Spring Boot介绍

springboot思想：

约定优于配置

什么是spring boot?

是 Spring 开源组织下的子项目，是 Spring 组件一站式解决方案，主要是简化了使用 Spring 的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

spring优缺点：

优点：

1. 容易上手，提高开发效率。
2. 不需要XML配置。
3. 提供了一系列大型项目通用的非业务性功能，例如：内嵌服务器、安全管理、运行数据监控、运行状况检查和外部化配置等。
4. 版本依赖集中管理，避免大量的 Maven 导入和各种版本冲突。
5. 通过一些相对简单的方法，通过依赖注入和面向切面编程，**用简单的java对象实现了EJB功能。**

缺点：

XML配置复杂，依赖管理耗时耗力，一旦选错版本，不兼容会严重阻碍项目开发进度。

springboot解决问题：

有效解决配置和业务问题思维切换，全身心投入到逻辑业务代码编写中。

起步依赖：

把具备某种功能的坐标打包到一起，并提供默认的功能。

自动配置：

1.会自动将一些配置类（指使用@Configuration的类，@Configuration: 指明当前类是一个配置类来替代之前的Spring配置文件）的bean注册进ioc容器，需要的地方使用@autowired或@rescource使用它；

2.只要引入想用功能的包，相关配置不用管，springboot会自动注入这些配置bean，直接使用即可。

springboot单元测试搭建：

1.导入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

2.编写单元测试类和测试方法

```java
@RunWith(SpringRunner.class) //测试启动器，加载Spring Boot测试注解
@SpringBootTest //标记为Spring Boot单元测试类，加载项目ApplicationContext上下文环境
class SpringbootDemoApplicationTests {
    @Autowired
    private DemoController demoController;

    @Test
    void contextLoads() {
        String demo = demoController.demo();
        System.out.println(demo);
    }
}
```

springboot热部署搭建：

devtools方式:

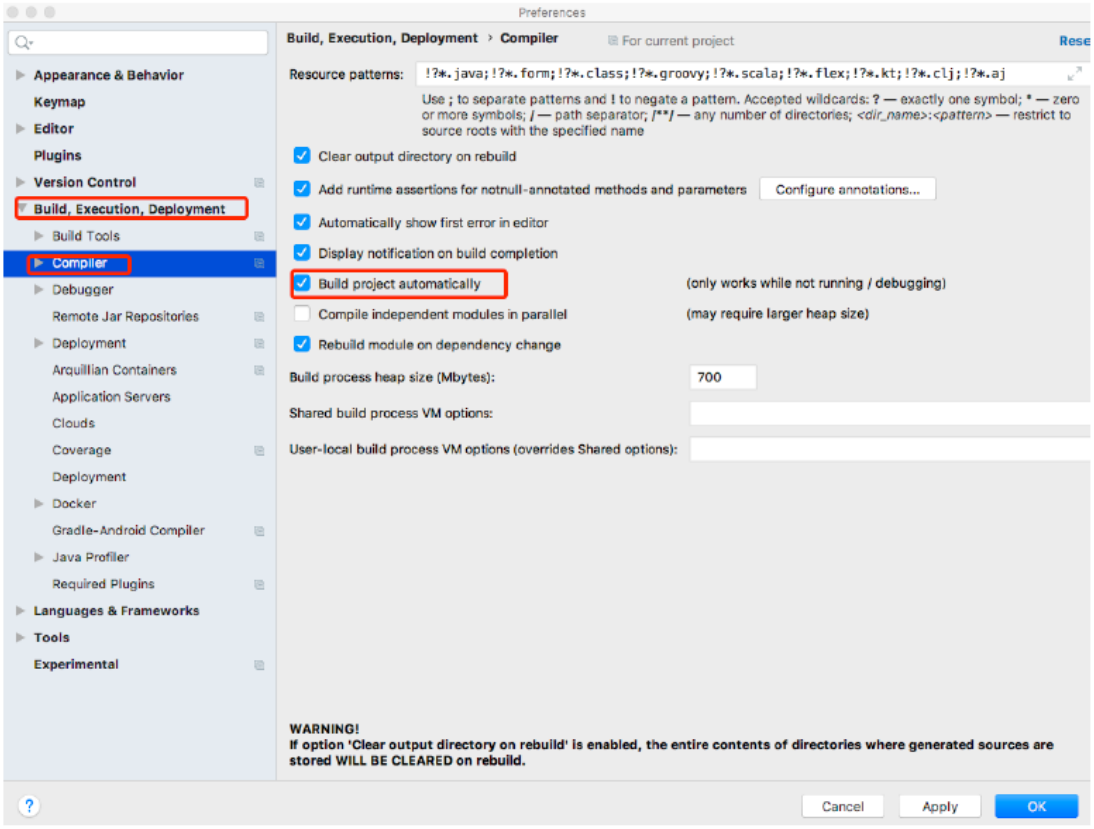
1.导入依赖

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

2.idea设置

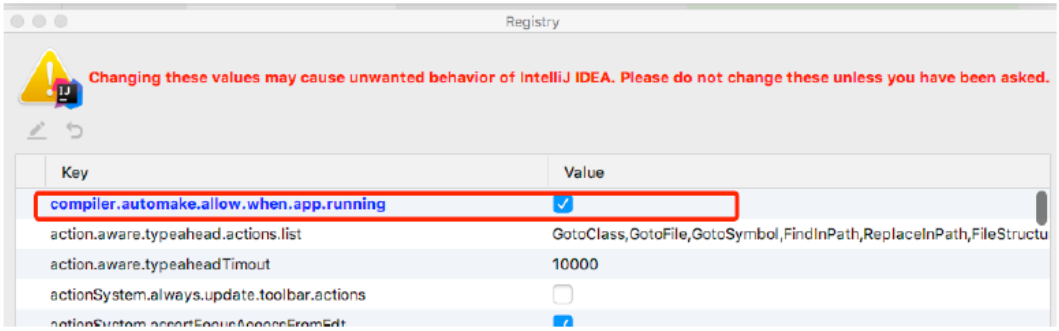
2. IDEA工具热部署设置

选择IDEA工具界面的【File】->【Settings】选项，打开Compiler面板设置页面



选择Build下的Compiler选项，在右侧勾选“Build project automatically”选项将项目设置为自动编译，单击【Apply】→【OK】按钮保存设置

在项目任意页面中使用组合快捷键“Ctrl+Shift+Alt+/”打开Maintenance选项框，选中并打开Registry页面，具体如图1-17所示



列表中找到“compiler.automake.allow.when.app.running”，将该选项后的Value值勾选，用于指定IDEA工具在程序运行过程中自动编译，最后单击【Close】按钮完成设置

3.添加properties文件配置

```
spring:
  devtools:
    restart:
      enabled: true  #设置开启热部署
      additional-paths: src/main/java #重启目录
      exclude: WEB-INF/**
  freemarker:
    cache: false    #页面不加载缓存，修改即时生效

```

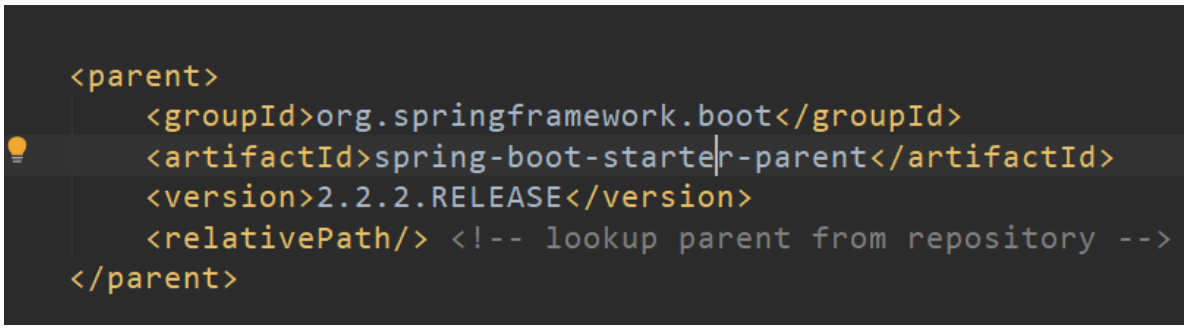
使用springloaded方式：

1. 使用springloaded依赖
2. 配置pom.xml文件，使用mvn spring-boot:run启动

配置JVM启动参数方式：

1. 本地下载springloaded包，配置jvm参数-javaagent:<jar包地址> -noverify

springboot配置讲解：



```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.2.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
```

```
    //上图是入口
      <resource>
       <filtering>true</filtering>
       <directory>${basedir}/src/main/resources</directory>
       <includes>
         <include>**/application*.yml</include>
         <include>**/application*.yaml</include>
         <include>**/application*.properties</include>
       </includes>
      </resource>
```

从上图中看出，yml文件是先加载的，如果**相同的配置**存在于两个文件中，最后会使用properties中的配置。

application.properties配置：

该文件可以是系统属性、环境变量、命令行参数等信息，也可以是自定义配置文件名称和位置。

```
server.port=8081
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.config.additional-location=
spring.config.location=
spring.config.name=application
```

demo:将配置文件属性注入到Person实体类对应属性中：

```
public class Pet {
private String type;
private String name;
//省略get和set方法
}
```

```java
@Component
@ConfigurationProperties(prefix = "person")//将配置文件中以person开头的属性通过set方法注入到实体类相应的属性中
public class Person {
private int id;
private String name;
private List hobby;
private String[] family;
private Map map;
private Pet pet;
  //省略get和set方法
}
```

以上的properties文件改造：

```yaml
person:
  id: 1
  name: lucy
  hobby: [吃饭,睡觉,打豆豆]
  family: [father,mother]
  map: {k1: v1,k2: v2}
  pet: {type: dog,name: 旺财}
```

导入依赖：

```xml
//有了该依赖，写properties文件时会有提示
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
    <optional>true</optional>
</dependency>
```

application.yaml配置：

缩进式（两种写法）：

```yaml
person:
  hobby:
    - play
    - read
    - sleep
```

或

```yaml
person:
  hobby:
    play,
    read,
    sleep
```

数组：

```yaml
person:
  hobby: [play,read,sleep]
```

Map：

```yaml
//缩进式
person:
  map:
    k1: v1
    k2: v2
```

```yaml
//行内式
person:
  map: {k1: v1,k2: v2}
```

@Value讲解：

配置文件属性值注入（使得上面配置文件生效）

```
public class Student {

    @Value("3")
    private int id;//相当于id=3,一般不会这样用

    @Value("${person.name}")
    private String name;//将properties或ymal文件的值注入进来，不需要set方法。
                        //对于包含Map、对象以及ymal文件格式的行内式写法的配置文件的属性注入都不支持，如果赋值会出现错误。
}
```

自定义配置properties文件：

需加@PropertySource

```
test.properties//不加注解是扫描不到的

test.id=110
test.name=test
```

```
@Component
@PropertySource("classpath:test.properties")  //配置自定义配置文件的名称及位置
@ConfigurationProperties(prefix = "test")
public class MyProperties {
    private int id;
    private String name;
}
```

使用@Configuration编写自定义配置类：

```
@Configuration //标明该类为配置类
public class MyConfig {

    @Bean(name = "iservice")   //将返回值对象作为组件添加到spring容器中，标识id默认是方法名
    public MyService myService() {
        return new MyService();
    }
}
```

测试：

```
/*
     @Configuration进行测试
    */

   @Autowired
   private ApplicationContext context;

   @Test
   void iocTest() {
       System.out.println(context.containsBean("iservice"));
   }
```

properties配置文件随机值设置和参数引用：

随机值设置：

```
my.secret=${random.value}          // 配置随机数
my.number=${random.int}            // 配置随机整数
my.bignumber=${random.long}       // 配置随机long类型数
my.uuid=${random.uuid}             //配置uuid类型数
my.number.less.than.ten=${random.int(10)}    // 配置小于10的随机整数
my.number.in.range=${random.int[1024,65536]} //配置范围在[1024,65536]之间的随机整数
```

参数引用:（省去多处修改的麻烦）

```
app.name=MyApp
app.description=${app.name} is a Spring Boot application //${app.name}拿到值：MyApp
```

springboot注解：

参考springboot注解.md

springboot解决中文乱码：

```
@RequestMapping(produces = "application/json; charset=utf-8")
```

或

```
spring.http.encoding.force-response=true #设置响应为utf-8
```

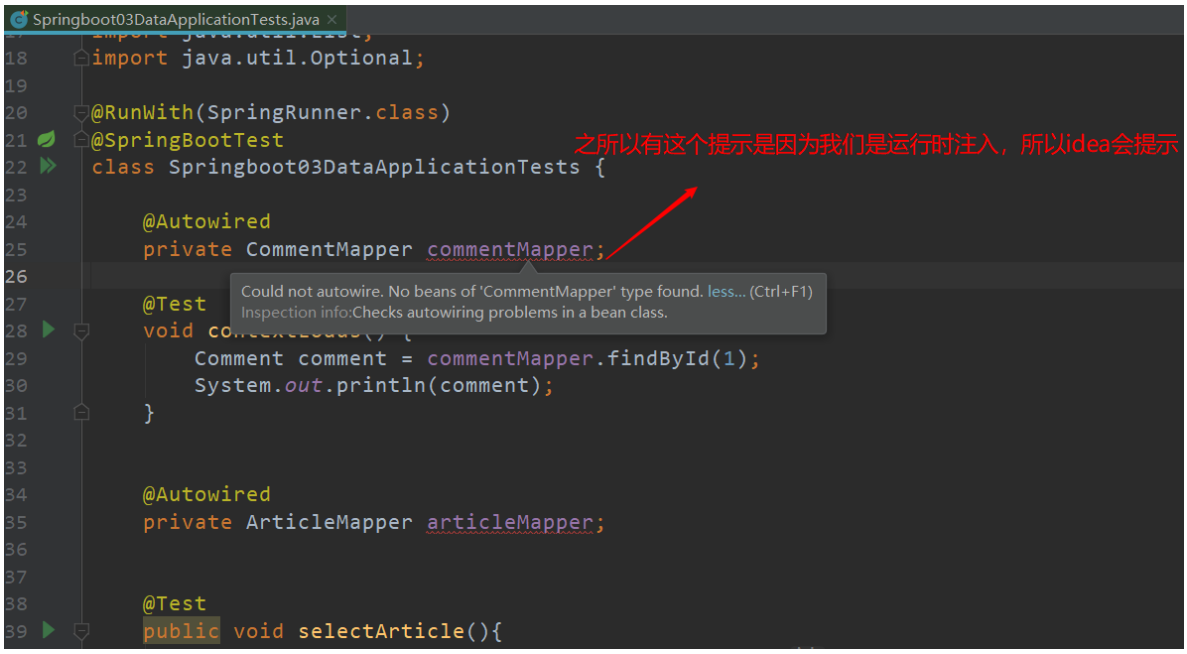
springboot解决的问题：

起步依赖：把具备某种功能的坐标打包到一起，并提供默认的功能。

自动配置：自动将一些配置类的bean注册进ioc容器，使用@autowired或@rescource使用它。

只要引入想用功能的包，相关配置不用管，springboot会自动注入这些配置bean，直接使用即可。

扩展



注解使用到的设计模式：

代理模式

自定义注解的底层：

先定义注解，然后扫描使用到注解的类，通过代理+反射执行逻辑代码。

使用过什么springboot组件

SpringBoot-Dubbo、SpringBoot-MyBatis、SpringBoot-Redis、SpringBoot-Redisson

SpringBoot-ActiveMQ　activemq-all v5.15.5　简单实例(activemq-demo)、整合Redis(activemq-redis)
SpringBoot-Admin spring-cloud Finchley.SR2 Spring Boot Admin是一个开源社区项目，用于管理和监控SpringBoot应用程序
SpringBoot-Cache　spring-boot-starter-cache　集成Caffeine的两种方式
SpringBoot-Chart　jfreechart v1.0.13　JFreeChart是JAVA平台上的一个开放的图表绘制类库
SpringBoot-Date-Jpa　spring-boot-starter-data-jpa　整合Jpa实现简单的增、删、改、查
SpringBoot-Docker　—　Docker中部署SpringBoot项目
SpringBoot-Dubbo com.alibaba.boot 0.2.0　Apache Dubbo（孵化）是一个由阿里巴巴开源的基于Java的高性能RPC框架
SpringBoot-Elasticsearch　Elasticsearch-5.5.0 集成Elasticsearch的简单实例
SpringBoot-Excel　poi-ooxml. v3.9　集成POI对excel导入导出的简单实例
SpringBoot-Mail jodd.mail v3.7.1 集成jodd发送邮件
SpringBoot-MongoDB　spring-boot-starter-data-mongodb 集成mongodb的简单实例
SpringBoot-MyBatis　mybatis.spring.boot v1.3.0 集成mybatis的简单实例
SpringBoot-MyBatisPlus　mybatis-plus-boot v3.0.1　代码生成器、多数据源配置、CRUD
SpringBoot-Quartz quartz v2.2.1　集成Quartz实现动态配置定时任务，支持mysql读库
SpringBoot-RabbitMQ　spring-boot-starter-amqp　集成RabbitMQ的简单实例

SpringBoot-Redis　jedis v2.5.0 集成jedis的简单实例
SpringBoot-Redisson redisson v3.12.3 集成redisson分布式锁的简单实例
SpringBoot-Shiro　shiro-core v1.2.3 集成Shiro实现权限验证的简单实例
SpringBoot-SSO — 结合redis实现一个简单单点登录实例
SpringBoot-Storm storm-core v1.1.1 简单实例、整合MySQL
SpringBoot-Thymeleaf spring-boot-starter-thymeleaf 集成Thymeleaf的简单实例
SpringBoot-Utils — 常用的时间、数子、数据处理工具类
SpringBoot-Webflux spring-boot-starter-webflux 集成Webflux的简单实例
SpringBoot-WebSocket spring-boot-starter-websocket 简单实例、webSocket实现聊天室

面试题

在使用Springboot遇到了什么问题？为什么推出Springboot,使用Springboot的好处?

1. **@Test和类名相同:**

2. **启动项目的时候报错:**

```
1.Error starting ApplicationContext.
To display the auto-configuration report re-run your application with 'debug'
enabled.
```

解决方法:

在yml配置文件中加入debug: true,因为默认的话是false

3. **在集成mybatis时mapper包中的类没被扫描:**

```
org.Springframework.beans.factory.NoSuchBeanDefinitionException:
 No qualifying bean of type 'com.app.mapper.UserMapper' available:
expected at least 1 bean which qualifies as autowire candidate. Dependency
annotations: {}
```

解决方法:

在Springboot的启动类中加入@MapperScan("mapper类的路径")
或者直接在Mapper类上面添加注解@Mapper,建议使用上面那种，不然每个mapper加个注解也挺麻烦的

4. **报以下错误：**

```
### Error querying database.  Cause: java.lang.IllegalArgumentException: invalid comparison: java.util.ArrayList and java.lang.String
### Cause: java.lang.IllegalArgumentException: invalid comparison: java.util.ArrayList and java.lang.String
```

原因：

这是一个根据list集合的查找数据的 sql，在接收list的时候加了判断 list ！ = ‘ ’ “”，引起了集合与Stirng类型的比较，故报错

```
<if test="list != null and list != '' ">
     AND roo_id IN
    <foreach collection="list" item="id" index="index" open="(" close=")"
separator=",">
       #{id}
    </foreach>
</if>
```

解决：

```
<if test="list != null and list.size > 0 "> //改为list.size > 0
     AND roo_id IN
    <foreach collection="list" item="id" index="index" open="(" close=")"
separator=",">
       #{id}
    </foreach>
</if>
```

5. **用mybatis查询时报错：**

```
org.mybatis.Spring.MyBatisSystemException:
nested exception is org.apache.ibatis.binding.BindingException:
Parameter 'user_type' not found. Available parameters are [2, 1, 0, param1,
param2, param3]
```

原因：@Param注解缺失，当只有一个参数时，Mapper接口中可以不使用

```
public User getUser(String name);
```

有多个参数时就必须使用

```
public User getUser(@Param("name") String name,@Param("password") String
password);
```

Spring Boot 中 “约定优于配置“的具体产品体现在哪里。

Spring Boot Starter、Spring Boot Jpa 都是“约定优于配置“的一种体现。
都是通过“约定优于配置“的设计思路来设计的，Spring Boot Starter 在启动的过程中会根据约定的信息对资源进行初始化；Spring Boot Jpa 通过约定的方式来自动生成 Sql ，避免大量无效代码编写。

Spring Boot 中如何实现定时任务 ？

定时任务也是一个常见的需求，Spring Boot 中对于定时任务的支持主要还是来自 Spring 框架。

在 Spring Boot 中使用定时任务主要有两种不同的方式，一个就是使用 Spring 中的 @Scheduled 注解，另一个则是使用第三方框架 Quartz。

使用 Spring 中的 @Scheduled 的方式主要通过 @Scheduled 注解来实现。

使用 Quartz ，则按照 Quartz 的方式，定义 Job 和 Trigger 即可。

Spring-boot-starter-parent 有什么用 ?

1、定义了 Java 编译版本为 1.8 。
2、使用 UTF-8 格式编码。
3、继承自 Spring-boot-dependencies，这个里边定义了依赖的版本，也正是因为继承了这个依赖，所以我们在写依赖时才不需要写版本号。
4、执行打包操作的配置。
5、自动化的资源过滤。
6、自动化的插件配置。
7、针对 application.properties 和 application.yml 的资源过滤，包括通过 profile 定义的不同环境的配置文件，例如 application-dev.properties 和 application-dev.yml。

Spring Boot 是否可以使用 XML 配置 ?

Spring Boot 推荐使用 Java 配置而非 XML 配置，但是 Spring Boot 中也可以使用 XML 配置，通过 @ImportResource 注解可以引入一个 XML 配置。

Spring Boot 打成的 jar 和普通的 jar 有什么区别 ?

Spring Boot 项目最终打包成的 jar 是可执行 jar ，这种 jar 可以直接通过 java -jar xxx.jar 命令来运行，这种 jar 不可以作为普通的 jar 被其他项目依赖，即使依赖了也无法使用其中的类。

Spring Boot 的 jar 无法被其他项目依赖，主要还是他和普通 jar 的结构不同。普通的 jar 包，解压后直接就是包名，包里就是我们的代码，而 Spring Boot 打包成的可执行 jar 解压后，在 \BOOT-INF\classes 目录下才是我们的代码，因此无法被直接引用。如果非要引用，可以在 pom.xml 文件中增加配置，将 Spring Boot 项目打包成两个 jar ，一个可执行，一个可引用。

application和bootstrap怎么加载

- 使用的区别：

  bootstrap用于应用程序上下文的引导阶段，可以理解成系统级别的参数配置，这些参数一般是不会变动的。

  application用于定义应用级别的，搭配 spring-cloud-config 使用 application.yml 里面定义的文件可以实现实时更新

- 加载顺序：

  在不指定要被加载文件时，默认的加载顺序：**由里向外加载**，所以**最外层**的**最后**被加载，会覆盖**里层**的属性

  若application.yml 和bootStrap.yml 在同一目录下，则bootStrap.yml 的加载顺序要高于application.yml

@Component, @Repository, @Service的区别

@Component是通用注解，其他三个注解是这个注解的拓展，并且具有了特定的功能。

@Controller：进行前端请求的处理，转发，重定向。包括调用Service层的方法
@Service：处理业务逻辑
@Repository：作为DAO对象（数据访问对象，Data Access Objects），这些类可以直接对数据库进行操作，具有将数据库操作抛出的原生异常翻译转化为spring的持久层异常的功能。

之所以区分这些注解，就能将请求处理，义务逻辑处理，数据库操作处理分离出来，为代码解耦，也方便了以后项目的维护和开发。

常用SpringBoot注解：

- **@SpringBootApplication**

  它也是 Spring Boot的核心注解，主要组合包含了以下 3 个：

  1. @SpringBootConfiguration：组合了 @Configuration 注解，实现配置文件的功能。
  2. @EnableAutoConfiguration：打开自动配置的功能，将所有符合自动配置条件的bean定义加载到IoC容器。
  3. @ComponentScan：Spring组件扫描
- **@PropertySource**

  导入properties文件
- **自动装配条件类**

  @Conditional，当指定的条件都满足时，组件才被注册
  @ConditionalOnBean，指定bean在上下文中时，才注册当前bean。用在方法上，则默认依赖类为方法的返回类型
  @ConditionalOnClass，指定类在classpath上时，才初始化当前bean。用在方法上，则默认依赖类为方法的返回类型

  ...
- **缓存类：**

  @EnableCaching，开启缓存配置，支持子类代理或者AspectJ增强
  @CacheConfig，在一个类下，提供公共缓存配置
  @Cacheable，放着方法和类上，缓存方法或类下所有方法的返回值
  @CachePut，每次先执行方法，再将结果放入缓存
  @CacheEvict，删除缓存
  @Caching，可以配置@Cacheable、@CachePut、@CacheEvict
- **定时器：**

  @EnableScheduling，开启定时任务功能
  @Scheduled，按指定执行周期执行方法
  @Schedules，包含多个@Scheduled，可同时运行多个周期配置
  @EnableAsync，开启方法异步执行的能力，通过@Async或者自定义注解找到需要异步执行的方法。通过实现AsyncConfigurer接口的getAsyncExecutor()和getAsyncUncaughtExceptionHandler()方法自定义Executor和异常处理。
  @Async，标记方法为异步线程中执行
- **Spring Cloud类：**

  1、@EnableEurekaServer
  用在springboot启动类上，表示这是一个eureka服务注册中心；

  2、@EnableDiscoveryClient
  用在springboot启动类上，表示这是一个服务，可以被注册中心找到；

  3、@LoadBalanced
  开启负载均衡能力；

  4、@EnableCircuitBreaker
  用在启动类上，开启断路器功能；

  5、@HystrixCommand(fallbackMethod=”backMethod”)
  用在方法上，fallbackMethod指定断路回调方法；

  6、@EnableConfigServer
  用在启动类上，表示这是一个配置中心，开启Config Server；

7、@EnableZuulProxy
开启zuul路由，用在启动类上；

8、@SpringCloudApplication
@SpringBootApplication
@EnableDiscovertyClient
@EnableCircuitBreaker
分别是SpringBoot注解、注册服务中心Eureka注解、断路器注解。对于SpringCloud来说，这是每一微服务必须应有的三个注解，所以才推出了@SpringCloudApplication这一注解集合。

9、@ConfigurationProperties