Spring Boot

**Spring Boot** - упрощает создание автономных приложений на основе Spring производственного уровня, которые вы можете "просто запустить". Он поднимает все используемые Фреймворки spring объединяет и запускает приложения.

**Основные аннотации из этого пакета:**

**@SpringBootApplication** – это аннотация, чтобы отметить основной класс приложения Spring Boot. @SpringBootApplication инкапсулирует аннотации @Configuration, @EnableAutoConfiguration и @ComponentScan с их атрибутами по умолчанию.

**@Configuration** - это аннотация на уровне класса, указывающая на то, что объект является источником определений бина. Классы, аннотированные @Configuration, объявляют бины через методы, аннотированные @Bean. Вызовы методов @Bean для классов @Configuration также могут быть использованы для определения межбиновых зависимостей.

**@EnableAutoConfiguration** – это аннотация которая включает автоматическую настройку Spring ApplicationContext путем сканирования компонентов пути к классам и регистрации бинов, соответствующих различным условиям.

**@ComponentScan** – это аннотация которая указывает, где Spring искать классы, помеченные аннотацией @Component или его производной.

**@Component** – это аннотация которая помечает класс в качестве кандидата для создания Spring бина.

**@Import** – это аннотация Spring импортирует один или несколько @Configuration классов. @Import Аннотация эквивалентна <import> элементу в Spring XML.

**@ImportResource** – это аннотация импортирует конфигурации XML в приложения на основе JavaConfig. Класс, аннотированный с помощью @Configuration, может использовать @ImportResource аннотации для загрузки XML-конфигураций.

**Разберем основные классы из этого пакета:**

**SpringApplication -** класс, который может использоваться для начальной загрузки и запуска приложения Spring из основного метода Java.

**Методы:**

**addBootstrapRegistryInitializer(BootstrapRegistryInitializer bootstrapRegistryInitializer)** - добавляет BootstrapRegistryInitializer экземпляры, которые могут быть использованы для инициализации BootstrapRegistry.

**addInitializers(ApplicationContextInitializer<?>... initializers)** - добавьте ApplicationContextInitializer файлы, которые будут применены к Spring ApplicationContext.

**addListeners(ApplicationListener<?>... listeners)** - добавьте ApplicationListener файлы, которые будут применены к SpringApplication и зарегистрированы в ApplicationContext.

**addPrimarySources(Collection<Class<?>> additionalPrimarySources)** - добавьте дополнительные элементы к основным источникам, которые будут добавлены в ApplicationContext при run(String...) вызове.

**afterRefresh(ConfigurableApplicationContext context, ApplicationArguments args)** - вызывается после обновления контекста.

**applyInitializers(ConfigurableApplicationContext context)** - примените любые ApplicationContextInitializer к контексту перед его обновлением.

**bindToSpringApplication(ConfigurableEnvironment environment)** - привязать среду к SpringApplication.

**configureEnvironment(ConfigurableEnvironment environment, String[] args)** - делегирование метода шаблона в configurePropertySources(ConfigurableEnvironment, String[]) и configureProfiles(ConfigurableEnvironment, String[]) в таком порядке.

**configureProfiles(ConfigurableEnvironment environment, String[] args) -** настройте, какие профили активны (или активны по умолчанию) для этой среды приложения.

**configurePropertySources(ConfigurableEnvironment environment, String[] args)** - добавляйте, удаляйте или переупорядочивайте любые PropertySource приложения в среде этого приложения.

**createApplicationContext()** - метод стратегии, используемый для создания ApplicationContext.

**createBeanDefinitionLoader(BeanDefinitionRegistry registry, Object[] sources)** - заводской метод, используемый для создания BeanDefinitionLoader.

**exit(ApplicationContext context, ExitCodeGenerator... exitCodeGenerators**) - статический помощник, который можно использовать для выхода из SpringApplication и получения кода, указывающего на успех (0) или иным образом.

**from(ThrowingConsumer<String[]> main)** - создайте приложение на основе существующего main метода, который может выполняться с дополнительными @Configuration классами или bean-классами.

**getAdditionalProfiles()** - возвращает неизменяемый набор любых дополнительных используемых профилей.

**getAllSources()** - возвращает неизменяемый набор всех исходных текстов, которые будут добавлены в ApplicationContext при run(String...) вызове.

**getApplicationLog()** - возвращает Log для приложения.

**getApplicationStartup()** - возвращаетApplicationStartup, используемый для сбора показателей запуска.

**getClassLoader()** - либо загрузчик классов, который будет использоваться в ApplicationContext (если resourceLoader задан), либо загрузчик класса контекста (если не null), либо загрузчик класса Spring ClassUtils.

**getEnvironmentPrefix()** - возвращает префикс, который следует применять при получении свойств конфигурации из системной среды.

**getInitializers()** - возвращает доступный только для чтения упорядоченный набор ApplicationContextInitializer файлов, которые будут применены к Spring ApplicationContext.

**getListeners()** - возвращает доступный только для чтения упорядоченный набор ApplicationListener файлов, которые будут применены к SpringApplication и зарегистрированы в ApplicationContext .

**getMainApplicationClass()** - возвращает основной класс приложения, который был выведен или явно настроен.

**getResourceLoader()** - загрузчик ресурсов, который будет использоваться в ApplicationContext.

**getShutdownHandlers()** - возвращает SpringApplicationShutdownHandlers экземпляр, который можно использовать для добавления или удаления обработчиков, выполняющих действия перед завершением работы JVM.

**getSources()** - возвращает изменяемый набор исходных текстов, которые будут добавлены в ApplicationContext при run(String...) вызове.

**getWebApplicationType()** - возвращает тип запускаемого веб-приложения.

**load(ApplicationContext context, Object[] sources**) - загрузите компоненты в контекст приложения.

**logStartupInfo(boolean isRoot)** - вызываемые для регистрации информации о запуске, подклассы могут переопределяться для добавления дополнительного ведения журнала.

**logStartupProfileInfo(ConfigurableApplicationContext context)** - вызывается для регистрации информации активного профиля.

**main(String[] args)** - базовый main, который можно использовать для запуска приложения.

**postProcessApplicationContext(ConfigurableApplicationContext context)** - примените любую соответствующую последующую обработкуApplicationContext.

**refresh(ConfigurableApplicationContext applicationContext)** - обновите базовый ApplicationContext.

**registerLoggedException(Throwable exception)** - зарегистрируйте, что данное исключение было зарегистрировано.

**run(Class<?>[] primarySources, String[] args)** - статический помощник, который может использоваться для запуска SpringApplication из указанных источников с использованием настроек по умолчанию и предоставленных пользователем аргументов.

**run(Class<?> primarySource, String... args) -** статический помощник, который можно использовать для запуска SpringApplication из указанного источника с использованием настроек по умолчанию.

**run(String... args)** - запустите приложение Spring, создав и обновив новоеApplicationContext.

**setAddCommandLineProperties(boolean addCommandLineProperties)** - задает, следует ли добавлять CommandLinePropertySource в контекст приложения для предоставления аргументов.

**setAddConversionService(boolean addConversionService)** - задает, следует ли ApplicationConversionService добавлять в контекст приложения Environment.

**setAdditionalProfiles(String... profiles)** - задайте дополнительные значения профиля для использования (поверх тех, которые заданы в свойствах системы или командной строки).

**setAllowBeanDefinitionOverriding(boolean allowBeanDefinitionOverriding)** - задает, должно ли быть разрешено переопределение определения компонента путем регистрации определения с тем же именем, что и существующее определение.

**setAllowCircularReferences(boolean allowCircularReferences)** - определяет, следует ли разрешать циклические ссылки между компонентами и автоматически пытаться разрешить их.

**setApplicationContextFactory(ApplicationContextFactory applicationContextFactory)** - задает фабрику, которая будет вызываться для создания контекста приложения.

**setApplicationStartup(ApplicationStartup applicationStartup)** - установите ApplicationStartup для использования для сбора показателей запуска.

**setBanner(Banner banner)** - задает Banner экземпляр, который будет использоваться для печати баннера, если не предоставлен статический файл баннера.

**setBannerMode(Banner.Mode bannerMode)** - устанавливает режим, используемый для отображения баннера при запуске приложения.

**setBeanNameGenerator(BeanNameGenerator beanNameGenerator)** - задает генератор имен компонентов, который следует использовать при генерации имен компонентов.

**setDefaultProperties(Map<String,Object> defaultProperties)** - установите свойства среды по умолчанию, которые будут использоваться в дополнение к существующим Environment.

**setDefaultProperties(Properties defaultProperties)** - удобная альтернатива setDefaultProperties(Map).

**setEnvironment(ConfigurableEnvironment environment)** - задает базовую среду, которая должна использоваться с созданным контекстом приложения.

**setEnvironmentPrefix(String environmentPrefix)** - установите префикс, который должен применяться при получении свойств конфигурации из системной среды.

**setHeadless(boolean headless)** - задает, является ли приложение безголовым и не должно ли оно создавать экземпляр AWT.

**setInitializers(Collection<? extends ApplicationContextInitializer<?>> initializers)** - задает, ApplicationContextInitializer что будет применено к Spring ApplicationContext.

**setLazyInitialization(boolean lazyInitialization)** - задает, следует ли лениво инициализировать компоненты.

**setListeners(Collection<? extends ApplicationListener<?>> listeners**) - задает ApplicationListener параметры, которые будут применены к SpringApplication и зарегистрированы в ApplicationContext.

**setLogStartupInfo(boolean logStartupInfo) -** задает, должна ли информация о приложении регистрироваться при запуске приложения.

**setMainApplicationClass(Class<?> mainApplicationClass) -** задайте конкретный класс основного приложения, который будет использоваться в качестве источника журнала и для получения информации о версии.

**setRegisterShutdownHook(boolean registerShutdownHook)** - задает, должен ли у созданного ApplicationContext быть зарегистрирован перехват завершения работы.

**setResourceLoader(ResourceLoader resourceLoader)** - задаетResourceLoader, которые должны использоваться при загрузке ресурсов.

**setSources(Set<String> sources)** - задайте дополнительные источники, которые будут использоваться для создания ApplicationContext.

**setWebApplicationType(WebApplicationType webApplicationType)** - задает тип веб-приложения, которое будет запущено.

**withHook(SpringApplicationHook hook, Runnable action)** - выполните указанное действие с данным SpringApplicationHook прикрепленным, если действие вызывает application run.

**withHook(SpringApplicationHook hook, ThrowingSupplier<T> action)** - выполните указанное действие с данным SpringApplicationHook прикрепленным, если действие вызывает application run.

**ApplicationArguments -** предоставляет доступ к аргументам, которые использовались для запуска SpringApplication.

**SpringApplication.Augmented -** используется для настройки и запуска расширенной системы, SpringApplication где должна быть применена дополнительная конфигурация.

**SpringApplication.Running -** предоставляет доступ к деталям SpringApplication запуска с использованием SpringApplication.Augmented.run(String...).

**SpringApplicationAotProcessor -**

точка входа для обработки AOT SpringApplication.

**SpringApplicationHook -** низкоуровневый хук, который можно использовать для прикрепления SpringApplicationRunListener к SpringApplication, чтобы наблюдать или изменять его поведение.

**SpringApplicationRunListener -** прослушиватель для SpringApplication run метода.

**SpringApplicationShutdownHandlers -** интерфейс, который можно использовать для добавления или удаления кода, который должен выполняться при завершении работы JVM.

**ApplicationContextFactory -** интерфейс стратегии для создания ConfigurableApplicationContext используется SpringApplication.

**ApplicationRunner -** интерфейс, используемый для указания того, что компонент должен выполнить когда он содержится в SpringApplication.

**Banner -** класс интерфейса для программного написания баннера.

**Banner.Mode -** перечисление возможных значений для настройки баннера.

**BootstrapContext -** простой контекст начальной загрузки, доступный во время запуска и Environment последующая обработка вплоть до того, что ApplicationContext готов.

**BootstrapContextClosedEvent -** applicationEvent опубликовано BootstrapContext когда он будет закрыт.

**BootstrapRegistry -** простой реестр объектов, который доступен во время запуска и Environment последующая обработка вплоть до того, что ApplicationContext готов.

**BootstrapRegistry.InstanceSupplier<T> -** поставщик используется для предоставления фактического экземпляра при необходимости.

**BootstrapRegistry.Scope -** область действия экземпляра.

**BootstrapRegistryInitializer -** интерфейс обратного вызова, который может использоваться для инициализации BootstrapRegistry перед его использованием.

**CommandLineRunner -** интерфейс, используемый для указания того, что компонент должен выполнить когда он содержится в SpringApplication.

**ConfigurableBootstrapContext -** bootstrapContextКоторый также предоставляет методы настройки через BootstrapRegistry интерфейс.

**DefaultApplicationArguments -** реализация по умолчанию ApplicationArguments.

**DefaultBootstrapContext -** сonfigurableBootstrapContextРеализация по умолчанию.

**DefaultPropertiesPropertySource -** mapPropertySource содержащий свойства по умолчанию, внесенные непосредственно в SpringApplication.

**ExitCodeEvent -** событие запускается, когда код выхода приложения был определен из ExitCodeGenerator.

**ExitCodeExceptionMapper -** интерфейс стратегии, который может использоваться для обеспечения сопоставления между исключениями и кодами выхода.

**ExitCodeGenerator -** интерфейс, используемый для генерации "кода выхода" из запущенной командной строки SpringApplication.

**LazyInitializationBeanFactoryPostProcessor -** beanFactoryPostProcessor установить lazy-init для определений компонентов, которые не являются excluded и еще не имели явно установленного значения.

**LazyInitializationExcludeFilter -** фильтр, который может использоваться для исключения того, чтобы определения beans не были lazy-init установлены LazyInitializationBeanFactoryPostProcessor.

**ResourceBanner -** реализация баннера, который печатается из исходного текста Resource.

**SpringBootConfiguration -** указывает, что класс предоставляет приложение Spring Boot @Configuration.

**SpringBootExceptionReporter -** интерфейс обратного вызова, используемый для поддержки пользовательских отчетов об SpringApplication ошибках запуска.

**SpringBootVersion -** предоставляет версию Spring Boot.

**WebApplicationType -** перечисление возможных типов веб-приложений.

**Решим задачи с сайта Spring:**

* **Создание веб-службы RESTful**

**Надо изучить:**

**https://spring.io/guides/tutorials/spring-boot-kotlin/**

[**https://spring.io/guides/tutorials/spring-webflux-kotlin-rsocket/**](https://spring.io/guides/tutorials/spring-webflux-kotlin-rsocket/)

**https://spring.io/guides/gs/graphql-server/**