БАНК

Цена докторской степени по Spring

Содержание доклада

- Что есть Spring Boot и его накопившиеся болячки
- Обсудим особенности Spring Boot разработки
- Протестируем Spring Boot
- По чем докторская?
- Как мы пришли к Kora

Инструментарий

Spring внедрение: хотим

```
@Service
public class SomeService {

    private final OtherService otherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Spring внедрение: имеем

```
@Service
public class SomeService {

    private final OtherService otherService;
    @Autowired
    private OtherService someOtherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Spring внедрение: имеем

```
@Service
public class SomeService {

    private final OtherService otherService;
    @Autowired
    private SomeOtherService someOtherService;
    private SomeOtherOtherService someOtherOtherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }

    @Autowired
    public void setSomeOtherService(SomeOtherService someOtherService) {
        this.someOtherService = someOtherService;
    }
}
```

Spring внедрение: имеем

```java
@Service
public class SomeService {

    private final OtherService otherService;
    @Autowired
    private SomeOtherService someOtherService;
    private SomeOtherOtherService someOtherOtherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }

    @Autowired
    public void setSomeOtherService(SomeOtherService someOtherService) {
        this.someOtherService = someOtherService;
    }
}
```

```xml
<bean id="exampleBean" class="examples.ExampleBean">
    <!-- constructor injection using the nested ref element -->
    <constructor-arg>
        <ref bean="anotherExampleBean"/>
    </constructor-arg>

    <!-- constructor injection using the neater ref attribute -->
    <constructor-arg ref="yetAnotherBean"/>
    <constructor-arg type="int" value="1"/>
</bean>

<bean id="anotherExampleBean" class="examples.AnotherBean"/>
<bean id="yetAnotherBean" class="examples.YetAnotherBean"/>
```

Spring внедрение: следим

```
@Service
public class SomeService {

    private final OtherService otherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Spring внедрение: следим

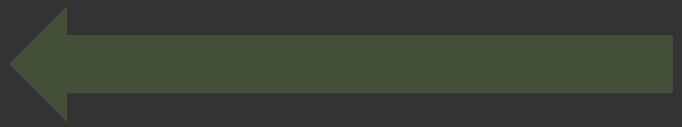
```
@Service
public class SomeService {

    private final OtherService otherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Spring внедрение точечно: хотим

```
@Configuration
public class BeanFactory {

  @Qualifier("c1")
  @Bean
  public String someStr() {
    return "one";
  }

  @Qualifier("c1")
  @Bean
  public String someOtherStr() {
    return "two";
  }

  @Bean
  public Supplier<String> combiner(@Qualifier("c1") List<String> strs) {
    return () -> String.join(", ", strs);
  }
}
```

Spring внедрение точечно: имеем

```
@Configuration
public class BeanFactory {

  @Qualifier("c1")
  @Bean
  public String someStr() {
    return "one";
  }

  @Qualifier("c1")
  @Bean
  public String someOtherStr() {
    return "two";
  }

  @Bean
  public Supplier<String> combiner(@Qualifier("c1") List<String> strs) {
    return () -> String.join(", ", strs);
  }
}
```



Русская буква ‘с’

Spring внедрение точечно: имеем

```
@Configuration
public class BeanFactory {

  @Bean(name = "c1")
  public String someStr() {
    return "one";
  }

  @Bean(name = "c1")
  public String someOtherStr() {
    return "two";
  }

  @Bean
  public Supplier<String> combiner(@Qualifier("c1") List<String> strs) {
    return () -> String.join(", ", strs);
  }
}
```

Spring внедрение точечно: следим

```
@Configuration
public class BeanFactory {

    public static final String TAG_C1 = "c1";

    @Qualifier(TAG_C1)
    @Bean
    public String someStr() {
        return "one";
    }

    @Qualifier(TAG_C1)
    @Bean
    public String someOtherStr() {
        return "two";
    }

    @Bean
    public Supplier<String> combiner(@Qualifier(TAG_C1) List<String> strs) {
        return () -> String.join(", ", strs);
    }
}
```

Spring внедрение точечно: следим строго

```java
@Configuration
public class BeanFactory {

  @Retention(RetentionPolicy.RUNTIME)
  @Qualifier
  public @interface SomeTag { }

  @SomeTag
  @Bean
  public String someStr() {
    return "one";
  }

  @SomeTag
  @Bean
  public String someOtherStr() {
    return "two";
  }

  @Bean
  public Supplier<String> combiner(@SomeTag List<String> strs) {
    return () -> String.join(", ", strs);
  }
}
```

Spring внедрение точечно: следим строго

```
@Configuration
public class BeanFactory {

  @Retention(RetentionPolicy.RUNTIME)
  @Qualifier
  public @interface SomeTag { }

  @SomeTag
  @Bean
  public String someStr() {
    return "one";
  }

  @SomeTag
  @Bean
  public String someOtherStr() {
    return "two";
  }

  @Bean
  public Supplier<String> combiner(@SomeTag List<String> strs) {
    return () -> String.join(", ", strs);
  }
}
```

Spring фабрика: хотим

```java
@Configuration
public class BeanFactoryOne {

  @Primary
  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return "one";
  }

  @Qualifier("c1")
  @Bean
  public String getBeanOtherStr() {
    return "two";
  }
}
```

```java
@Configuration
public class BeanFactoryTwo {


  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return ”three";
  }

  @Qualifier("c1")
  @Bean
  public String getAllBeans(List<String> strs) {
    return String.join(", ", strs);
  }
}
```

Spring фабрика: имеем

```java
@Configuration
public class BeanFactoryOne {

  @Primary
  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return "one";
  }

  @Qualifier("c1")
  @Bean
  public String getBeanOtherStr() {
    return "two";
  }
}
```

```java
@Configuration
public class BeanFactoryTwo {


  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return "three";
  }

  @Qualifier("c1")
  @Bean
  public String getAllBeans(List<String> strs) {
    return String.join(", ", strs);
  }
}
```

getBeanStr()

```
***************************
APPLICATION FAILED TO START
***************************

Description:

The bean 'getBeanStr', defined in class path resource [ru/tinkoff/spring/basic/FactoryPrototype$BeanFactoryTwo.class], could not be registered. A bean with that name has already been defined in class path resource [ru/tinkoff/spring/basic/FactoryPrototype$BeanFactoryOne.class] and overriding is disabled.
```

Spring фабрика: следим

```java
@Configuration
public class BeanFactoryOne {

  @Primary
  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return "one";
  }

  @Qualifier("c1")
  @Bean
  public String getBeanOtherStr() {
    return "two";
  }
}
```

```java
@Configuration
public class BeanFactoryTwo {


  @Qualifier("c1")
  @Bean
  public String getBeanStr() {
    return ”three";
  }

  @Qualifier("c1")
  @Bean
  public String getAllBeans(List<String> strs) {
    return String.join(", ", strs);
  }
}
```

Spring жизненный цикл: хотим

```
@Service
public class SomeService {

}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy

```
@Service
public class SomeService {

    @PostConstruct
    void setup() {
        System.out.println("Initializing...");
    }

    @PreDestroy
     void closeup() {
        System.out.println("Releasing...");
    }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope

```
@RequestScope
@Service
public class SomeService {

    @PostConstruct
     void setup() {
        System.out.println("Initializing...");
    }

    @PreDestroy
    void closeup() {
        System.out.println("Releasing...");
    }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя

```
@RequestScope
@Service
public class SomeService extends ParentService {

   @PostConstruct
    void setup() { System.out.println("Init..."); }

   @PreDestroy
    void closeup() { System.out.println("Close..."); }
}

public abstract class ParentService {

   @PostConstruct
    void setupParent() { System.out.println("Init p..."); }

   @PreDestroy
    void closeupParent() { System.out.println("Close p..."); }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса

```
@RequestScope
@Service
public class SomeService extends ParentService {

  private int state = 0;

  @PostConstruct
   void setup() {
     this.getStateAndInc();
     super.parentState++;
     System.out.println("Initializing...");
  }

  @PreDestroy
   void closeup() {
     System.out.println("Releasing...");
  }

  int getStateAndInc() { return state++; }
}

public abstract class ParentService {

  private int parentState = 0;

  @PostConstruct
   void setupParent() {  System.out.println("Initializing parent..."); }

  @PreDestroy
   void closeupParent() {  System.out.println("Releasing parent..."); }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса
        - Используем SmartLifecycle

```
@RequestScope
@Service
public class SomeService extends ParentService implements  SmartLifecycle {

  private int state = 0;

  @PostConstruct
   void setup() {
     this.getStateAndInc();
     super.parentState++;
     System.out.println("Initializing...");
  }

  @PreDestroy
   void closeup() {  System.out.println("Releasing...");    }

   int getStateAndInc() {
     return state++;
  }

   void start() {getStateAndInc(); }
   void stop() { state--; }
}

public abstract class ParentService {

   private int parentState = 0;

  @PostConstruct
   void setupParent() {   System.out.println("Initializing parent..."); }

  @PreDestroy
   void closeupParent() {  System.out.println("Releasing parent...");  }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса
        - Используем SmartLifecycle
          - Используем аспекты

```
@RequestScope
@Service
public class SomeService extends ParentService implements  SmartLifecycle {

  private int state = 0;

  @PostConstruct
   void setup() {
     this.getStateAndInc();
     super.parentState++;
     System.out.println("Initializing...");
   }

  @PreDestroy
   void closeup() {  System.out.println("Releasing...");    }

  @Loggable
   int getStateAndInc() { return state++;  }

   void start() {getStateAndInc(); }
   void stop() { state--; }
}

public abstract class ParentService {

  private int parentState = 0;

  @PostConstruct
   void setupParent() {   System.out.println("Initializing parent..."); }

  @PreDestroy
   void closeupParent() {  System.out.println("Releasing parent...");  }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса
        - Используем SmartLifecycle
          - Используем аспекты внутри аспекта

```
@RequestScope
@Service
public class SomeService extends ParentService implements  SmartLifecycle {

   private int state = 0;

   @PostConstruct
    void setup() {
      this.getStateAndInc();
      super.parentState++;
      System.out.println("Initializing...");
   }

   @PreDestroy
    void closeup() {  System.out.println("Releasing...");    }

   @Loggable
    int getStateAndInc() { return this.getParentStateAndInc() + state++;  }

    void start() {getStateAndInc(); }
    void stop() { state--; }
}

public abstract class ParentService {

   private int parentState = 0;

   @PostConstruct
    void setupParent() {   System.out.println("Initializing parent..."); }

   @PreDestroy
    void closeupParent() {  System.out.println("Releasing parent...");  }

   @Loggable
    int getParentStateAndInc() { return parentState++;  }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса
        - Используем SmartLifecycle
          - Используем аспекты внутри аспекта
            - Используем аспекты над @PostConstruct

```
@RequestScope
@Service
public class SomeService extends ParentService implements  SmartLifecycle {

   private int state = 0;

   @PostConstruct @Loggable
    void setup() {
      this.getStateAndInc();
      super.parentState++;
      System.out.println("Initializing...");
   }

   @PreDestroy @Loggable
    void closeup() {  System.out.println("Releasing...");  }

   @Loggable
    int getStateAndInc() { return this.getParentStateAndInc() + state++; }

    void start() {getStateAndInc(); }
    void stop() { state--; }
}

public abstract class ParentService {

   private int parentState = 0;

   @PostConstruct @Loggable
    void setupParent() {   System.out.println("Initializing parent..."); }

   @PreDestroy @Loggable
    void closeupParent() {  System.out.println("Releasing parent...");  }

   @Loggable
    int getParentStateAndInc() { return parentState++;  }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy
  - Singleton
  - Prototype
  - Request Scope
  - Session Scope
    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя
      - Обращаемся к поля класса
      - Обращаемся к методам класса
        - Используем SmartLifecycle
          - Используем аспекты внутри аспекта
            - Используем аспекты над @PostConstruct
              - Используем аспект в классе которые наследует интерфейсы

```
@Service
public class SomeService extends ParentService implements  SmartLifecycle,
                                                            Comparable<SomeService> {
  private int state = 0;

  @PostConstruct @Loggable
  void setup() {
    this.getStateAndInc();
    super.parentState++;
    System.out.println("Initializing...");
  }

  @PreDestroy @Loggable
  void closeup() {  System.out.println("Releasing...");  }

  @Loggable
  int getStateAndInc() { return this.getParentStateAndInc() + state++; }

  void start() {getStateAndInc(); }
  void stop() { state--; }

  @Override
  public int compareTo(SomeService o) { return Integer.compare(state, o.state); }
}

public abstract class ParentService {

  private int parentState = 0;

  @PostConstruct @Loggable
  void setupParent() {   System.out.println("Initializing parent..."); }

  @PreDestroy @Loggable
  void closeupParent() {  System.out.println("Releasing parent...");  }

  @Loggable
  int getParentStateAndInc() { return parentState++;  }
}
```

Spring жизненный цикл: имеем

- @PostConstruct
- @PreDestroy

  - Singleton
  - Prototype
  - Request Scope
  - Session Scope

    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя

      - Обращаемся к поля класса
      - Обращаемся к методам класса

        - Используем SmartLifecycle

          - Используем аспекты внутри аспекта

            - Используем аспекты над @PostConstruct

              - Используем аспект в классе
                которые наследует интерфейсы

Spring жизненный цикл: думаем

- @PostConstruct
- @PreDestroy

  - Singleton
  - Prototype
  - Request Scope
  - Session Scope

    - @PostConstruct у класса родителя
    - @PreDestroy у класса родителя

      - Обращаемся к поля класса
      - Обращаемся к методам класса

        - Используем SmartLifecycle

          - Используем аспекты внутри аспекта

            - Используем аспекты над @PostConstruct

              - Используем аспект в классе
                которые наследует интерфейсы

Kora: приятно познакомится

Kora - фреймворк общего назначения для написания серверных приложений

Kora внедряем: хотим

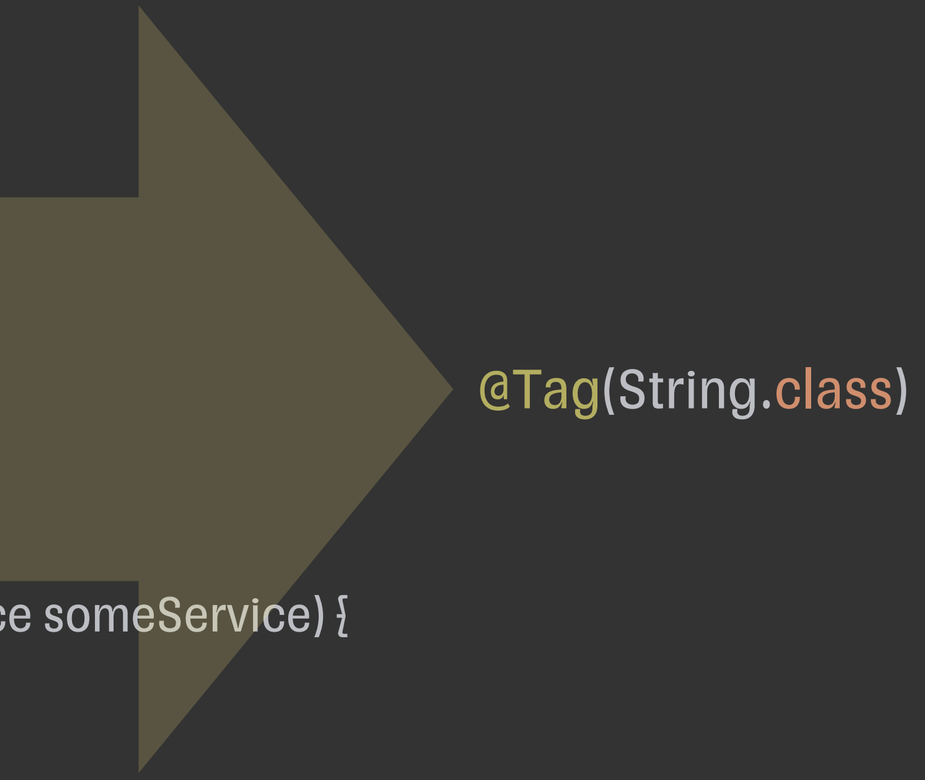
```
@Component
public class SomeService {

    private final OtherService otherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Kora внедряем: имеем

```
@Component
public class SomeService {

    private final OtherService otherService;

    public SomeService(OtherService otherService) {
        this.otherService = otherService;
    }
}
```

Kora внедряем: точечно

```
@Module
public interface SomeModule {

  @Tag(String.class)
  default SomeService someService() {
    return new SomeService();
  }

  @Tag(String.class)
  default OtherService otherService(SomeService someService) {
    return new OtherService(someService);
  }
}
```



Kora жизненный цикл: имеем

```java
@Component
public static class SomeService extends ParentService implements Comparable<SomeService> {

  private int state = 0;

  @Log
  @Override
  public void init() throws Exception {
    super.init();
    this.getStateAndInc();
    super.parentState++;
    System.out.println("Initializing...");
  }

  @Log
  @Override
  public void release() throws Exception {
    super.release();
    System.out.println("Releasing...");
  }

  @Log
  public int getStateAndInc() {
    return state++;
  }

  @Log
  @Override
  public int getParentStateAndInc() {
    return super.getParentStateAndInc();
  }

  @Override
  public int compareTo(SomeService o) {
    return Integer.compare(state, o.state);
  }
}
```

```java
public abstract static class ParentService implements Lifecycle {

  protected int parentState = 0;

  @Override
  public void init() throws Exception {  System.out.println("Initializing..."); }

  @Override
  public void release() throws Exception { System.out.println("Releasing...");  }

  public int getParentStateAndInc() { return parentState++; }
}
```

Kora жизненный цикл: имеем

```java
@Component
public static class SomeService extends ParentService implements Comparable<SomeService> {

  private int state = 0;

  @Log
  @Override
  public void init() throws Exception {
    super.init();
    this.getStateAndInc();
    super.parentState++;
    System.out.println("Initializing...");
  }

  @Log
  @Override
  public void release() throws Exception {
    super.release();
    System.out.println("Releasing...");
  }

  @Log
  public int getStateAndInc() {
    return state++;
  }

  @Log
  @Override
  public int getParentStateAndInc() {
    return super.getParentStateAndInc();
  }

  @Override
  public int compareTo(SomeService o) {
    return Integer.compare(state, o.state);
  }
}
```

```java
public abstract static class ParentService implements Lifecycle {

  protected int parentState = 0;

  @Override
  public void init() throws Exception {  System.out.println("Initializing..."); }

  @Override
  public void release() throws Exception { System.out.println("Releasing...");  }

  public int getParentStateAndInc() { return parentState++; }
}
```

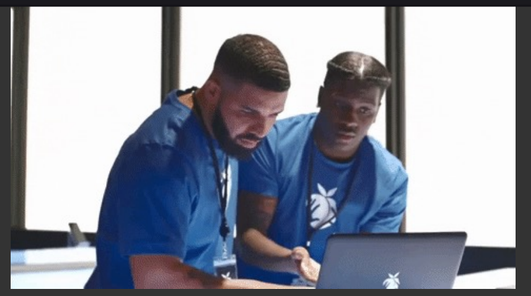
Kora жизненный цикл: имеем

```
    @Log
    @Override
    public void init() throws Exception {
        super.init();
        System.out.println("Initializing...");
    }

    @Log
    @Override
    public void release() throws Exception {
        super.release();
        System.out.println("Releasing...");
    }
}
```

Kora жизненный цикл: имеем

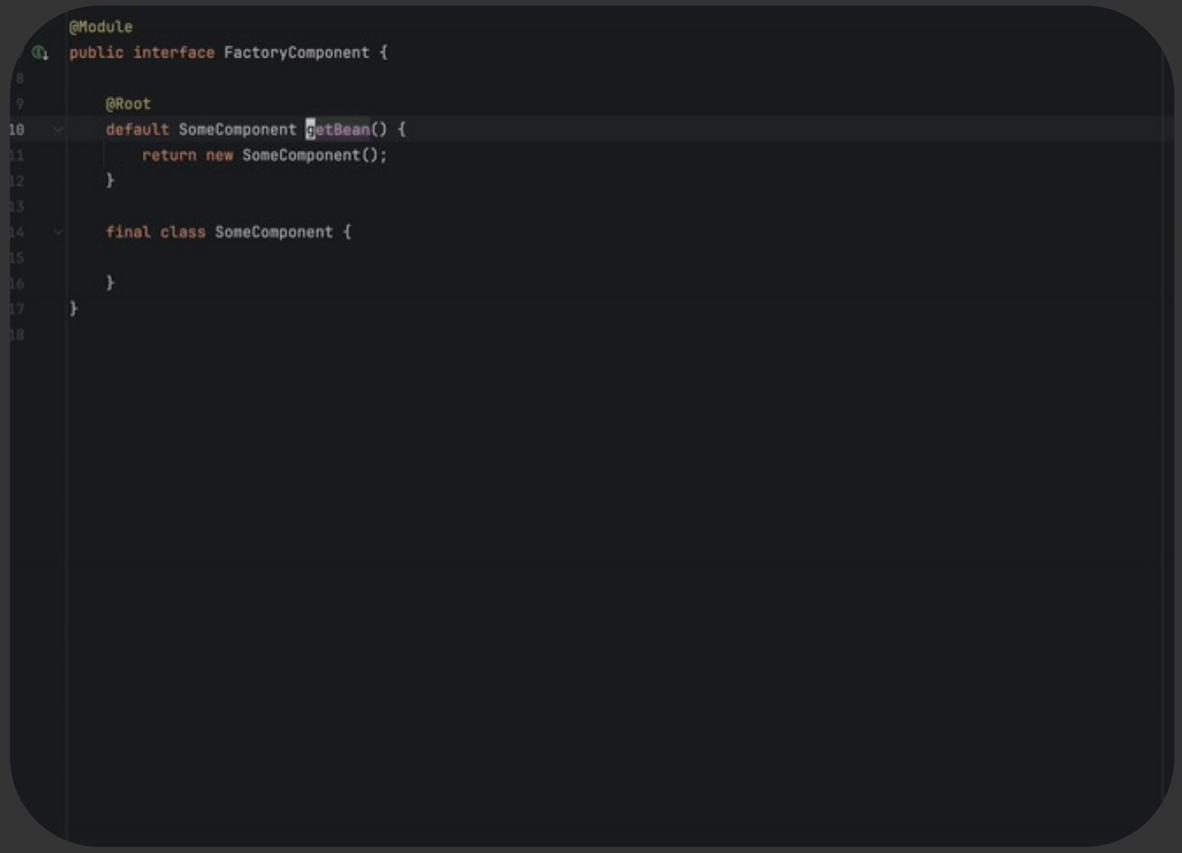
```
    @Log
    @Override
    public void init() throws Exception {
        super.init();
        System.out.println("Initializing...");
    }

    @Log
    @Override
    public void release() throws Exception {
        super.release();
        System.out.println("Releasing...");
    }
}
```

```
@Root
@Component
public static class SomeService extends ParentService implements

    private int state = 0;

    @Log
    @Override
    public void init() throws Exception {
        super.init();
        System.out.println("Initializing...");
    }

    @Log
    @Override
    public void release() throws Exception {
        super.release();
        System.out.println("Releasing...");
    }
```



Kora жизненный цикл и внедрение

- Обработчики аннотаций как основной столп на котором строится фреймворк Kora
- Контейнер зависимостей создается и проверяется на этапе компиляции кода
- Не используется ни Reflection API, ни генерация байт-кода, ни динамические прокси, только прозрачный исходный код



Spring конфигурация: хотим

Типы конфигураций и настроек:

1. По умолчанию
2. Системные переменные
3. Файл конфигурации
4. Переменных окружения
5. Переопределять в тестах

Spring конфигурация: имеем

Источники рассматриваются в следующем порядке:

1. Свойства по умолчанию (задаются установкой SpringApplication.setDefaultProperties).
2. Аннотации @PropertySource на ваших классах @Configuration. Обратите внимание, что такие источники свойств не добавляются в среду до тех пор, пока контекст приложения не обновится. Это слишком поздно для настройки некоторых свойств, таких как logging.* и spring.main.*, которые считываются до начала обновления.
3. Данные конфигурации (например, файлы application.properties).
4. RandomValuePropertySource, который имеет свойства только в random.*.
5. Переменные окружения ОС.
6. Свойства Java System (System.getProperties()).
7. Атрибуты JNDI из java:comp/env.
8. Параметры инициализации ServletContext.
9. Параметры инициализации ServletConfig.
10. Свойства из SPRING_APPLICATION_JSON (встроенный JSON, встроенный в переменную окружения или системное свойство).
11. Аргументы командной строки.
12. Атрибуты свойств в ваших тестах. Доступен в @SpringBootTest и тестовых аннотациях для тестирования определенного фрагмента вашего приложения.
13. Аннотации @DynamicPropertySource в ваших тестах.
14. Аннотации @TestPropertySource в ваших тестах.
15. Свойства глобальных настроек Devtools в каталоге $HOME/.config/spring-boot, когда devtools активен.

Файлы данных конфигурации рассматриваются в следующем порядке:

1. Свойства приложения, упакованные в ваш jar (application.properties и варианты YAML).
2. Свойства приложения для конкретного профиля, упакованные в jar (application-{profile}.properties и варианты YAML).
3. Свойства приложения вне упакованного jar (application.properties и варианты YAML).
4. Свойства приложения, специфичные для профиля, вне вашего упакованного jar (application-{profile}.properties и варианты YAML).

Spring конфигурация: имеем

15

источников конфигурации

4

источника файлов конфигурации

Spring конфигурация: имеем

15

источников конфигурации

4

источника файлов конфигурации

5

вариантов мягкой привязки свойств

Spring конфигурация: имеем

Все варианты привязываются к свойству **hostName**:

1. hostName
2. hostname
3. host_name
4. host-name
5. HOST_NAME

Spring конфигурация: имеем

Все варианты привязываются к свойству **hostName**:

1. hostName
2. hostname
3. host_name
4. host-name
5. HOST_NAME

15
источников конфигурации

4
источника файлов конфигурации

5
вариантов мягкой привязки свойств

Spring конфигурация: имеем

```
@Configuration
@ConfigurationProperties("foo")
public class Config {

    private String bar;

    public String getBar() {
        return bar;
    }

    public void setBar(String bar) {
        this.bar = bar;
    }
}
```

Spring конфигурация: имеем

```
@Component
public class Config {

    private final String bar;

    public Config(@Value("staticValue") String bar) {
        this.bar = bar;
    }

    public String getBar() {
        return bar;
    }
}
```

Spring конфигурация: имеем

```
@Component
public class Config {

    private final String bar;

    public Config(@Value("${foo.bar}") String bar) {
        this.bar = bar;
    }

    public String getBar() {
        return bar;
    }
}
```

Spring конфигурация: имеем

```
@Component
public class Config {

    private final String bar;

    public Config(@Value("${foo.bar:defaultValue}") String bar) {
        this.bar = bar;
    }

    public String getBar() {
        return bar;
    }
}
```

Spring конфигурация: имеем

```
@Component
public class Config {

    private final String bar;

    public Config(@Value("#{'${foo.bar}'.replace('source', 'awesome')}") String bar) {
        this.bar = bar;
    }

    public String getBar() {
        return bar;
    }
}
```

Spring конфигурация: имеем

```
@Validated
@Configuration
@ConfigurationProperties("foo")
public class ConfigClassValid {

    @NotNull
    private String bar;
    private String baz;
    private String bag = "someBag";
}
```

Требуется подключать отдельно

По умолчанию не обязательно

Spring конфигурация: имеем

```
@Validated
@Configuration
@ConfigurationProperties("foo")
public class ConfigClassValid {

    @NotNull
    private String bar;
    private String baz;
    private String bag = "someBag";
}
```

Kora конфигурация: имеем

```
@ConfigSource("foo")
public interface FooConfig {

    String bar();

    @Nullable
    String baz();

    default String bag() {
        return "someBag";
    }
}
```

По умолчанию обязательно

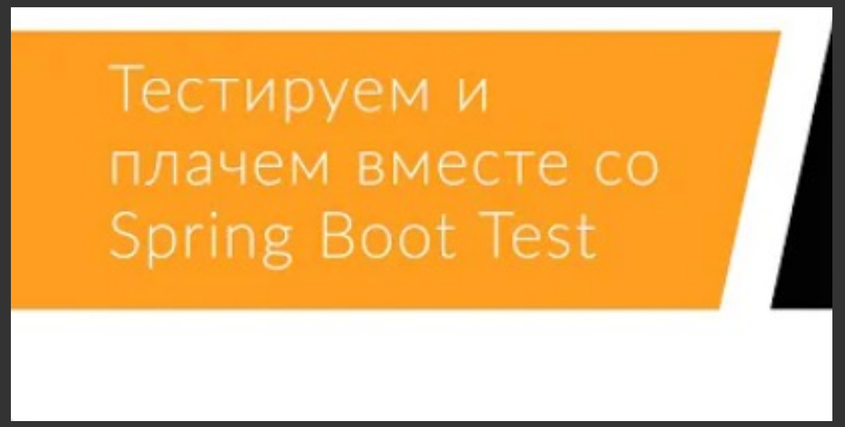
Spring модульные тесты: имеем

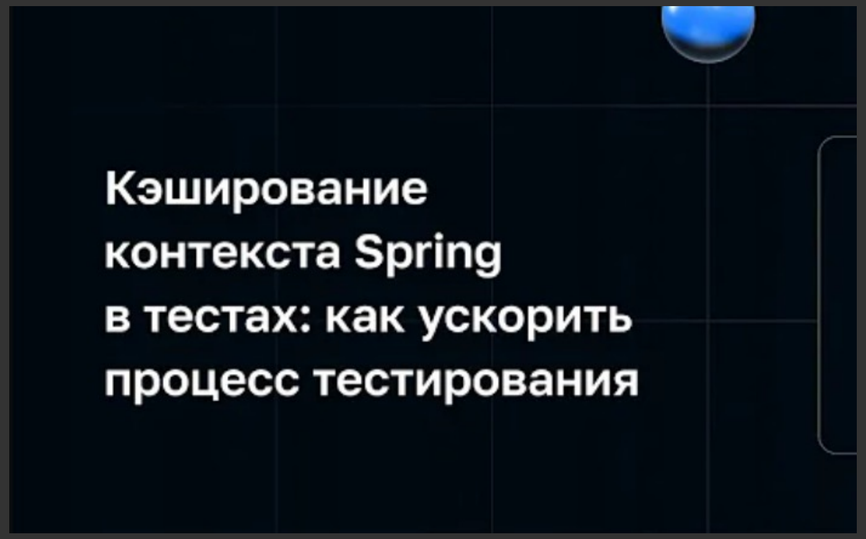
Spring тесты: модульные

- Переиспользовать или нет контекст?
- Контекст с заглушками или нет?
- Переопределять компоненты или нет?
- Кешировать контекст или нет?
- Как и когда кешировать контекст?
- Откатывать транзакции в тестах или нет?
- Подкладывать ли конфигурации/профили или нет?

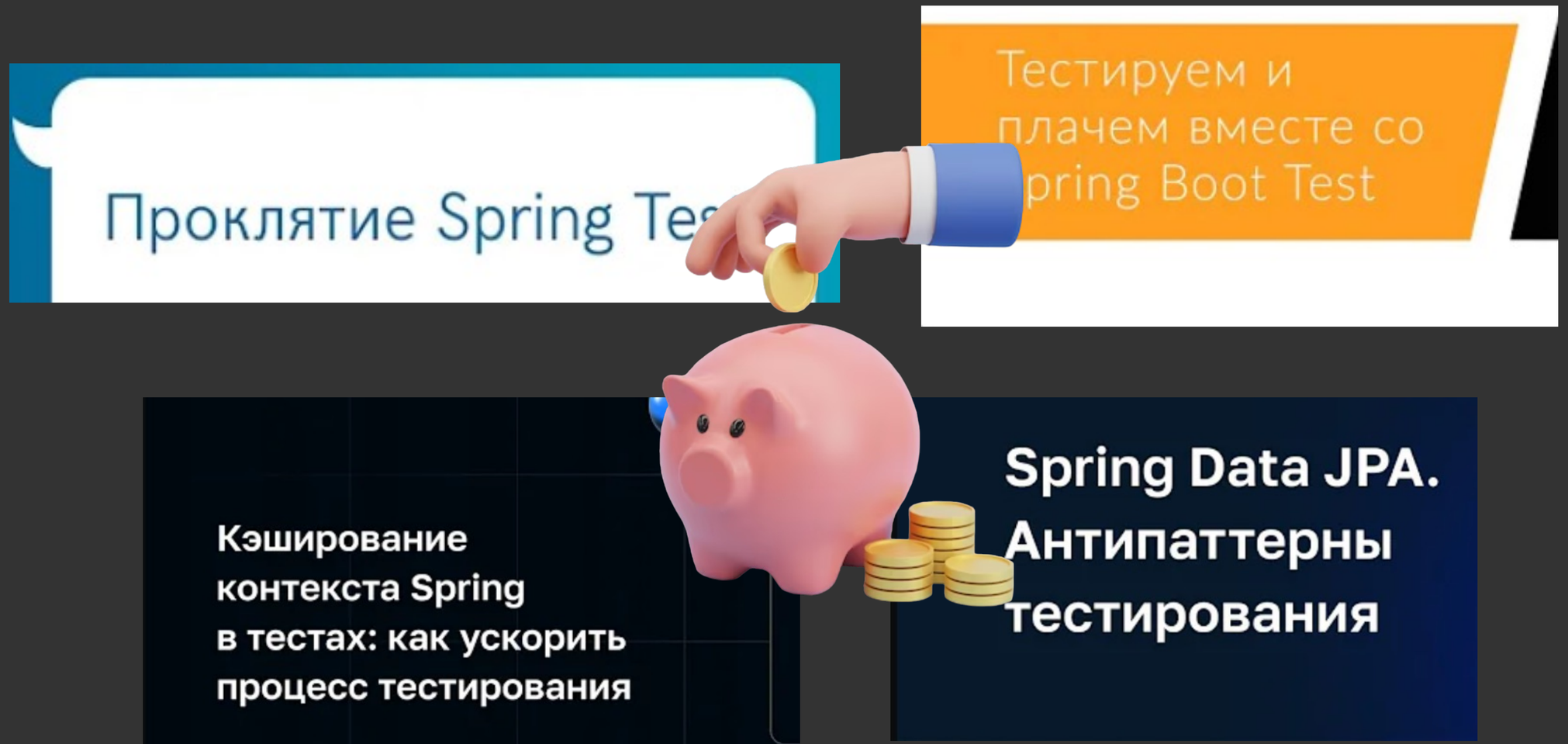
```
@DisabledInAotMode
@TestComponent
@TestConfiguration
@DataJpaTest
@JsonTest
@JooqTest
@JdbcTest
@GraphQlTest
@DataCassandraTest
@DataCouchbaseTest
@DataElasticsearchTest
@DataJdbcTest
@DataMongoTest
@DataNeo4jTest
@DataR2dbcTest
@DataRedisTest
@RestClientTest
@WebFluxTest
@WebMvcTest
@WebServiceClientTest
@WebServiceServerTest
@ActiveProfiles
@BootstrapWith
@TestPropertySource
@RunWith(SpringRunner.class)
@AutoConfigureMockMvc
@MockBean
@Transactional
@Rollback
@DirtiesContext
@Commit
@Repeat
@Timed(millis = 1000)
@LocalServerPort
@LocalRSocketServerPort
@SpringBootTest
```

Spring тесты: плачем





Spring тесты: плачем



Kora тесты: имеем

1. Все что описано то и тестируется
2. Все что заглушка — есть заглушка
3. Все в конфигурации теста — есть конфигурация
4. Понятный жизненный цикл в тесте
5. Быстрая скорость работы

```
@KoraAppTest(Application.class)
class PetServiceTests implements KoraAppTestConfigModifier {

  @Mock @TestComponent
  private PetCache petCache;

  @TestComponent
  private PetService petService;

  @NotNull
  @Override
  public KoraConfigModification config() {
    return KoraConfigModification.ofResourceFile("test.conf");
  }

  @Test
  void test() {
    // сам тест
  }
```

```
 @KoraAppTest test method 'PetServiceTests#updatePetWithNewCategoryCreated()' setup took: 0.798s
 @KoraAppTest test method 'PetServiceTests#updatePetWithNewCategoryCreated()' cleanup took: 0s
 @KoraAppTest test method 'PetServiceTests#updatePetWithSameCategory()' setup started...
der - VirtualThreadExecutor enabled
 @KoraAppTest test method 'PetServiceTests#updatePetWithSameCategory()' setup took: 0.002s
 @KoraAppTest test method 'PetServiceTests#updatePetWithSameCategory()' cleanup took: 0s
```

Spring инструментарий: цена

1. Переусложненный инструментарий
2. Ошибки архитектуры
3. Большой контекст который надо держать в голове

Kora инструментарий: резюме

1. Устоявшиеся подходы к разработке приложений
2. Современные и естественные конструкции языка
3. Очевидный подход без эффекта **черной коробки**
4. Прямое написание кода, а не «**магические**» практики

Аспекты

Аспекты: Spring хотим

```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public Pet getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

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

Аспекты: Spring имеем

```
Exception in thread "main" io.github.resilience4j.spring6.timelimiter.configure.IllegalReturnTypeException Create breakpoint : ru.tinkoff.benchmark.aop.WorkProdService$User ru.tinkoff.benchmark.aop
 .SpringServiceProdAspect#doWork has unsupported by @TimeLimiter return type. CompletionStage expected.
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.proceed(TimeLimiterAspect.java:106)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.lambda$timeLimiterAroundAdvice$0(TimeLimiterAspect.java:90)
    at io.github.resilience4j.spring6.fallback.FallbackExecutor.execute(FallbackExecutor.java:37)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.timeLimiterAroundAdvice(TimeLimiterAspect.java:91) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:627)
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:173)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:768)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at io.github.resilience4j.circuitbreaker.CircuitBreaker.lambda$decorateCheckedSupplier$0(CircuitBreaker.java:71)
    at io.github.resilience4j.circuitbreaker.CircuitBreaker.executeCheckedSupplier(CircuitBreaker.java:739)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.defaultHandling(CircuitBreakerAspect.java:179)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.proceed(CircuitBreakerAspect.java:126)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.lambda$circuitBreakerAroundAdvice$0(CircuitBreakerAspect.java:108)
    at io.github.resilience4j.spring6.fallback.FallbackExecutor.execute(FallbackExecutor.java:37)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.circuitBreakerAroundAdvice(CircuitBreakerAspect.java:109) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:627)
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:173)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:768)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at io.github.resilience4j.retry.Retry.lambda$decorateCheckedSupplier$1(Retry.java:135)
    at io.github.resilience4j.retry.Retry.executeCheckedSupplier(Retry.java:350)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.handleDefaultJoinPoint(RetryAspect.java:175)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.proceed(RetryAspect.java:132)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.lambda$retryAroundAdvice$0(RetryAspect.java:116)
    at io.github.resilience4j.spring6.fallback.FallbackExecutor.execute(FallbackExecutor.java:37)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.retryAroundAdvice(RetryAspect.java:117) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:627)
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:173)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:768)
    at org.springframework.aop.interceptor.ExposeInvocationInterceptor.invoke(ExposeInvocationInterceptor.java:97)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:184)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:768)
    at org.springframework.aop.framework.CglibAopProxy$DynamicAdvisedInterceptor.intercept(CglibAopProxy.java:720)
    at ru.tinkoff.benchmark.aop.SpringServiceProdAspect$$SpringCGLIB$$0.doWork(<generated>)
```

Аспекты: Spring имеем

```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public CompletableFuture<Pet> getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

Не все сигнатуры поддерживаются

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

Аспекты: Spring имеем

SPeL

```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "#id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public CompletableFuture<Pet> getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

Не все сигнатуры поддерживаются

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

Аспекты: Spring имеем

SPeL

```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "#id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public CompletableFuture<Pet> getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

Не все сигнатуры поддерживаются

Забыли подключить

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

Аспекты: Spring порядок ?

```
 @Validated                                          1
@Component
public class PetService {

   @Cacheable(value = "pet", key = "#id")            2
   @CircuitBreaker(name = "pet")                     3
   @Retry(name = "pet")                              4
   @TimeLimiter(name = "pet")                        5
   public CompletableFuture<Pet> getPet(@NotBlank String id) {
      return // делаю что-то
   }
}
```

Аспекты: Spring не порядок

```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "#id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public CompletableFuture<Pet> getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

Может быть реализован по средствам Ordered интерфейса в реализации аспекта

Иногда может быть изменен для всех аннотаций в рамках всего приложения

Аспекты: Spring не порядок

```
 @Validated                                          5
@Component
public class PetService {

    @Cacheable(value = "pet", key = "#id")           4
    @CircuitBreaker(name = "pet")                    2
    @Retry(name = "pet")                             1
    @TimeLimiter(name = "pet")                       3
    public CompletableFuture<Pet> getPet(@NotBlank String id) {
        return // делаю что-то
    }
}
```

Аспекты: Spring не порядок

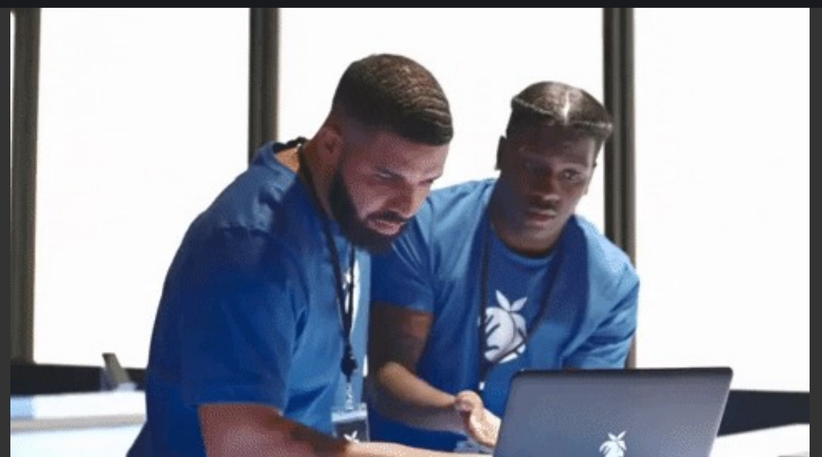
```
@Validated
@Component
public class PetService {

    @Cacheable(value = "pet", key = "#id")
    @CircuitBreaker(name = "pet")
    @Retry(name = "pet")
    @TimeLimiter(name = "pet")
    public CompletableFuture<Pet> getPet(@NotBlan
        return // делаю что-то
    }
}
```

Аспекты: Kora порядок?

```
@Validate
@Cacheable(PetCache.class)
@CircuitBreaker("pet")
@Retry("pet")
@Timeout("pet")
public Pet getPet(@NotBlank String id) {
    return // делаю что-то
}
```

Аспекты: Kora порядок

```java
@Root
@Component
public static class SomeService extends ParentService implements

    private int state = 0;

    @Log
    @Override
    public void init() throws Exception {
        super.init();
        System.out.println("Initializing...");
    }
```



Аспекты: Kora порядок на любой вкус

```
1       @Validate
2       @Cacheable(PetCache.class)
3       @CircuitBreaker("pet")
4       @Retry("pet")
5       @Timeout("pet")
        public Pet getPet(@NotBlank String id) {
            return // делаю что-то
        }
```

Аспекты: Kora порядок на любой вкус

```
1       @Validate
2       @Cacheable(PetCache.class)
3       @Timeout("pet")
4       @CircuitBreaker("pet")
5       @Retry("pet")
        public Pet getPet(@NotBlank String id) {
            return // делаю что-то
        }
```

Аспекты: замеряем работу

Сравнивать будем с помощью JMH:

1. Эмуляция на голой Java (эталон)
2. Spring аспект
3. Kora аспект

Аспекты: кандидаты

```
@Component
public class SomeService {


  @Override
  public String doCall1(String arg1) {
    return arg1;
  }









  @Override
  public String doCall5(String arg1) {
    return arg1;
  }
}
```

Аспекты: кандидаты

```
@Component
public class SomeService {

  @Loggable1
  @Override
  public String doCall1(String arg1) {
    return arg1;
  }

  @Loggable1
  @Loggable2
  @Loggable3
  @Loggable4
  @Loggable5
  @Override
  public String doCall5(String arg1) {
    return arg1;
  }
}
```

Аспекты: кандидаты

```
@Component
public class SomeService {

  @Loggable1                                  2 вызова
  @Override
  public String doCall1(String arg1) {
    return arg1;
  }

  @Loggable1
  @Loggable2
  @Loggable3                                  10 вызовов
  @Loggable4
  @Loggable5
  @Override
  public String doCall5(String arg1) {
    return arg1;
  }
}
```

Четыре вида работ аспектов:

1. Пустышка
2. Маленький
3. Средний
4. Большой

Аспекты: кандидаты

```
@Component
public class SomeService {

  @Loggable1
  @Override
  public String doCall1(String arg1) {
    return arg1;
  }

  @Loggable1
  @Loggable2
  @Loggable3
  @Loggable4
  @Loggable5
  @Override
  public String doCall5(String arg1) {
    return arg1;
  }
}
```

2 вызова

10 вызовов

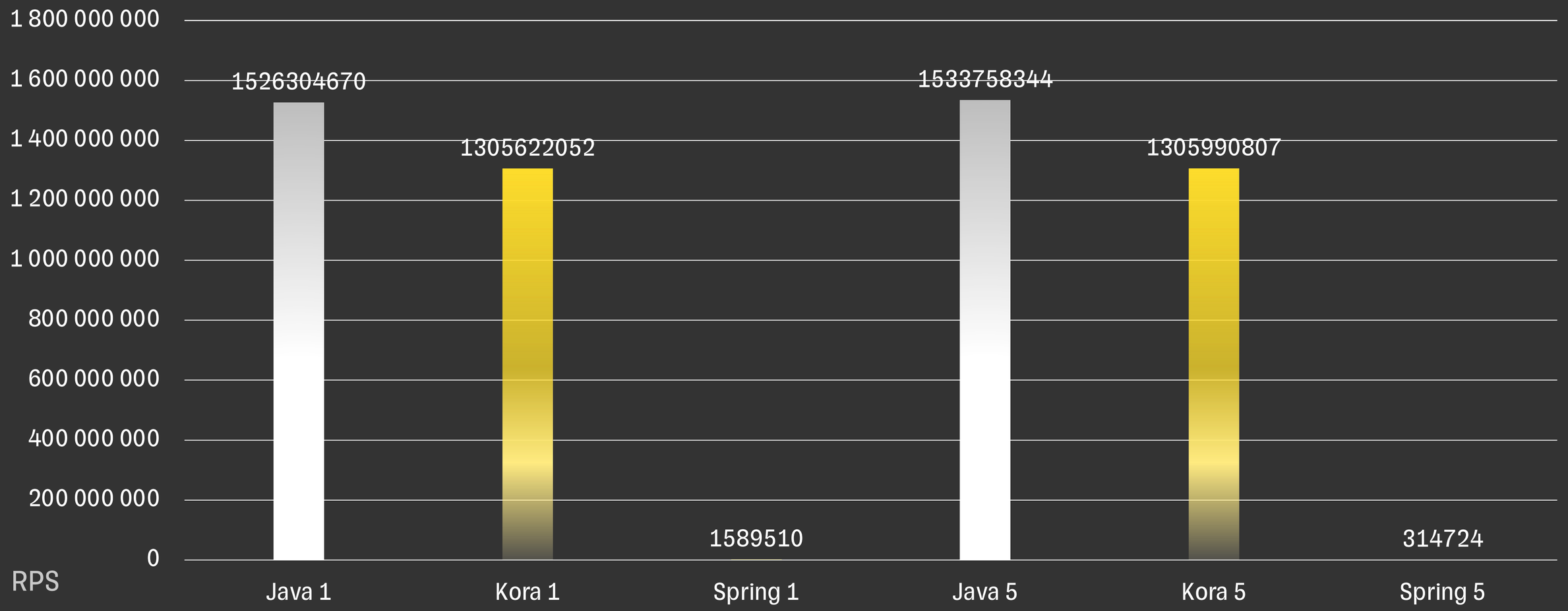
```
public String doNothing1(String arg1) {
  var workType = WorkType.NOTHING;
  aspectBefore(arg1, workType, blackhole);
  String value = service.doNothing(arg1, blackhole);
  aspectAfter(arg1, workType, blackhole);
  return value;
}
```

```
public String doNothing12345(String arg1) {
  var workType = WorkType.NOTHING;
  doAspectBefore12345(arg1, workType, blackhole);
  String value = service.doNothing(arg1, blackhole);
  doAspectAfter12345(arg1, workType, blackhole);
  return value;
}
```

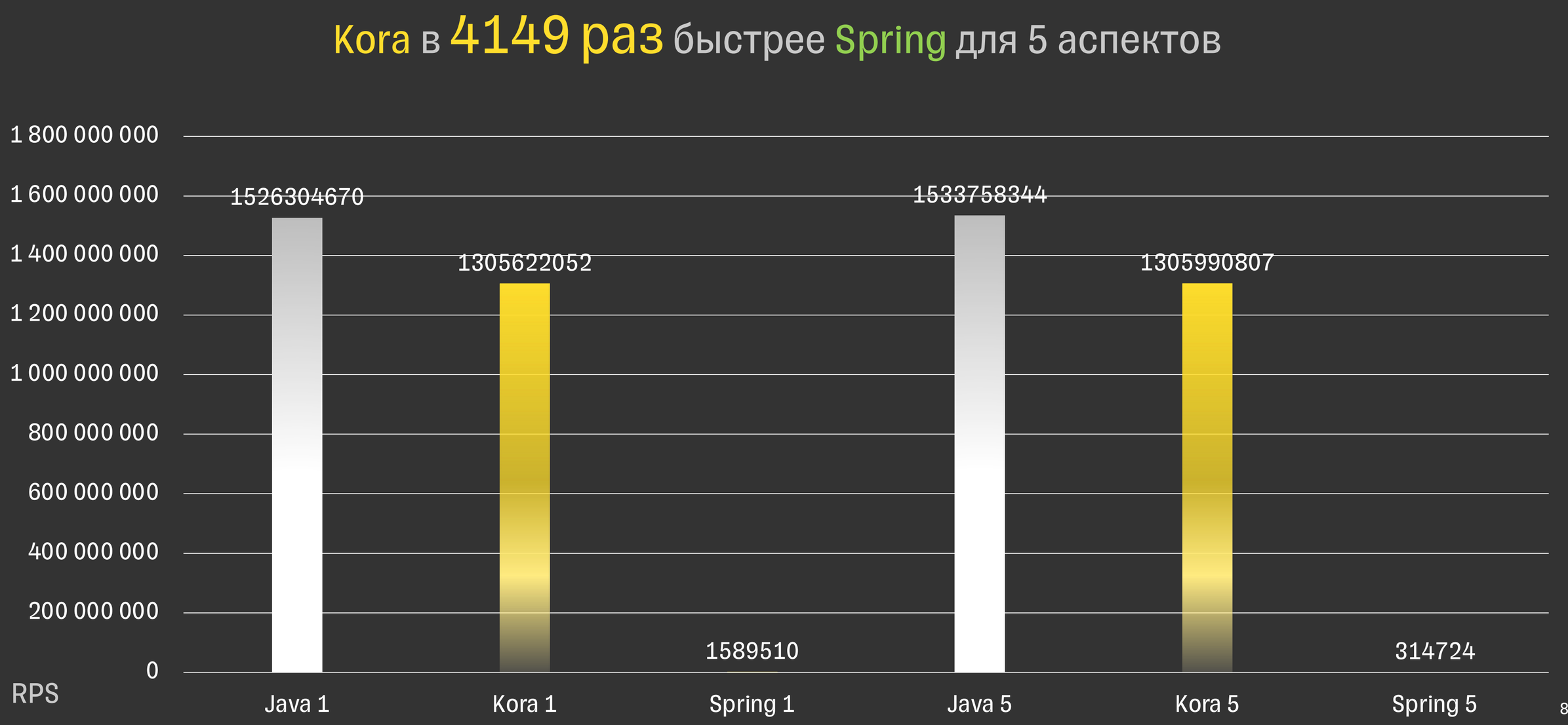
Аспекты: пустышка

Работа пустышка – аспект ничего не делает

Аспекты: пустышка



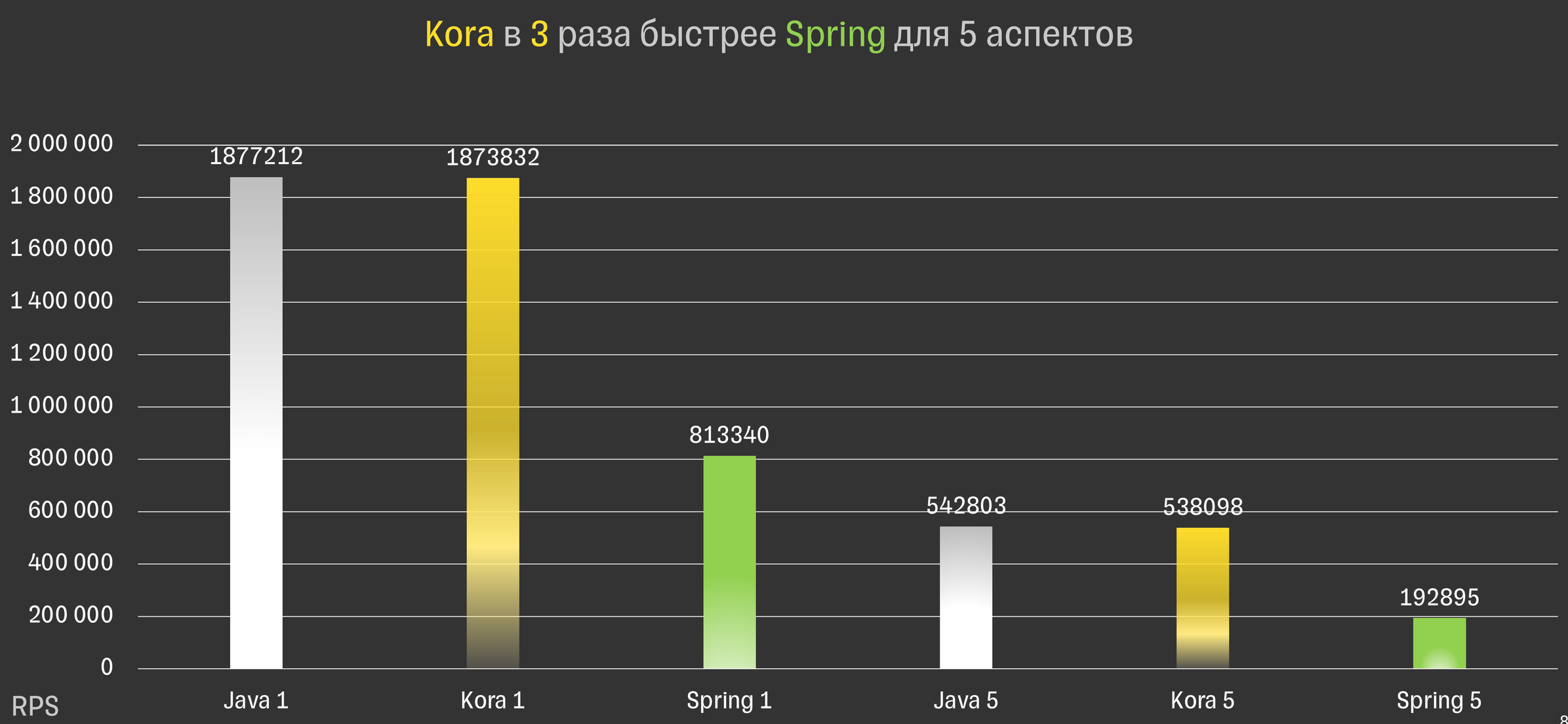
Аспекты: пустышка из золота



Аспекты: маленький

Работа маленькая – создаем 1 UUID

Аспекты: маленький

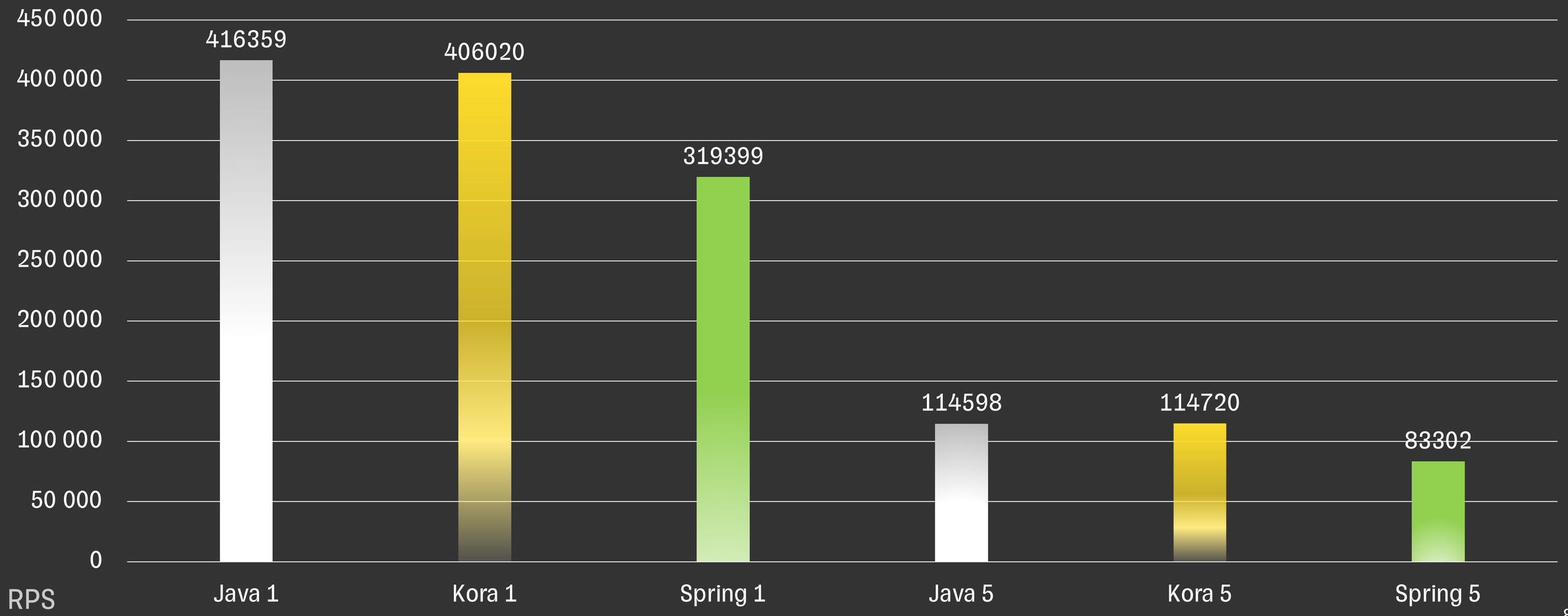


Аспекты: побольше

Работа средняя – создаем 5 UUID

Аспекты: побольше

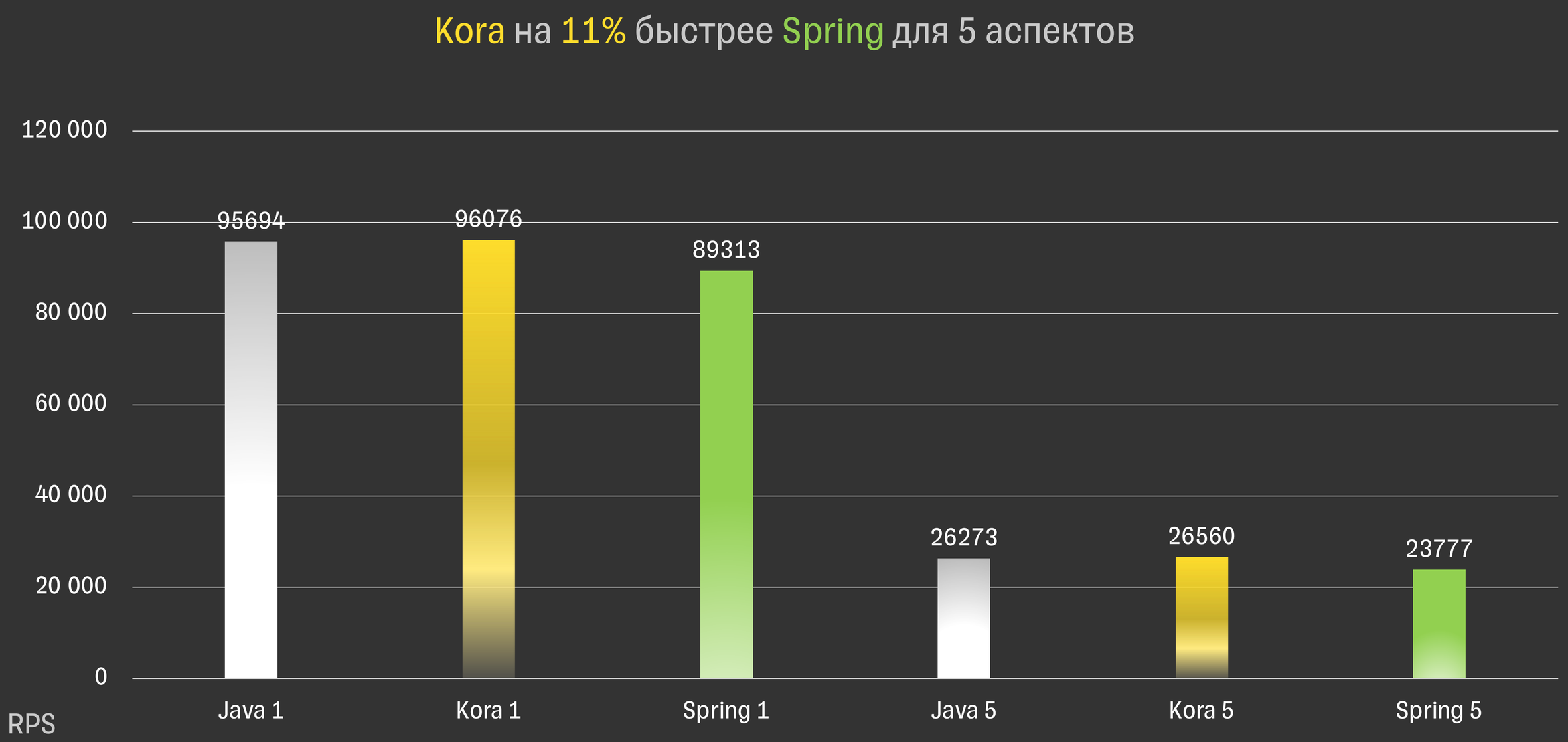
Kora на 38% быстрее Spring для 5 аспектов



Аспекты: ваще большой жесть

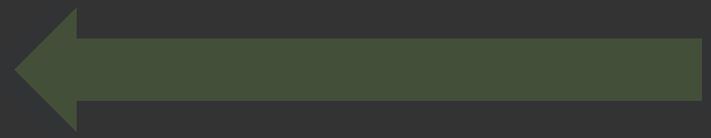
Работа большая – создаем 25 UUID

Аспекты: ваще большой жесть



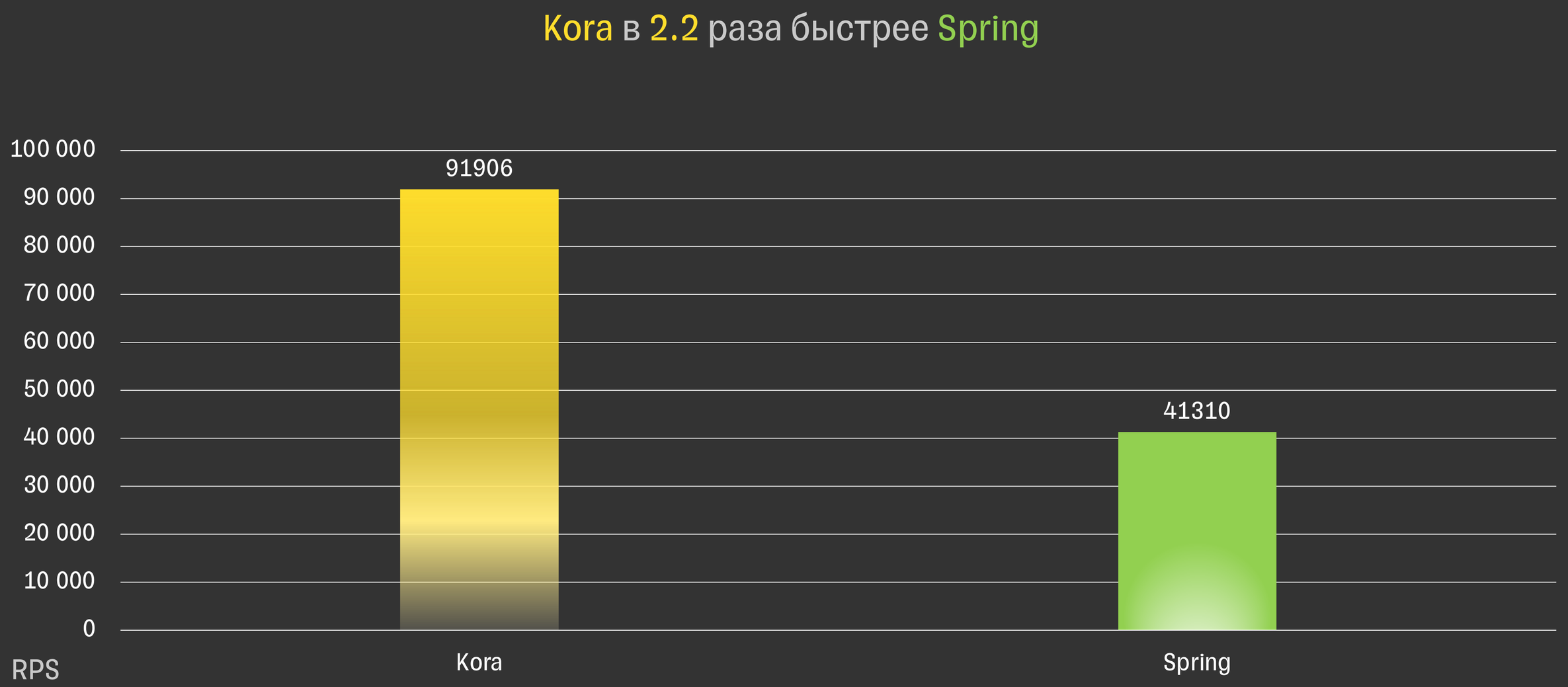
Аспекты: Spring реальный мир

```
@Validated
@Component
public class SpringService {

    @Cacheable(value = "bench", key = ”#id")
    @CircuitBreaker(name = "bench")
    @Retry(name = "bench")
    @TimeLimiter(name = "bench")
    public CompletableFuture<User> doWork(@NotBlank String id) {
        return // creates user
    }
}
```



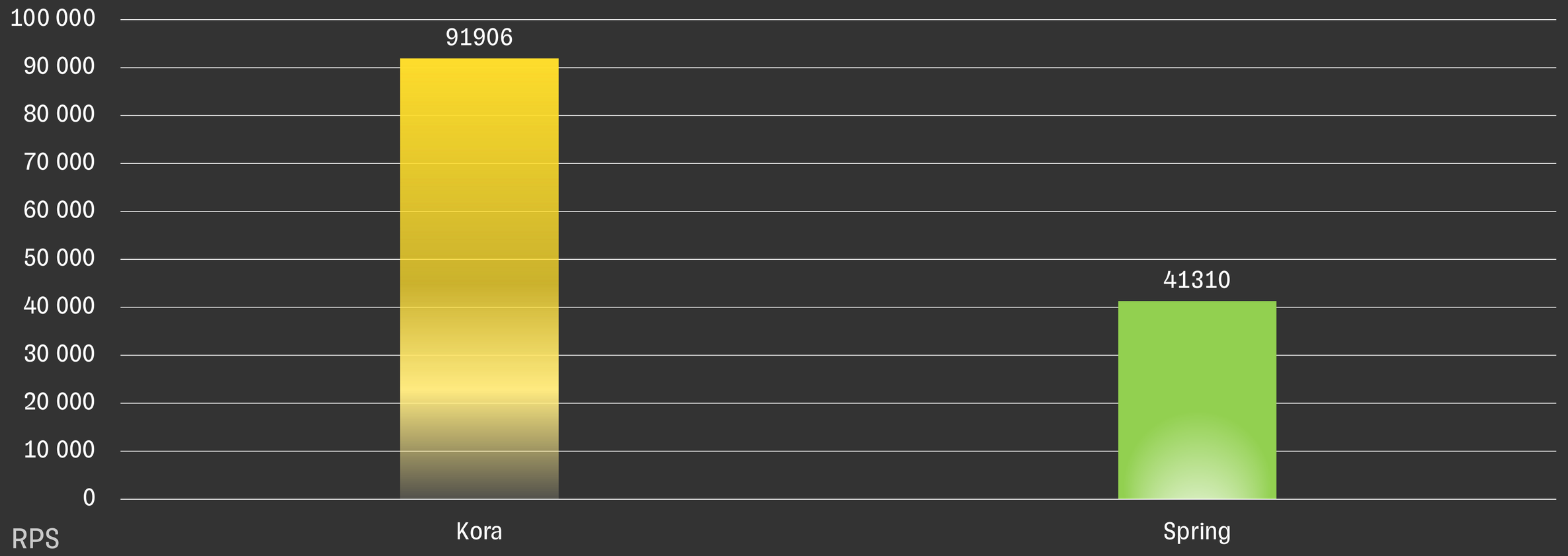
@Cacheable всегда промахивается так как размер кеша равен 0

Аспекты: Spring реальный мир



Аспекты: Spring реальный мир

Вы тратите в 2.2 раза больше циклов процессора на Spring аспекты



Аспекты: Spring цепочка вызовов

```
java.util.concurrent.CompletionException Create breakpoint : java.lang.IllegalStateException: Something happened <5 internal lines>
    at org.springframework.cache.interceptor.CacheAspectSupport$CachePutRequest.apply(CacheAspectSupport.java:1004)
    at org.springframework.cache.interceptor.CacheAspectSupport.evaluate(CacheAspectSupport.java:560)
    at org.springframework.cache.interceptor.CacheAspectSupport.execute(CacheAspectSupport.java:433)
    at org.springframework.cache.interceptor.CacheAspectSupport.execute(CacheAspectSupport.java:395)
    at org.springframework.cache.interceptor.CacheInterceptor.invoke(CacheInterceptor.java:74)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:184)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.lambda$handleJoinPointCompletableFuture$1(TimeLimiterAspect.java:141)
    at io.github.resilience4j.timelimiter.internal.TimeLimiterImpl.lambda$decorateCompletionStage$2(TimeLimiterImpl.java:77)
    at io.github.resilience4j.timelimiter.TimeLimiter.executeCompletionStage(TimeLimiter.java:165)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.handleJoinPointCompletableFuture(TimeLimiterAspect.java:139)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.proceed(TimeLimiterAspect.java:110)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.lambda$timeLimiterAroundAdvice$0(TimeLimiterAspect.java:90)
    at io.github.resilience4j.spring6.fallback.FallbackExecutor.execute(FallbackExecutor.java:37)
    at io.github.resilience4j.spring6.timelimiter.configure.TimeLimiterAspect.timeLimiterAroundAdvice(TimeLimiterAspect.java:91) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:627)
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:173)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.lambda$handleJoinPointCompletableFuture$1(CircuitBreakerAspect.java:167)
    at io.github.resilience4j.circuitbreaker.CircuitBreaker.lambda$decorateCompletionStage$2(CircuitBreaker.java:107)
    at io.github.resilience4j.circuitbreaker.CircuitBreaker.executeCompletionStage(CircuitBreaker.java:715)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.handleJoinPointCompletableFuture(CircuitBreakerAspect.java:165)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.proceed(CircuitBreakerAspect.java:124)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.lambda$circuitBreakerAroundAdvice$0(CircuitBreakerAspect.java:108)
    at io.github.resilience4j.spring6.fallback.FallbackExecutor.execute(FallbackExecutor.java:37)
    at io.github.resilience4j.spring6.circuitbreaker.configure.CircuitBreakerAspect.circuitBreakerAroundAdvice(CircuitBreakerAspect.java:109) <2 internal lines>
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(AbstractAspectJAdvice.java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.java:627)
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:173)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:89)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.lambda$handleJoinPointCompletableFuture$1(RetryAspect.java:188)
    at io.github.resilience4j.retry.Retry$AsyncRetryBlock.run(Retry.java:551) <6 internal lines>
Caused by: java.lang.IllegalStateException Create breakpoint : Something happened
    at ru.tinkoff.benchmark.aop.SpringServiceProdAspect.doWork(SpringServiceProdAspect.java:28) <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:351)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:196)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.validation.beanvalidation.MethodValidationInterceptor.invoke(MethodValidationInterceptor.java:174)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:184)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.cache.interceptor.CacheInterceptor.lambda$invoke$0(CacheInterceptor.java:64)
    at org.springframework.cache.interceptor.CacheAspectSupport.invokeOperation(CacheAspectSupport.java:416)
    at org.springframework.cache.interceptor.CacheAspectSupport.evaluate(CacheAspectSupport.java:545)
    ... 46 more
```

Аспекты: Spring цепочка вызовов

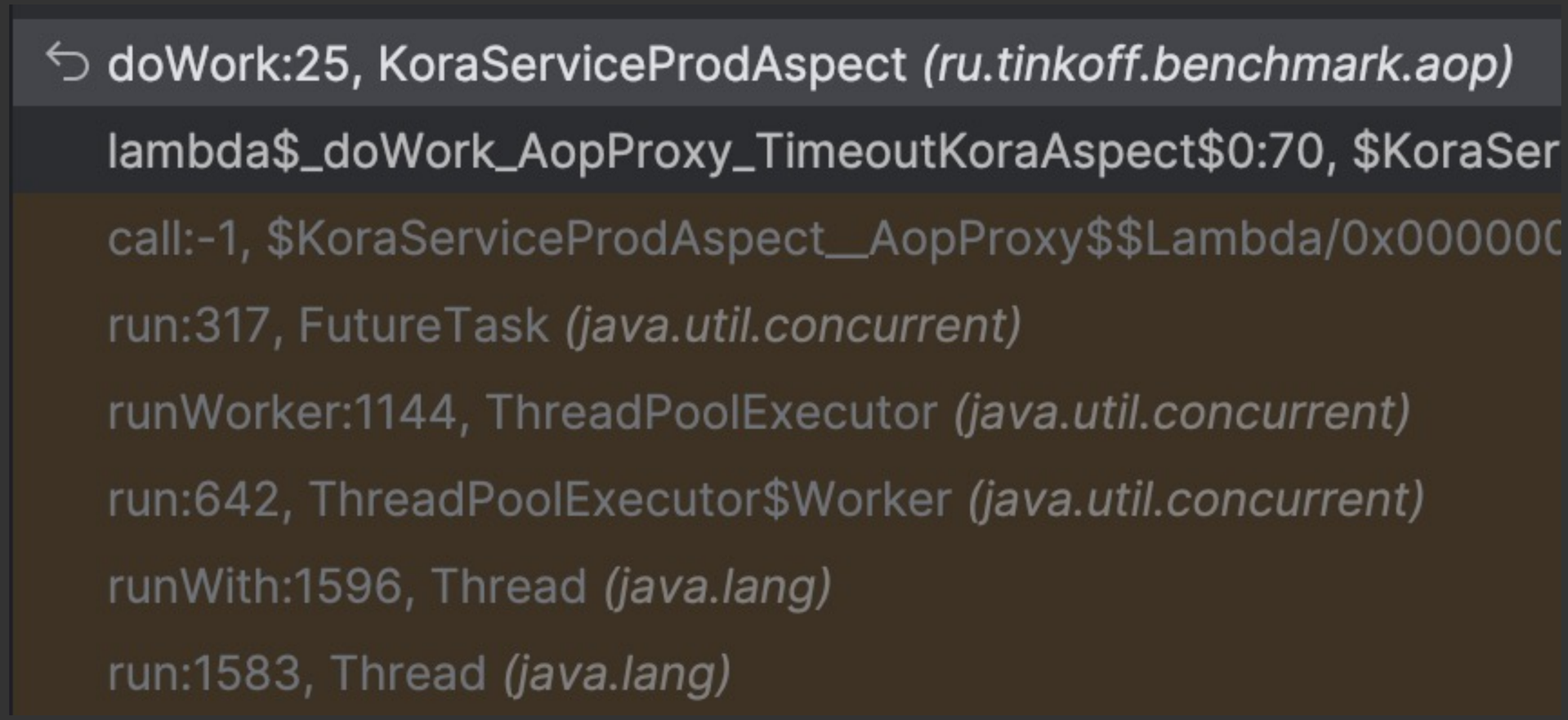
```
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethodWithGivenArgs(Abst  ectJA  .java:637)
    at org.springframework.aop.aspectj.AbstractAspectJAdvice.invokeAdviceMethod(AbstractAspectJAdvice.  a:62
    at org.springframework.aop.aspectj.AspectJAroundAdvice.invoke(AspectJAroundAdvice.java:71)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedin  t.java:89)
    at io.github.resilience4j.spring6.retry.configure.RetryAspect.lambda$handleJoinPointCompletableFuture$1(  pect.java:188)
```

```
Caused by: java.lang.IllegalStateException Create breakpoint : Something happened
    at ru.tinkoff.benchmark.aop.SpringServiceProdAspect.doWork(SpringServiceProdAspect.java:28) <2 internal lines>
    at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:351)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:196)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:163)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
    at org.springframework.validation.beanvalidation.MethodValidationInterceptor.invoke(MethodValidationInterceptor.java:174)
    at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:184)
    at org.springframework.aop.framework.CglibAopProxy$CglibMethodInvocation.proceed(CglibAopProxy.java:765)
```

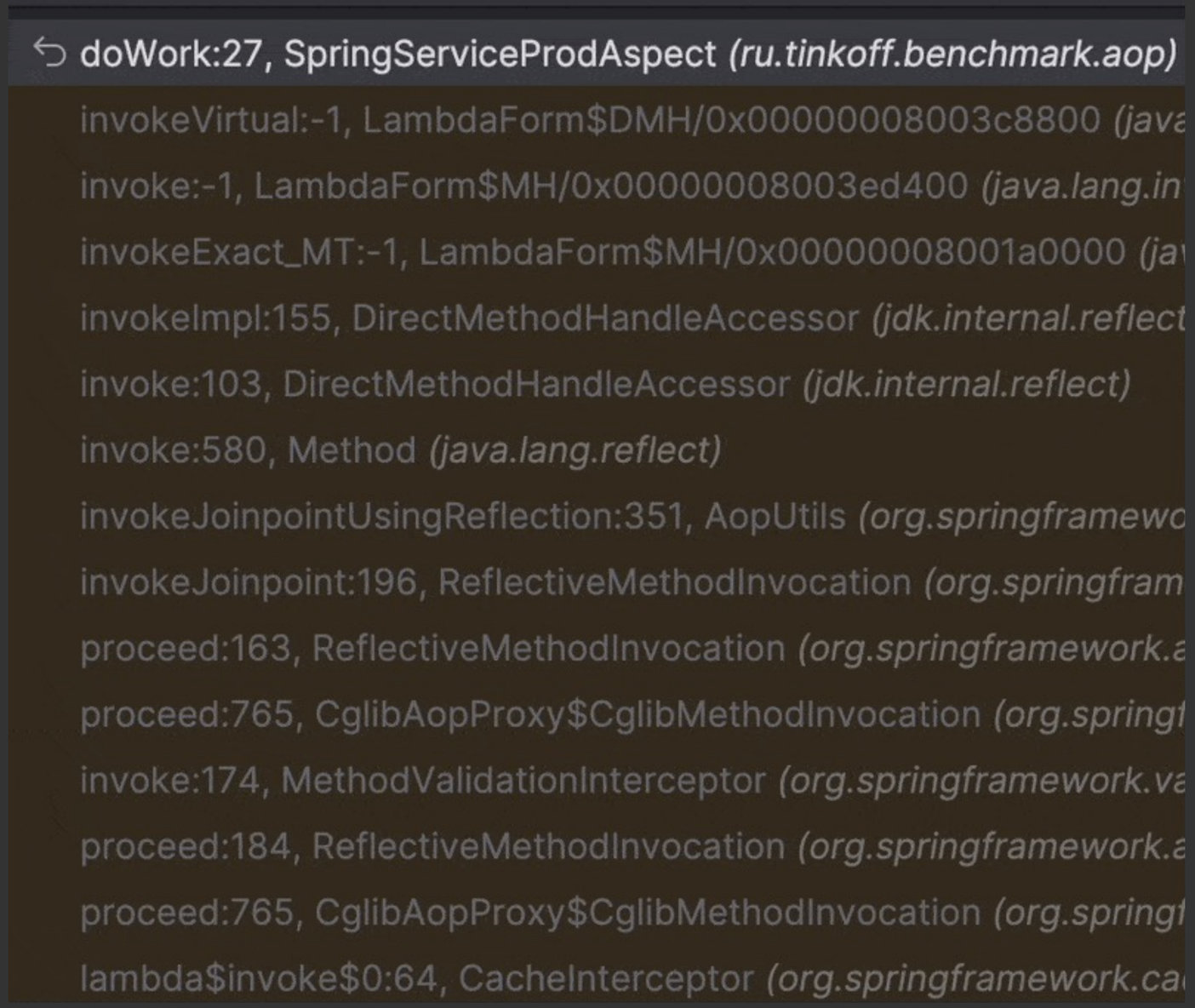
Аспекты: Kora цепочка вызовов

```
Exception in thread "main" ru.tinkoff.kora.resilient.retry.RetryExhaustedException Create breakpoint : java.lang.IllegalStateException: Something happened
    at ru.tinkoff.kora.resilient.retry.KoraRetry.internalRetry(KoraRetry.java:90)
    at ru.tinkoff.kora.resilient.retry.KoraRetry.retry(KoraRetry.java:53)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy._doWork_AopProxy_RetryKoraAspect($KoraServiceProdAspect__AopProxy.java:74)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy._doWork_AopProxy_CircuitBreakerKoraAspect($KoraServiceProdAspect__AopProxy.java:81)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy.lambda$_doWork_AopProxy_CacheableAopKoraAspect$2($KoraServiceProdAspect__AopProxy.java:95)
    at com.github.benmanes.caffeine.cache.BoundedLocalCache.lambda$doComputeIfAbsent$14(BoundedLocalCache.java:2688) <1 internal line>
    at com.github.benmanes.caffeine.cache.BoundedLocalCache.doComputeIfAbsent(BoundedLocalCache.java:2686)
    at com.github.benmanes.caffeine.cache.BoundedLocalCache.computeIfAbsent(BoundedLocalCache.java:2669)
    at com.github.benmanes.caffeine.cache.LocalCache.computeIfAbsent(LocalCache.java:112)
    at com.github.benmanes.caffeine.cache.LocalManualCache.get(LocalManualCache.java:62)
    at ru.tinkoff.kora.cache.caffeine.AbstractCaffeineCache.computeIfAbsent(AbstractCaffeineCache.java:69)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy._doWork_AopProxy_CacheableAopKoraAspect($KoraServiceProdAspect__AopProxy.java:95)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy._doWork_AopProxy_ValidateMethodKoraAspect($KoraServiceProdAspect__AopProxy.java:111)
    at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy.doWork($KoraServiceProdAspect__AopProxy.java:117)
    at ru.tinkoff.benchmark.aop.Bench.main(Bench.java:52)
    Suppressed: java.lang.IllegalStateException: Something happened
        at ru.tinkoff.benchmark.aop.KoraServiceProdAspect.doWork(KoraServiceProdAspect.java:26)
        at ru.tinkoff.benchmark.aop.$KoraServiceProdAspect__AopProxy.lambda$_doWork_AopProxy_TimeoutKoraAspect$0($KoraServiceProdAspect__AopProxy.java:70) <4 internal lines>
```

Аспекты: Kora отладка



Аспекты: Spring цена аспектов



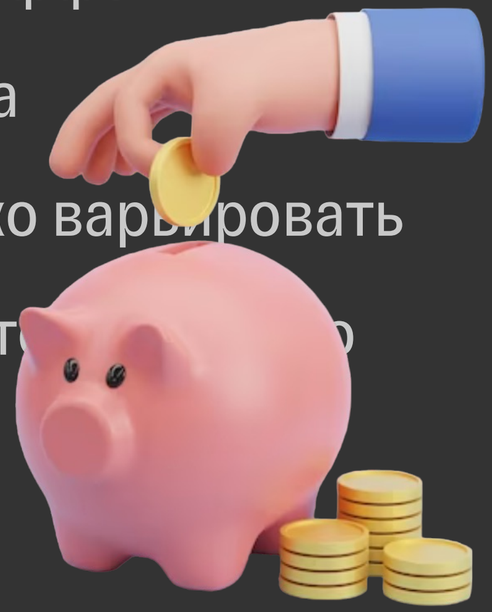
Аспекты: Spring резюме

1. Стеклянный потолок
2. Негативный накопительный эффект
3. Неочевидный порядок вызова
4. Порядок вызовов нельзя гибко варьировать
5. Цепочка вызовов и отладка это неприятно

Надо признать что пользовательские аспекты на Spring писать легче

Аспекты: Spring резюме

1. Стеклянный потолок
2. Негативный накопительный эффект
3. Неочевидный порядок вызова
4. Порядок вызовов нельзя гибко варьировать
5. Цепочка вызовов и отладка эт[illegible]



Надо признать что пользовательские аспекты на Spring писать легче

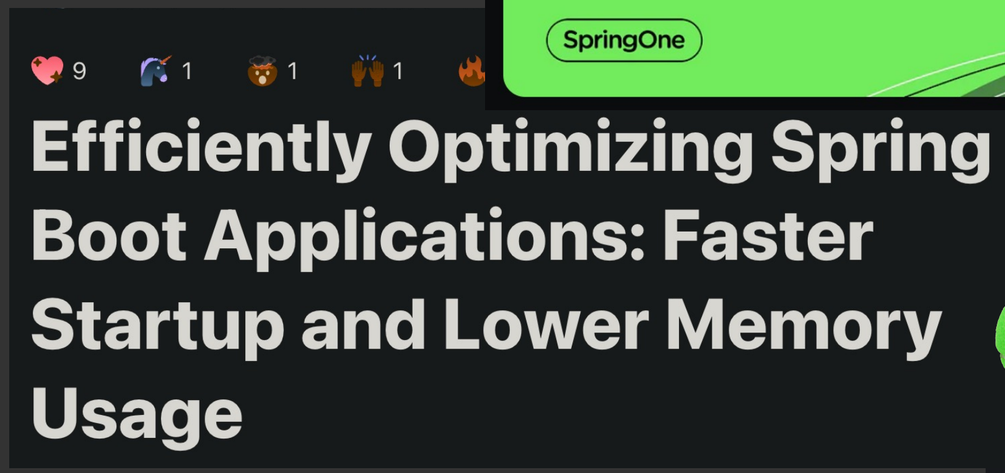
Аспекты: Kora резюме

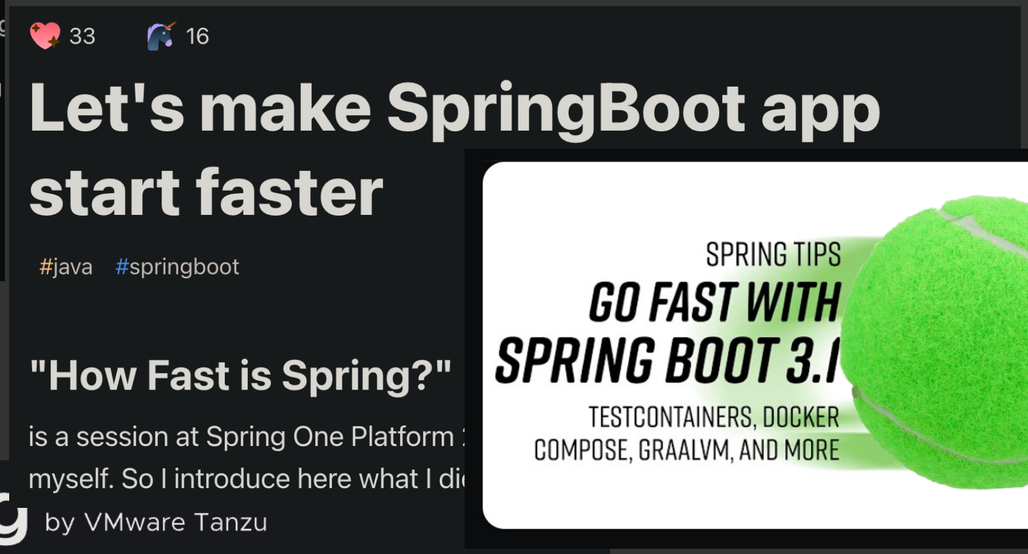
1. Бесплатная стоимость механизма аспектов
2. Оптимизированные аспекты с поддержкой синхронных/асинхронных/реактивных сигнатур
3. Прозрачный порядок вызова
4. Прозрачное понимание работы и работает ли в принципе
5. Прозрачная цепочка вызовов

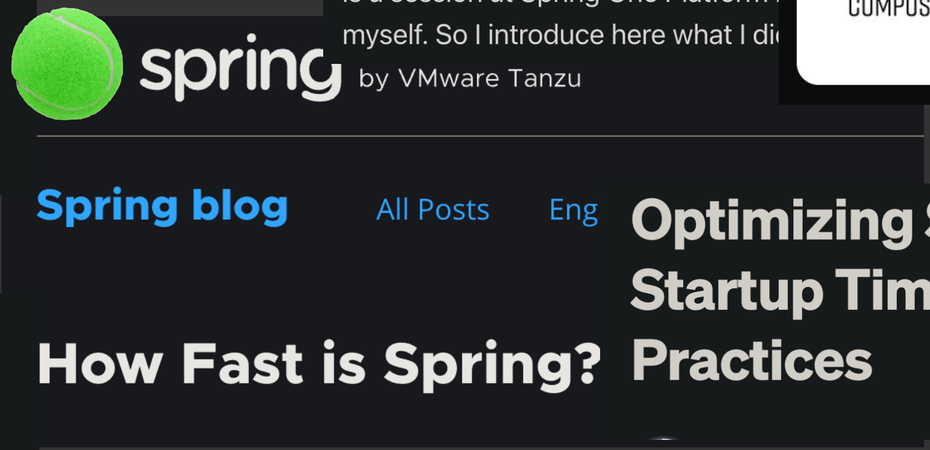
Аспекты в Kora это коллега программист который написал оптимизированный код и положил его целиком в соседний метод

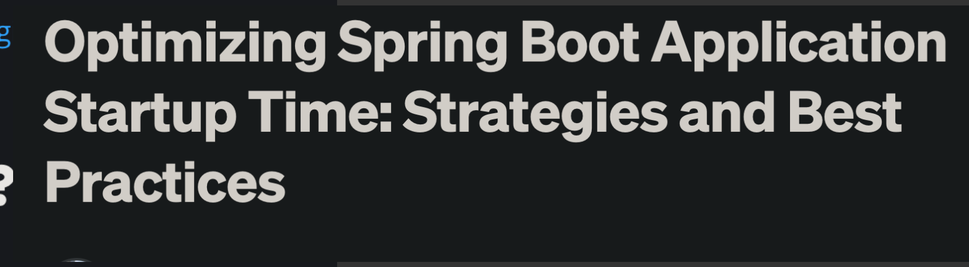
Оптимизация

Оптимизация: Spring у них дома









Оптимизация: Spring ленивая инициализация

Benefits of Lazy Initialization

Lazy initialization can result in significantly reduced startup times as fewer classes are loaded and fewer beans are created during application startup. For example, a small web application that's using Actuator and Spring Security that normally starts in 2500ms will start in 2000ms with lazy initialization enabled. The exact improvement will vary from application to application depending on the structure of their beans' dependency graphs.

Ленивая загрузка бинов:

- Была доступна всегда с `@Lazy`
- Начиная с Spring Boot 2.3 можно включить глобально
  `spring.main.lazy-initialization=true`
- Сохранят время запуска, но жертвует временем первого запроса

Spring Boot быстрый

Low hanging fruits для быстрого запуска

- Ленивая загрузка бинов
  `spring.main.lazy-initialization=true`
- Ленивую инициализацию Spring Data JPA репозиториев
  `spring.data.jpa.repositories.bootstrap-mode=lazy`

4. Lazy Initialization

Tried Lazy Init.

```
@Configuration
public class LazyInitBeanFactoryPostProcessor implements BeanFacto
  @Override
  public void postProcessBeanFactory(ConfigurableListableBeanFacto
    for (String beanName : beanFactory.getBeanDefinitionNames()) {
      beanFactory.getBeanDefinition(beanName).setLazyInit(true);
    }
  }
}
```

↓Here's the result. It became just a little bit faster.

```
                                Mode  Cnt  Score   Error
case01_FluxBaseline               ss   10  2.938 ± 0.287
case04_WithLazyInit               ss   10  2.844 ± 0.129
```

Оптимизация: Spring ленивая инициализация

1. Сервис не рабочий пока не инициализируется
2. Сервис инициализируется под нагрузкой
3. Вывели под нагрузку не рабочий сервис, просели по SLA

Оптимизация: Spring ленивая деградация

1. Сервис не рабочий пока не инициализируется
2. Сервис инициализируется под нагрузкой
3. Вывели под нагрузку не рабочий сервис, просели по SLA

MICRONAUT®

Если нужен нерабочий сервис, то какая разница сколько запускается?

Если нужен рабочий сервис, зачем ленивая инициализация?

Оптимизация: Spring запуск за секунды

Spring Boot быстрый

Spring Boot 2.3+ можно запустить за 1 секунду

Benchmark	(sample)	Mode	Cnt	Score	Error	Units	Beans	Classes
MainBenchmark	actr	avgt	10	1.316	± 0.060	s/op	186	5666
MainBenchmark	jdbc	avgt	10	1.237	± 0.050	s/op	147	5625
MainBenchmark	demo	avgt	10	1.056	± 0.040	s/op	111	5266
MainBenchmark	slim	avgt	10	1.003	± 0.011	s/op	105	5208
MainBenchmark	thin	avgt	10	0.855	± 0.028	s/op	60	4892
MainBenchmark	lite	avgt	10	0.694	± 0.015	s/op	30	4580
MainBenchmark	func	avgt	10	0.652	± 0.017	s/op	25	4378

Dave Syer - How fast is Spring?

https://spring.io/blog/2018/12/12/how-fast-is-spring

“… откуда вообще этот странный миф, что Spring запускается медленно, я слышал вообще истории что Spring запускается минутами! Минутами! Господи что они там делают я вообще не представляю… ”

Докладчик (с.)

Оптимизация: Spring запуск на все деньги

Spring Boot быстрый

Spring Boot 2.3+ можно запустить за 1 секунду

Benchmark	(sample)	Mode	Cnt	Score	Error	Units	Beans	Classes
MainBenchmark	actr	avgt	10	1.316	± 0.060	s/op	186	5666
MainBenchmark	jdbc	avgt	10	1.237	± 0.050	s/op	147	5625
MainBenchmark	demo	avgt	10	1.056	± 0.040	s/op	111	5266
MainBenchmark	slim	avgt	10	1.003	± 0.011	s/op	105	5208
MainBenchmark	thin	avgt	10	0.855	± 0.028	s/op	60	4892
MainBenchmark	lite	avgt	10	0.694	± 0.015	s/op	30	4580
MainBenchmark	func	avgt	10	0.652	± 0.017	s/op	25	4378

Dave Syer - How fast is Spring?

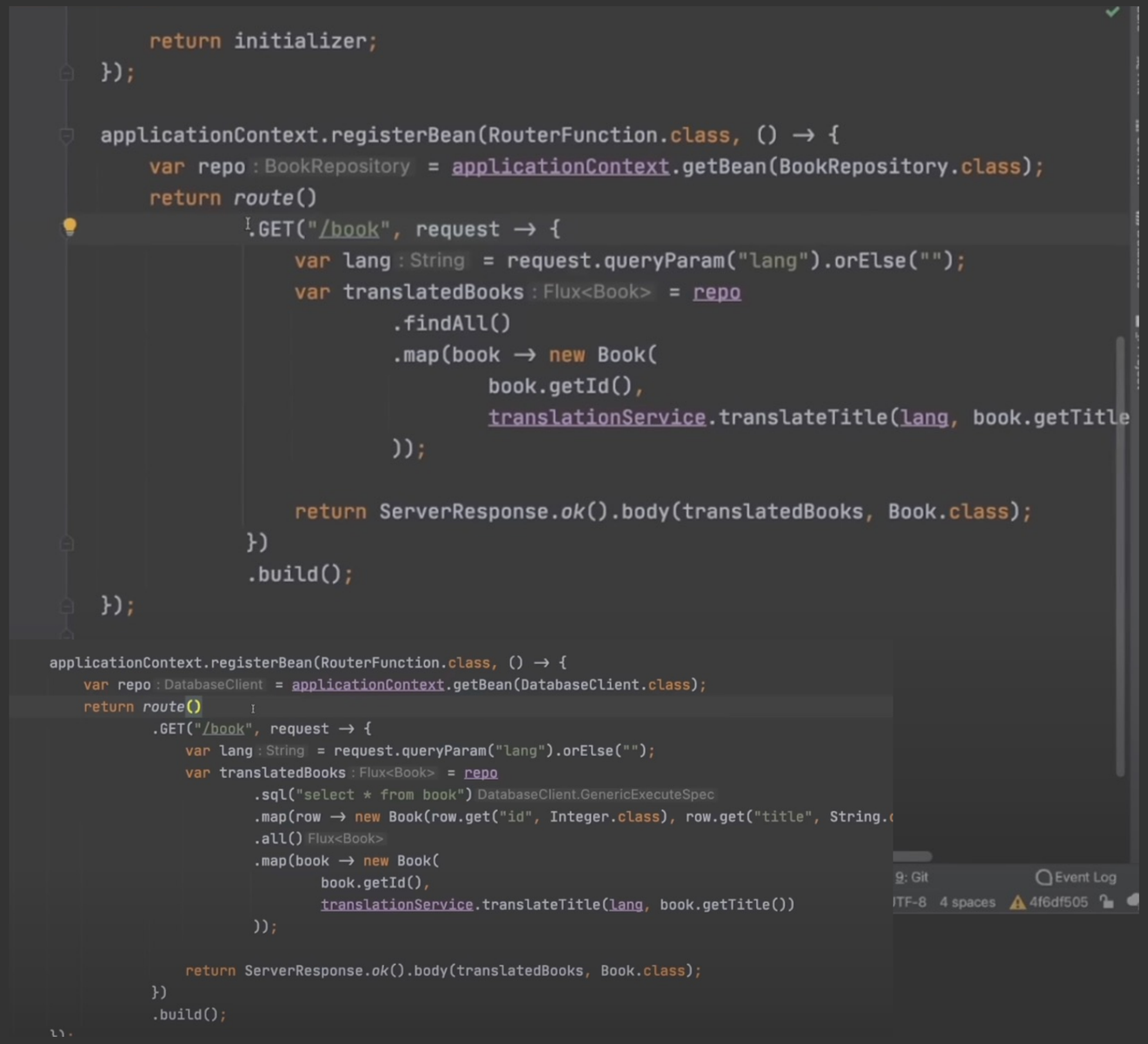
https://spring.io/blog/2018/12/12/how-fast-is-spring

- Spring 5.0
- Много оптимизаций
- Синтетические, не боевые приложения
- «Функциональный Spring»

Машина для тестов:

- Intel Core i7 (i7-3770, 4 cores, 8 threads), 3.4GHz, 32G RAM, SSD

Оптимизация: Spring функциональный



```
        return initializer;
    });

    applicationContext.registerBean(RouterFunction.class, () -> {
        var repo : BookRepository = applicationContext.getBean(BookRepository.class);
        return route()
                .GET("/book", request -> {
                    var lang : String = request.queryParam("lang").orElse("");
                    var translatedBooks : Flux<Book> = repo
                            .findAll()
                            .map(book -> new Book(
                                    book.getId(),
                                    translationService.translateTitle(lang, book.getTitle
                            ));

                    return ServerResponse.ok().body(translatedBooks, Book.class);
                })
                .build();
    });
```

```
applicationContext.registerBean(RouterFunction.class, () -> {
    var repo : DatabaseClient = applicationContext.getBean(DatabaseClient.class);
    return route()
            .GET("/book", request -> {
                var lang : String = request.queryParam("lang").orElse("");
                var translatedBooks : Flux<Book> = repo
                        .sql("select * from book") DatabaseClient.GenericExecuteSpec
                        .map(row -> new Book(row.get("id", Integer.class), row.get("title", String.
                        .all() Flux<Book>
                        .map(book -> new Book(
                                book.getId(),
                                translationService.translateTitle(lang, book.getTitle())
                        ));

                return ServerResponse.ok().body(translatedBooks, Book.class);
            })
            .build();
});
```

9: Git Event Log

ITF-8 4 spaces 4f6df505

«Фреймворк 25 века» (с.) Докладчик

Оптимизация: Spring «функциональный»


```java
@SpringBootApplication
@RestController
public class Application {

    @GetMapping("/")
    public String home() { return "Hello"; }

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

Оптимизация: Spring «нефункциональный»

Вы держитесь здесь, всего доброго, хорошего настроения, здоровья!

Оптимизация: Spring модули

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}

@SpringBootApplication
public class Application {

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

Оптимизация: Spring модули

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}
```

@EnableAutoConfiguration

Все зависимости на пути протекают и автоматически подключаются

Оптимизация: Spring модули

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}

@SpringBootApplication
public class Application {

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

Если вы ничего не подключали, это не значит что оно не включено

Оптимизация: Spring модули руками

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}

@SpringBootConfiguration
@ComponentScan(basePackages = {"ru.tinkoff.spring"})
@Import({ ValidationAutoConfiguration.class,
     JpaRepositoriesAutoConfiguration.class,
     HibernateJpaAutoConfiguration.class })
public class Application {

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

Оптимизация: Spring модули руками

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}
```

```
@SpringBootConfiguration
@ComponentScan(basePackages = {"ru.tinkoff.spring"})
@Import({ ValidationAutoConfiguration.class,
    JpaRepositoriesAutoConfiguration.class,
    HibernateJpaAutoConfiguration.class })
public class Application {

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

Не учитывает порядок конфигураций

Оптимизация: Spring модули руками

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}

@SpringBootConfiguration
@ComponentScan(basePackages = {"ru.tinkoff.spring"})
@ImportAutoConfiguration({ ValidationAutoConfiguration.class,
    JpaRepositoriesAutoConfiguration.class,
    HibernateJpaAutoConfiguration.class })
public class Application {

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

Оптимизация: Spring модули руками

```
dependencies {
   implementation "org.springframework.boot:spring-boot-starter"
   implementation "org.springframework.boot:spring-boot-starter-aop"
   implementation "org.springframework.boot:spring-boot-starter-validation"
   implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}
```

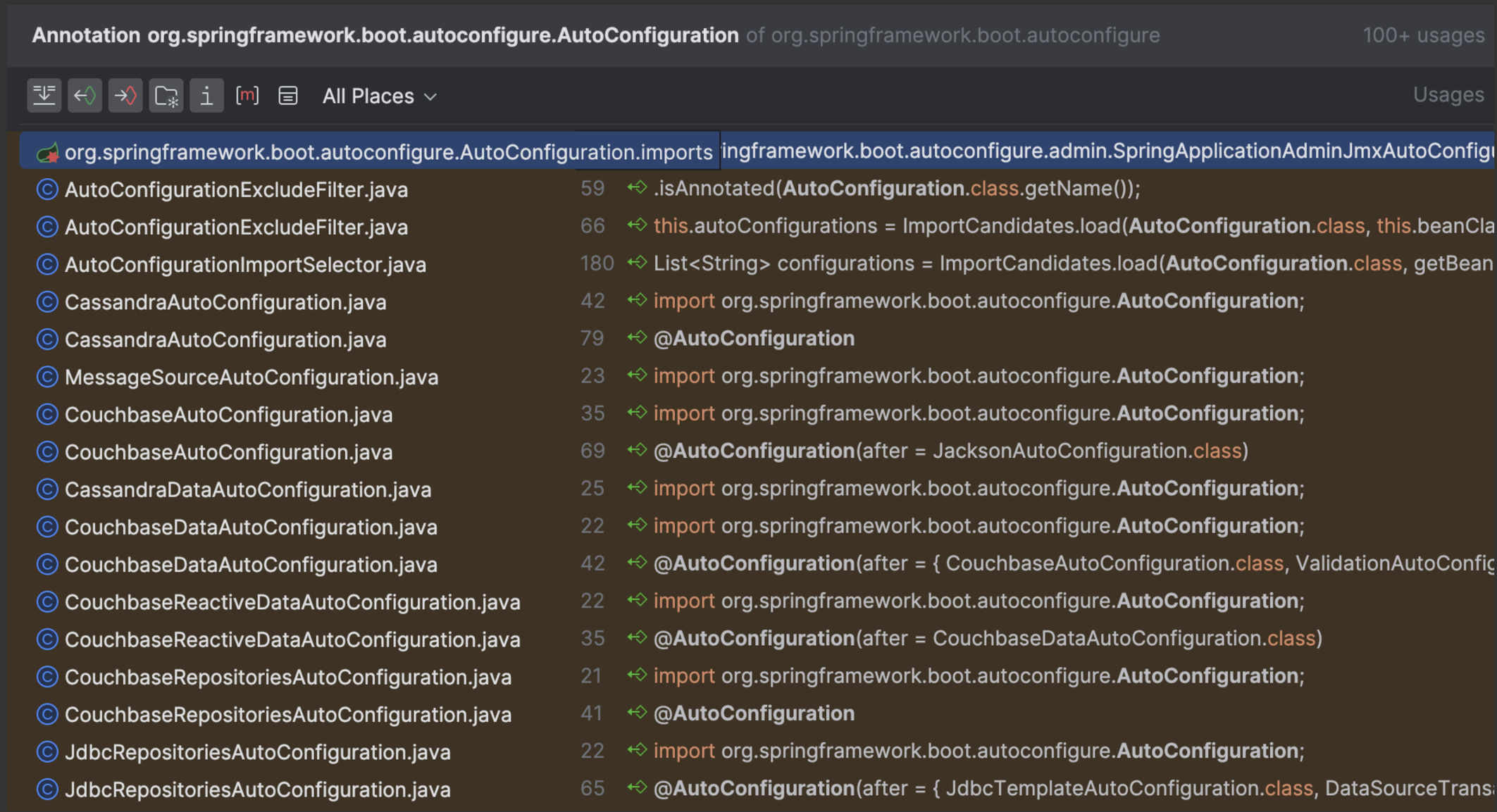
```
@SpringBootConfiguration
@ComponentScan(basePackages = {"ru.tinkoff.spring"})
@ImportAutoConfiguration({ ValidationAutoConfiguration.class,
    JpaRepositoriesAutoConfiguration.class,
    HibernateJpaAutoConfiguration.class })
public class Application {

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

```
@Configuration
@ConfigurationProperties("foo")
public class ConfigClass {

   String bar;

   String getBar() { return bar; }

   void setBar(String bar) { this.bar = bar; }
}
```

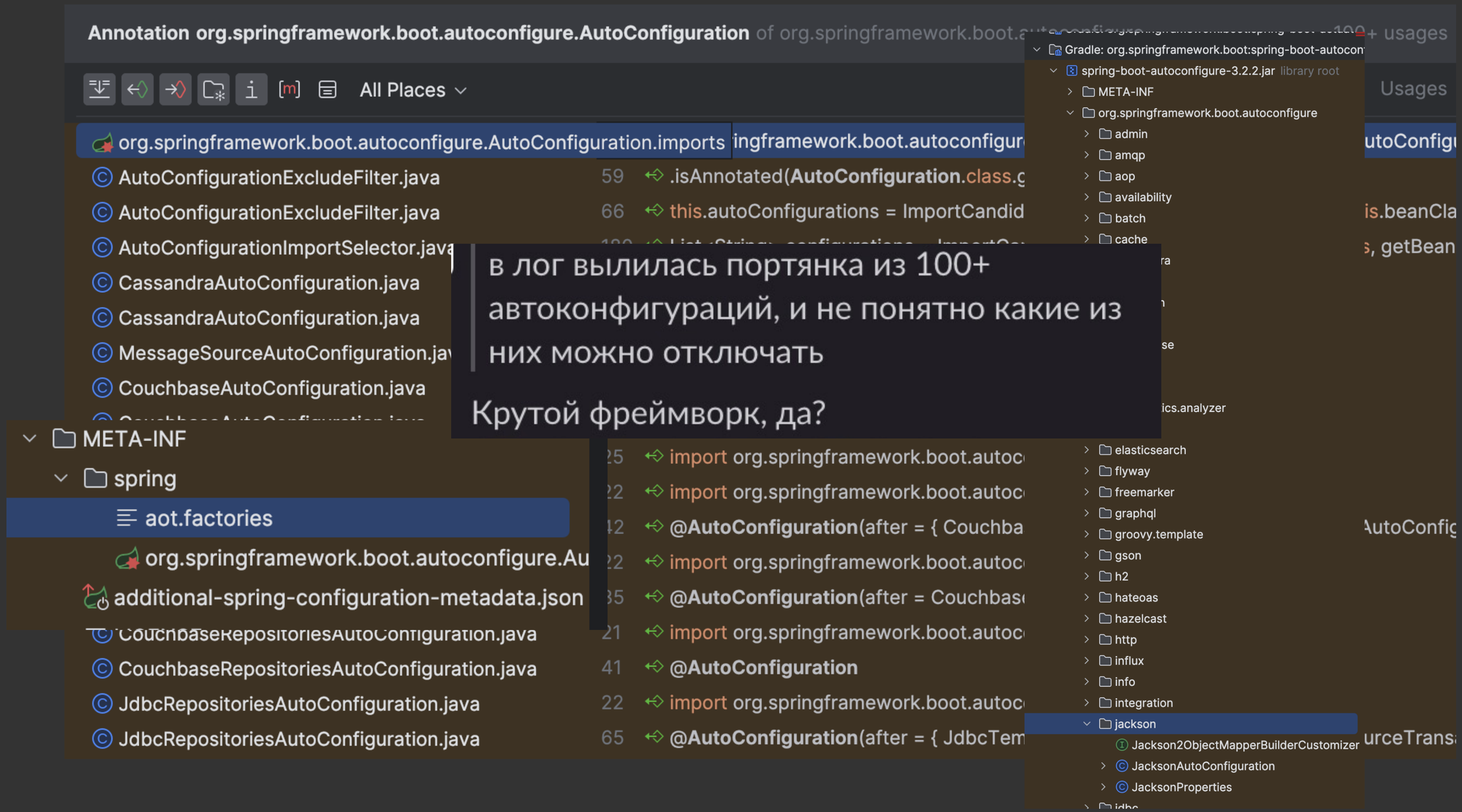
Оптимизация: Spring модули руками

```
dependencies {
    implementation "org.springframework.boot:spring-boot-starter"
    implementation "org.springframework.boot:spring-boot-starter-aop"
    implementation "org.springframework.boot:spring-boot-starter-validation"
    implementation "org.springframework.boot:spring-boot-starter-data-jpa"
}

@SpringBootConfiguration
@ComponentScan(basePackages = {"ru.tinkoff.spring"})
@ImportAutoConfiguration({ ValidationAutoConfiguration.class,
    JpaRepositoriesAutoConfiguration.class,
    HibernateJpaAutoConfiguration.class,
    ConfigurationPropertiesAutoConfiguration.class })
public class Application {

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

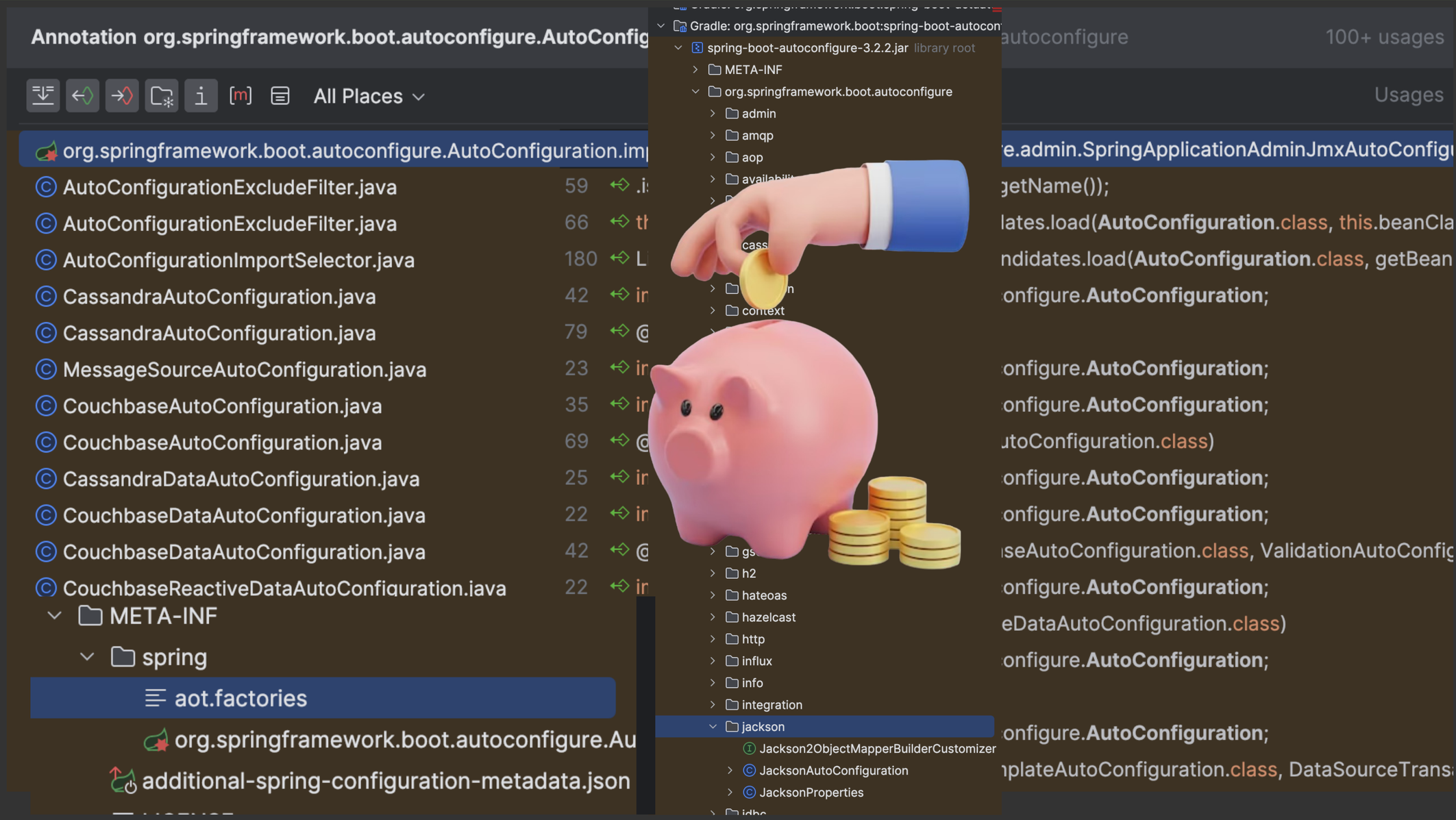
Оптимизация: Spring модули по рукам



Оптимизация: Spring модули по рукам



Оптимизация: Spring модули по рукам



Оптимизировано: Kora внешние модули

```
public interface SomeModule {


    default SomeService someService() {
        return new SomeService();
    }


    default OtherService someOtherService(SomeService someService) {
        return new OtherService(someService);
    }
}
```

Оптимизировано: Kora внешние модули

```
public interface SomeModule {


    default SomeService someService() {
        return new SomeService();
    }


    default OtherService someOtherService(SomeService someService) {
        return new OtherService(someService);
    }
}


@KoraApp
public interface Application extends SomeModule, SomeOtherModule { }
```

Оптимизировано: Kora внешние модули

Подключение

Java Kotlin

Зависимость `build.gradle`:

```
implementation "ru.tinkoff.kora:http-client-async"
```

Модуль:

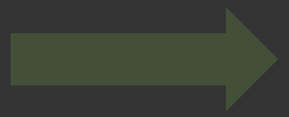
```
@KoraApp
public interface Application extends AsyncHttpClientModule { }
```

Оптимизация: Spring все оптимизации

1. Используйте только необходимые проекту зависимости
2. Используйте эффективные реализации зависимостей (Tomcat -> Undertow)
3. Используйте ~~spring-context-indexer~~ (deprecated) -> ./**gradlew processAot**
4. Ограничьте сканирование пути класса только необходимыми пакетами
5. Используйте ленивую инициализацию компонент
6. Используйте ленивую инициализацию репозиториев
7. Оптимизируйте Spring Boot Actuator
8. Используйте многослойный распакованный **jar**
9. Используйте последнюю версию JVM
10. Настройте параметры сборки мусора JVM
11. Оптимизируйте настройки JVM, используйте флаги JVM
12. Исключите ненужные авто-конфигурации
13. Удалите Spring Boot Actuator
14. Используйте AppCDS
15. Отключите внедрение зависимостей через аннотации
16. Удалите Spring Boot
17. Отключите JMX, Logback, Jackson, Hibernate Validation
18. Удалите Spring

Оптимизация: Spring рабочие оптимизации

Применяем



1. Используйте только необходимые проекту зависимости
2. Используйте эффективные реализации зависимостей (Tomcat -> Undertow)
3. Используйте ~~spring-context-indexer~~ (deprecated) -> ./**gradlew processAot**
4. Ограничьте сканирование пути класса только необходимыми пакетами
5. ~~Используйте ленивую инициализацию компонент~~
6. ~~Используйте ленивую инициализацию репозиториев~~
7. Оптимизируйте Spring Boot Actuator
8. Используйте многослойный распакованный **jar**
9. Используйте последнюю версию JVM
10. Настройте параметры сборки мусора JVM
11. Оптимизируйте настройки JVM, используйте флаги JVM
12. Исключите ненужные авто-конфигурации
13. ~~Удалите Spring Boot Actuator~~
14. ~~Используйте AppCDS~~
15. ~~Отключите внедрение зависимостей через аннотации~~
16. ~~Удалите Spring Boot~~
17. ~~Отключите JMX, Logback, Jackson, Hibernate Validation~~
18. ~~Удалите Spring~~

Оптимизация: Spring цель оптимизации

Используем Spring Boot 3.2.2 в сервисе Hello World:

- Spring Web

Оптимизация: Spring цель оптимизации

Используем Spring Boot 3.2.2 в сервисе PetClinic:

- Spring Web
- Spring Data JDBC

Оптимизация: Spring цель оптимизации

Используем Spring Boot 3.2.2 в сервисе PetClinic x10:

- Spring Web
- Spring Data JDBC

Оптимизация: Spring цель оптимизации

Используем Spring Boot 3.2.2 в сервисе PetClinic x10:

- Spring Web
- Spring Data JDBC
- Spring Actuator
- Springdoc OpenAPI

Оптимизация: Spring цель оптимизации

Используем Spring Boot 3.2.2 в сервисе PetClinic x10:

- Spring Web
- Spring Data JDBC
- Spring Actuator
- Springdoc OpenAPI
- Validation Hibernate
- Cache Caffeine
- Resilient4j
- Quartz
- Logback

Оптимизация: Spring цена оптимизации

Конфигурация сервиса

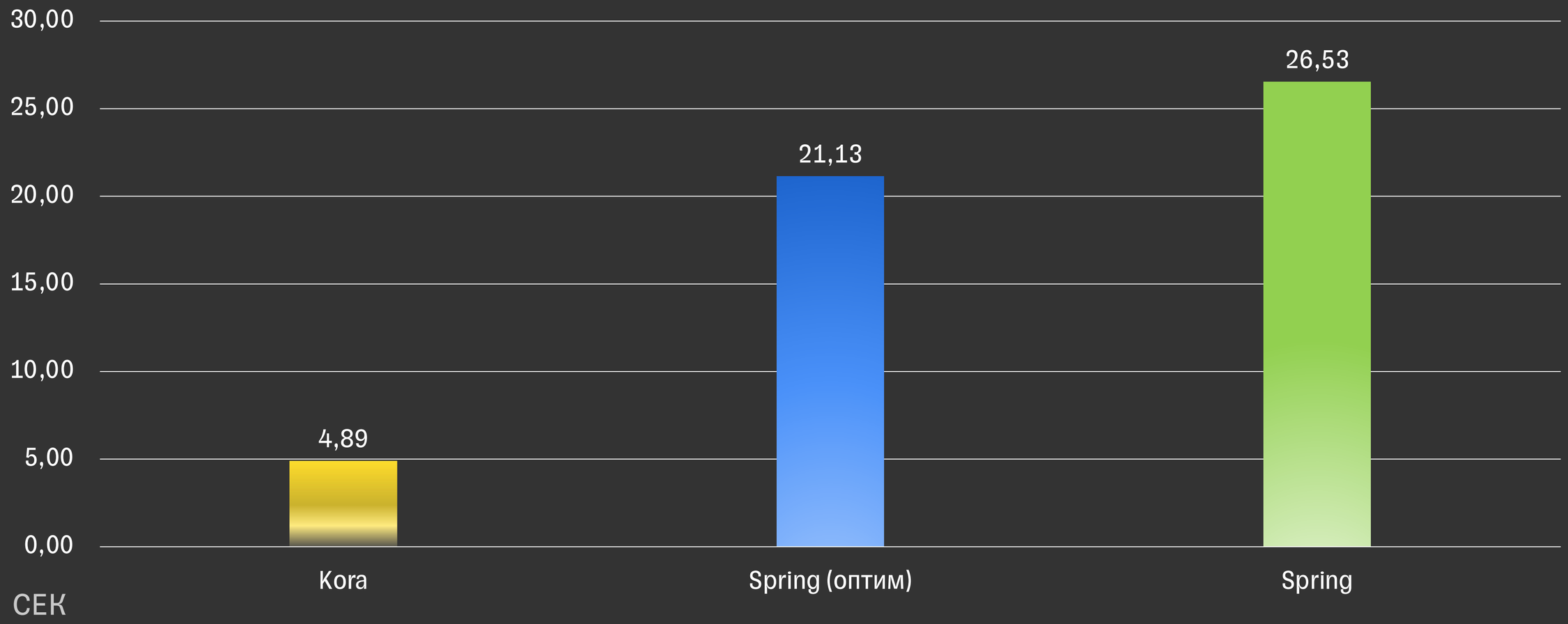
```
@ImportAutoConfiguration({
    ConfigurationPropertiesAutoConfiguration.class,
    AopAutoConfiguration.class,
    JacksonAutoConfiguration.class,
    DispatcherServletAutoConfiguration.class,
    TaskExecutionAutoConfiguration.class,
    ValidationAutoConfiguration.class,
    ProxyCachingConfiguration.class,
    PetCacheConfiguration.class,
    CacheAutoConfiguration.class,
    CircuitBreakerAutoConfiguration.class,
    CircuitBreakerMetricsAutoConfiguration.class,
    TimeLimiterAutoConfiguration.class,
    TimeLimiterMetricsAutoConfiguration.class,
    RetryAutoConfiguration.class,
    RetryMetricsAutoConfiguration.class,
    ServletWebServerFactoryAutoConfiguration.class,
    WebMvcAutoConfiguration.class,
    HttpEncodingAutoConfiguration.class,
    ServletManagementContextAutoConfiguration.class,
    ActuatorConfiguration.class,
    EndpointAutoConfiguration.class,
    ServletEndpointManagementContextConfiguration.class,
    WebMvcEndpointManagementContextConfiguration.class,
    AvailabilityHealthContributorAutoConfiguration.class,
    ApplicationAvailabilityAutoConfiguration.class,
    AvailabilityProbesAutoConfiguration.class,
    WebEndpointAutoConfiguration.class,
    ManagementContextAutoConfiguration.class,
    JdbcConverterConfiguration.class,
    DataSourceAutoConfiguration.class,
    JdbcTemplateAutoConfiguration.class,
    DataSourceTransactionManagerAutoConfiguration.class,
    TransactionManagerCustomizationAutoConfiguration.class,
    TransactionAutoConfiguration.class,
    JdbcRepositoriesAutoConfiguration.class,
    SpringDocConfiguration.class,
    SpringDocWebMvcConfiguration.class,
    SwaggerUiConfigProperties.class,
    SwaggerUiOAuthProperties.class,
    SwaggerConfig.class,
    MultipleOpenApiSupportConfiguration.class,
    SpringDocUIConfiguration.class,
    SpringDocConfigProperties.class,
})
@EnableJdbcRepositories
@EnableConfigurationProperties
@EnableCaching
@EnableTransactionManagement
@EnableAspectJAutoProxy
@SpringBootConfiguration
@ComponentScan(nameGenerator = FullyQualifiedAnnotationBeanNameGenerator.class,
        basePackages = {
            "ru.tinkoff.spring.crud.big",
            "ru.tinkoff.spring.crud.big.openapi.server" },
        excludeFilters = {
            @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
            @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
public class OptimizedOpenApiGeneratorApplication {

  public static void main(String[] args) {
    SpringApplication.run(OptimizedOpenApiGeneratorApplication.class, args);
  }

  @Bean(name = "ru.tinkoff.spring.crud.big.OpenApiGeneratorApplication.jsonNullableModule")
  public Module jsonNullableModule() {
    return new JsonNullableModule();
  }
}
```

```
@Configuration
@Import({
    RabbitHealthContributorAutoConfiguration.class,
    AuditAutoConfiguration.class,
    AuditEventsEndpointAutoConfiguration.class,
    AvailabilityHealthContributorAutoConfiguration.class,
    AvailabilityProbesAutoConfiguration.class,
    BeansEndpointAutoConfiguration.class,
    CachesEndpointAutoConfiguration.class,
    CassandraHealthContributorAutoConfiguration.class,
    CassandraReactiveHealthContributorAutoConfiguration.class,
    ConditionsReportEndpointAutoConfiguration.class,
    ConfigurationPropertiesReportEndpointAutoConfiguration.class,
    ShutdownEndpointAutoConfiguration.class,
    EndpointAutoConfiguration.class,
    JacksonEndpointAutoConfiguration.class,
    JmxEndpointAutoConfiguration.class,
    WebEndpointAutoConfiguration.class,
    EnvironmentEndpointAutoConfiguration.class,
    HealthContributorAutoConfiguration.class,
    HealthEndpointAutoConfiguration.class,
    InfoContributorAutoConfiguration.class,
    InfoEndpointAutoConfiguration.class,
    IntegrationGraphEndpointAutoConfiguration.class,
    DataSourceHealthContributorAutoConfiguration.class,
    JmsHealthContributorAutoConfiguration.class,
    LdapHealthContributorAutoConfiguration.class,
    LogFileWebEndpointAutoConfiguration.class,
    LoggersEndpointAutoConfiguration.class,
    HeapDumpWebEndpointAutoConfiguration.class,
    ThreadDumpEndpointAutoConfiguration.class,
    CompositeMeterRegistryAutoConfiguration.class,
    JvmMetricsAutoConfiguration.class,
    KafkaMetricsAutoConfiguration.class,
    Log4J2MetricsAutoConfiguration.class,
    LogbackMetricsAutoConfiguration.class,
    MetricsAspectsAutoConfiguration.class,
    MetricsAutoConfiguration.class,
    MetricsEndpointAutoConfiguration.class,
    SystemMetricsAutoConfiguration.class,
    CacheMetricsAutoConfiguration.class,
    RepositoryMetricsAutoConfiguration.class,
    JmxMetricsExportAutoConfiguration.class,
    OtlpMetricsExportAutoConfiguration.class,
    PrometheusMetricsExportAutoConfiguration.class,
    SimpleMetricsExportAutoConfiguration.class,
    DataSourcePoolMetricsAutoConfiguration.class,
    StartupTimeMetricsListenerAutoConfiguration.class,
    TaskExecutorMetricsAutoConfiguration.class,
    ObservationAutoConfiguration.class,
    WebMvcObservationAutoConfiguration.class,
    org.springframework.boot.actuate.autoconfigure.opentelemetry.OpenTelemetryAutoConfiguration.class,
    StartupEndpointAutoConfiguration.class,
    DiskSpaceHealthContributorAutoConfiguration.class,
    BraveAutoConfiguration.class,
    MicrometerTracingAutoConfiguration.class,
    NoopTracerAutoConfiguration.class,
    org.springframework.boot.actuate.autoconfigure.tracing.OpenTelemetryAutoConfiguration.class,
    OtlpAutoConfiguration.class,
    PrometheusExemplarsAutoConfiguration.class,
    HttpExchangesAutoConfiguration.class,
    HttpExchangesEndpointAutoConfiguration.class,
    MappingsEndpointAutoConfiguration.class,
    ReactiveManagementContextAutoConfiguration.class,
    ManagementContextAutoConfiguration.class,
    ServletManagementContextAutoConfiguration.class,
})
public class OptimizedActuatorConfiguration {

}
```

Порядок
Обычных
Конфигураций
Может
Иметь
Значение

Оптимизация: Spring жертва оптимизации

Kora стартует в 5.4 раз быстрее Spring из коробки
и в 4.3 раза быстрее Spring (оптимизированного)



Запуск в Docker контейнере (1 CPU и 1 Gb) на Macbook Pro 2019 (Intel I7-9750H, 6 ядер, 2.6-4.5 ГГц) среднее по 5 прогонам

Оптимизация: Spring жертва сборки

А сколько стоит сборка?

Оптимизация: Spring жертва сборки

distTar (для Kora)

./gradlew clean bootJar (для Spring)

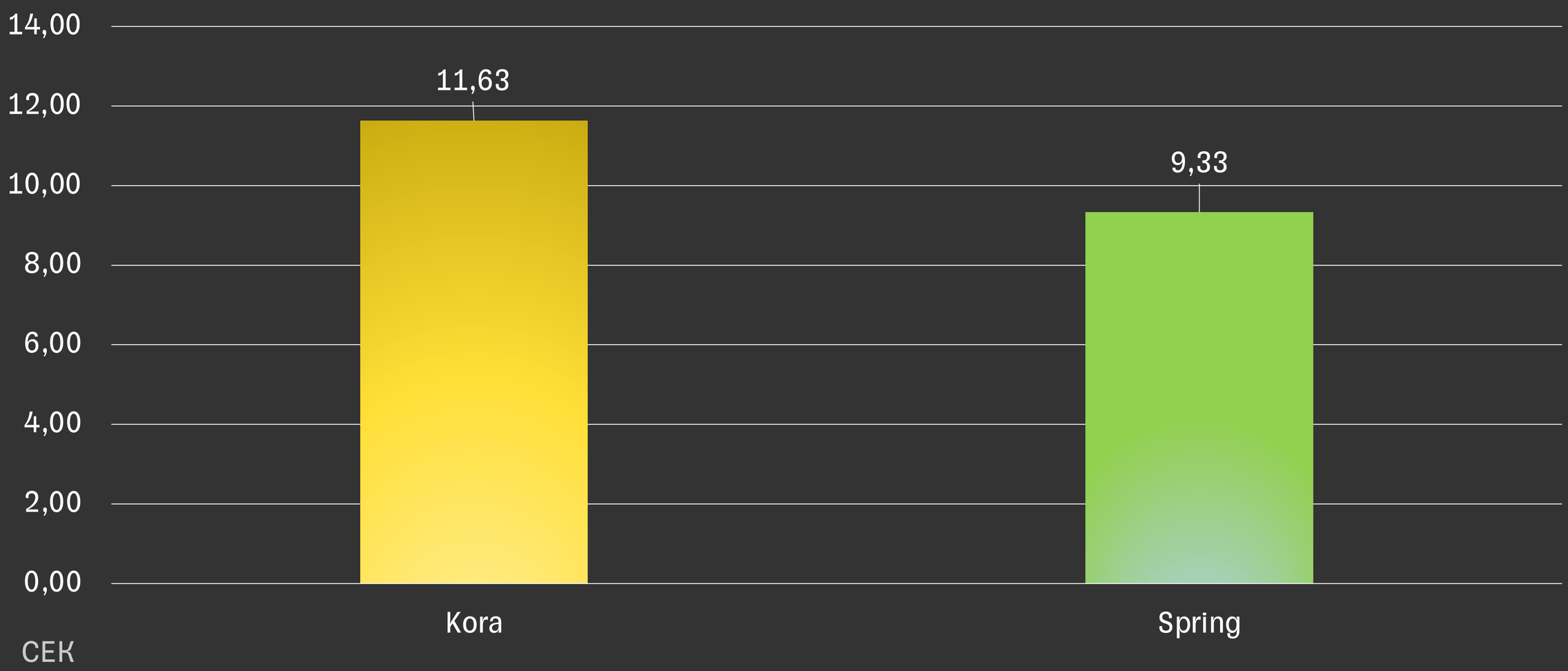
--no-build-cache

--no-configuration-cache

--no-parallel

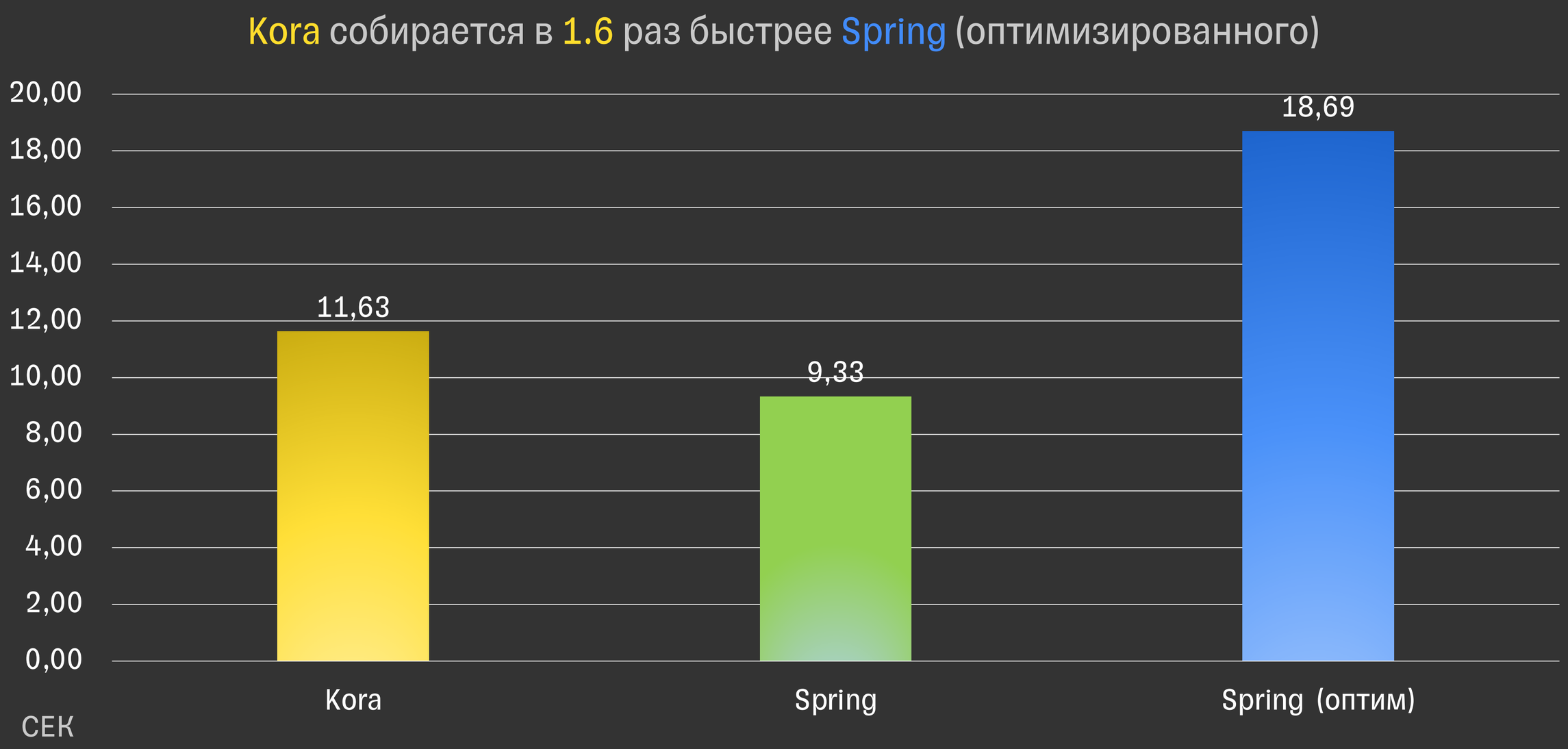
-Porg.gradle.daemon=false

Оптимизация: Spring жертва сборки



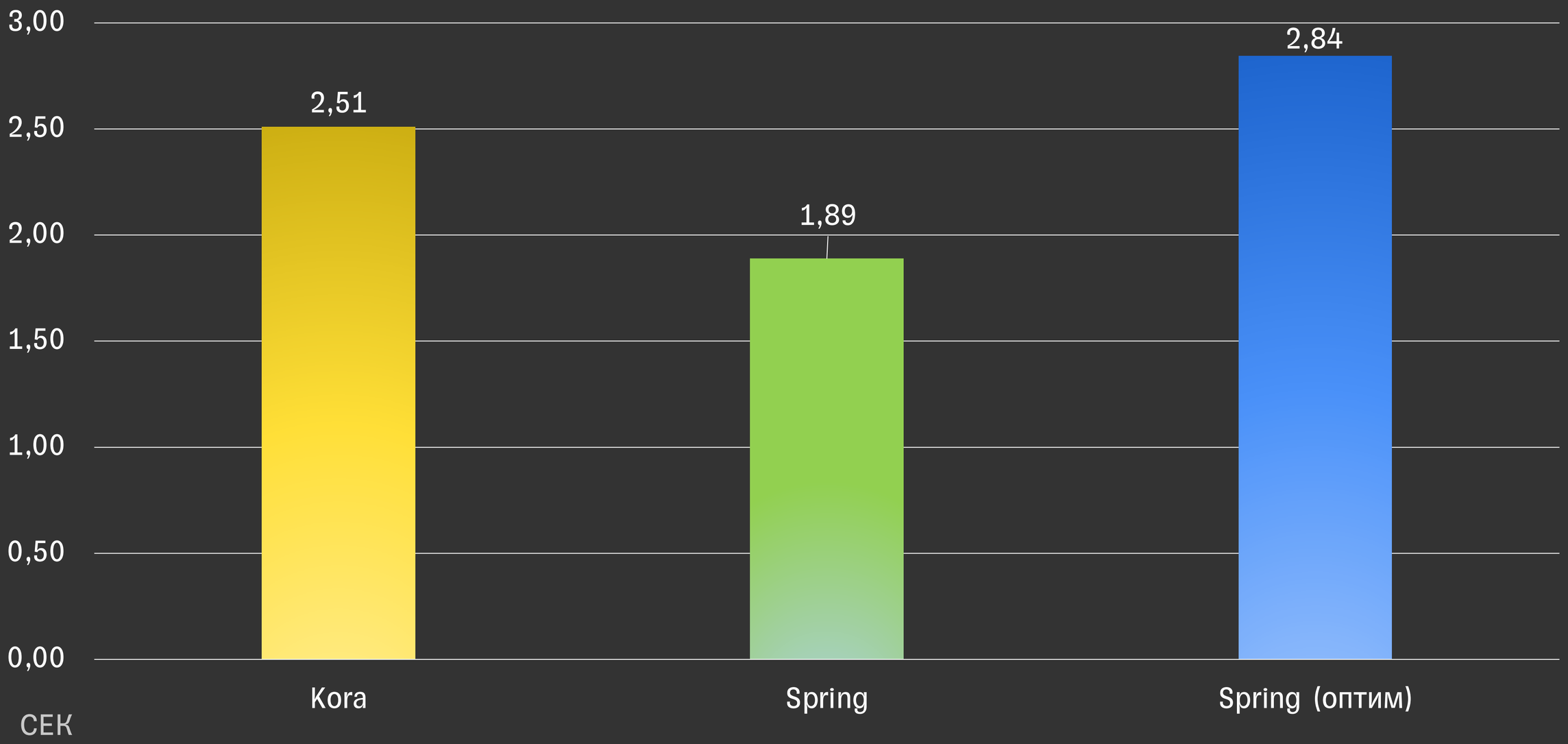
Сборка на Macbook Pro 2019 (Intel I7-9750H, 6 ядер, 2.6-4.5 ГГц) среднее по 5 прогонам

Оптимизация: Spring жертва сборки



Сборка на Macbook Pro 2019 (Intel I7-9750H, 6 ядер, 2.6-4.5 ГГц) среднее по 5 прогонам

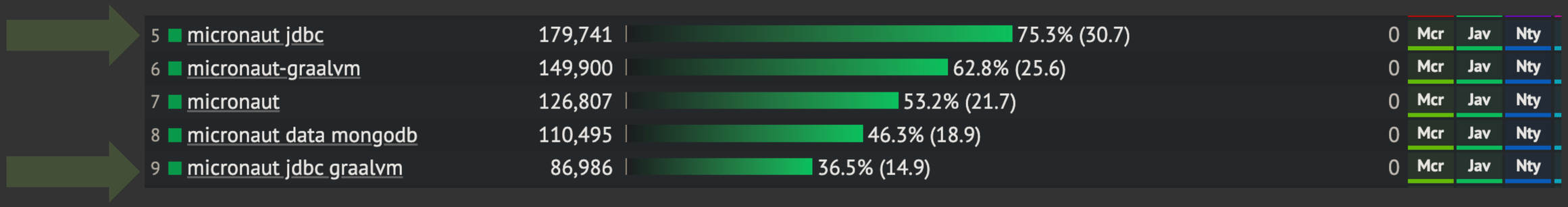
Оптимизация: Spring обычная сборка



Сборка (с вкл. опциями) на Macbook Pro 2019 (Intel I7-9750H, 6 ядер, 2.6-4.5 ГГц) среднее по 5 прогонам

Оптимизация: Spring и GraalVM Native Image

1. Порождает кучу новых проблем
2. Другая парадигма разработки и доставки приложений
3. Лишение привычных инструментов отладки и профилирования
4. Жертвует предельной производительностью



Оптимизация: Spring цена

1. Разработчики тратят уйму времени на «оптимизации»
2. Ленивая инициализация это не оптимизация
3. Оптимизации могут требовать издержек по функционалу
4. Значительные оптимизации ни кто не готов «оплачивать»
5. Значительные оптимизации не стоят того за что их продают

Оптимизация: Kora доступные оптимизации

1. Используйте последнюю версию JVM
2. Настройте параметры сборки мусора JVM
3. Оптимизируйте настройки JVM, используйте флаги JVM

Итог


свинья-копилка накопила
обиду, ненависть и боль

Итог: Spring бинго

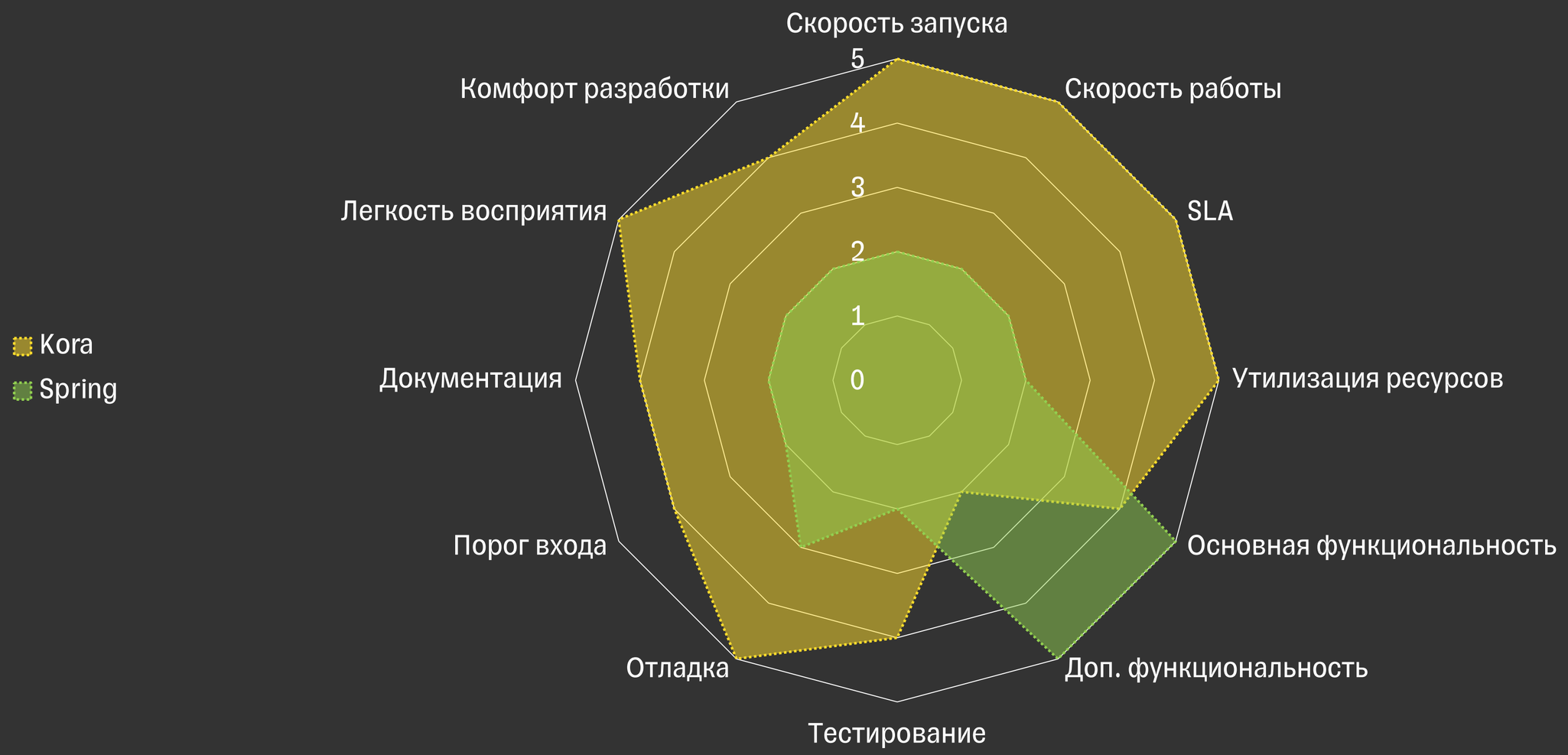
1. Архаичный инструментарий и ошибки архитектуры
2. Много способов сделать одну и ту же работу
3. Переусложненные инструменты и тестирование
4. Держим огромный контекст в голове
5. Теряем временя на оптимизации, костыли, магию
6. Неочевидная обратная связь
7. Порой нет рекомендаций и практик от самих создателей

Вечное бинго где балансируем между функциональностью и ее последствиями

Считаем

1. Скорость запуска
2. Скорость работы
3. SLA
4. Утилизация ресурсов
5. Основная функциональность
6. Доп. функциональность
7. Тестирование
8. Отладка
9. Порог входа
10. Документация
11. Легкость восприятия
12. Комфорт разработки

Итог: цена на карте



Итог: карта фреймворков

Сайт

Telegram