Spring Boot启动原理源码剖析

Spring Boot启动原理源码剖析
概念：
原理&源码：

概念&原理：
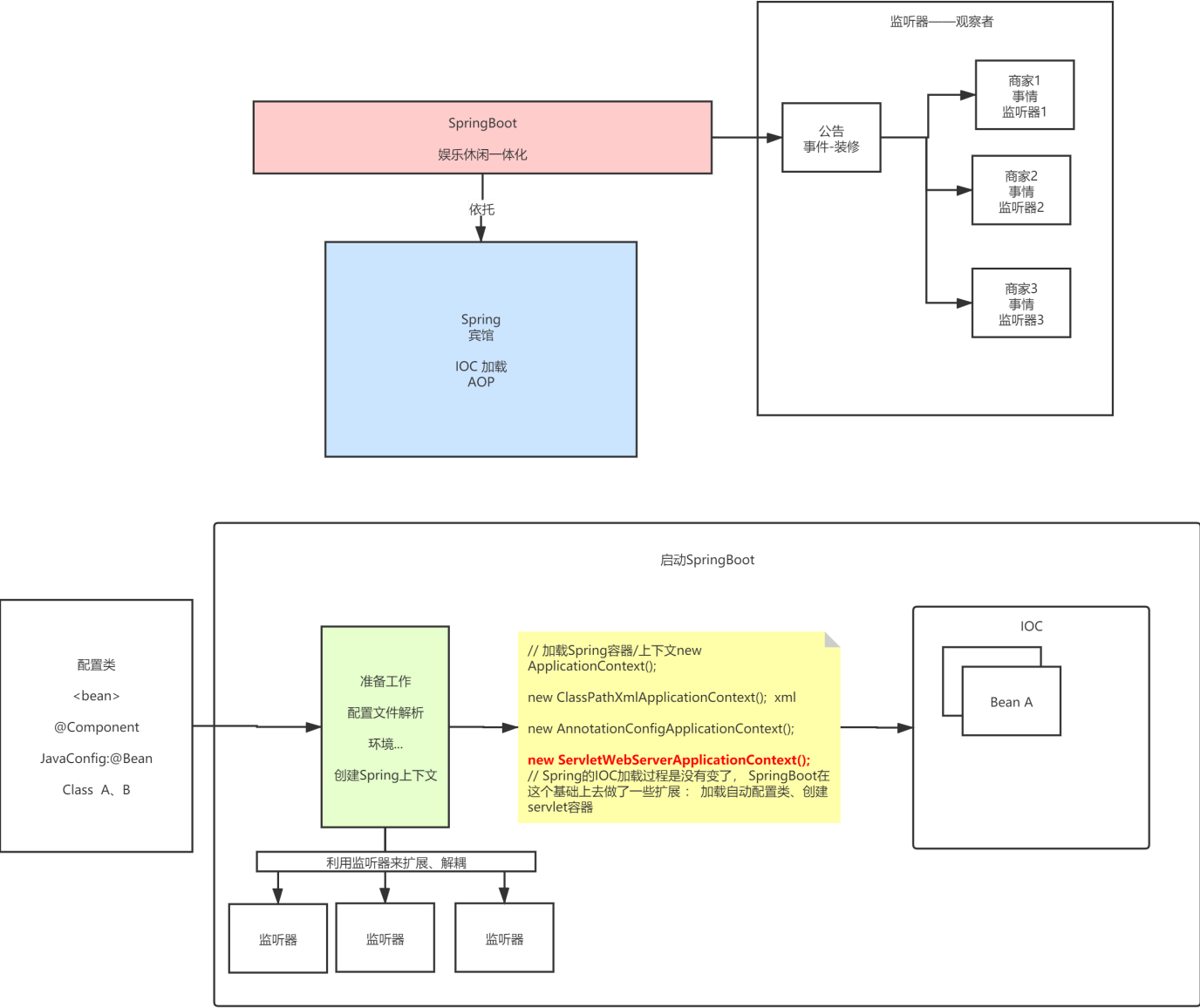
源码：

SpringBoot 事假监听器发布顺序：

```
1  1.ApplicationStartingEvent在运行开始时发送，但在进行任何处理之前（侦听器和初始化程序的注册除外）发送。
```

```
2    2.在创建上下文之前，将发送ApplicationEnvironmentPreparedEvent。
3    3.准备ApplicationContext并调用ApplicationContextInitializers之后，将发送ApplicationContextInitializedEvent。
4    4.读取完配置类后发送ApplicationPreparedEvent。
5    5.在刷新上下文之后但在调用任何应用程序和命令行运行程序之前，将发送ApplicationStartedEvent。
6    6.紧随其后发送带有LivenessState.CORRECT的AvailabilityChangeEvent，以指示该应用程序被视为处于活动状态。
7    7.在调用任何应用程序和命令行运行程序之后，将发送ApplicationReadyEvent。
8    8.紧随其后发送ReadabilityState.ACCEPTING_TRAFFIC的AvailabilityChangeEvent，以指示应用程序已准备就绪，可以处理请求
9    如果启动时发生异常，则发送ApplicationFailedEvent。
```

1.ApplicationStartingEvent

```
▼ ∞ getApplicationListeners(event, type) = {ArrayList@1931}  size = 5
   ▶ ≣ 0 = {RestartApplicationListener@1836}
   ▶ ≣ 1 = {LoggingApplicationListener@1933}
   ▶ ≣ 2 = {BackgroundPreinitializer@1934}
   ▶ ≣ 3 = {DelegatingApplicationListener@1935}
   ▶ ≣ 4 = {LiquibaseServiceLocatorApplicationListener@1936}
```

2.ApplicationEnvironmentPreparedEvent

```
▼ ∞ getApplicationListeners(event, type) = {ArrayList@2457}  size = 8
   ▶ ≣ 0 = {RestartApplicationListener@2319}
   ▶ ≣ 1 = {ConfigFileApplicationListener@2450}
   ▶ ≣ 2 = {AnsiOutputApplicationListener@2451}
   ▶ ≣ 3 = {LoggingApplicationListener@2452}
   ▶ ≣ 4 = {BackgroundPreinitializer@2453}
   ▶ ≣ 5 = {ClasspathLoggingApplicationListener@2454}
   ▶ ≣ 6 = {DelegatingApplicationListener@2455}
   ▶ ≣ 7 = {FileEncodingApplicationListener@2456}
```

1 调用SpringApplication.run启动springboot应用

```
1  SpringApplication.run(Application.class, args);
```

2. 使用自定义SpringApplication进行启动

```
1  public static ConfigurableApplicationContext run(Class<?>[] primarySources, String[] args) {
2      return new SpringApplication(primarySources).run(args);
3  }
```

1. 创建SpringApplication

- new SpringApplication(primarySources)

```
1  public SpringApplication(ResourceLoader resourceLoader, Class<?>... primarySources) {
2      this.resourceLoader = resourceLoader;

3      Assert.notNull(primarySources, "PrimarySources must not be null");
```

```
4      // 将启动类放入primarySources
5      this.primarySources = new LinkedHashSet<>(Arrays.asList(primarySources));
6      // 根据classpath 下的类，推算当前web应用类型(webFlux, servlet)
7      this.webApplicationType = WebApplicationType.deduceFromClasspath();
8      // 就是去spring.factories 中去获取所有key:org.springframework.context.ApplicationContextInitializer
9      setInitializers((Collection) getSpringFactoriesInstances(ApplicationContextInitializer.class));
10     //就是去spring.factories 中去获取所有key: org.springframework.context.ApplicationListener
11     setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class));
12     // 根据main方法推算出mainApplicationClass
13     this.mainApplicationClass = deduceMainApplicationClass();
14  }
```

- org.springframework.context.ApplicationContextInitializer

- org.springframework.context.ApplicationListener

总结：

1. 获取启动类

2.获取web应用类型

3.读取了对外扩展的ApplicationContextInitializer ,ApplicationListener

4. 根据main推算出所在的类

就是去初始化了一些信息

2. 启动

- run
  - 启动springboot最核心的逻辑

```java
public ConfigurableApplicationContext run(String... args) {
    // 用来记录当前springboot启动耗时
    StopWatch stopWatch = new StopWatch();
    // 就是记录了启动开始时间
    stopWatch.start();
    // 它是任何spring上下文的接口, 所以可以接收任何ApplicationContext实现
    ConfigurableApplicationContext context = null;
    Collection<SpringBootExceptionReporter> exceptionReporters = new ArrayList<>();
    // 开启了Headless模式:
    configureHeadlessProperty();
    // 去spring.factroies中读取了SpringApplicationRunListener 的组件,  就是用来发布事件或者运行监听器
    SpringApplicationRunListeners listeners = getRunListeners(args);
    // 发布1.ApplicationStartingEvent事件, 在运行开始时发送
    listeners.starting();
    try {
        // 根据命令行参数 实例化一个ApplicationArguments
        ApplicationArguments applicationArguments = new DefaultApplicationArguments(args);
        // 预初始化环境:  读取环境变量, 读取配置文件信息(基于监听器)
        ConfigurableEnvironment environment = prepareEnvironment(listeners, applicationArguments);
        // 忽略beaninfo的bean
        configureIgnoreBeanInfo(environment);
        // 打印Banner 横幅
        Banner printedBanner = printBanner(environment);
        // 根据webApplicationType创建Spring上下文
        context = createApplicationContext();
        exceptionReporters = getSpringFactoriesInstances(SpringBootExceptionReporter.class,
            new Class[] { ConfigurableApplicationContext.class }, context);
        //预初始化spring上下文
        prepareContext(context, environment, listeners, applicationArguments, printedBanner);
        // 加载spring ioc 容器    **相当重要    由于是使用AnnotationConfigServletWebServerApplicationContext 启动的
        //  加载自动配置类: invokeBeanFactoryPostProcessors ,   创建servlet容器onRefresh
        refreshContext(context);
        afterRefresh(context, applicationArguments);
        stopWatch.stop();
        if (this.logStartupInfo) {
            new StartupInfoLogger(this.mainApplicationClass).logStarted(getApplicationLog(), stopWatch);
        }
        listeners.started(context);
        callRunners(context, applicationArguments);
    }
    catch (Throwable ex) {
        handleRunFailure(context, ex, exceptionReporters, listeners);
        throw new IllegalStateException(ex);
    }

    try {
```

```
47        listeners.running(context);
48    }
49    catch (Throwable ex) {
50        handleRunFailure(context, ex, exceptionReporters, null);
51        throw new IllegalStateException(ex);
52    }
53    return context;
54 }
```

- prepareEnvironment

```
1  private ConfigurableEnvironment prepareEnvironment(SpringApplicationRunListeners listeners,
2          ApplicationArguments applicationArguments) {
3      // 根据webApplicationType 创建Environment  创建就会读取： java环境变量和系统环境变量
4      ConfigurableEnvironment environment = getOrCreateEnvironment();
5      // 将命令行参数读取环境变量中
6      configureEnvironment(environment, applicationArguments.getSourceArgs());
7      // 将@PropertieSource的配置信息 放在第一位， 因为读取配置文件@PropertieSource优先级是最低的
8      ConfigurationPropertySources.attach(environment);
9      // 发布了ApplicationEnvironmentPreparedEvent 的监听器  读取了全局配置文件
10     listeners.environmentPrepared(environment);
11     // 将所有spring.main 开头的配置信息绑定SpringApplication
12     bindToSpringApplication(environment);
13     if (!this.isCustomEnvironment) {
14         environment = new EnvironmentConverter(getClassLoader()).convertEnvironmentIfNecessary(environment,
15             deduceEnvironmentClass());
16     }
17     //更新PropertySources
18     ConfigurationPropertySources.attach(environment);
19     return environment;
20 }
```

- prepareContext
  - 预初始化上下文

```
1  private void prepareContext(ConfigurableApplicationContext context, ConfigurableEnvironment environment,
2          SpringApplicationRunListeners listeners, ApplicationArguments applicationArguments, Banner printedBann
3      context.setEnvironment(environment);
4      postProcessApplicationContext(context);
5      // 拿到之前读取到所有ApplicationContextInitializer的组件， 循环调用initialize方法
6      applyInitializers(context);
7      // 发布了ApplicationContextInitializedEvent
8      listeners.contextPrepared(context);
9      if (this.logStartupInfo) {
10         logStartupInfo(context.getParent() == null);
11         logStartupProfileInfo(context);
12     }
```

```
13        // 获取当前spring上下文beanFactory (负责创建bean)
14        ConfigurableListableBeanFactory beanFactory = context.getBeanFactory();
15        beanFactory.registerSingleton("springApplicationArguments", applicationArguments);
16        if (printedBanner != null) {
17            beanFactory.registerSingleton("springBootBanner", printedBanner);
18        }
19        // 在Spring下 如果出现2个重名的bean，则后读取到的会覆盖前面
20        // 在SpringBoot 在这里设置了不允许覆盖， 当出现2个重名的bean 会抛出异常
21        if (beanFactory instanceof DefaultListableBeanFactory) {
22            ((DefaultListableBeanFactory) beanFactory)
23                    .setAllowBeanDefinitionOverriding(this.allowBeanDefinitionOverriding);
24        }
25        // 设置当前spring容器是不是要将所有的bean设置为懒加载
26        if (this.lazyInitialization) {
27            context.addBeanFactoryPostProcessor(new LazyInitializationBeanFactoryPostProcessor());
28        }
29        // Load the sources
30        Set<Object> sources = getAllSources();
31        Assert.notEmpty(sources, "Sources must not be empty");
32        // 读取主启动类 （因为后续要根据配置类解析配置的所有bean）
33        load(context, sources.toArray(new Object[0]));
34        //4.读取完配置类后发送ApplicationPreparedEvent。
35        listeners.contextLoaded(context);
36    }
```

总结：

1. 初始化SpringApplication 从spring.factories 读取 listener ApplicationContextInitializer 。

2.运行run方法

3.读取 环境变量 配置信息......

4. 创建springApplication上下文:**ServletWebServerApplicationContext**

5. 预初始化上下文： 读取启动类

6.调用refresh 加载ioc容器

   加载所有的自动配置类

   创建servlet容器

7.在这个过程中springboot会调用很多监听器对外进行