

# Spring boot: назначение, внутреннее устройство



Григорий  
Скобелев



**Григорий Скобелев**

Java Developer, ЮMoney



Григорий Скобелев



# План занятия

1. [Spring boot](#)
2. [Автоконфигурации](#)
3. [Конфигурирование](#)
4. [Spring boot actuator](#)
5. [Итоги](#)
6. [Домашнее задание](#)



# Spring boot

---

# Spring boot

**Spring boot** - это современный framework для создания Java приложений, который отличается высоким уровнем абстракции и предоставляет быстрый способ создания production-ready приложений.



---

# Ключевые достоинства Spring boot

К ключевым особенностям Spring boot можно отнести:

- **Stand-alone приложения.** Помогает легко и быстро создавать stand-alone приложения, использующие Spring.
- **Embedded containers.** Теперь Spring сам за вас запускает Tomcat, который уже находится в вашем jar файле.
- **Автоконфигурации.** Spring boot заботится за вас о создании часто используемых бинов.
- **'Starter' dependencies.** Большое количество зависимостей для gradle и maven, в которых содержится все необходимое для быстрого запуска приложения.
- **Metrics.** Из коробки Spring boot вам предлагает возможность для сбора метрик вашего приложения.

# Пример простого приложения

Основная аннотация  
Spring boot

Стартовый класс приложения

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

# Пример простого приложения

Основная аннотация  
Spring boot

Стартовый класс приложения

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```



# Пример простого приложения

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

```
@Configuration
@EnableAutoConfiguration
@ComponentScan
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Оба варианта  
эквивалентны

---

# @SpringBootApplication

Функционал, включаемый аннотацией `@SpringBootApplication` по умолчанию:

- **Конфигурирование.** Класс может содержать декларацию бинов с помощью `@Bean`.
- **Автоконфигурации.** Включает функционал автоконфигурирования.
- **Поиск бинов и конфигураций.** С текущего пакета начнется поиск классов, помеченных аннотациями `@Component` или `@Configuration`.

# @SpringBootApplication

Маркер класса конфигураций

Аннотация для включения  
автоконфигураций

```
@SpringBootConfiguration  
@EnableAutoConfiguration  
@ComponentScan(excludeFilters = { @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),  
    @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })  
public @interface SpringBootApplication {
```

Основная аннотация Spring boot,  
которой мы помечаем класс с main  
методом

---

# @SpringBootApplication

```
/** Exclude specific auto-configuration classes such that they will never be applied. ...*/  
@AliasFor(annotation = EnableAutoConfiguration.class)  
Class<?>[] exclude() default {};
```

```
/** Exclude specific auto-configuration class names such that they will never be ...*/  
@AliasFor(annotation = EnableAutoConfiguration.class)  
String[] excludeName() default {};
```

```
/** Base packages to scan for annotated components. Use {@link #scanBasePackageClasses} ...*/  
@AliasFor(annotation = ComponentScan.class, attribute = "basePackages")  
String[] scanBasePackages() default {};
```

```
/** Type-safe alternative to {@link #scanBasePackages} for specifying the packages to ...*/  
@AliasFor(annotation = ComponentScan.class, attribute = "basePackageClasses")  
Class<?>[] scanBasePackageClasses() default {};
```

---

Аннотация @AliasFor нужна, чтобы значение атрибута текущей аннотации применялось к аннотации, которая лежит в атрибуте annotation

---

# @SpringBootApplication

Функционал, настраиваемый в аннотации

**@SpringBootApplication:**

- Выключение автоконфигураций через указание класса или через указание полное название класса.
- Указание пакета, с которого следует начать поиск классов, аннотированных **@Component** или **@Configuration**.

# Старт приложения

## Версия spring boot

[illegible]

```

2020-10-21 00:29:06.900 INFO 34721 --- [main] ru.sbt.spring.Application : Starting Application on MacBook-Pro-Bocarov.local with PID 34721
(/Users/ivan/spring-sbt-maven/target/classes started by ivan in /Users/ivan/spring-sbt-maven)
2020-10-21 00:29:06.905 INFO 34721 --- [main] ru.sbt.spring.Application : No active profile set, falling back to default profiles: default
2020-10-21 00:29:06.967 INFO 34721 --- [main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context
.AnnotationConfigServletWebServerApplicationContext@436a4e4b: startup date [Wed Oct 21 00:29:06 MSK 2020]; root of context hierarchy
2020-10-21 00:29:07.752 INFO 34721 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2020-10-21 00:29:07.767 INFO 34721 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2020-10-21 00:29:07.767 INFO 34721 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.32
2020-10-21 00:29:08.238 INFO 34721 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup
2020-10-21 00:29:08.272 INFO 34721 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2020-10-21 00:29:08.275 INFO 34721 --- [main] ru.sbt.spring.Application : Started Application in 1.669↑seconds (JVM running for 2.338)

```

## Порт на котором был запущен веб контейнер



# Автоконфигурации

# Автоконфигурации

**Автоконфигурации** - это механизм, появившийся вместе со Spring boot, который конфигурирует и добавляет бины в ApplicationContext на основе зависимостей, который есть в classpath приложения. Включается аннотацией **@EnableAutoConfiguration**.



# Пример автоконфигурации

```
@Configuration
@ConditionalOnClass(ObjectMapper.class)
public class JacksonAutoConfiguration {

    private static final Map<?, Boolean> FEATURE_DEFAULTS;

    static {...}

    @Bean
    public JsonComponentModule jsonComponentModule() { return new JsonComponentModule(); }

    @Configuration
    @ConditionalOnClass(Jackson20objectMapperBuilder.class)
    static class JacksonObjectMapperConfiguration {...}

    @Configuration
    @ConditionalOnClass({ Jackson20objectMapperBuilder.class, DateTime.class,
        DateTimeSerializer.class, JacksonJodaDateFormat.class })
    static class JodaDateTimeJacksonConfiguration {
```



## @Conditional

**Аннотация Conditional** - это механизм, который используется для включения в контекст приложения те или иные бины в зависимости от выполнения заданных условий, заданных в классе, реализующем интерфейс Condition.

---

# @Conditional

```
@Conditional(ProfileCondition.class)
public @interface Profile {

    class ProfileCondition implements Condition {

        @Override
        public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
            MultiValueMap<String, Object> attrs = metadata.getAllAnnotationAttributes(Profile.class.getName());
            if (attrs != null) {
                for (Object value : attrs.get("value")) {
                    if (context.getEnvironment().acceptsProfiles((String[]) value)) {
                        return true;
                    }
                }
                return false;
            }
            return true;
        }
    }
}
```

---

# Виды @Conditional

- **@ConditionalOnBean** - регистрация бина в контексте, в случае если в контексте уже присутствует бин с заданным в аннотации названием
- **@ConditionalOnMissingBean** - регистрация бина в контексте, в случае если в контексте еще не зарегистрирован бин с заданным в аннотации названием
- **@ConditionalOnClass** - регистрация бина в контексте, в случае если в classpath есть класс, заданный в аннотации
- **@ConditionalOnProperty** - регистрация бина в контексте, в случае если задан определенный параметр в application.properties



# Конфигурирование

# Конфигурирование

Spring Boot позволяет **реализовать внешнюю конфигурацию**, чтобы вы могли работать с одним и тем же кодом приложения в разных средах.

Вы можете использовать различные внешние источники конфигурации, включая файлы, переменные среды и аргументы командной строки. По умолчанию это файл **application.properties**, в котором в формате ключ-значение заданы параметры для вашего приложения. Этот механизм, в частности, используется в автоконфигурациях.

---

Также можно задавать параметры конфигураций в файле `application.yaml`, и у него есть некоторые преимущества перед `application.properties`

# Пример конфигурирования

```
server.port=8080
spring.datasource.schema=testdb
netology.config.name=netology
netology.config.password=1234
```

Параметры для  
автоконфигураций

Пользовательские  
параметры

Все параметры, которые используются для автоконфигураций бинов Spring, можно посмотреть в файле `spring-configuration-metadata.json`. Там же есть и дефолтные значения некоторых из них.

# Пример конфигурирования

```
app.prop=spring
app.new.prop=${app.prop}/additional
netology.config.name[0]=ivan
netology.config.name[1]=sergey
netology.config.map[first]=sergey
netology.config.map[second]=sergey
netology.config.password=1234
```

Использование  
placeholder'a

Параметры в виде List

Параметры в виде  
Map



# Порядок поиска

Порядок поиска application.(properties|yaml)  
по директориям:

- Корень classpath
- Папка /config в classpath
- Текущая директория
- Папка /config в текущей директории
- Дочерние директории поддиректории /config

# @ConfigurationProperties

Позволяет внедрять пользовательские и параметры Spring с определенным префиксом в объекты:

- в поля объекта;
- в конструктор с помощью `@ConstructorBinding`

```
@ConfigurationProperties(prefix = "my.properties")
public class MyProperties {

    private boolean enabled;

    private int id;

    private String name;

    public boolean isEnabled() { return enabled; }

}
```

# Relaxed binding

**Relaxed binding** позволяет использовать параметры в различных форматах:

- kebab case: `my-property.id`(рекомендован)
- camel case: `myProperty.id`
- snake case: `my_property.id`
- upper underscore: `MY_PROPERTY.ID`

---

# @Value

Позволяет внедрять пользовательские параметры в поля:

- подставляет значения по полному совпадению placeholder'a
- через двоеточие можно указать дефолтное значение, если этот параметр не будет найден

```
public class MyProperties {  
  
    @Value("my.properties.enabled:true")  
    private boolean enabled;  
  
    @Value("my.properties.id:1111")  
    private int id;  
  
    @Value("my.properties.name")  
    private String name;  
  
    public boolean isEnabled() { return enabled; }  
  
}
```

# @ConfigurationProperties vs @Value

Сравнения функциональности:

Фича	@ConfigurationProperties	@Value
Relaxed binding	Да	Ограниченно
Использование SpEL	Нет	Да



# Spring boot actuator



## Spring boot actuator

Spring Boot включает ряд дополнительных функций, которые помогут вам **контролировать и управлять вашим приложением**.

Вы можете управлять своим приложением с помощью HTTP endpoint или JMX.

# Endpoint

Actuator предоставляет следующие модули, для мониторинга:

- **configprops** - показывает весь список параметров приложения
- **health** - показывает состояние приложения
- **metrics** - показывает специальные метрики приложения
- и другие



# Включение Endpoint

- По умолчанию все метрики включены, кроме shutdown
- Включение/выключение какого-либо endpoint производится проставлением значения параметра в application.properties

```
management.endpoint.shutdown.enabled=true
```

Название  
endpoint'a

Состояние  
endpoint'a



# Итоги



## Итоги

Spring boot является одним из самых удобных фреймворков для прототипирования и сборки production-ready приложений. Также, он заботится о том, чтобы вы не писали каждый раз один и те же бины, и могли все это конфигурировать в одном файле.

---

# Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

Григорий Скобелев