1.

@Controller@Service("iCategoryService")

都包含@Component注解，所以会被自动扫描扫描到

放在类的上面

示例：

@Component  
public class Knight {  
 public void speak() {  
 System.*out*.println("knight speak");  
 }  
}

扩展：

1. 别名：为bean指定别名

示例:

@Component("main\_battle")  
public class Battle {  
  
 @Autowired  
 private Knight knight;  
  
 public void battle() {  
 knight.speak();  
 }  
}

[2.@ComponentScan](mailto:2.@ComponentScan)默认会扫描同包下的@Component注解

默认属性会扫描参数包下的@Component注解

放在配置类的上面

示例：

@Configuration  
@ComponentScan("com.springjiemi.test")  
public class ScanConfig {  
}

扩展：

1. basePackages = “com.springjiemi.test” //指定要扫描的包，注意是复数形式，故可以添加多个包

示例:

@Configuration  
@ComponentScan(basePackages = {"com.springjiemi.test", "com.springjiemi.dao"})  
public class ScanConfig {  
}

1. basePackageClasses = DaoForScan.class //指定要扫描的包，通过包里面所包含的某个类或接口

示例：

@Configuration  
@ComponentScan(basePackageClasses = DaoForScan.class)  
public class ScanConfig {  
}

3. @Configuration

用于定义配置文件，代替web.xml

放在配置类的上面

示例：

@Configuration  
@ComponentScan("com.springjiemi.test")  
public class ScanConfig {  
}

4.@Autowired注解用于自动装配bean，使用此注解的类需要类上面有@Component注解

示例：

@Component  
public class Battle {  
   
 @Autowired  
 private Knight knight;  
   
 public void battle() {  
 knight.speak();  
 }  
}

扩展：

1. 此注解也可以用于其他方法上，spring会尝试满足所声明的依赖。
2. 可以使用required = false 属性设置这个依赖不是必须的（但有可能抛出NPE异常，谨慎）

示例:

@Autowired(required = false)  
private Ghost ghost;

5．@bean 和 @Component类似，都是声明为bean，但@bean是用在javaConfig类的方法上

示例1：

@Configuration  
public class ScanConfig {  
 @Bean  
 public Ghost ghost() {  
 return new Ghost();  
 }  
}

示例2：添加参数，spring会自动装配参数

@Configuration  
public class ScanConfig {  
 @Bean  
 public Ghost ghost(LittleRedHat littleRedHat) {  
 return new Ghost(littleRedHat);  
 }  
}

扩展：

1. name=“” //设置bean的名字
2. javaConfig中注册的Bean的高级应用:随机生成bean

示例：

@Configuration  
public class ScanConfig {  
 @Bean  
 public Human human() {  
 int random = (int) Math.*floor*(Math.*random*() \* 2);  
 if (random == 0) {  
 return new Knight();  
 } else {  
 return new LittleRedHat();  
 }  
 }  
}

6，@Import(xxxConfig.class) //引入其他配置文件

@Configuration  
@ComponentScan("com.springjiemi.test")  
@Import(BeanConfig.class)  
public class ScanConfig {  
}

扩展：

1. 使用更高级别的Config组合低级别的Config

示例：

@Configuration  
@Import({BeanConfig.class, ScanConfig.class})  
public class Config {  
}

1. @ImportResource(“classpath:xxxx.xml”) //引入基于xml的配置文件

示例：

@Configuration  
@Import(ScanConfig.class)  
@ImportResource("classpath:bean-config.xml")  
public class Config {  
}

7. 通过使用@MapperScan可以指定要扫描的Mapper类的包的路径，扫描并添加上@Mapper注解

@Mapper注解的的作用

1:为了把mapper这个DAO交給Spring管理

2:为了不再写mapper映射文件

3:为了给mapper接口 自动根据一个添加@Mapper注解的接口生成一个实现类 （自动生成的实现类中，自动帮我们添加了@Component注解。因此我们在其他地方可以通过@AutoWired的方式自动注入。）

[8.@Profile(“dev”)](mailto:8.@Profile() 只有在对于dev profile被激活时才会创建

可以用在javaConfig类上或者方法上

示例：

@Configuration  
@MapperScan("com.springjiemi.dao")  
@Profile("dev")  
public class DataSourceConfig {

有spring.profiles.active和spring.profiles.default两个属性控制profile的状态，先查找spring.profiles.active如果找不到再查找spring.profiles.default

可以：1）作为DispatcherServlet的初始化参数

2）作为Web应用的上下文参数

3）。。。

如在web.xml文件里面

1）为上下文设置默认profile

<context-param>  
 <param-name>spring.profiles.default</param-name>  
 <param-value>dev</param-value>  
</context-param>

2）为Servlet设置默认profile

<!--配置SpringMVC-->  
<!--\*.do的请求都会被SpringMVC拦截-->  
<servlet>  
 <servlet-name>dispatcher</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <init-param>  
 <param-name>spring.profiles.default</param-name>  
 <param-value>dev</param-value>  
 </init-param>  
 <load-on-startup>1</load-on-startup><!--如果大于0Servlet启动时会调用servlet-class的init方法-->  
</servlet>  
<servlet-mapping>  
 <servlet-name>dispatcher</servlet-name>  
 <url-pattern>\*.do</url-pattern>  
</servlet-mapping>

[9.@Conditional(GhostCondition.class)](mailto:9.@Conditional(GhostCondition.class)) //指定创建bean的条件，满足时才会创建该bean

示例：

@Bean  
@Conditional(GhostCondition.class)  
public Ghost ghost(LittleRedHat littleRedHat) {  
 return new Ghost(littleRedHat);  
}

其中GhostCondition.class类要实现Condition接口

public class GhostCondition implements Condition{  
 @Override  
 public boolean matches(ConditionContext conditionContext, AnnotatedTypeMetadata annotatedTypeMetadata) {  
 return true;  
 }  
}

其中：

ConditionContext可以提供相应的信息，如：检查bean的定义，检查bean是否存在，勘察bean的属性，检查环境变量是否存在以及它的值是什么

AnnotatedTypeMetadata可以检查带有@Bean注解的方法上还有什么其他的注解

[10.@Qualifier(“bean](mailto:10.@Qualifier() id”) //指定加载哪一个bean（如果出现歧义时）

示例：

@Autowired  
@Qualifier("knight")  
private Human human;

扩展：

1. 可以在@Bean 或 @Component注解上面用@Qualifier注解为bean指定限定符，这也在使用@Qualifier指定bean时就可以使用这个限定符指定，而不直接依赖与beanId

如：

@Component  
@Qualifier("red")  
public class LittleRedHat implements Human {  
 @Override  
 public void speak() {  
 System.*out*.println("apple apple ");  
 }  
}

public class Ghost {  
 @Autowired  
 @Qualifier("red")  
 private Human human;  
}

[11.@Scope()](mailto:11.@Scope()) //设置bean的作用域，可以用在@Bean或@Component注解上

示例：

@Component  
@Scope(ConfigurableBeanFactory.*SCOPE\_PROTOTYPE*)  
public class Knight implements Human {  
 @Override  
 public void speak() {  
 System.*out*.println("knight speak");  
 }  
}

有4种作用域

1）singleton （单一实例）

2）prototype 每次注入或者通过spring应用上下文获取的时候，都会创建一个新的bean实例

3）request 每个请求都会创建一个bean实例

4）session 每个session都会创建一个bean实例

[12.@AspectJ](mailto:12.@AspectJ)注解

有前置通知（before），后置通知（after）（方法完成后通知），返回通知（after-returning）（方法返回后通知），异常通知（after-throwing），环绕通知（around）（调用之前和调后之后通知）

示例：

@Aspect  
public class PoPo {  
 @Pointcut("execution(\* com.springjiemi.test.Ghost.dodo(..))") //定义切点  
 public void popo() {};  
 @After("execution(\* com.springjiemi.test.Ghost.dodo(..))") //为Ghost类的dodo方法加上这个切面  
 public void pai() {  
 System.*out*.println("ssssssssssssssssssssssssspai");  
 }  
 @Before("popo()")  
 public void d() {  
 System.*out*.println("aaaaaaaaaaaaaaaaasss");  
 }  
}

配置文件

@Configuration  
@EnableAspectJAutoProxy  
public class AspectConfig {  
 @Bean  
 public PoPo poPo() {  
 return new PoPo();  
 }  
}

13@DeclareParents注解 //为类引入新的方法

示例：

JavaConfig类

@Configuration  
@EnableAspectJAutoProxy(proxyTargetClass = true) //开启Aspect自动代理（默认为false）  
public class AspectConfig {  
}

引入功能接口类

public interface Encoreable {  
 void encore();  
}

引入功能实现类

@Component  
public class EncoreableImpl implements Encoreable {  
 @Override  
 public void encore() {  
 System.*out*.println("eeeeeeeeeeeeeeeeccccoooorrrree");  
 }  
}

引入方法装配类

@Aspect  
@Component  
public class EncoreableIntroducer {  
 @DeclareParents(  
 value = "com.springjiemi.test.inte.Human+", //指定需要代理的对象  
 defaultImpl = EncoreableImpl.class) //指定引入功能的实现类  
 public static Encoreable *encoreable*; //指定引入功能的接口类  
}

测试类

public static void main(String[] args) throws IOException {  
 ApplicationContext context = new AnnotationConfigApplicationContext(com.springjiemi.config.Config.class);  
LittleRedHat littleRedHat=context.getBean(LittleRedHat.class);  
 Encoreable encoreable = (Encoreable) littleRedHat;  
 encoreable.encore();  
}