Spring Boot启动过程分析

Spring Boot是通过指定容器启动main方法，然后以命令行形式启动Jar包，主程序如下所示：

*@SpringBootApplication*

*public class SampleApplication {*

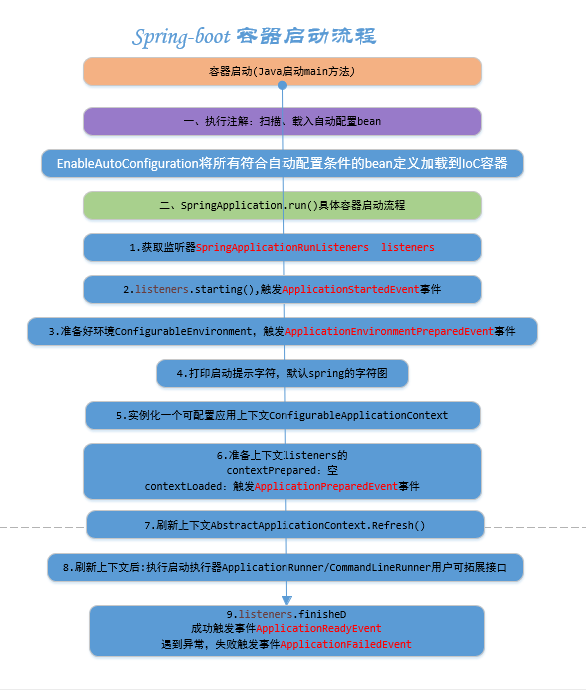
*public static void main(String[] args) {*

*SpringApplication.run(SampleApplication.class, args);*

*}*

*}*

其启动流程图如下所示：



核心点：

* @SpringBootApplication注解
* SpringApplication.run静态方法

# @SpringBootApplication

源码如下：

*@Target(ElementType.TYPE)*

*@Retention(RetentionPolicy.RUNTIME)*

*@Documented*

*@Inherited*

*@SpringBootConfiguration*

*@EnableAutoConfiguration*

*@ComponentScan(excludeFilters = {*

*@Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),*

*@Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })*

*public @interface SpringBootApplication {*

核心注解：

* @SpringBootConfiguration，实际上就是@Configuration

*@Configuration*

*public @interface SpringBootConfiguration*

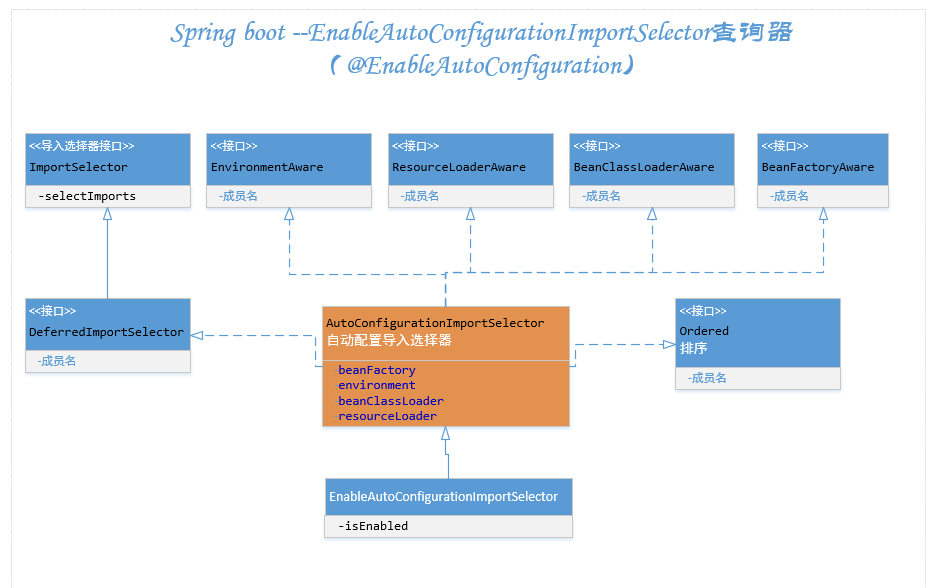
表示这是一个JavaConfig配置类，可以在这个类中定义bean，依赖关系等。

* EnableAutoConfiguration，借助

*@AutoConfigurationPackage  
@Import(AutoConfigurationImportSelector.class)*

*public @interface EnableAutoConfiguration*

借助@Import，把所有符合自动配置条件的bean定义加载到IoC容器，扫描子包和类，其核心是AutoConfigurationImportSelector，类图如下：



顶级接口为selectImports中，源码如下：

*@Override*

*public String[] selectImports(AnnotationMetadata annotationMetadata) {*

*AutoConfigurationMetadata autoConfigurationMetadata =*

*AutoConfigurationMetadataLoader.loadMetadata(this.beanClassLoader);*

*AutoConfigurationEntry autoConfigurationEntry =*

*getAutoConfigurationEntry(autoConfigurationMetadata, annotationMetadata);*

*return StringUtils.toStringArray(autoConfigurationEntry.getConfigurations());*

*}*

*protected AutoConfigurationEntry getAutoConfigurationEntry(*

*AutoConfigurationMetadata autoConfigurationMetadata,*

*AnnotationMetadata annotationMetadata) {*

*...*

*AnnotationAttributes attributes = getAttributes(annotationMetadata);*

*List<String> configurations = getCandidateConfigurations(annotationMetadata,attributes);*

*configurations = removeDuplicates(configurations);*

*Set<String> exclusions = getExclusions(annotationMetadata, attributes);*

*checkExcludedClasses(configurations, exclusions);*

*configurations.removeAll(exclusions);*

*configurations = filter(configurations, autoConfigurationMetadata);*

*fireAutoConfigurationImportEvents(configurations, exclusions);*

*return new AutoConfigurationEntry(configurations, exclusions);*

*}*

* loadMetadata，加载配置，使用类加载器加载META-INF/spring-autoconfigure-metadata.

properties文件中定义的配置，返回PropertiesAutoConfigurationMetadata

* getCandidateConfigurations，获取默认支持的自动配置类名列表调用SpringFactoriesLoad

-er.loadFactoryNames从META-INFO/spring.factories文件中获取自动配置类，其配置示例如下：

*org.springframework.boot.autoconfigure.EnableAutoConfiguration*

*=com.fys.starter.HelloAutoConfiguration*

* filter过滤器，根据onClassCondition注解把不满足条件的过滤掉

*for (AutoConfigurationImportFilter filter : getAutoConfigurationImportFilters()) {*

*invokeAwareMethods(filter);*

*boolean[] match = filter.match(candidates, autoConfigurationMetadata);*

*for (int i = 0; i < match.length; i++) {*

*if (!match[i]) {*

*skip[i] = true;*

*skipped = true;*

*}*

*}*

*}*

# SpringApplication.run

启动流程如下：

*public ConfigurableApplicationContext run(String... args) {*

*......*

*//获取监听器*

*SpringApplicationRunListeners listeners = getRunListeners(args);*

*listeners.starting();*

*try {*

*ApplicationArguments applicationArguments =*

*new DefaultApplicationArguments(args);*

*//初始化环境*

*ConfigurableEnvironment environment = prepareEnvironment(listeners,*

*applicationArguments);*

*configureIgnoreBeanInfo(environment);*

*Banner printedBanner = printBanner(environment);*

*//实例化一个可配置应用上下文*

*context = createApplicationContext();*

*exceptionReporters = getSpringFactoriesInstances(*

*SpringBootExceptionReporter.class,*

*new Class[] { ConfigurableApplicationContext.class }, context);*

*prepareContext(context, environment, listeners, applicationArguments,*

*printedBanner);*

*refreshContext(context);*

*afterRefresh(context, applicationArguments);*

*//启动程序*

*listeners.started(context);*

*callRunners(context, applicationArguments);*

*}*

*return context;*

*}*

## 2.1 SpringApplicationRunListeners

通过getRunListeners获取监听器

*private SpringApplicationRunListeners getRunListeners(String[] args) {*

*Class<?>[] types = new Class<?>[] { SpringApplication.class, String[].class };*

*return new SpringApplicationRunListeners(logger,*

*getSpringFactoriesInstances(SpringApplicationRunListener.class, types, this, args));*

*}*

*private <T> Collection<T> getSpringFactoriesInstances(Class<T> type,*

*Class<?>[] parameterTypes, Object... args) {*

*ClassLoader classLoader = Thread.currentThread().getContextClassLoader();*

*// 载入工厂名称集合*

*Set<String> names = new LinkedHashSet<>(*

*SpringFactoriesLoader.loadFactoryNames(type, classLoader));*

*//创建工厂实例*

*List<T> instances = createSpringFactoriesInstances(type, parameterTypes,*

*classLoader, args, names);*

*AnnotationAwareOrderComparator.sort(instances);*

*return instances;*

*}*

SpringFactoresLoader从META-INF/spring.factores文件中获取SpringApplicationRunListener类配置，内容如下：



这些都是类名称，根据配置获取EventPublishingRunListener事件发布启动监听器。

## **2.2 上下文的创建及使用**

ConfigurableApplicationContext上下文，SpringApplication#run中相关流程

* createApplicationContext，初始化context，
* prepareContext，通过BeanNameGenerator将ResourceLoader设置进上下文，并载入资源，添加单例Bean
* refreshContext，刷新上下文，调用AbstractApplicationContext抽象类的refresh方法，获取依赖jar中的bean
* afterRefresh

## **2.3 callRunner**

Spring Boot中提供ApplicationRunner和CommandLineRunner，在容器启动完毕后执行操作

*private void callRunners(ApplicationContext context, ApplicationArguments args) {*

*List<Object> runners = new ArrayList<>();*

*runners.addAll(context.getBeansOfType(ApplicationRunner.class).values());*

*runners.addAll(context.getBeansOfType(CommandLineRunner.class).values());*

*AnnotationAwareOrderComparator.sort(runners);*

*for (Object runner : new LinkedHashSet<>(runners)) {*

*if (runner instanceof ApplicationRunner) {*

*callRunner((ApplicationRunner) runner, args);*

*}*

*if (runner instanceof CommandLineRunner) {*

*callRunner((CommandLineRunner) runner, args);*

*}*

*}*

*}*

在CommandLineRunner中执行参数是原始的java启动类main方法的spring[] args字符串

https://www.cnblogs.com/dennyzhangdd/p/8028950.html