

weixin\_39610085

码龄4年

暂无认证

136

原创

-

周排名

97万+

总排名

1万+

访问量

等级

29

积分

2

粉丝

6

获赞

0

评论

33

收藏

私信

关注

搜博文文章

热门文章

springboot启动原理\_SpringBoot启动原理及相关流程

3084

运行python程序的两种方式交互式 and 文件式\_执行Python程序的两种方式

813

黑苹果intel 9560无线网卡驱动\_NUC8 安装黑苹果(Hackintosh)链接集合

681

qt中添加注释的快捷键\_Qt Creator快捷键

630

python经纬度是否在范围内\_如何判断一个指定的经纬点是否落在一个多边形区域内？ ...

568

最新文章

e映射oracle视图\_Ef code first 如何映射一个数据视图？

android php get post,Android\_android使用url connection示例(get和pos数据获取返回数据), 一定要加上对Sd卡读写文件的 - phpStudy...

php权限二进制,使用二进制进行权限控制

2021

04月

03月

02月

01月

1篇

4篇

30篇

20篇

2020年

186篇

## springboot启动原理\_SpringBoot启动原理及相关流程

weixin\_396100852020-11-24 09:39:303130收藏 9

文章标签：

springboot启动原理

springboot指定logstash type

工程师小C的小店

我也想开通小店

Python编程三剑客：Python编程从入门到实践第2版+快速上手第2版+极客编...

作者：[美] 埃里克·马瑟斯 (Eric Matthes)

出版社：人民邮电出版社

好评：100.0%

销售量：5

¥149

更多

### 一、springboot启动原理及相关流程概览

springboot是基于spring的新型的轻量级框架，最厉害的地方当属自动配置。那我们就可以根据启动流程和相关原理来看看，如何实现传奇的自动配置

### 二、springboot的启动类入口

用过springboot的技术人员很显而易见的两者之间的差别就是视觉上很直观的：springboot有自己独立的启动类（独立程序）

```
1 | @SpringBootApplication
2 | public class Application {
3 |     public static void main(String[] args) {
4 |         SpringApplication.run(Application.class, args);
5 |     }
6 | }
```

从上面代码可以看出，Annotation定义（@SpringBootApplication）和类定义（SpringApplication.run）最为耀眼，所以要揭开SpringBoot的神秘面纱，我们要从这两位开始就可以了。

### 三、单单是SpringBootApplication接口用到了这些注解

```
1 |
2 | @Target(ElementType.TYPE) // 注解的适用范围，其中TYPE用于描述类、接口（包括包注解类型）或enum声明
3 | @Retention(RetentionPolicy.RUNTIME) // 注解的生命周期，保留到class文件中（三个生命周期）
4 | @Documented // 表明这个注解应该被javadoc记录
5 | @SpringBootConfiguration // 继承了Configuration，表示当前是注解类
6 |
7 | @EnableAutoConfiguration // 开启springboot的注解功能，springboot的四大神器之一，其借助@import
8 | @ComponentScan(excludeFilters = { // 扫描路径设置（具体使用待确认）
9 | @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
10 | @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
11 | public interface SpringBootApplication {
12 | }
```

在其中比较重要的有三个注解，分别是：

- 1) @SpringBootConfiguration // 继承了Configuration，表示当前是注解类
- 2) @EnableAutoConfiguration // 开启springboot的注解功能，springboot的四大神器之一，其借助@import的帮助
- 3) @ComponentScan(excludeFilters = { // 扫描路径设置（具体使用待确认）

接下来对三个注解——详解，增加对springbootApplication的理解

#### 1) @Configuration注解

按照原来xml配置文件的形式，在springboot中我们大多用配置类来解决配置问题

配置bean方式的不同：

##### a) xml配置文件的形式配置bean

```
1 | <?xml version="1.0" encoding="UTF-8"?>
2 | <beans xmlns="http://www.springframework.org/schema/beans"
3 | xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 | xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springfram
5 | default-lazy-init="true">
6 | <!-- bean定义 -->
7 | </beans>
```

##### b) javaconfiguration的配置形式配置bean

```
1 | @Configuration
2 | public class MockConfiguration{
3 |     //bean定义
4 | }
```

注入bean方式的不同：

##### a) xml配置文件的形式注入bean

```
1 | <bean id="mockService" class="..MockServiceImpl">
2 | ...
3 | </bean>
```

##### b) javaconfiguration的配置形式注入bean

```
1 | @Configuration
2 | public class MockConfiguration{
3 |     @Bean
4 |     public MockService mockService(){
5 |         return new MockServiceImpl();
6 |     }
7 | }
```

任何一个标注了@Bean的方法，其返回值将作为一个bean定义注册到Spring的IoC容器，方法名将默认成该bean定义的id。

表达bean之间依赖关系的不同：

##### a) xml配置文件的形式表达依赖关系

```
1 | <bean id="mockService" class="..MockServiceImpl">
2 | <property name="dependencyService" ref="dependencyService" />
3 | </bean>
4 | <bean id="dependencyService" class="DependencyServiceImpl"></bean>
```

https://blog.csdn.net/weixin\_39610085/article/details/110509034

点赞3

评论

分享

收藏9

关注

一键三连

1/6

b) javaconfiguration配置的形式表达依赖关系

```
1 | @Configuration
2 | public class MockConfiguration{
3 |     @Bean
4 |     public MockService mockService(){
5 |         return new MockServiceImpl(dependencyService());
6 |     }
7 |     @Bean
8 |     public DependencyService dependencyService(){
9 |         return new DependencyServiceImpl();
10 |    }
11 | }
```

如果一个bean的定义依赖其他bean,则直接调用对应的JavaConfig类中依赖bean的创建方法就可以了。

2) @ComponentScan注解

作用: a) 对应xml配置中的元素;

b) ComponentScan的功能其实就是自动扫描并加载符合条件的组件 (比如@Component和@Repository等) 或者bean定义;

c) 将这些bean定义加载到IoC容器中.

我们可以通过basePackages等属性来定制@ComponentScan自动扫描的范围, 如果不指定, 则默认Spring框架实现会从声明@ComponentScan所在类的package进行扫描。

1

注: 所以SpringBoot的启动类最好是放在root package下, 因为默认不指定basePackages。

3) @EnableAutoConfiguration

此注解顾名思义是可以自动配置, 所以应该是springboot中最为重要的注解。

在spring框架中就提供了各种以@Enable开头的注解, 例如: @EnableScheduling、@EnableCaching、@EnableMBeanExport等; @EnableAutoConfiguration的理念和做事方式其实一脉相承简单概括一下就是, 借助@Import的支持, 收集和注册特定场景相关的bean定义。

- @EnableScheduling是通过@Import将Spring调度框架相关的bean定义都加载到IoC容器【定时任务、时间调度任务】
- @EnableMBeanExport是通过@Import将JMX相关的bean定义加载到IoC容器【监控JVM运行时状态】

@EnableAutoConfiguration也是借助@Import的帮助, 将所有符合自动配置条件的bean定义加载到IoC容器。

@EnableAutoConfiguration作为一个复合Annotation,其自身定义关键信息如下:

```
1 | @SuppressWarnings("deprecation")
2 | @Target(ElementType.TYPE)
3 | @Retention(RetentionPolicy.RUNTIME)
4 | @Documented
5 | @Inherited
6 | @AutoConfigurationPackage【重点注解】
7 | @Import(EnableAutoConfigurationImportSelector.class)【重点注解】
8 | public @interface EnableAutoConfiguration {
9 |     ...
10 | }
```

其中最重要的两个注解已经标注: 1、@AutoConfigurationPackage【重点注解】2、@Import(EnableAutoConfigurationImportSelector.class)【重点注解】

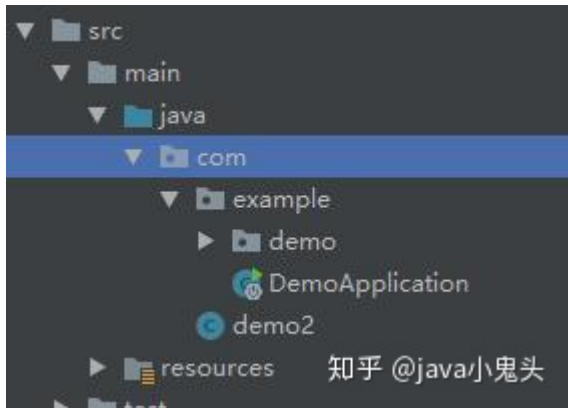
当然还有其中比较重要的一个类就是: EnableAutoConfigurationImportSelector.class

AutoConfigurationPackage注解:

```
1 |
2 | static class Registrar implements ImportBeanDefinitionRegistrar, DeterminableImports {
3 |     3 | @Override
4 |
5 |     public void registerBeanDefinitions(AnnotationMetadata metadata,BeanDefinitionRegistry
6 |         register(registry, new PackageImport(metadata).getPackageName()); 6 | }
```

它其实是注册了一个Bean的定义。

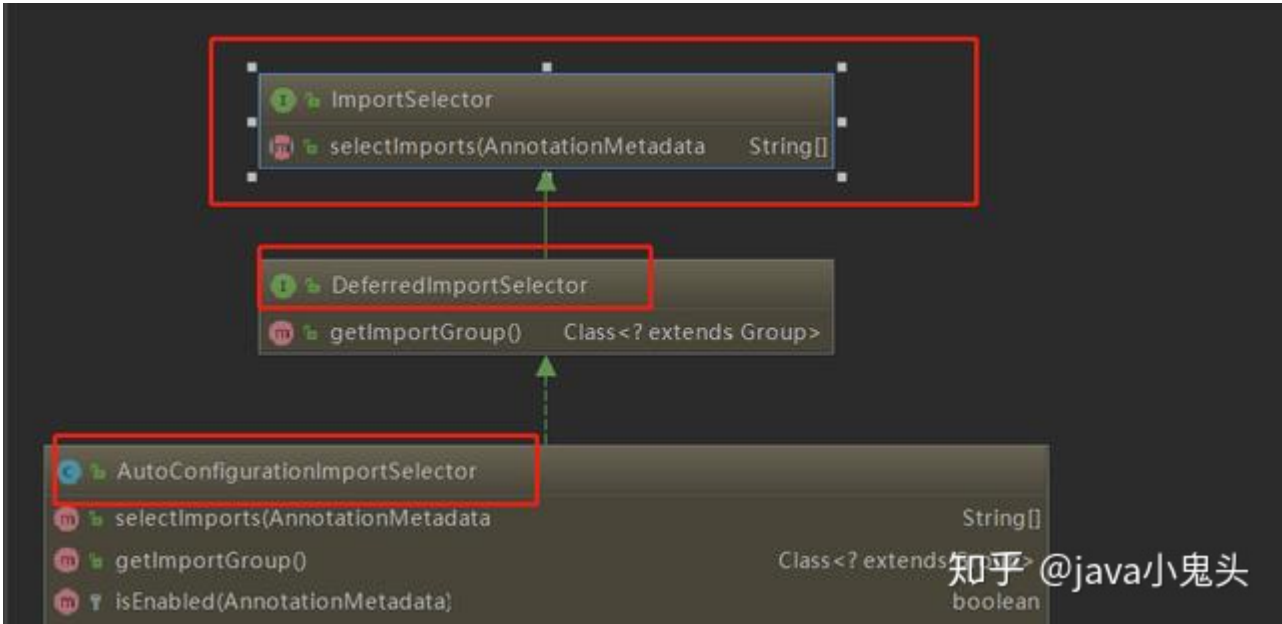
new PackageImport(metadata).getPackageName(), 它其实返回了当前主程序类的 同级以及子级 的包组件。



以上图为例, DemoApplication是和demo包同级, 但是demo2这个类是DemoApplication的父级, 和example包同级

也就是说, DemoApplication启动加载的Bean中, 并不会加载demo2, 这也就是为什么, 我们要把DemoApplication放在项目的最高级中。

Import(AutoConfigurationImportSelector.class)注解:



可以从图中看出 AutoConfigurationImportSelector 继承了 DeferredImportSelector 继承了 ImportSelector

ImportSelector有一个方法为: selectImports。

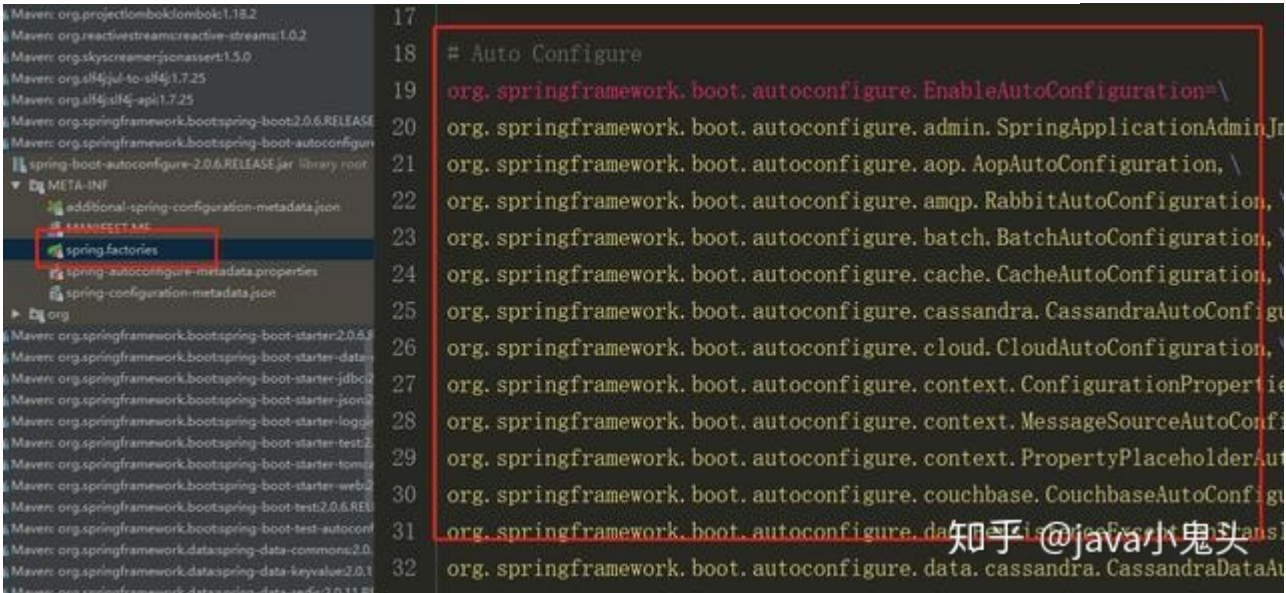
```
1 | @Override
2 | public String[] selectImports(AnnotationMetadata annotationMetadata) {
3 |     if (!isEnabled(annotationMetadata)) {
4 |         return NO_IMPORTS;
5 |     }
6 |     AutoConfigurationMetadata autoConfigurationMetadata = AutoConfigurationMetadataLoader
7 |     AnnotationAttributes attributes = getAttributes(annotationMetadata);
8 |     List<String> configurations = getCandidateConfigurations(annotationMetadata,attribute
9 |     configurations = removeDuplicates(configurations);
10 |    Set<String> exclusions = getExclusions(annotationMetadata, attributes);
11 |    checkExcludedClasses(configurations, exclusions);
12 |    configurations.removeAll(exclusions);
13 |    configurations = filter(configurations, autoConfigurationMetadata);
14 |    fireAutoConfigurationImportEvents(configurations, exclusions);
15 |    return StringUtils.toStringArray(configurations);
16 | }
```

✍

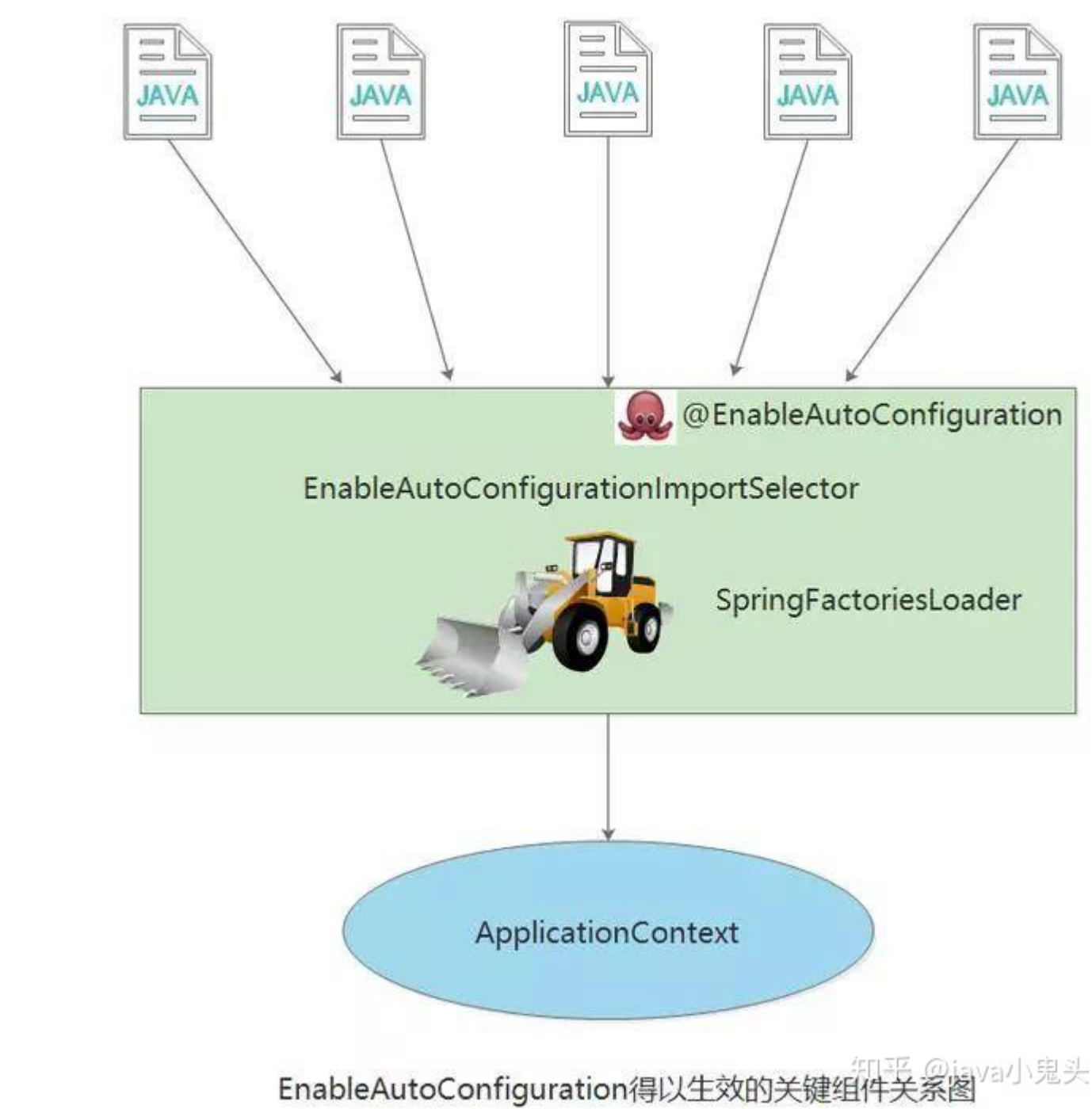
🔊

🚩





其中，最关键的要属@Import(EnableAutoConfigurationImportSelector.class)，借助EnableAutoConfigurationImportSelector，@EnableAutoConfiguration可以帮助SpringBoot应用将所有符合条件的@Configuration配置都加载到当前SpringBoot创建并使用的IoC容器。就像一只“八爪鱼”一样。



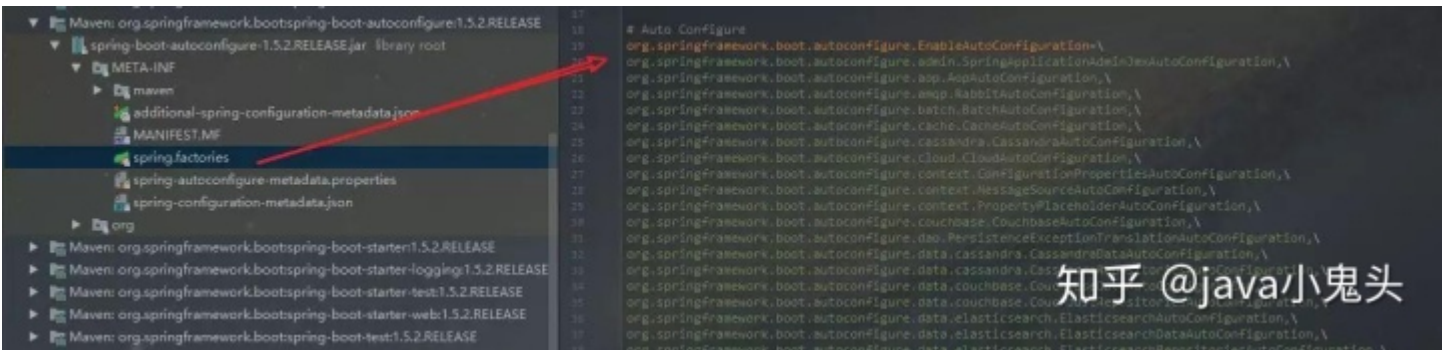
自动配置幕后英雄：SpringFactoriesLoader详解

借助于Spring框架原有的一个工具类：SpringFactoriesLoader的支持，@EnableAutoConfiguration可以智能的自动配置功效才得以大功告成！

SpringFactoriesLoader属于Spring框架私有的一种扩展方案，其主要功能就是从指定的配置文件META-INF/spring.factories加载配置。

```
1 public abstract class SpringFactoriesLoader {
2     //...
3
4     public static <T> List<T> loadFactories(Class<T> factoryClass, ClassLoader classLoader
5     ... 5 | }
6
7
8
9     public static List<String> loadFactoryNames(Class<?> factoryClass, ClassLoader classLo
10    ... 10 | }
11 }
```

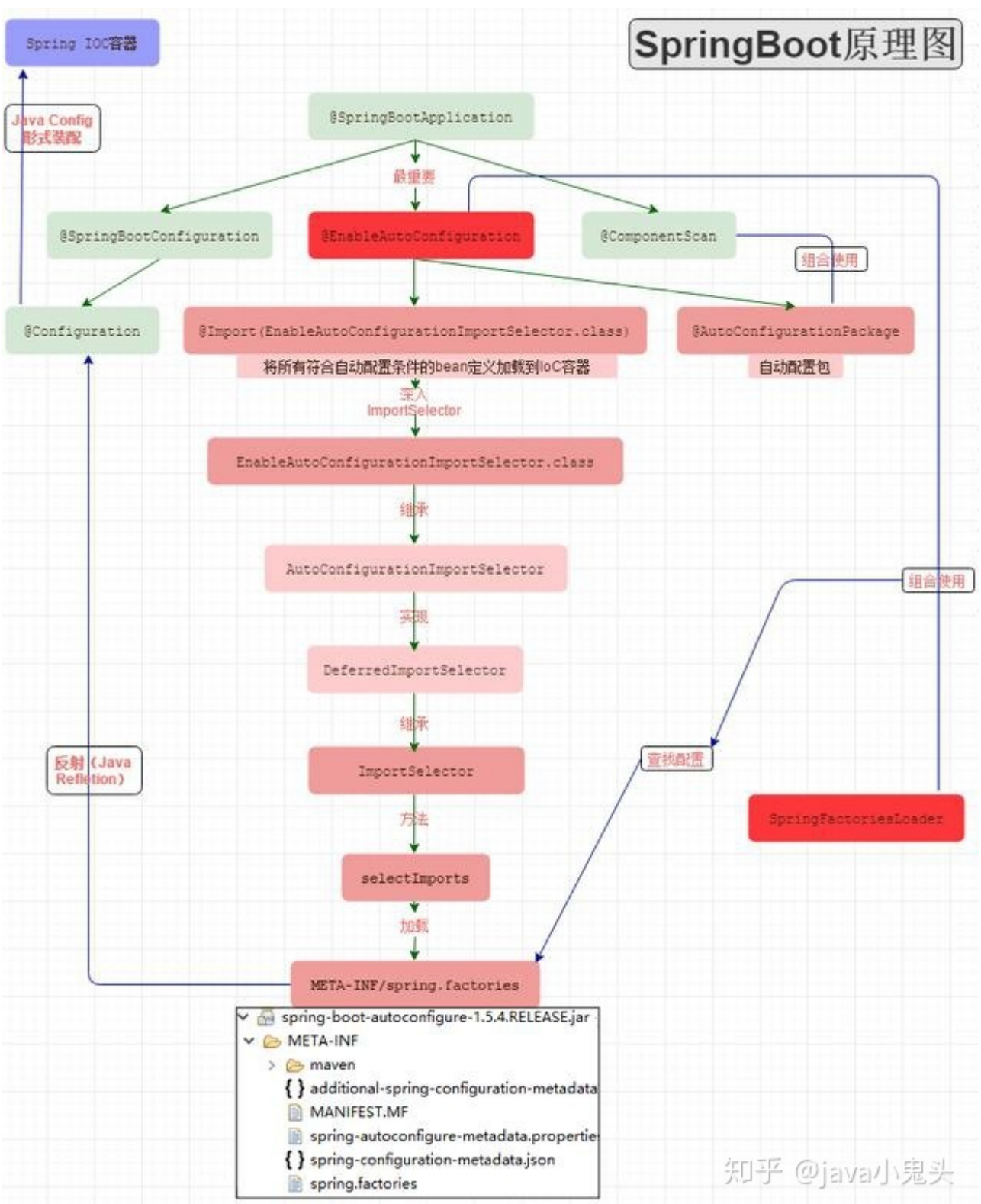
配合@EnableAutoConfiguration使用的话，它更多是提供一种配置查找的功能支持，即根据@EnableAutoConfiguration的完整类名org.springframework.boot.autoconfigure.EnableAutoConfiguration作为查找的Key,获取对应的一组@Configuration类



上图就是从SpringBoot的autoconfigure依赖包中的META-INF/spring.factories配置文件中摘录的一段内容，可以很好地说明问题。

所以，@EnableAutoConfiguration自动配置的魔法骑士就变成了：从classpath中搜寻所有的META-INF/spring.factories配置文件，并将其其中org.springframework.boot.autoconfigure.EnableAutoConfiguration对应的配置项通过反射（Java Reflection）实例化为对应的标注了@Configuration的JavaConfig形式的IoC容器配置类，然后汇总为一个并加载到IoC容器。

四、springboot启动流程概览图



深入探索SpringApplication执行流程

SpringApplication的run方法的实现是我们本次旅程的主要线路，该方法的主要流程大体可以归纳如下：

- 1) 如果我们使用的是SpringApplication的静态run方法，那么，这个方法里面首先要创建一个SpringApplication对象实例，然后调用这个创建好的SpringApplication的实例方法。在SpringApplication实例



```
1 public static ConfigurableApplicationContext run(Object[] sources, String[] args) {
2     return new SpringApplication(sources).run(args);
3 }
```

- 根据classpath里面是否存在某个特征类 (org.springframework.web.context.ConfigurableWebApplicationContext) 来决定是否应该创建一个为Web应用使用的ApplicationContext类型。
- 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationContextInitializer。
- 使用SpringFactoriesLoader在应用的classpath中查找并加载所有可用的ApplicationListener。
- 推断并设置main方法的定义类。
- 

```
1 @SuppressWarnings({ "unchecked", "rawtypes" })
2 private void initialize(Object[] sources) {
3     if (sources != null && sources.length > 0) {
4         this.sources.addAll(Arrays.asList(sources));
5     }
6     this.webEnvironment = deduceWebEnvironment();
7     setInitializers((Collection) getSpringFactoriesInstances(
8         ApplicationContextInitializer.class));
9     setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class));
10    this.mainApplicationClass = deduceMainApplicationClass();
11 }
```

- 2) SpringApplication实例初始化完成并且完成设置后，就开始执行run方法的逻辑了，方法执行伊始，首先遍历执行所有通过SpringFactoriesLoader可以查找到并加载的SpringApplicationRunListener。调用它们的started()方法，告诉这些SpringApplicationRunListener，“嘿，SpringBoot应用要开始执行咯！”。
- 

```
1 public ConfigurableApplicationContext run(String... args) {
2     Stopwatch stopwatch = new Stopwatch();
3     stopwatch.start();
4     ConfigurableApplicationContext context = null;
5     FailureAnalyzers analyzers = null;
6     configureHeadlessProperty();
7     SpringApplicationRunListeners listeners = getRunListeners(args);
8     listeners.starting();
9     try {
10        ApplicationArguments applicationArguments = new DefaultApplicationArguments(
11            args);
12        ConfigurableEnvironment environment = prepareEnvironment(listeners,
13            applicationArguments);
14        Banner printedBanner = printBanner(environment);
15        context = createApplicationContext();
16        analyzers = new FailureAnalyzers(context);
17        prepareContext(context, environment, listeners, applicationArguments,
18            printedBanner);
19        // 核心点：会打印springboot的启动标志，直到server.port端口启动
20        refreshContext(context);
21        afterRefresh(context, applicationArguments);
22        listeners.finished(context, null);
23        stopwatch.stop();
24        if (this.logStartupInfo) {
25            new StartupInfoLogger(this.mainApplicationClass)
26                .logStarted(getApplicationLog(), stopwatch);
27        }
28        return context;
29    }
30    catch (Throwable ex) {
31        handleRunFailure(context, listeners, analyzers, ex);
32        throw new IllegalStateException(ex);
33    }
34 }
```

- 3) 创建并配置当前Spring Boot应用将要使用的Environment（包括配置要使用的PropertySource以及Profile）。

```
1 private ConfigurableEnvironment prepareEnvironment(SpringApplicationRunListeners list
2 // Create and configure the environment
3 ConfigurableEnvironment environment = getOrCreateEnvironment();
4 configureEnvironment(environment, applicationArguments.getSourceArgs());
5 listeners.environmentPrepared(environment);
6 if (!this.webEnvironment) {
7     environment = new EnvironmentConverter(getClassLoader()).convertToStandardEnvironmen
8 }
9 return environment;
10 }
```

4) 遍历调用所有SpringApplicationRunListener的environmentPrepared()的方法，告诉他们：“当前SpringBoot应用使用的Environment准备好了咯！”。

```
1 public void environmentPrepared(ConfigurableEnvironment environment) {
2     for (SpringApplicationRunListener listener : this.listeners) {
3         listener.environmentPrepared(environment);
4     }
5 }
```

5) 如果SpringApplication的showBanner属性被设置为true，则打印banner。

```
1 private Banner printBanner(ConfigurableEnvironment environment) {
2     if (this.bannerMode == Banner.Mode.OFF) {
3         return null;
4     }
5     ResourceLoader resourceLoader = this.resourceLoader != null ? this.resourceLoader: n
6     SpringApplicationBannerPrinter bannerPrinter = new SpringApplicationBannerPrinter(re
7     if (this.bannerMode == Mode.LOG) {
8         return bannerPrinter.print(environment, this.mainApplicationClass, logger);
9     }
10    return bannerPrinter.print(environment, this.mainApplicationClass, System.out);
11 }
```

6) 根据用户是否明确设置了applicationContextClass类型以及初始化阶段的推断结果，决定该为当前SpringBoot应用创建什么类型的ApplicationContext并创建完成，然后根据条件决定是否添加ShutdownHook，决定是否使用自定义的BeanNameGenerator，决定是否使用自定义的ResourceLoader，当然，最重要的，将之前准备好的Environment设置给创建好的ApplicationContext使用。

7) ApplicationContext创建好之后，SpringApplication会再次借助Spring-FactoriesLoader，查找并加载classpath中所有可用的ApplicationContext-Initializer，然后遍历调用这些ApplicationContextInitializer的initialize（applicationContext）方法来对已经创建好的ApplicationContext进行进一步的处理。

```
1 @SuppressWarnings({ "rawtypes", "unchecked" })
2 protected void applyInitializers(ConfigurableApplicationContext context) {
3     for (ApplicationContextInitializer initializer : getInitializers()) {
4         Class<?> requiredType = GenericTypeResolver.resolveTypeArgument(initializer.getClass
5         Assert.isInstanceOf(requiredType, context, "Unable to call initializer.");
6         initializer.initialize(context);
7     }
8 }
```

8) 遍历调用所有SpringApplicationRunListener的contextPrepared()方法。



举报

```
(1条消息) springboot启动原理_SpringBoot启动原理及相关流程_weixin_39610085的博客-CSDN博客
2 | context.setEnvironment(environment); 3 | postProcessApplicationContext(context);
4 | applyInitializers(context);
5 | listeners.contextPrepared(context);
6 | if (this.logStartupInfo) {
7 | logStartupInfo(context.getParent() == null);
8 | logStartupProfileInfo(context);
9 | }
10 | // Add boot specific singleton beans
11 | context.getBeanFactory().registerSingleton("springApplicationArguments",applicationA
12 | if (printedBanner != null) {
13 | context.getBeanFactory().registerSingleton("springBootBanner", printedBanner);
14 | }
15 | // Load the sources
16 | Set<Object> sources = getSources();
17 | Assert.notEmpty(sources, "Sources must not be empty");
18 | load(context, sources.toArray(new Object[sources.size()]));
19 | listeners.contextLoaded(context);
20 | }
```

9)最核心的一步，将之前通过@EnableAutoConfiguration获取的所有配置以及其他形式的IoC容器配置加载到已经准备完毕的ApplicationContext。

```
1 |
private void prepareAnalyzer(ConfigurableApplicationContext context,FailureAnalyzer ana
2 | if (analyzer instanceof BeanFactoryAware) { 3 |
((BeanFactoryAware) analyzer).setBeanFactory(context.getBeanFactory()); 4 | }
5 | }
```

10) 遍历调用所有SpringApplicationRunListener的contextLoaded()方法。

```
1 | public void contextLoaded(ConfigurableApplicationContext context) {
2 | for (SpringApplicationRunListener listener : this.listeners) {
3 | listener.contextLoaded(context);
4 | }
5 | }
```

11) 调用ApplicationContext的refresh()方法，完成IoC容器可用的最后一道工序。

```
1 | private void refreshContext(ConfigurableApplicationContext context) {
2 | refresh(context);
3 | if (this.registerShutdownHook) {
4 | try {
5 | context.registerShutdownHook();
6 | }catch (AccessControlException ex) {
7 | // Not allowed in some environments.
8 | }
9 | }
10 | }
```

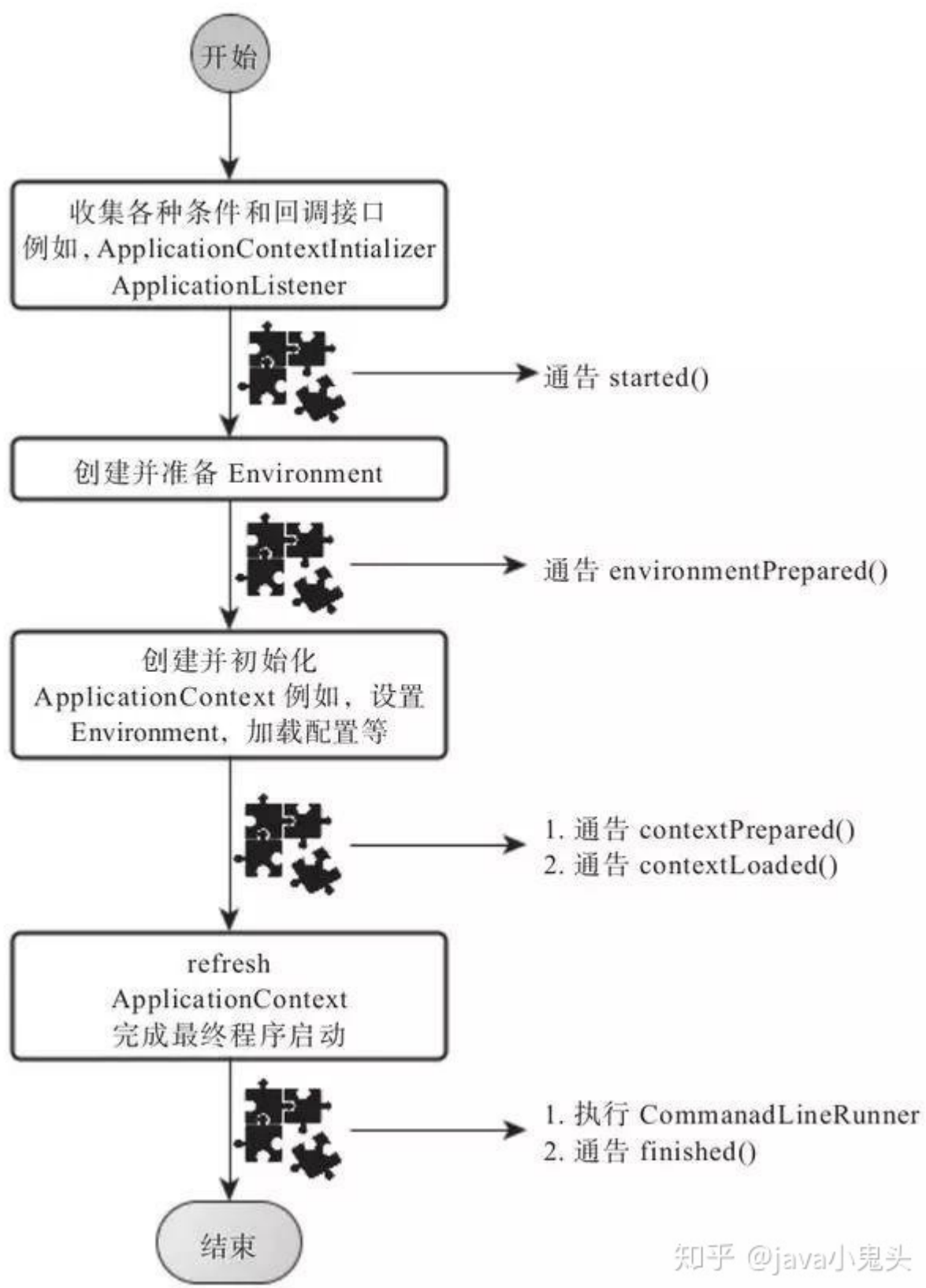
12) 查找当前ApplicationContext中是否注册有CommandLineRunner，如果有，则遍历执行它们。

```
1 | private void callRunners(ApplicationContext context, ApplicationArguments args) {
2 | List<Object> runners = new ArrayList<Object>();
3 | runners.addAll(context.getBeansOfType(ApplicationRunner.class).values());
4 | runners.addAll(context.getBeansOfType(CommandLineRunner.class).values());
5 | AnnotationAwareOrderComparator.sort(runners);
6 | for (Object runner : new LinkedHashSet<Object>(runners)) {
7 | if (runner instanceof ApplicationRunner) {
8 | callRunner((ApplicationRunner) runner, args);
9 | }
10 | if (runner instanceof CommandLineRunner) {
11 | callRunner((CommandLineRunner) runner, args);
12 | }
13 | }
14 | }
```

13) 正常情况下，遍历执行SpringApplicationRunListener的finished()方法、（如果整个过程出现异常，则依然调用所有SpringApplicationRunListener的finished()方法，只不过这种情况下会将异常信息一并传入处理）

去除事件通知点后，整个流程如下：

```
1 | public void finished(ConfigurableApplicationContext context, Throwable exception) {
2 | for (SpringApplicationRunListener listener : this.listeners) {
3 | callFinishedListener(listener, context, exception);
4 | }
5 | }
```



知乎 @java小鬼头

### 总结

到此，SpringBoot的核心组件完成了基本的解析，综合来看，大部分都是Spring框架背后的一些概念和实践方式，SpringBoot只是在这些概念和实践上对特定的场景事先进行了固化和升华，而也恰恰是这些固化让我们开发基于Sping框架的应用更加方便高效。

欢迎工作一到五年的Java工程师朋友们加入Java程序员开发： 721575865

群内提供免费的Java架构学习资料（里面有高可用、高并发、高性能及分布式、Jvm性能调优、Spring源码，MyBatis， Netty,Redis,Kafka,Mysql,Zookeeper,Tomcat,Docker,Dubbo,Nginx等多个知识点的架构资料）合理利用自己每一分每一秒的时间来学习提升自己，不要再用“没有时间”来掩饰自己思想上的懒惰！趁年轻，使劲拼，给未来的自己一个交代！





工程师小C的小店

我也想开通小店



缓存中间件Redis技术入门与应用场景实战（SpringBoot2.x + 抢红包系统设计...

讲师：修罗debug

好评：100.0%    销售量：1


¥ 99

更多

SpringBoot启动及自动装配原理过程详解

08-19


主要介绍了SpringBoot启动及自动装配原理过程详解,文中通过示例代码介绍的非常详细，对大家的学习或者工作具有一定...



优质评论可以帮助作者获得更高权重

抢沙发

评论



weixin\_39610085(博主): 这篇文章对你有帮助吗？作为一名程序工程师，在评论区留下你的困惑或你的见解，大家一起来交流吧！

相关推荐

SpringBoot启动原理(超详细)\_孤利

4-11

由于该系统是底层系统,以微服务形式对外暴露dubbo服务,所以本流程中SpringBoot不基于jetty或者tomcat等容器启动方式发...

SpringBoot启动配置原理\_蜗牛君

4-7

配置结束后,Springboot做了一些基本的收尾工作,返回了应用环境上下文。回顾整体流程,Springboot的启动,主要创建了配置...

springboot启动流程图.rp

06-01

使用Axure画的Springboot启动流程图源文件《springboot启动流程图.rp》，可以下载编辑。导出的图片，请查看博客：http...

【源码解读系列五】深入剖析Springboot启动原理的底层源码

12-22

写在前面：我是 扬帆向海，这个昵称来源于我的名字以及女朋友的名字。我热爱技术、热爱开源、热爱编程。技术是开源...

SpringBoot启动原理解析\_不忘初心

4-5

这里的@Configuration对我们来说不陌生,它就是JavaConfig形式的Spring Ioc容器的配置类使用的那个@Configuration,Spr...

SpringBoot启动过程原理一\_RainSun\_springboot启动过程

4-24

从上面代码看,调用了SpringApplication的静态方法run。这个run方法会构造一个SpringApplication的实例,然后再调用这里实...

瑞金医院糖尿病数据集

08-16

数据集来自天池大赛。此数据集旨在通过糖尿病相关的教科书、研究论文来做糖尿病文献挖掘并构建糖尿病知识图谱。

Spring系列-Spring boot启动原理

山楂树的博客 5339

我们上一篇文章新建了一个Spring boot的项目：spring-boot-test，稍微做了一些配置就运行起来了。和以前运行普通的项目...

深入解析springboot启动原理

qq\_22933035的博客 648

1. 概述 用过springboot的人都知道，springboot一个特点是能直接将springboot的项目打包成一个jar就可以直接运行。但为...

SpringBoot启动流程解析

weixin\_33804990的博客 1423

写在前面： 由于该系统是底层系统，以微服务形式对外暴露dubbo服务，所以本流程中SpringBoot不基于jetty或者tomcat等...

spring boot(二)：启动原理解析

weixin\_34004576的博客 524

我们开发任何一个Spring Boot项目，都会用到如下的启动类 1 @SpringBootApplication 2 public class Application { 3 public...

Spring Boot 详细启动原理

06-08

Spring Boot 详细启动原理,主要解释Spring Boot如何基于spring的框架下为我们提供简洁的配置

Spring Boot启动原理

wang\_jianqiang99的博客 2916

参考文章来源：来源网址：http://www.cnblogs.com/hafiz 原文地址：https://www.cnblogs.com/hafiz/p/9131264.html 1. 通过...

SpringBoot启动原理

xiexiaojing的博客 268

背景1> 大家都知道SpringBoot是通过main函数启动的，这里面跟踪代码到处都没有找到while(true)，为什么启动后可以一...

springboot的启动原理\_SpringBoot启动原理及相关流程

weixin\_39559015的博客 119

一、springboot启动原理及相关流程概览 springboot是基于spring的新型的轻量级框架，最厉害的地方当属自动配置。那...

SpringBoot启动原理及相关流程

javarrn的博客 3万+

一、springboot启动原理及相关流程概览 springboot是基于spring的新型的轻量级框架，最厉害的地方当属自动配置。那...

深度 | 面试官：能说下 SpringBoot 启动原理吗？

Java进阶架构师 6846

点击上方"java进阶架构师"，选择右上角"置顶公众号"20大进阶架构专题每日送达SpringBoot为我们做的自动配置，确实方...

经典面试题：SpringBoot启动原理

CW\_SZDX的博客 3047

抄自：https://www.jianshu.com/p/ef6f0c0de38f SpringBoot整个启动流程分为两个步骤： 初始化一个SpringApplication对...

springboot启动原理\_SpringBoot启动原理及相关流程

weixin\_39782752的博客 65

一、springboot启动原理及相关流程概览springboot是基于spring的新型的轻量级框架，最厉害的地方当属自动配置。那我们...

Springboot启动源码详解

刘建平Pinard的博客 930

我们开发任何一个Spring Boot项目，都会用到如下的启动类 @SpringBootApplication public class Application { public static...

【SpringBoot】Spring Boot 2小时入门基础教程

06-05

授课环境：mac+idea+jdk8。课程通过实践编码，针对常用功能进行讲解。 第一章：以hello word为切入点详细讲解返回s...

人工智能糖尿病数据集训练及测试

05-22

利用深度学习算法，对UCL机器学习数据库里的一个糖尿病数据集 进行训练学习并预测。主要利用了 python 的 sklearn 神...

HTML个人简历

10-08

利用HTML5语言编写的个人简历有兴趣的朋友可以下载下来学习一下

©2020 CSDN 皮肤主题: 游动-白 设计师: 白松林 返回首页

关于我们 招贤纳士 广告服务 开发助手 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 网络110报警服务 中国互联网举报中心 家长监护 Chrome商店下载 ©1999-2021北京创新乐知网络技术有限公司 版权与免责声明 版权申诉 出版物许可证 营业执照

举报

点赞3

评论

分享

收藏9

关注

一键三连

https://blog.csdn.net/weixin\_39610085/article/details/110509034

6/6