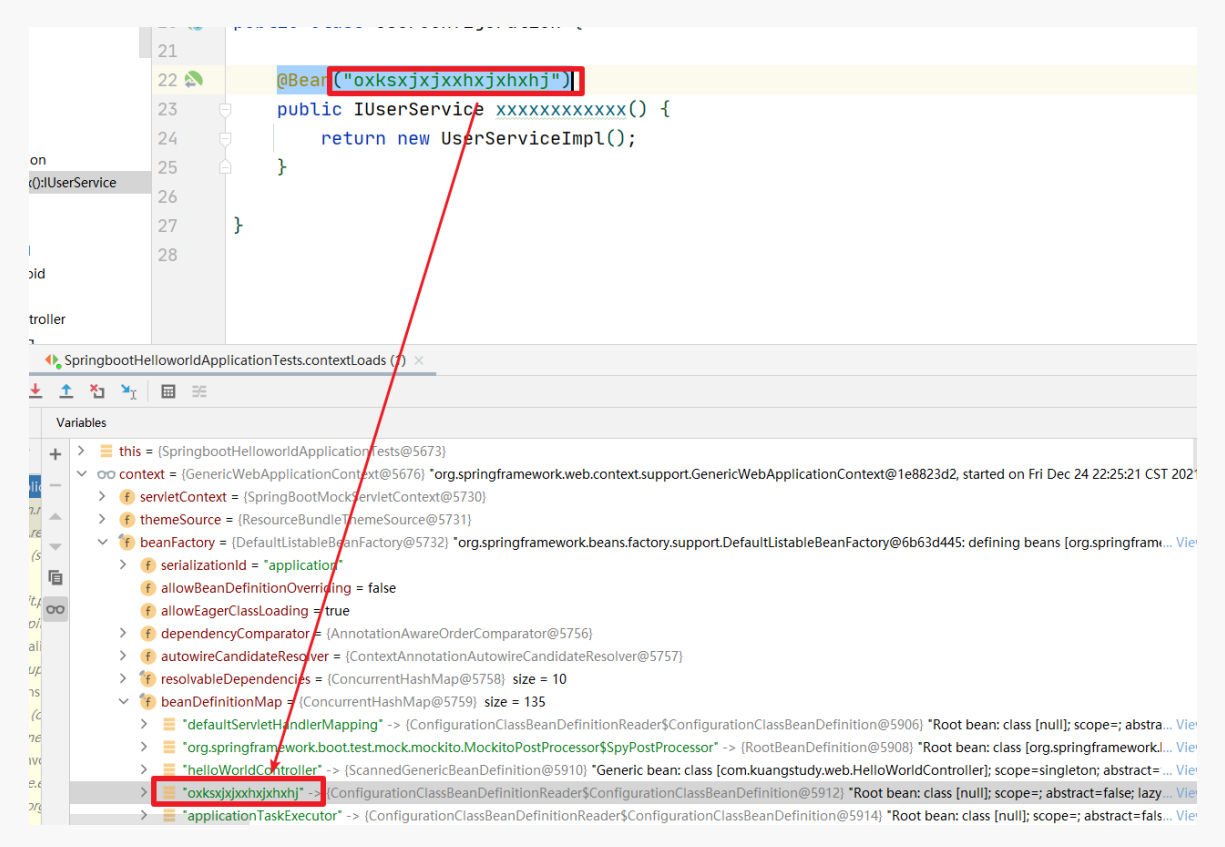
```
 */
@Configuration
public class UserConfiguration {

    @Bean
    public IUserService getUserService() {
        return new UserServiceImpl();
    }

}

```

- 注意事项1：@Bean必须定义在配置类中@Configuration或者加了spring注解（@Service，@Mapper，@Controller，@RestController，@Component，@Repository），如果没有@Bean不会有效。
- 注意事项2：如果是扫包的情况下：默认是以类名小写作为key进行映射。如果是@Bean，以方法名作为key进行容器映射。如果你在@Bean("oxksxjxjxxhxjxhxhj") 那么就用你指定的名字进行映射。



- 注意事项3：虽然@Bean可以结合spring的其他注解也可以加载到ioc容器中，但是还是推荐使用@Configuration,这样更加明确。因为它们黄金搭档。

05、条件注解

默认情况，假设没有条件注解，那么发生什么？因为如果bean指定@Configuration和@Bean是必须一定加载ioc容器中去的.

思考一个问题：springboot的starter机制中提供了100~200个配置类，这么的多配置类，我们不可能全不让ioc容器加载，因为项目可能之用到了redis，mybatis，那么另外100多个，应该不要加载才行。所以条件注解就专门来做过滤使用。满足配置就加载，不满足的就忽略。

```
/*
 * Copyright 2012-2021 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      https://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

```

```java
package org.springframework.boot.autoconfigure.data.redis;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnSingleCandidate;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisOperations;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;

/**
 * {@link EnableAutoConfiguration Auto-configuration} for Spring Data's Redis support.
 *
 * @author Dave Syer
 * @author Andy Wilkinson
 * @author Christian Dupuis
 * @author Christoph Strobl
 * @author Phillip Webb
 * @author Eddú Meléndez
```

```
41  * @author Stephane Nicoll
42  * @author Marco Aust
43  * @author Mark Paluch
44  * @since 1.0.0
45  */
46 @Configuration(proxyBeanMethods = false)
47 @ConditionalOnClass(RedisOperations.class)
48 @EnableConfigurationProperties(RedisProperties.class)
49 @Import({ LettuceConnectionConfiguration.class,
    JedisConnectionConfiguration.class })
50 public class RedisAutoConfiguration {
51
52     @Bean
53     @ConditionalOnMissingBean(name = "redisTemplate")
54
    @ConditionalOnSingleCandidate(RedisConnectionFactory.class)
55     public RedisTemplate<Object, Object>
    redisTemplate(RedisConnectionFactory redisConnectionFactory)
    {
56         RedisTemplate<Object, Object> template = new
    RedisTemplate<>();
57
    template.setConnectionFactory(redisConnectionFactory);
58         return template;
59     }
60
61     @Bean
62     @ConditionalOnMissingBean
63
    @ConditionalOnSingleCandidate(RedisConnectionFactory.class)
64     public StringRedisTemplate
    stringRedisTemplate(RedisConnectionFactory
    redisConnectionFactory) {
65         return new
    StringRedisTemplate(redisConnectionFactory);
66     }
67
68 }
```

```
69
```

07、相同类型的怎么办？

```
1  package com.kuangstudy.config;
2
3  import com.kuangstudy.service.IUserService;
4  import com.kuangstudy.service.UserServiceImpl;
5  import
   org.springframework.beans.factory.annotation.Qualifier;
6  import
   org.springframework.boot.autoconfigure.condition.ConditionalO
   nBean;
7  import
   org.springframework.boot.autoconfigure.condition.ConditionalO
   nClass;
8  import
   org.springframework.boot.autoconfigure.condition.ConditionalO
   nMissingBean;
9  import org.springframework.context.annotation.Bean;
10 import org.springframework.context.annotation.Configuration;
11 import org.springframework.context.annotation.Primary;
12 import org.springframework.stereotype.Component;
13 import org.springframework.stereotype.Controller;
14 import org.springframework.stereotype.Service;
15
16 /**
17  * @author 飞哥
18  * @Title: 学相伴出品
19  * @Description: 飞哥B站地址：
   https://space.bilibili.com/490711252
20  * 记得关注和三连哦!
21  * @Description: 我们有一个学习网站：https://www.kuangstudy.com
22  * @date 2021/12/24 22:15
23  */
```

```
@Configuration
public class UserConfiguration {


    @Bean
    public IUserService getUserService1() {

 System.out.println("22222222222222222222222222222");
        return new UserServiceImpl();
    }

    @Bean
    public IUserService getUserService2() {
        System.out.println("11111111111111111111111111");
        return new UserServiceImpl();
    }

}

```

上面的代码：出现一个接口有两个子类，如果初始化就会报错：

解决方案1：使用@Qualifier

```
@Autowired
@Qualifier("getUserService1")
private IUserService userService;
```

解决方案2：名字匹配

```
@Autowired
private IUserService getUserService1;
@Autowired
private IUserService getUserService2;
```

解决方案3：@Primary

```

```

```java
package com.kuangstudy.config;

import com.kuangstudy.service.IUserService;
import com.kuangstudy.service.UserServiceImpl;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.autoconfigure.condition.ConditionalOnBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Controller;
import org.springframework.stereotype.Service;

/**
 * @author 飞哥
 * @Title: 学相伴出品
 * @Description: 飞哥B站地址：https://space.bilibili.com/490711252
 * 记得关注和三连哦!
 * @Description: 我们有一个学习网站：https://www.kuangstudy.com
 * @date 2021/12/24 22:15
 */
@Configuration
public class UserConfiguration {


    @Bean
    @Primary
    public IUserService getUserService1() {
```

```
31
    System.out.println("22222222222222222222222222222222");
32          return new UserServiceImpl();
33      }
34
35      @Bean
36      public IUserService getUserService2() {
37          System.out.println("11111111111111111111111111");
38          return new UserServiceImpl();
39      }
40
41  }
```

08、相同名字的bean怎么办？

```
1  spring:
2    main:
3      allow-bean-definition-overriding: true
```

上面配置是告诉，如果在项目中出现名字相同的bean，要注入的时候，允许容器map的key 允许覆盖。默认是false，不允许之间报错。在项目不要打开，因为名字相同是不行。你应该是你项目重复初始化了。应该去删掉一个，或者更换名字，这才正确的打开方式。否则你项目中会造成误解和无厘头的问题。

09、总结

- 为什么springboot的starter都才用配置类来开发呢？
- 为什么不是扫包+@Service，@Mapper，@Controller，@RestController，@Component，@Repository呢？

因为组件redis,jpa它包名是不可能统一，redis可能叫：com.redis.xxx 比如：jpa可能是：org.jpa.xxx 这个你扫包肯定不可取。所以必须要找一种能够让其不用配置和修改，也能够加载ioc容器机制，当然就是：@Configuration+@Bean。它们的组合就可以脱离扫包+注解的机制，可以变更加灵活和随心所欲。这也就是为什么说配置类是让你去扩展的和去定义，去加载别人的bean的一种机制。反思：如果没有这种机制，你思考这些第三方的如何把的bean加载到ioc为我们项目所用呢？

> 配置类+@*Bean*就是一种：把*bean*加载到*ioc*容器的一种机制。