**Spring Boot自动配置原理(自动装配)是什么？**

通过EnableAutoConfiguration注解开启自动配置，该注解包含Import注解，导入ImportSelector接口的实现类AutoConfigurationImportSelector，在selectImports方法中会加载所有starter包的META-INF目录下的spring.factories文件。以EnableAutoConfiguration作为key，获取到对应的配置类，然后返回这些类的名字，注入到spring容器当中。通过conditional注解，按照条件进行生效

**spring boot启动原理**

一、创建springbootApplication对象

确定应用程序类型，加载所有的初始化器、监听器，设置程序运行的主类

二、调用springbootApplication对象的run方法

开启计时器，获取并启用监听器，设置应用程序参数，准备环境变量，打印banner，创建应用程序的上下文，在容器中实例化Bean对象，结束计时器，发布上下文准备就绪事件，执行自定义的run方法。

(ApplicationRunner或者 CommandLineRunner的run方法)

**如何在 Spring Boot 启动的时候运行一些特定的代码？**

可以实现接口 ApplicationRunner 或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个 run 方法

**Spring Boot 配置加载顺序**

1.命令行参数；java -jar app.jar --server.port=8081

2.Java 系统属性

3.操作系统环境变量

4.外部配置文件

当前目录下的config目录下的application-dev.yml、application.yml

当前目录下的application-dev.yml、application.yml

5.打包在 JAR 内部的配置文件

classpath下的config目录下的application-dev.yml、application.yml

classpath下的application-dev.yml、application.yml

6. @PropertySource注解；@PropertySource("classpath:custom.properties")

7.SpringApplication 默认属性

PS: 如果激活多个 Profile（如 `dev` 和 `prod`），后激活的 Profile 配置会覆盖前者。例如，激活顺序为 `dev,prod`，则 `application-prod.yml` 中的属性会覆盖 `application-dev.yml` 中的同名属性。

**Spring Boot 有哪几种读取配置的方式**

Spring Boot 可以通过 @PropertySource,@Value,@Environment, @ConfifigurationPropertie注解来绑定变量

**Spring Boot 的核心注解是哪个？它主要由哪几个注解组成的？**

@SpringBootApplication，它是 Spring Boot 的核心注解，主要组合包含了以下 3 个注解：

@SpringBootConfiguration：继承自Configuration，支持JavaConfig的方式进行配置

@EnableAutoConfiguration：开启自动配置，也可以关闭某个自动配置的选项，如关闭数据源自动配置功能： @SpringBootApplication(exclude = { DataSourceAutoConfiguration.class })。

@ComponentScan：指定初始化容器时要扫描的包。

**什么是 JavaConfig？**

JavaConfig 是 Spring 社区的产品，它提供了配置 Spring IoC 容器的纯Java 方法。有助于避免使用 XML 配置。使用 JavaConfig 的优点在于：

（1）面向对象的配置。由于配置被定义为 JavaConfig 中的类，可以充分利用 Java 中的面向对象功能。一个配置类可以继承另一个，重写它的@Bean方法等。

（2）有助于避免使用 XML 配置。

**Spring Boot 可以兼容老 Spring 项目吗，如何做？**

可以兼容，使用 @ImportResource 注解导入老 Spring 项目配置文件。

**YAML 配置的优势**

配置有序，支持数组，数组中的元素可以是基本数据类型也可以是对象

**spring boot 核心配置文件是什么？**

bootstrap (. yml 或者 . properties)：boostrap 由父Spring ApplicationContext 加载的，比 applicaton 优先加载，且 boostrap 里面的属性不能被覆盖。

application (. yml 或者 . properties)：由ApplicatonContext 加载，用于 spring boot 项目的自动化配置。

**如何在自定义端口上运行 Spring Boot 应用程序？**

application.properties 中指定端口，server.port = 8090

**spring boot 配置文件有哪几种类型？它们有什么区别？**

配置文件有 . properties 格式和 . yml 格式，区别是写法风格不同。

.properties 配置如下：

spring.RabbitMQ.port=5672

.yml 配置如下：

spring:

RabbitMQ:

port: 5672

.yml 格式不支持@PropertySource注解导入。

**Spring Boot 中如何解决跨域问题 ?**

在后端通过CORS来解决跨域问题。可以通过实现WebMvcConfigurer接口然后重写addCorsMappings方法解决跨域问题。

@Configuration

public class CorsConfig implements WebMvcConfigurer {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowCredentials(true)

allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")

.maxAge(3600);

}

}

**前后端分离，如何维护接口文档 ?**

Spring Boot 中，这个问题常见的解决方案是 Swagger ，使用 Swagger 我们可以快速生成一个接口文档网站，接口一旦发生变化，文档就会自动更新，所有开发工程师访问这一个在线网站就可以获取到最新的接口文档，非常方便。

**spring boot 有哪些方式可以实现热部署？**

使用 devtools 启动热部署，添加 devtools 库，在配置文件中把 spring. devtools. restart. enabled 设置为 true。（热部署，不重启更新代码）

**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。

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。

**开启 Spring Boot 特性有哪几种方式？**

1）继承spring-boot-starter-parent项目

2）导入spring-boot-dependencies项目依赖.

**微服务中如何实现 session 共享 ?**

Spring Session + Redis 来实现 session 共享。将所有微服务的 session 统一保存在 Redis 上，当各个微服务对 session 有相关的读写操作时，都去操作 Redis 上的 session 。

**Spring Boot 中如何实现定时任务?**

@Scheduled 注解。默认是单线程的，也就是说当我们定义了多个定时任务时，如果有本应该是相同时间触发的定时任务， 会进行排队， 如果某个定时任务执行时间过长， 就会导致其他定时并未按照设置时间来触发执行。 异步任务可以部分解决这个问题， 在定时任务上再加上@Asyns注解，定时任务就会被作为异步任务多线程执行。

**如何实现 Spring Boot 应用程序的安全性？**

为了实现 Spring Boot 的安全性，我们使用 spring-boot-starter-security 依赖项，并且必须添加安全配置。它只需要很少的代码。

**为什么要用 spring boot**

提升开发效率、配置简单、独立运行、无代码生成和 xml 配置、避免大量的 Maven 导入和各种版本冲突。