课程内容：

1、SpringBoot启动过程源码解析

2、SpringBoot启动过程中扩展点源码解析

3、SpringBoot配置文件优先级解析

4、SpringBoot内置Tomcat启动源码解析

启动过程思维导图（重点）https://www.processon.com/view/link/64c664e0470d721c4e38bfe3

有道云链接：https://note.youdao.com/s/dTt7CSTP

构造SpringApplication对象

1、推测web应用类型

1. 如果项目依赖中存在**org.springframework.web.reactive.DispatcherHandler**，并且不存在**org.springframework.web.servlet.DispatcherServlet**，那么应用类型为**WebApplicationType.REACTIVE**
2. 如果项目依赖中不存在**org.springframework.web.reactive.DispatcherHandler**，也不存在**org.springframework.web.servlet.DispatcherServlet**，那么应用类型为**WebApplicationType.NONE**
3. 否则，应用类型为**WebApplicationType.SERVLET**

2、获取BootstrapRegistryInitializer对象

1. 从"META-INF/spring.factories"中读取key为**BootstrapRegistryInitializer**类型的扩展点，并实例化出对应扩展点对象
2. BootstrapRegistryInitializer的作用是可以**初始化BootstrapRegistry**
3. 上面的DefaultBootstrapContext对象就是一个BootstrapRegistry，可以用来注册一些对象，这些对象可以用在**从SpringBoot启动到Spring容器初始化完成**的过程中
4. 我的理解：没有Spring容器之前就利用BootstrapRegistry来共享一些对象，有了Spring容器之后就利用Spring容器来共享一些对象

3、获取ApplicationContextInitializer对象

1. 从"META-INF/spring.factories"中读取key为**ApplicationContextInitializer**类型的扩展点，并实例化出对应扩展点对象
2. 顾名思义，ApplicationContextInitializer是用来初始化Spring容器ApplicationContext对象的，比如可以利用ApplicationContextInitializer来向Spring容器中添加ApplicationListener

4、获取ApplicationListener对象

1. 从"META-INF/spring.factories"中读取key为**ApplicationListener**类型的扩展点，并实例化出对应扩展点对象
2. ApplicationListener是Spring中的监听器，并不是SpringBoot中的新概念，不多解释了

5、推测出Main类（main()方法所在的类）

没什么具体的作用，逻辑是根据当前线程的调用栈来判断main()方法在哪个类，哪个类就是Main类

run(String... args)方法

1. 创建DefaultBootstrapContext对象
2. 利用BootstrapRegistryInitializer初始化DefaultBootstrapContext对象
3. 获取SpringApplicationRunListeners

这三个步骤没什么特殊的

5、触发SpringApplicationRunListener的starting()

默认情况下SpringBoot提供了一个EventPublishingRunListener，它实现了SpringApplicationRunListener接口，默认情况下会利用EventPublishingRunListener发布一个ApplicationContextInitializedEvent事件，程序员可以通过定义ApplicationListener来消费这个事件

6、创建Environment对象

Environment对象表示环境变量，该对象内部主要包含了:

1. 当前操作系统的环境变量
2. JVM的一些配置信息
3. -D方式所配置的JVM环境变量

7、触发SpringApplicationRunListener的environmentPrepared()

默认情况下会利用EventPublishingRunListener发布一个ApplicationEnvironmentPreparedEvent事件，程序员可以通过定义ApplicationListener来消费这个事件，比如默认情况下会有一个EnvironmentPostProcessorApplicationListener来消费这个事件，而这个ApplicationListener接收到这个事件之后，就会解析application.properties、application.yml文件，并添加到Environment对象中去。

8、打印Banner

没什么特殊的

9、创建Spring容器对象（ApplicationContext）

会利用ApplicationContextFactory.DEFAULT，根据应用类型创建对应的Spring容器。

ApplicationContextFactory.DEFAULT为：

```
ApplicationContextFactory DEFAULT = (webApplicationType) -> {
    try {
        switch (webApplicationType) {
            case SERVLET:
                return new AnnotationConfigServletWebServerApplicationContext();
            case REACTIVE:
                return new AnnotationConfigReactiveWebServerApplicationContext();
            default:
                return new AnnotationConfigApplicationContext();
        }
    }
    catch (Exception ex) {
        throw new IllegalStateException("Unable create a default ApplicationContext instance, "
                                        + "you may need a custom ApplicationContextFactory", ex);
    }
};
```

所以：

1. 应用类型为SERVLET，则对应AnnotationConfigServletWebServerApplicationContext
2. 应用类型为REACTIVE，则对应AnnotationConfigReactiveWebServerApplicationContext
3. 应用类型为普通类型，则对应AnnotationConfigApplicationContext

10、利用ApplicationContextInitializer初始化Spring容器对象

默认情况下SpringBoot提供了多个ApplicationContextInitializer，其中比较重要的有ConditionEvaluationReportLoggingListener，别看到它的名字叫XXXListener，但是它确实是实现了ApplicationContextInitializer接口的。

在它的initialize()方法中会：

1. 将Spring容器赋值给它的applicationContext属性
2. 并且往Spring容器中添加一个ConditionEvaluationReportListener
   （ConditionEvaluationReportLoggingListener的内部类），它是一个ApplicationListener
3. 并生成一个ConditionEvaluationReport对象赋值给它的report属性

ConditionEvaluationReportListener会负责接收ContextRefreshedEvent事件，也就是Spring容器一旦启动完毕就会触发ContextRefreshedEvent，ConditionEvaluationReportListener就会打印自动配置类的条件评估报告。

11、触发SpringApplicationRunListener的contextPrepared()

默认情况下会利用EventPublishingRunListener发布一个ApplicationContextInitializedEvent事件，默认情况下暂时没有ApplicationListener消费了这个事件

12、调用DefaultBootstrapContext对象的close()

没什么特殊的，忽略

13、将启动类作为配置类注册到Spring容器中（load()方法）

将SpringApplication.**run**(MyApplication.class);中传入进来的类，比如MyApplication.class，作为Spring容器的配置类

14、 触发SpringApplicationRunListener的contextLoaded()

默认情况下会利用EventPublishingRunListener发布一个ApplicationPreparedEvent事件

15、刷新Spring容器

调用Spring容器的refresh()方法，结合第9、13步，相当于执行了这样一个流程：

1. AnnotationConfigApplicationContext applicationContext = new AnnotationConfigApplicationContext();
2. applicationContext .register(MyApplication.class)
3. applicationContext .refresh()

16、触发SpringApplicationRunListener的started()

发布ApplicationStartedEvent事件和AvailabilityChangeEvent事件，AvailabilityChangeEvent事件表示状态变更状态，变更后的状态为LivenessState.**CORRECT**

LivenessState枚举有两个值：

1. CORRECT：表示当前应用正常运行中
2. BROKEN：表示当前应用还在运行，但是内部出现问题，暂时还没发现哪里用到了

17、调用ApplicationRunner和CommandLineRunner

1. 获取Spring容器中的ApplicationRunner类型的Bean
2. 获取Spring容器中的CommandLineRunner类型的Bean
3. 执行它们的run()

18、触发SpringApplicationRunListener的ready()

发布ApplicationReadyEvent事件和AvailabilityChangeEvent事件，AvailabilityChangeEvent事件表示状态变更状态，变更后的状态为ReadinessState.**ACCEPTING_TRAFFIC**

ReadinessState枚举有两个值：

1. ACCEPTING_TRAFFIC：表示当前应用准备接收请求
2. REFUSING_TRAFFIC：表示当前应用拒绝接收请求，比如Tomcat关闭时，就会发布AvailabilityChangeEvent事件，并且状态为REFUSING_TRAFFIC

19、上述过程抛异常了就触发SpringApplicationRunListener的failed()

发布ApplicationFailedEvent事件

配置文件解析

流程图：https://www.processon.com/view/link/62d399e71e08530a89222b23

SpringBoot完整的配置优先级

https://docs.spring.io/spring-boot/docs/current/reference/html/features.html#features.external-config

优先级由低到高（注意和我写的顺序是反的，我写的是由高到底）

1. Default properties (specified by setting SpringApplication.setDefaultProperties).
2. @PropertySource annotations on your @Configuration classes. Please note that such property sources

are not added to the Environment until the application context is being refreshed. This is too late to configure certain properties such as logging.* and spring.main.* which are read before refresh begins.

3. Config data (such as application.properties files).
4. A RandomValuePropertySource that has properties only in random.*.
5. OS environment variables.
6. Java System properties (System.getProperties()).
7. JNDI attributes from java:comp/env. 不管它
8. ServletContext init parameters.
9. ServletConfig init parameters.
10. Properties from SPRING_APPLICATION_JSON (inline JSON embedded in an environment variable or system property).
11. Command line arguments.
12. properties attribute on your tests. Available on @SpringBootTest and the test annotations for testing a particular slice of your application.
13. @TestPropertySource annotations on your tests.
14. Devtools global settings properties in the $HOME/.config/spring-boot directory when devtools is active.