Тестируем и плачем

Вместе со Spring Boot Test

@tolkv

@lavcraft

ЦИАН

alfalab

РАЗБОР ПОЛЁТОВ
#RAZBORPOLETOV
SINCE 2011
EST 5772
ПОДКАСТ
ДОБРЫЙ ДЕНЬ ДОБРЫЙ ВЕЧЕР
ДОБРОЕ УТРО ДОБРОЙ НОЧИ

#cianfamily

ЦИАН

@jekaborisov

@jeka1978

NAYA TECHNOLOGIES

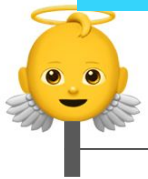
В программе

Тестирование живого приложения

- Старые подходы
  - **@ContextConfiguration**
  - **@ContextHierarchy && @DirtiesContext**
  - **@ActiveProfiles**
- Что нового нам приготовил Spring Boot?
  - **@SpringBootTest**
  - **@TestConfiguration**
  - **@MockBean && @SpyBean && @*Beans**
  - **@DataJpaTest**
  - **@MvcTest**
- Кэширование **spring** контекстов
- Шкала тестов

Немного теории

Шкала Тестов



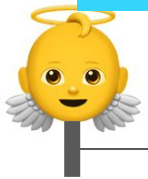
Unit

Component Test

Microservice Test

System Test

Шкала Тестов



Unit

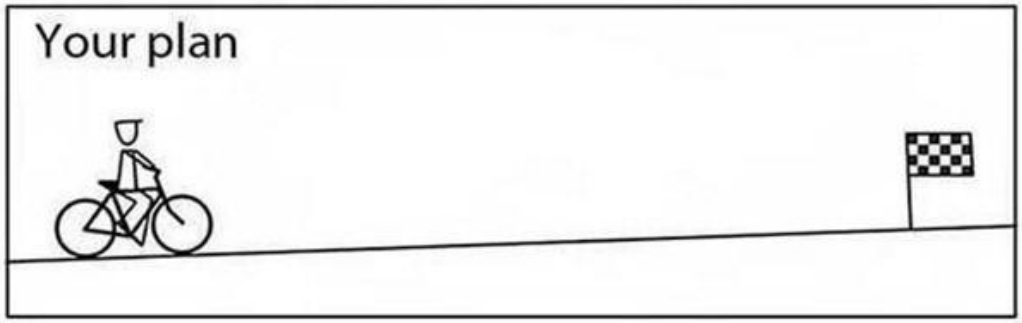
Component Test

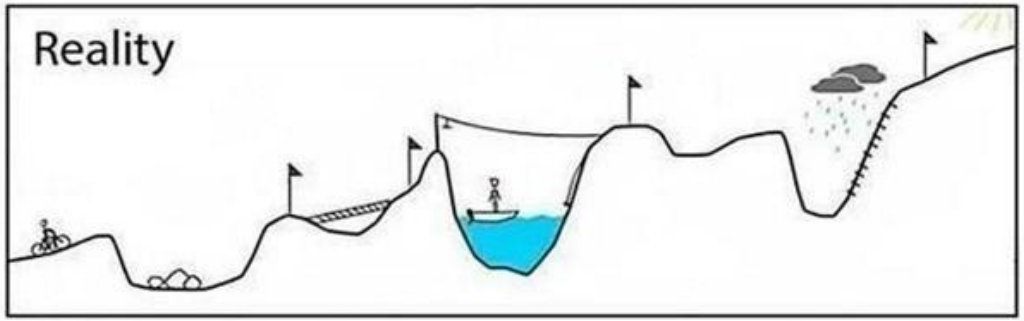
Microservice Test

System Test

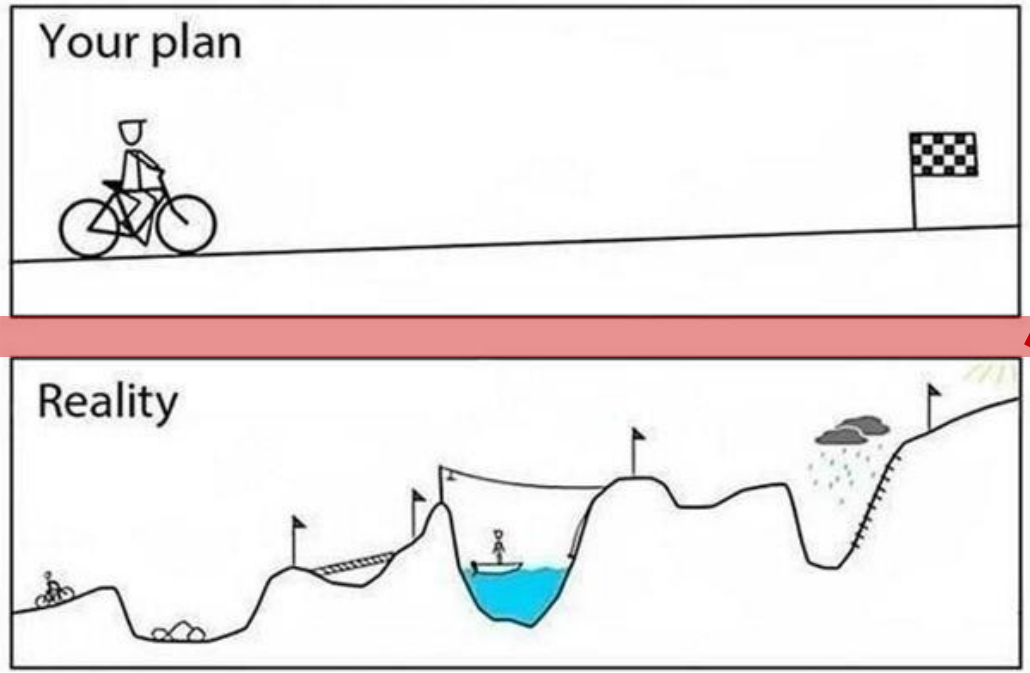
Unit/Component тесты. Для чего?

Unit/Component тесты. Для чего?



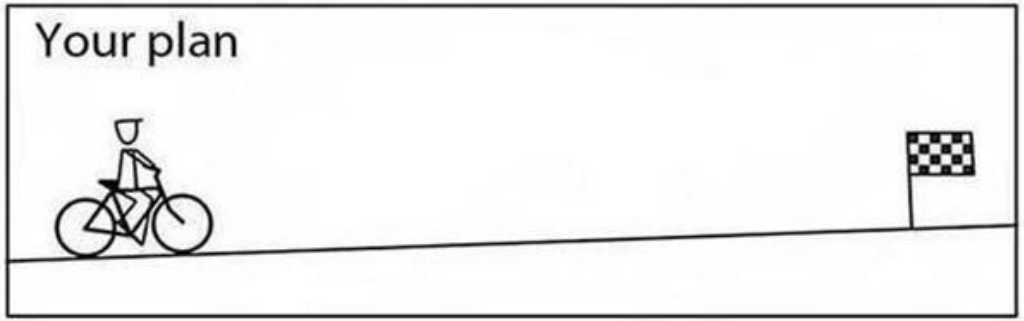


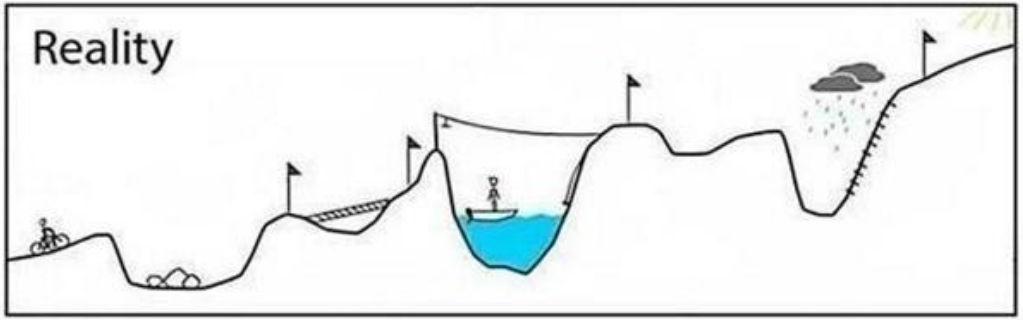
Unit/Component тесты. Для чего?



Ваши тесты
Тут

Unit/Component тесты. Для чего?

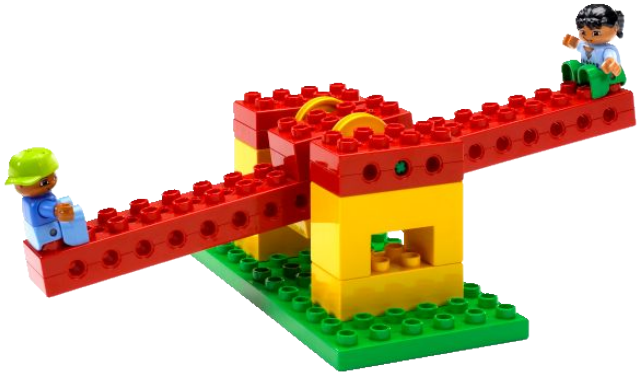




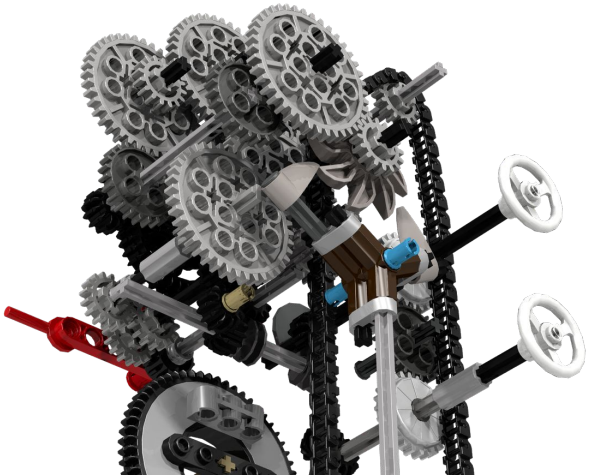
Тесты уменьшают неопределённость

Есть два типа тестов

Простой

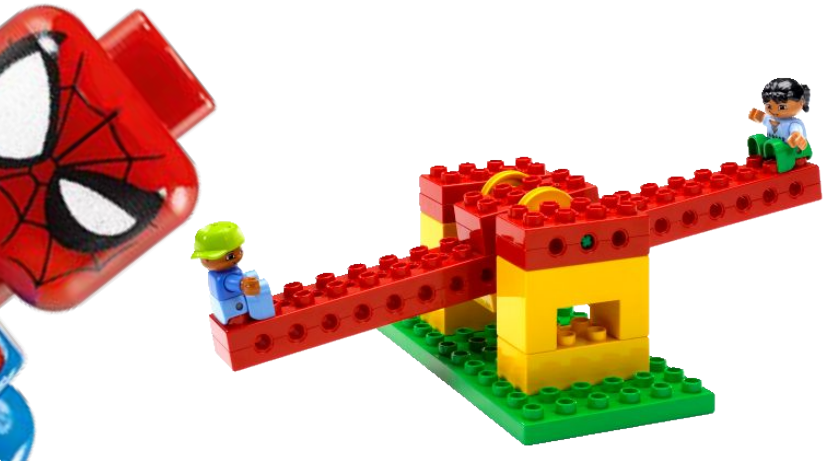


Сложный

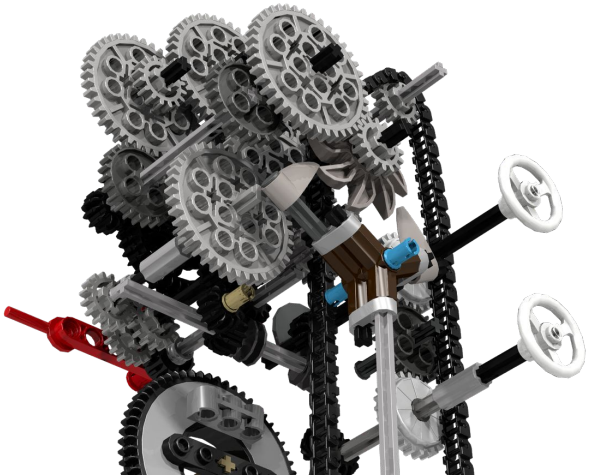


Есть два типа тестов
Какой сам выберешь

Простой

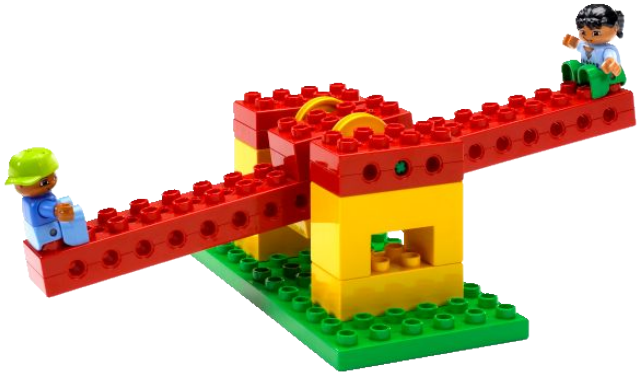


Сложный

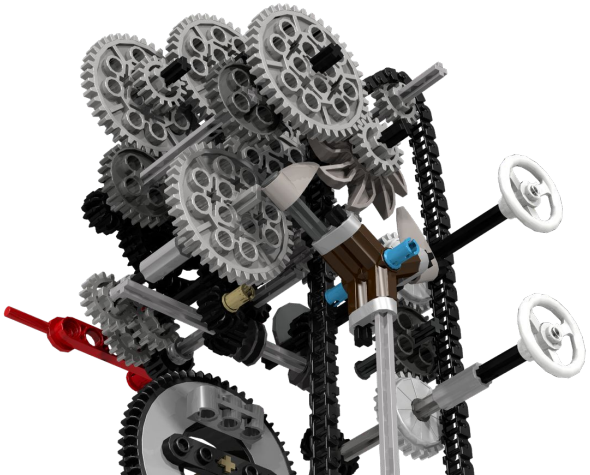


Есть два типа тестов
Какой сам выберешь,
а какой разработчику оставишь?

Простой



Сложный



Когда пишут тесты?

Когда пишут тесты?

1. Требование заказчика

Когда пишут тесты?

1. Требование заказчика
2. **Культура**

Когда пишут тесты?

1. Требование заказчика
2. **Культура** → **Перед кодом**

Когда пишут тесты?

1. Требование заказчика
2. **Культура**

Перед кодом

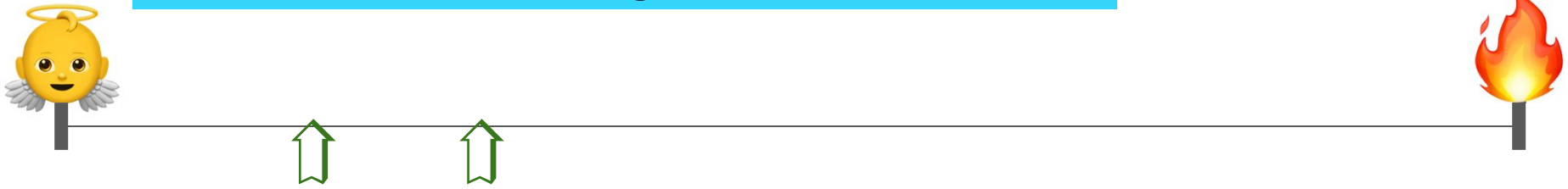
Вместе кодом

Когда пишут тесты?

1. Требование заказчика
2. **Культура**

- **Перед кодом**
- **Вместе кодом**
- **После кода**

Про какие тесты будем говорить?



Unit

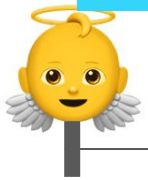
Component Test

Перед кодом

Вместе кодом

После кода

Про какие тесты будем говорить?



Unit

Component Test

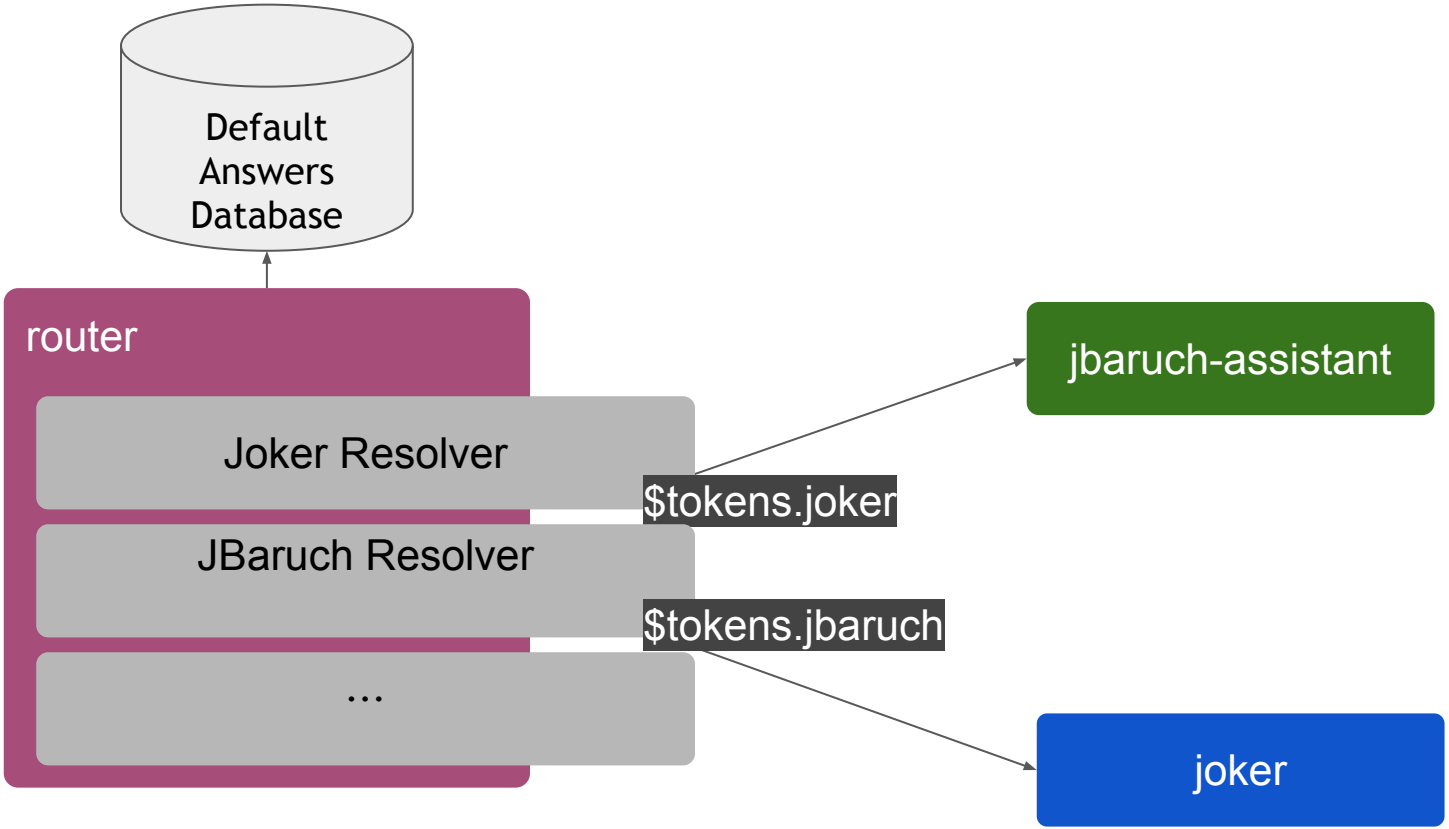
Перед кодом

Вместе кодом

После кода

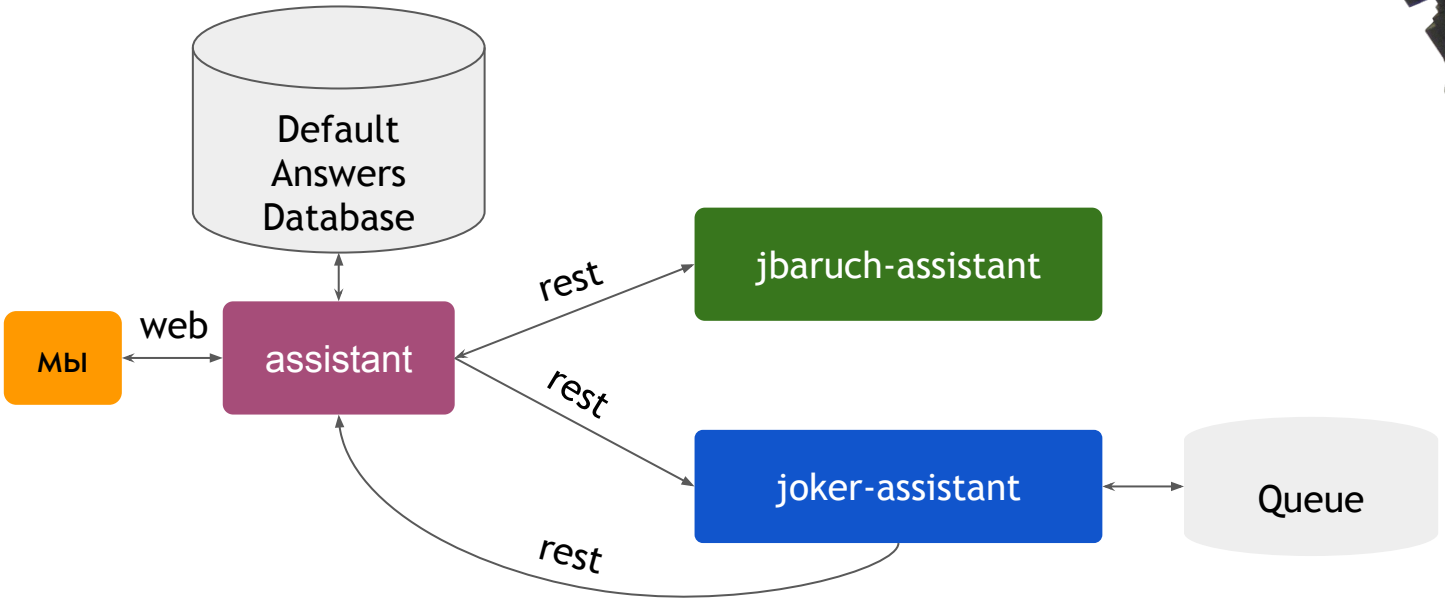
Начнём

Дано


Default
Answers
Database
router
Joker Resolver
JBaruch Resolver
...
$tokens.joker
$tokens.jbaruch
jbaruch-assistant
joker

Дано

Чат поддержки тестировщиков



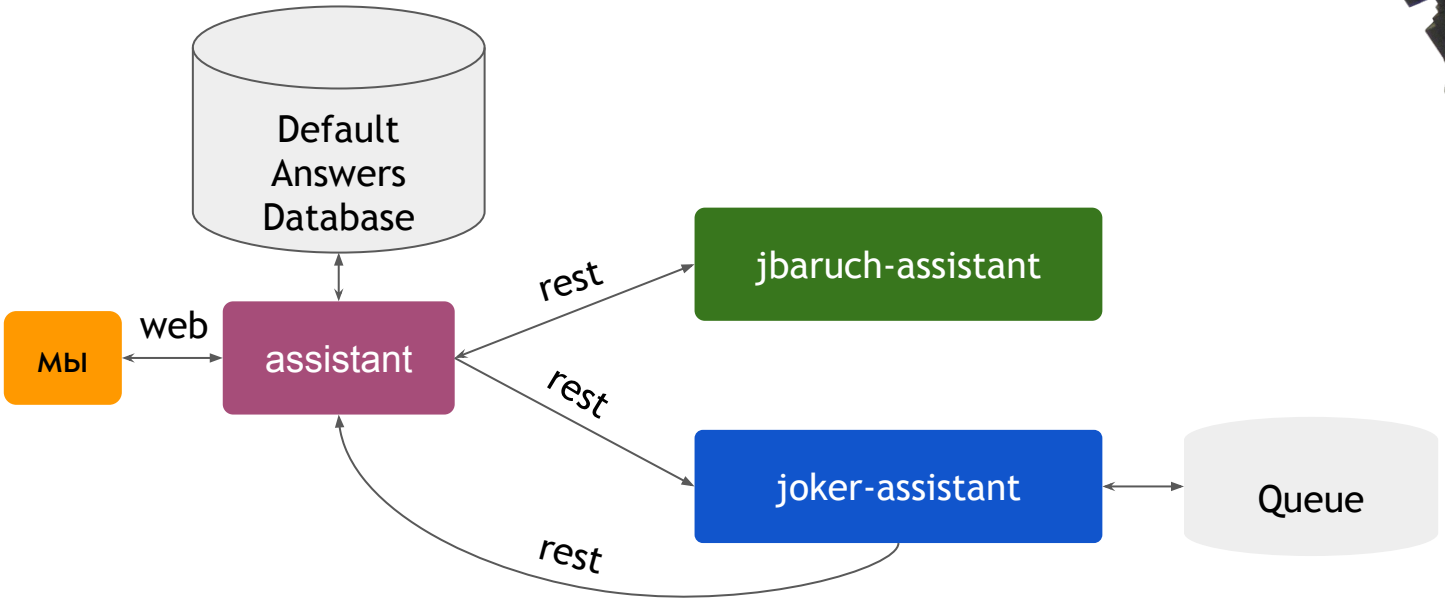
Эксперты

Эксперты


Demo

Дано

Чат поддержки тестировщиков



спросить

↪🐸 ✏ nexus это хороший телефон, всё

↪🐸 ✏ самый лушчий бинарный репозиторий – artifactory

↪🧠 ✏ 🕺 Connected

какой лучший язык программирования

спросить

↪🦄 ✏ Если не помог, заворачивайте декоратор в декоратор пока не поможет!

↪🦄 ✏ Лучший дизайн паттерн – Декоратор. Декоратор может решить любую проблему!

↪🐸 ✏ nexus это хороший телефон, всё

↪🐸 ✏ самый лушчий бинарный репозиторий – artifactory

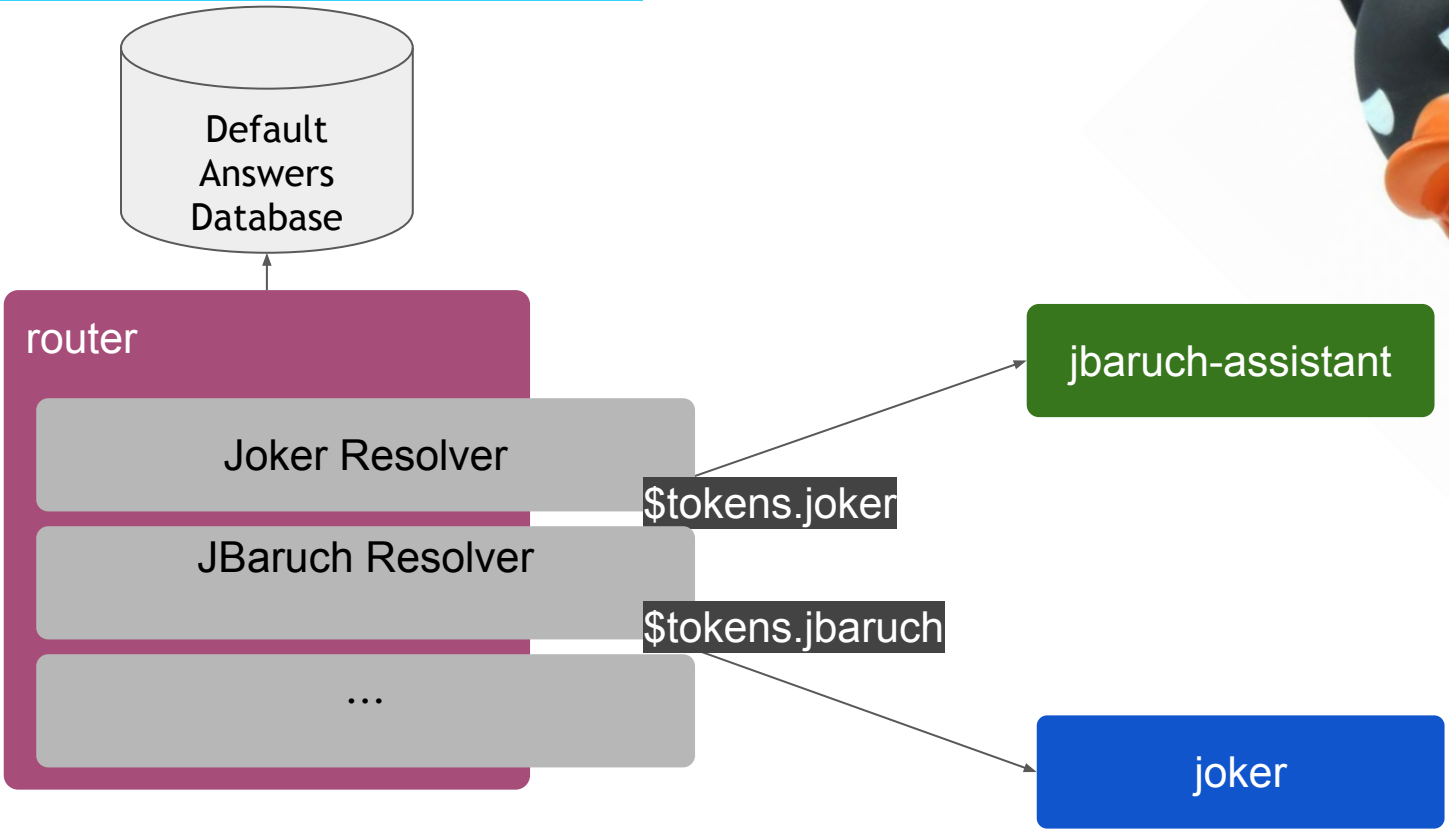
↪🐙 ✏ 🤾 Connected

спросить

↪🦄 ✏ Если не помог, заворачивайте декоратор в декоратор пока не поможет!
↪🦄 ✏ Лучший дизайн паттерн – Декоратор. Декоратор может решить любую проблему!
↪🐸 ✏ nexus это хороший телефон, всё
↪🐸 ✏ самый лушчий бинарный репозиторий – artifactory
↪🐙 ✏ 🕺 Connected

А давайте тестировать

А давайте тестировать



Demo

`JokerWordsFrequencyResolverTest`

А давайте тестировать. Тест #1

1. Пишем `JokerWordsFrequencyResolverTest`.


Demo

Кого тестируем

```
@Component
public class JokerWordsFrequencyResolver  extends AbstractWordsFreqResolver {

 @Value("${tokens.joker}")
 private String answers;

 public JokerWordsFrequencyResolver(WordsComposer wordsComposer) {
   super(wordsComposer);
 }

 @Override
 public QuestionType getQuestionType() {
   return JOKER;
 }
}
```

Тест №1.5

```
public class JokerWordsFrequencyResolverTest {
 @Test
 public void name() throws Exception {
   JokerWordsFrequencyResolver jokerWordsFrequencyResolver =
     new JokerWordsFrequencyResolver(
         ...
       )
   );
   jokerWordsFrequencyResolver.setAnswers( "objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Тест №1.5

```
public class JokerWordsFrequencyResolverTest {
 @Test
 public void name() throws Exception {
   JokerWordsFrequencyResolver jokerWordsFrequencyResolver =
     new JokerWordsFrequencyResolver(
       new WordsComposer(
           ...
       )
   );
   jokerWordsFrequencyResolver.setAnswers( "objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Тест №1.5

```
public class JokerWordsFrequencyResolverTest {
 @Test
 public void name() throws Exception {
   JokerWordsFrequencyResolver jokerWordsFrequencyResolver =
     new JokerWordsFrequencyResolver(
       new WordsComposer(
           new GarbageProperties()
       )
   );
   jokerWordsFrequencyResolver.setAnswers( "objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Тест №1

```
public class JokerWordsFrequencyResolverTest {
 @Test
 public void name() throws Exception {
   JokerWordsFrequencyResolver jokerWordsFrequencyResolver =
     new JokerWordsFrequencyResolver(
       new WordsComposer(
           new GarbageProperties()
       )
   );
   jokerWordsFrequencyResolver.setAnswers( "objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Тест №1

```
public class JokerWordsFrequencyResolverTest {
 @Test
 public void name() throws Exception {
   JokerWordsFrequencyResolver jokerWordsFrequencyResolver =
     new JokerWordsFrequencyResolver(
       new WordsComposer(
           new GarbageProperties()
       )
   );
   jokerWordsFrequencyResolver.setAnswers( "objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

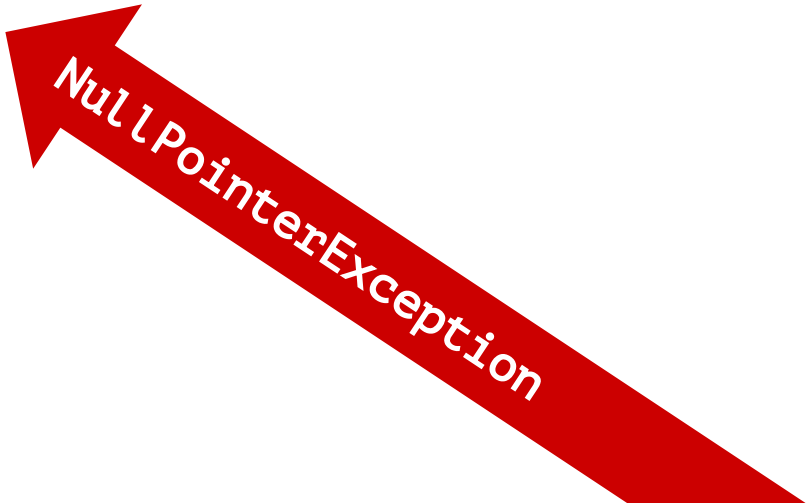
Результат

```
java.lang.NullPointerException
    at … .(WordsComposer.java:48)
```

Not Passed

WordsComposer:48

```
garbageProperties.getGarbage()
    .contains(s.toLowerCase())
```

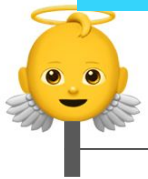


WordsComposer:48

А давайте тестировать. Тест #1

1. Пишем `JokerWordsFrequencyResolverTest`.
2. Как ни крути, но нужен более “интеграционный тест”

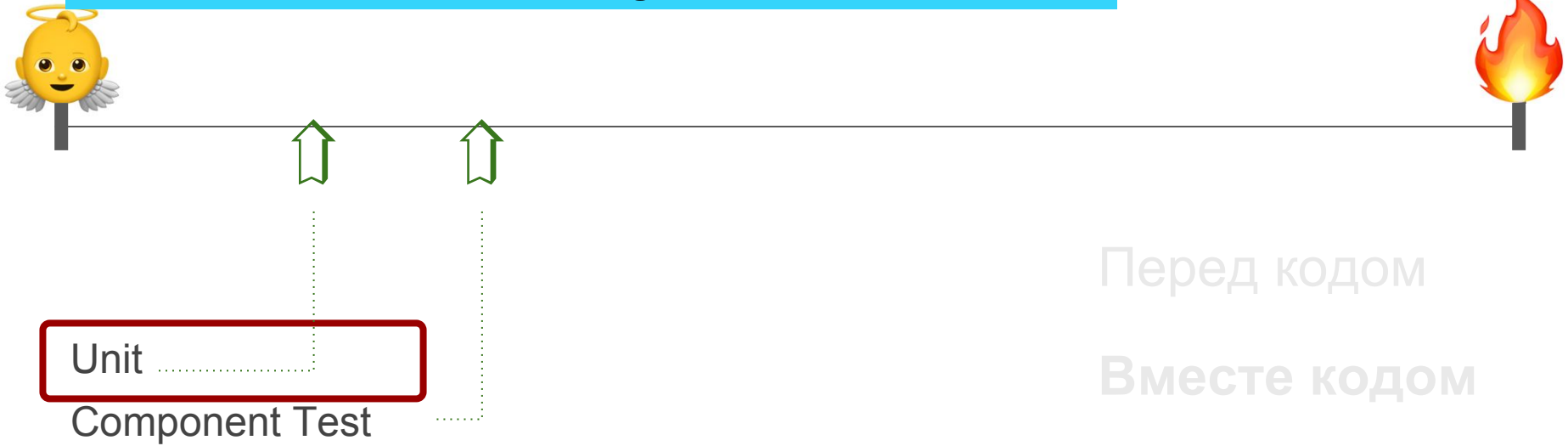
Шкала Тестов



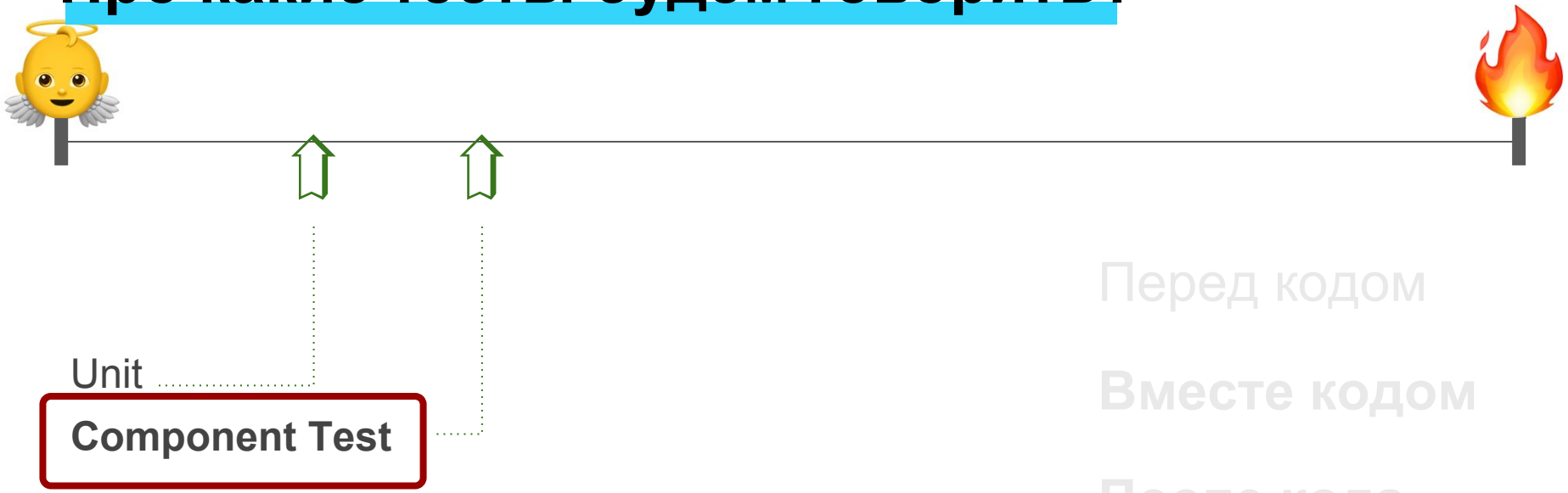
Unit

Шкала Тестов

Про какие тесты будем говорить?



Про какие тесты будем говорить?



Unit

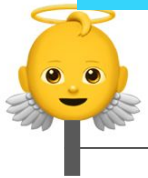
Component Test

Перед кодом

Вместе кодом

После кода

Про какие тесты будем говорить?



Инициализируется Spring`ом

Перед кодом

Вместе кодом

Unit

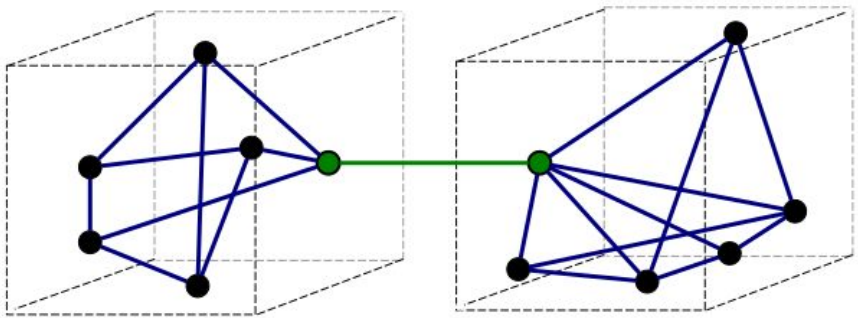
Component Test

```
@Value("${garbage}")
void setGarbage(String[] garbage) {
```

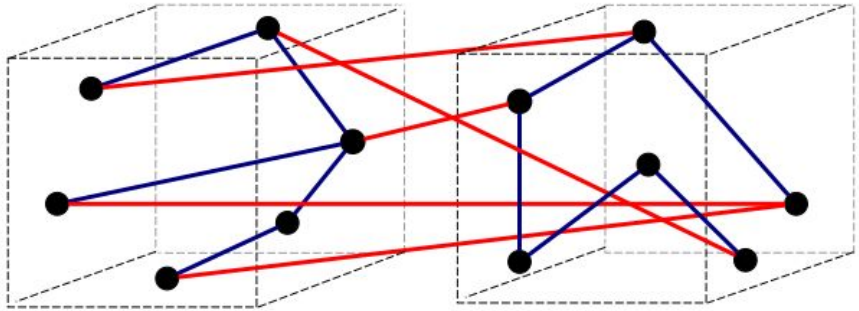
Ещё немного теории

IoC, DI, Spring и друзья

IoC, DI, Spring и друзья

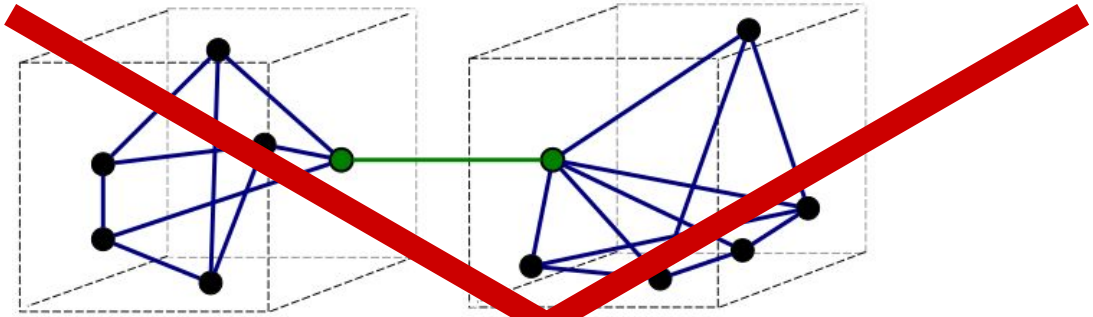


a) Слабое зацепление, сильная связность

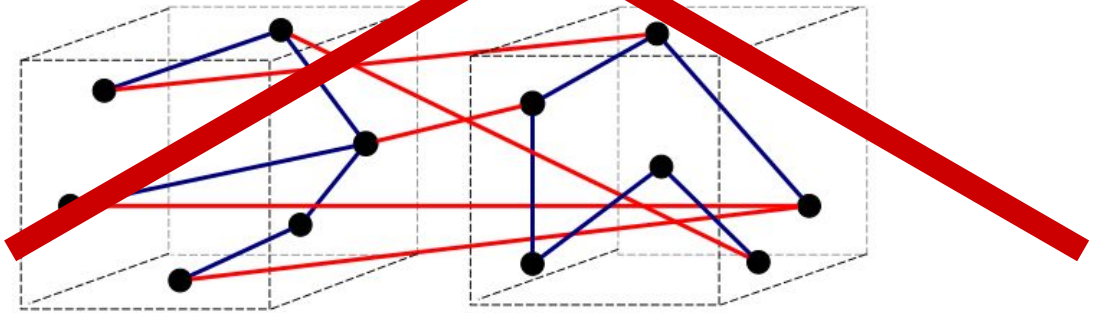


b) Сильное зацепление, слабая связность

IoC, DI, Spring и друзья



a) Слабое зацепление, сильная связность



b) Сильное зацепление, слабая связность

IoC, DI, Spring

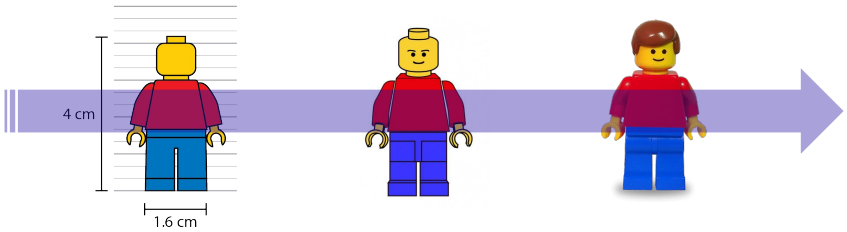
Кино про супергероев

IoC, DI, Spring

Кино про супергероев

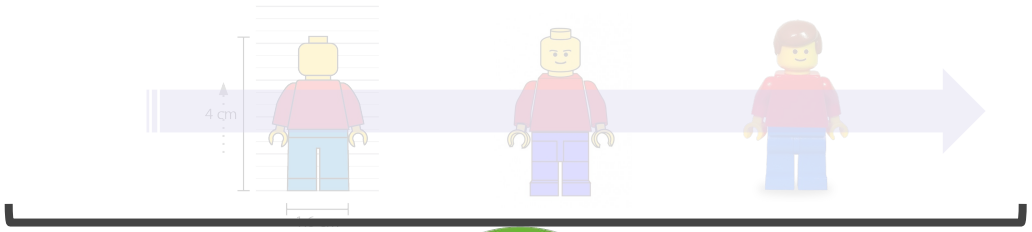
IoC, DI, Spring

Кино про супергероев

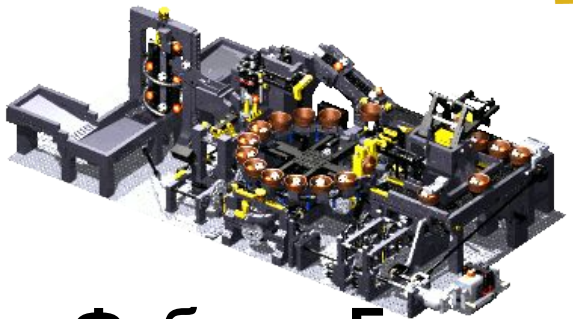


IoC, DI, Spring

Кино про супергероев



IoC, DI, Spring



ФабрикаГероев

IoC, DI, Spring и друзья

IoC для инверсии поведения

IoC, DI, Spring и друзья

IoC для инверсии поведения

```
public class СуперЗлодейТест {

    @Before
    public void setUp() throws Exception {
        ...
    }

}
```

Тоже инверсия контроля

IoC, DI, Spring и друзья

```
public class СъемочнаяПлощадка {
    public static void main(String[] args) {
        Киношка съёмка = new Киношка().снимать();

        съёмка.герой.бить();
        съёмка.злодей.страдать();
        съёмка.злодей.бить();
        съёмка.герой.страдать();
        съёмка.герой.страдать();
    }
}
```

IoC, DI, Spring и друзья

```
public class СъемочнаяПлощадка {
    public static void main(String[] args) {
        Киношка съёмка = new Киношка().снимать();

        съёмка.герой.бить();
        съёмка.злодей.страдать();
        съёмка.злодей.бить();
        съёмка.герой.страдать();
        съёмка.герой.страдать();
    }
}
```

NullPointerException

IoC, DI, Spring и друзья

```
public class Киношка {
    СуперГерой  герой;
    СуперЗлодей злодей;

    public Киношка снимать() {
        return new Киношка();
    }
}
```

IoC, DI, Spring и друзья

```
public class СуперГерой implements Герой {
    private СуперЗлодей вражина;

    @Override
    public void бить() {
        вражина.бить();
    }
}
```

Кто проставляет?

```
public class СуперЗлодей implements Герой {
    private СуперГерой вражина;

    @Override
    public void бить() {
        вражина.страдать();
    }
}
```

IoC, DI, Spring и друзья

```
public class ФабрикаГероев {
    public Object родить() {
        if (new Random().nextBoolean()) {
            return new СуперГерой();
        }
        return new СуперЗлодей();
    }
}
```

IoC, DI, Spring и друзья

```
public class ФабрикаГероев {
    public Object родить() {
        if (new Random().nextBoolean()) {
            return new СуперГерой();
        }
        return new СуперЗлодей();
    }
}
```

IoC, DI, Spring и друзья

IoC, DI, Spring и друзья

- @Autowired
- @Component/@Service
- @Configuration

IoC, DI, Spring и друзья

- @Autowired
- @Component/@Service
- @Configuration

```
@Component
public class Киношка {
    @Autowired СуперГерой герой;
    @Autowired СуперЗлодей злодей;

    public static Киношка снимать() {
        return new Киношка();
    }
}
```

IoC, DI, Spring и друзья

- @Autowired
- @Component/@Service
- @Configuration

```
@Component
public class Киношка {
    @Autowired СуперГерой герой;
    @Autowired СуперЗлодей злодей;

    public static Киношка снимать() {
        return new Киношка();
    }
}
```

IoC, DI, Spring и друзья

- @Autowired
- @Component/@Service
- @Configuration

```
@Component
public class СуперГерой implements Герой {
    @Autowired СуперЗлодей вражина;

    @Override
    public void бить() {
        вражина.бить();
    }
}
```


Demo

Тест №1.5

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = JokerWordsFrequencyResolverTestConfig. class)
public class JokerWordsFrequencyResolverTest {

 @Autowired
 JokerWordsFrequencyResolver jokerWordsFrequencyResolver;

 @Test
 public void name() throws Exception {
   jokerWordsFrequencyResolver.setAnswers("objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Тест №1.5

```
@Configuration
public class JokerWordsFrequencyResolverTestConfig {

 @Bean
 public JokerWordsFrequencyResolver jokerWordsFrequencyResolver(
     WordsComposer wordsComposer) {
   return new JokerWordsFrequencyResolver(wordsComposer);
 }

}
```

Тест №1.5

```
@Configuration
public class JokerWordsFrequencyResolverTestConfig {

 @Bean
 public JokerWordsFrequencyResolver jokerWordsFrequencyResolver(
     WordsComposer wordsComposer) {
   return new JokerWordsFrequencyResolver(wordsComposer);
 }

}
```

Тест №1.5

```
@Configuration
@ComponentScan("com.conference.spring.test.common")
public class JokerWordsFrequencyResolverTestConfig {

 @Bean
 public JokerWordsFrequencyResolver jokerWordsFrequencyResolver(
     WordsComposer wordsComposer) {
   return new JokerWordsFrequencyResolver(wordsComposer);
 }

}
```

Тест №1.5

```
@Configuration
@ComponentScan("com.conference.spring.test.common")
public class JokerWordsFrequencyResolverTestConfig {

 @Bean
 public JokerWordsFrequencyResolver jokerWordsFrequencyResolver(
     WordsComposer wordsComposer) {
   return new JokerWordsFrequencyResolver(wordsComposer);
 }

}
```

Тест №1.5

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = JokerWordsFrequencyResolverTestConfig. class)
public class JokerWordsFrequencyResolverTest {

 @Autowired
 JokerWordsFrequencyResolver  jokerWordsFrequencyResolver;

 @Test
 public void name() throws Exception {
   jokerWordsFrequencyResolver.setAnswers("objects");

   int match = jokerWordsFrequencyResolver.match(
       Question.builder().body("objects ...").build());

   assertThat(match, equalTo(1));
 }
}
```

Passed

Ещё немного теории

SpringRunner

```
/**
 * @author Sam Brannen
 * @since 4.3
 * @see SpringJUnit4ClassRunner
 */
public final class SpringRunner extends SpringJUnit4ClassRunner
```

SpringRunner & SpringJUnit4ClassRunner

```
/**
 * @author Sam Brannen
 * @author Juergen Hoeller
 * ...
 */
public class SpringJUnit4ClassRunner extends BlockJUnit4ClassRunner
```

SpringExtension — Junit5

```
/**
 * {@code SpringExtension} integrates the <em>Spring TestContext … </em>
 * into JUnit 5's <em>Jupiter</em> programming model.
 ...
 * @author Sam Brannen
 * @since 5.0
 */
public class SpringExtension implements BeforeAllCallback, … {
```

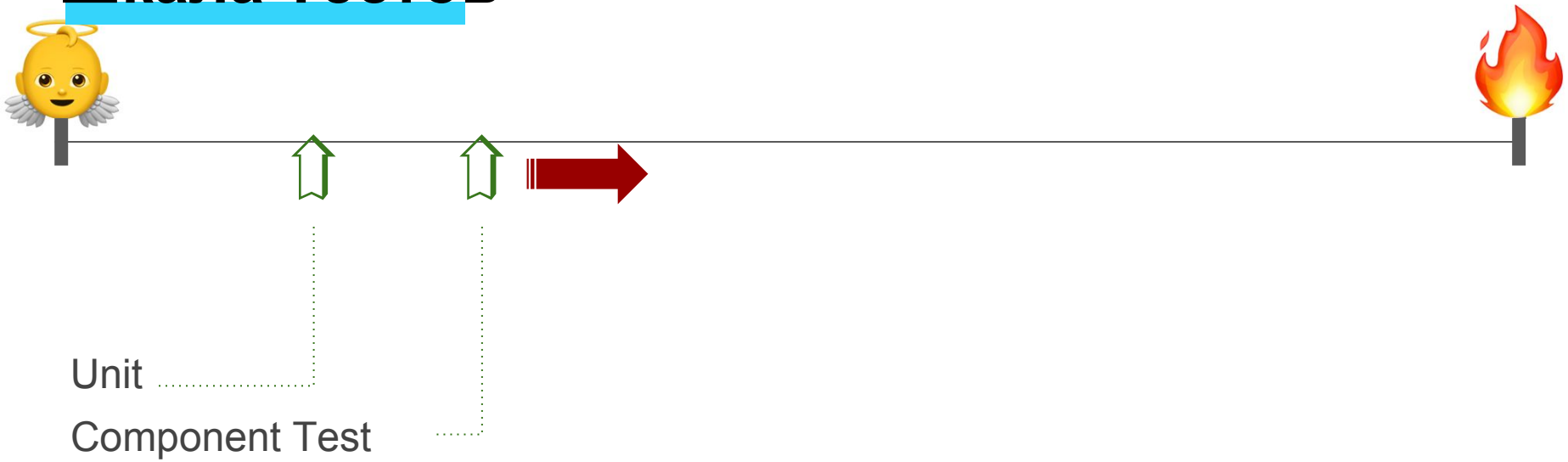
SpringExtension — Junit5

```
@SpringJUnitConfig
@SpringJUnitWebConfig
```

А давайте тестировать. Тест #2

1. Пишем `TextBasedQuestionTypeResolverTest`

Шкала Тестов



А давайте тестировать. Тест #2

1. Пишем `TextBasedQuestionTypeResolverTest`
2. Вручную создаем три бина для тестирования `TextBasedQuestionTypeResolver` на примере Барух vs Джокер кейса


Demo
TextBasedQuestionTypeResolverTest

Тест #2

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = TextBasedQuestionTypeResolverTestConfig.class)
public class TextBasedQuestionTypeResolverTest {
 @Autowired
 TextBasedQuestionTypeResolver questionResolver;

 @Test
 public void name() throws Exception {
  QuestionType groovy = questionResolver.resolveType(new Question("groovy"));
  QuestionType objects = questionResolver.resolveType(new Question("псих"));

  assertThat(groovy, equalTo(JBARUCH));
  assertThat(objects, equalTo(JOKER));
 }
}
```

Тест #2

```
@Configuration
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }


}
```

Тест #2

```
@Configuration
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }

}
```

Тест #2

```
@Configuration
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }
}
```

Для них нужен WordsComposer

```
@ComponentScan("com.conference.spring.test.common")
```
?

Тест #2

```
@Configuration
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }
}
```

Для них нужен WordsComposer

~~@ComponentScan("com.conference.spring.test.common")~~ ?

Тест #2

```
@Configuration
@Import(CommonConfig.class)
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }

}
```

Тест #2

```
@Configuration
@ComponentScan("com.conference.spring.test.common")
public class CommonConfig {

}
```

Not Passed

Что случилось

```
class JokerWordsFrequencyResolver
    @Value("${tokens.joker}")
    private String answers;

class JBaruchWordsFrequencyResolver
    @Value("${tokens.jbaruch}")
    private String answers;
```

Кто считывает?

Что случилось

```
class JokerWordsFrequencyResolver
    @Value("${tokens.joker}")
    private String answers;

class JBaruchWordsFrequencyResolver
    @Value("${tokens.jbaruch}")
    private String answers;
```

Кто считывает?

Отсюда считываем

```
application.yml:
tokens:
 jbaruch: npm leftpad artifactory groovy object ***
 joker: objects
```

Тест #2

```
@Configuration
@Import(CommonConfig.class)
@PropertySource("classpath*:application.yml")
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }

}
```

Тест #2

```
@Configuration
@Import(CommonConfig.class)
@PropertySource("classpath*:application.yml")
public class TextBasedQuestionTypeResolverTestConfig {

 @Bean
 public TextBasedQuestionTypeResolver textBasedQuestionTypeResolver(
     List<WordsFrequencyResolver> c) {
   return new TextBasedQuestionTypeResolver(c);
 }

 @Bean
 public JokerWordsFrequencyResolver … { … }

 @Bean
 public JBaruchWordsFrequencyResolver … { … }

}
```

Not Passed

А давайте тестировать. Тест #2

1. Пишем `TextBasedQuestionTypeResolverTest`
2. Вручную создаем три бина для тестирования `TextBasedQuestionTypeResolver` на примере Барух vs Егор кейса
3. Все падает потому что не подтягивается application.yml
4. `@PropertySource` ...

А давайте тестировать. Тест #2

```
@ContextConfiguration(classes = ....class,
initializers =
YamlFileApplicationContextInitializer.class)
public class OurTest {
   @Test
   public test(){
...
   }
}
```

Spring Boot обновки

1. `@SpringBootTest`
2. `@TestConfiguration`
3. `@MockBean && @SpyBean`
4. `@DataJpaTest`
5. `@MockMvcTest`

Углубляемся в Spring. Тест #2

1. Применяем `@SpringBootTest`


Demo

Тест #2

```
@RunWith(SpringRunner.class)
@ContextConfiguration(classes = TextBasedQuestionTypeResolverTestConfig. class)
public class TextBasedQuestionTypeResolverTest {
 @Autowired
 TextBasedQuestionTypeResolver  questionResolver;

 @Test
 public void name() throws Exception {
  QuestionType groovy =  questionResolver.resolveType(new Question("groovy"));
  QuestionType objects =  questionResolver.resolveType(new Question("objects"));

  assertThat(groovy, equalTo(JBARUCH));
  assertThat(objects, equalTo(JOKER));
 }
}
```

Тест #2

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class TextBasedQuestionTypeResolverTest {
 @Autowired
 TextBasedQuestionTypeResolver  questionResolver;

 @Test
 public void name() throws Exception {
  QuestionType groovy =  questionResolver.resolveType(new Question("groovy"));
  QuestionType objects =  questionResolver.resolveType(new Question("objects"));

   assertThat(groovy, equalTo(JBARUCH));
   assertThat(objects, equalTo(JOKER));
 }
}
```

Not Passed

Тест #2

```
@RunWith(SpringRunner.class)
@SpringBootTest
@ActiveProfiles("joker_vs_jbaruch")
public class TextBasedQuestionTypeResolverTest {
 @Autowired
 TextBasedQuestionTypeResolver questionResolver;

 @Test
 public void name() throws Exception {
  QuestionType groovy = questionResolver.resolveType(new Question("groovy"));
  QuestionType objects = questionResolver.resolveType(new Question("objects"));

  assertThat(groovy, equalTo(JBARUCH));
  assertThat(objects, equalTo(JOKER));
 }
}
```

Для подгрузки

application-joker_vs_jbaruch.yml

Passed

Углубляемся в Spring. Тест #2

1. Применяем `@SpringBootTest`
2. Долго…
3. `@SpringBootTest(classes = ...class)`

Углубляемся в Spring. Тест #2

1. Применяем `@SpringBootTest`
2. Долго…
3. `@SpringBootTest(classes = ...class)`
4. Стало быстрее


Demo - но можно лучше

Тест #2

```
@Configuration
@ComponentScan("com.conference.spring.test.common")
public class CommonConfig {

 @PostConstruct
 public void init() {
   System.out.println("Only once " + CommonConfig.class);
 }

}
```

Запустим тест **№1** и **№2** за раз

Only once

```
...

only once

...

only once
```

Only once

...

```
only once
```

...

```
only once
```

Дважды...

Углубляемся в Spring. Тест #2

1. Применяем `@SpringBootTest`
2. Долго…
3. `@SpringBootTest(classes = ...class)`
4. Стало быстрее
5. С кэшированием конфигураций – еще быстрее

Углубляемся в Spring. Тест #2

```
@ContextHierarchy({
 @ContextConfiguration(classes=WordsCommonConfiguration.class),
 @ContextConfiguration(classes= ...class)
})
```


Demo

```
@SpringBootTest
@ContextHierarchy({
   @ContextConfiguration(classes =
        TextBasedQuestionTypeResolverTestConfig.class),
   @ContextConfiguration(classes = CommonConfig.class)
})
@ActiveProfiles("joker_vs_jbaruch")
@RunWith(SpringRunner.class)
public class TextBasedQuestionTypeResolverTest {
...
```

Запустим тест **№1** и **№2** за раз

Only once

```
...

only once
...

only once
...

only once
...

only once
```

Четыре раза...

```java
@Configuration
@Import(CommonConfig.class)
public class JokerWordsFrequencyResolverTestConfig {


@Configuration
@Import(CommonConfig.class)
public class TextBasedQuestionTypeResolverTestConfig {
```

Убираем

```
@Import(CommonConfig.class)
```

Not Passed

Не найден spring bean `WordsComposer`

Углубляемся в Spring. Тест #2

```
@ContextHierarchy({
 @ContextConfiguration(classes=WordsCommonConfiguration.class),
 @ContextConfiguration(classes= ...class)
})
```

Порядок важен! Т.к другая конфигурация использует бины из `WordsCommonConfiguration`

Меняем порядок в @ContextHierarchy

CommonConfig **теперь первый**

```
@SpringBootTest
@ContextHierarchy({
    @ContextConfiguration(classes = CommonConfig.class),
    @ContextConfiguration(classes =
        TextBasedQuestionTypeResolverTestConfig.class)
})
@ActiveProfiles("joker_vs_jbaruch")
@RunWith(SpringRunner.class)
public class TextBasedQuestionTypeResolverTest {
...
```

Passed

Only once

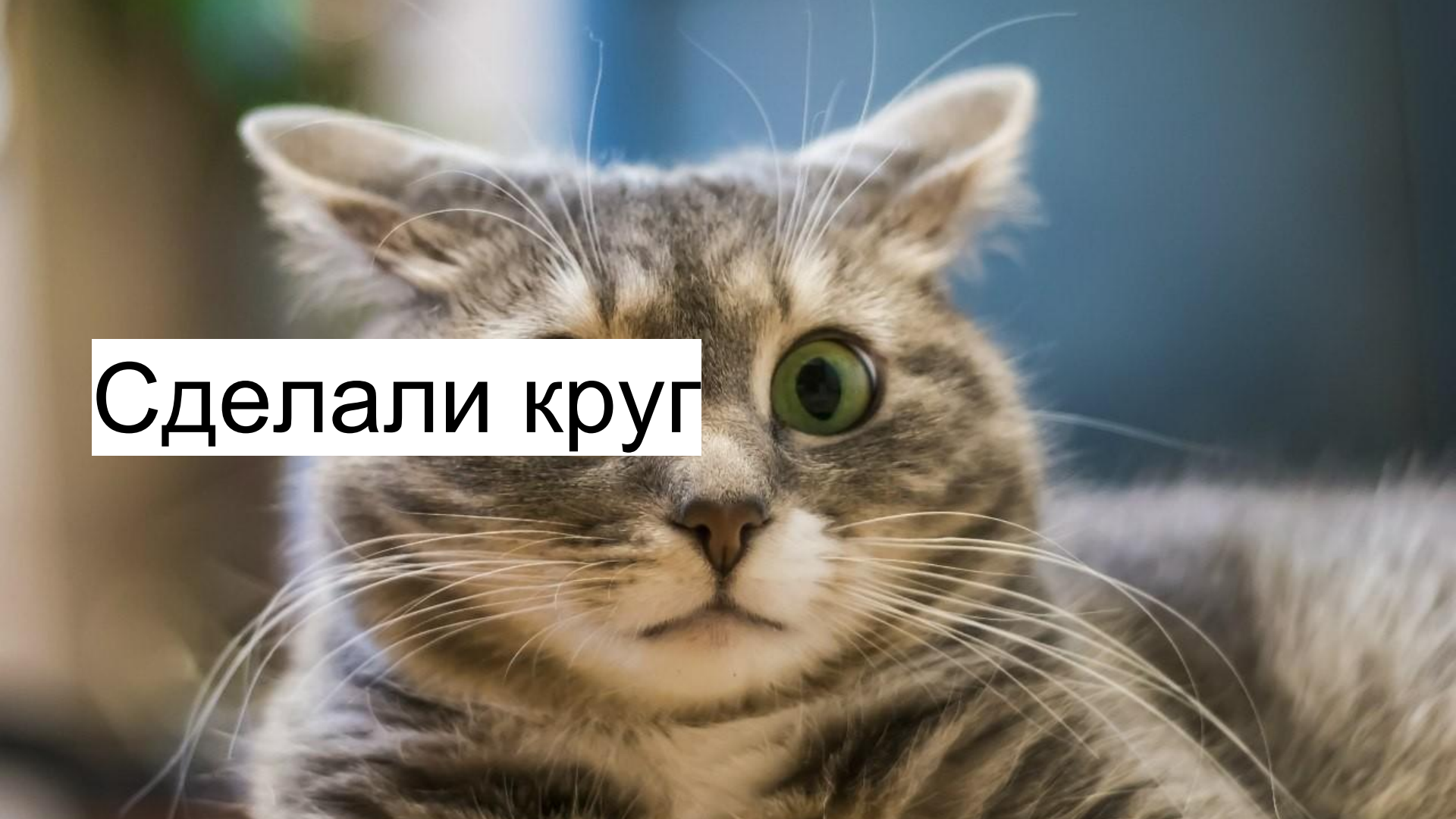
...

```
only once
```

...

```
only once
```

Дважды...


Сделали круг

Опять не закешировалось. Тест #2

Правила кэширования контекстов. Тест #2

`@SpringBootTest` – должен быть **везде**

`@Import` – должен быть **нигде**

`@ActiveProfiles` – **один** на всех

`SpringBootTest.properties` – должны быть **одинаковые**

Правила кэширования контекстов. Тест #2

`@SpringBootTest` – должен быть **везде**

`@Import` – должен быть **нигде**

`@ActiveProfiles` – **один** на всех

`SpringBootTest.properties` – должны быть **одинаковые**

Правила кэширования контекстов. Тест #2

`@SpringBootTest` – должен быть **везде**

`@Import` – должен быть **нигде**

`@ActiveProfiles` – **один** на всех

`SpringBootTest.properties` – должны быть **одинаковые**

Порядок важен!

Любая перестановка – cache miss

Правила кэширования контекстов. Тест #2

```
@SpringBootTest(properties={"a=b","b=a"})
@SpringBootTest(properties={"b=a","a=b"})
```

Правила кэширования контекстов. Тест #2

```
@SpringBootTest(properties={"a=b","b=a"})
@SpringBootTest(properties={"b=a","a=b"})
```

Кэш не сработает

Правила кэширования контекстов. Тест #2

`@SpringBootTest` – должен быть **везде**

`@Import` – должен быть **нигде**

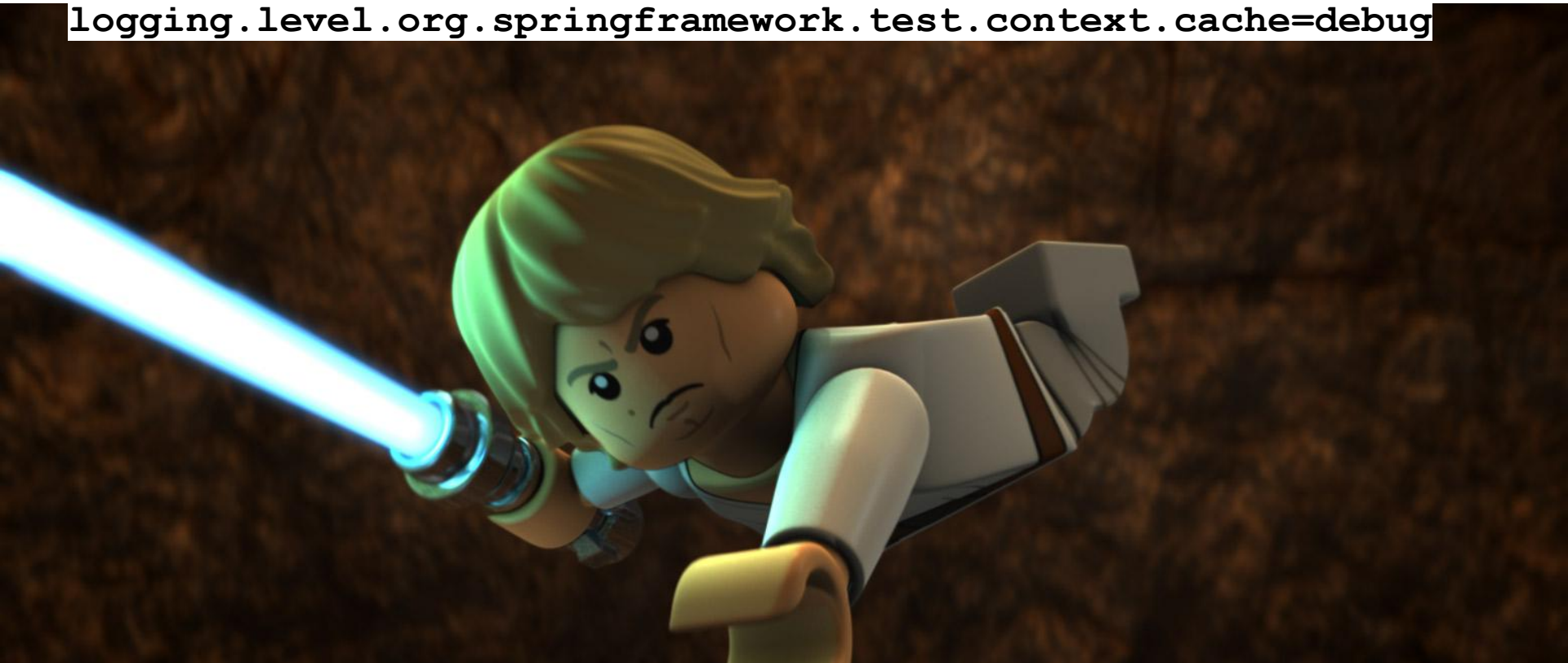
`@ActiveProfiles` – **один** на всех

`SpringBootTest.properties` – должны быть **одинаковые**

Все может привести к потере кэша

Пользуемся силой

```
logging.level.org.springframework.test.context.cache=debug
```



Б – безопасность

```
@SpringBootTest
@ActiveProfiles("joker_vs_jbaruch")
public abstract class ResolversAbstractCommonConfiguration {

}
```

Only once

...

`only once`

Один!...

А если наоборот? (как не кэшировать)

А если наоборот? (как не кэшировать)

```
@DirtiesContext(...)
```

А если наоборот? (как не кэшировать)

```
@DirtiesContext(...)

methodMode()     default MethodMode.AFTER_METHOD
classMode()      default ClassMode.AFTER_CLASS
...
```

Проверим наши знания. Тест #3

1. протестируем `AnswerCacheServiceJPABackend`

Demo

AnswerCacheServiceJPABackend

Что тестируем

```
@Service
@RequiredArgsConstructor
public class AnswerCacheServiceJPABackend implements AnswerCacheService {
 private final QuestionRepository questionRepository;
 private final AnswersRepository answersRepository;

 @Override
 public Answer find(Question question) { … }

 …

}
```

Что тестируем

```
@Service
@RequiredArgsConstructor
public class AnswerCacheServiceJPABackend implements AnswerCacheService {
 private final QuestionRepository questionRepository;
 private final AnswersRepository answersRepository;

 @Override
 public Answer find(Question question) { … }

 …

}
```

Что тестируем

```
@Service
@RequiredArgsConstructor
public class AnswerCacheServiceJPABackend implements AnswerCacheService {
 private final QuestionRepository questionRepository;
 private final AnswersRepository answersRepository;

 @Override
 public Answer find(Question question) { … }

 …

}
```

Spring Boot обновки

1. `@SpringBootTest`
2. **`@MockBean && @SpyBean`**
3. `@TestConfiguration`
4. `@DataJpaTest`
5. `@MockMvcTest`

Как тестируем

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = AnswerCacheServiceJPABackendTestConfig.class)
public class AnswerCacheServiceJPABackendTest {
 @Autowired
 AnswerCacheService answerCacheService;

 @MockBean
 AnswersRepository answersRepository;
 @MockBean
 QuestionRepository questionRepository;

 @Test
 public void should_not_fail() throws Exception {
   … test …
 }
}
```

Как тестируем

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = AnswerCacheServiceJPABackendTestConfig.class)
public class AnswerCacheServiceJPABackendTest {
 @Autowired
 AnswerCacheService answerCacheService;

 @MockBean
 AnswersRepository answersRepository;
 @MockBean
 QuestionRepository questionRepository;

 @Test
 public void should_not_fail() throws Exception {
   … test …
 }
}
```

Как тестируем

```
@RunWith(SpringRunner.class)
@SpringBootTest(classes = AnswerCacheServiceJPABackendTestConfig.class)
public class AnswerCacheServiceJPABackendTest {
 @Autowired
 AnswerCacheService answerCacheService;

 @MockBean
 AnswersRepository answersRepository;
 @MockBean
 QuestionRepository questionRepository;

 @Test
 public void should_not_fail() throws Exception {
   … test …
 }
}
```

Как тестируем – Конфигурация

```
@Configuration
public class AnswerCacheServiceJPABackendTestConfig {
 @Bean
 public AnswerCacheServiceJPABackend answerCacheServiceJpaBackend(
            QuestionRepository qR,
            AnswersRepository aR) {
   return new AnswerCacheServiceJPABackend(qR, aR);
 }

}
```

Как тестируем – сам тест

```
 @Test
 public void should_not_fail() throws Exception {
   Mockito.doThrow(new RuntimeException("Database is down"))
        .when(questionRepository)
        .findFirstByText(Matchers.anyString());

   Answer answer = answerCacheService.find(Question.builder().build());

   assertNull(answer);

 }
}
```

Наш @MockBean

Passed

Синергия с Mockito

1. `@MockBean/@SpyBean`
2. `@PostConstruct` для настройки
3. `@Bean` для настройки конкретных моков

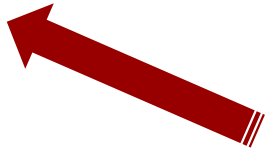
Все ли хорошо?

1. Запустим все тесты

Not Passed

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает



Стандартный тест на запуск контекст
см start.spring.io

Почему упал

Два бина одного типа в контексте

- `answerCacheServiceJPABackend`
- `answerCacheServiceJpaBackend`

Почему упал

Два бина одного типа в контексте

- `answerCacheService`**`JPA`**`Backend`
- `answerCacheService`**`Jpa`**`Backend`

Как Spring называет бины?

почему имена бинов разные

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает
3. Загрузил бины из другого теста!

Spring Boot обновки

1. `@SpringBootTest`
2. `@MockBean && @SpyBean`
3. **`@TestConfiguration`**
4. `@DataJpaTest`
5. `@MockMvcTest`

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает
3. Загрузил бины из другого теста!
4. `@TestConfiguration`!

@TestConfiguration

1. Не сканируется `@SpringBootTest`
2. Не сканируется другими конфигурациями и тестами
3. Не прерывает процесс сканирования `@SpringBootTest`

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает
3. Загрузил бины из другого теста!
4. `@TestConfiguration`!
5. `DeveloperAssistantApplicationTests.contextLoad` работает

Почему упал

Два бина одного типа в контексте

- `answerCacheServiceJPABackend`
- `answerCacheServiceJpaBackend`

Почему упал

Два бина одного типа в контексте

- `answerCacheServiceJPABackend`
- `answerCacheServiceJpaBackend`

Опять двадцать пять!

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает
3. Загрузил бины из другого теста!
4. `@TestConfiguration`!
5. `DeveloperAssistantApplicationTests.contextLoad` работает
6. А **`AnswerCacheServiceJPABackendTest`** перестал
7. Загрузил бины из другого теста!


Spring Заговор

Как @SpringBootTest сканирует пакеты

1.

Два процесса сканирования

1. `@SpringBootTest` сканирование
2. `@SpringBootApplication (@ComponentScan)`

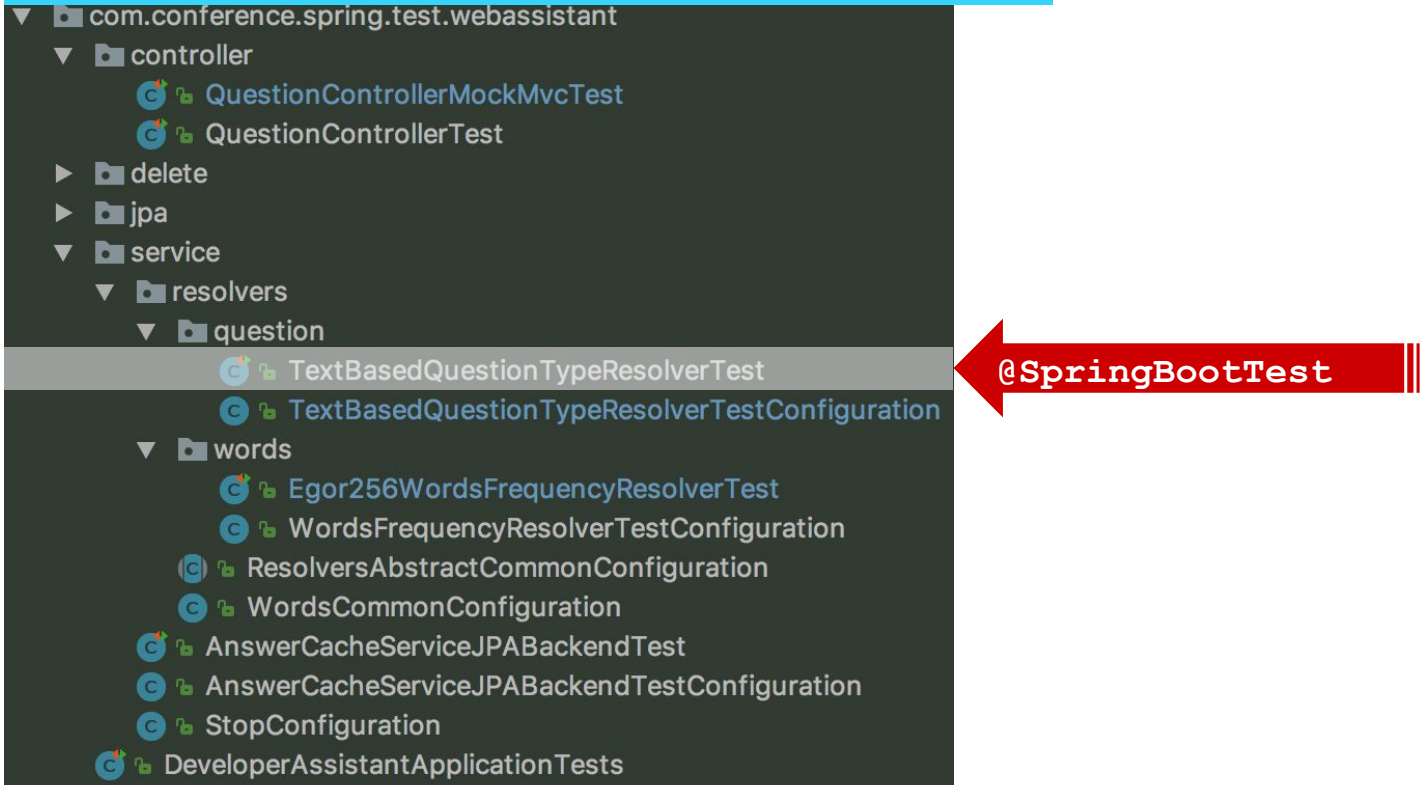
Два процесса сканирования

1. `@SpringBootTest` сканирование
2. `@SpringBootApplication` (`@ComponentScan`)

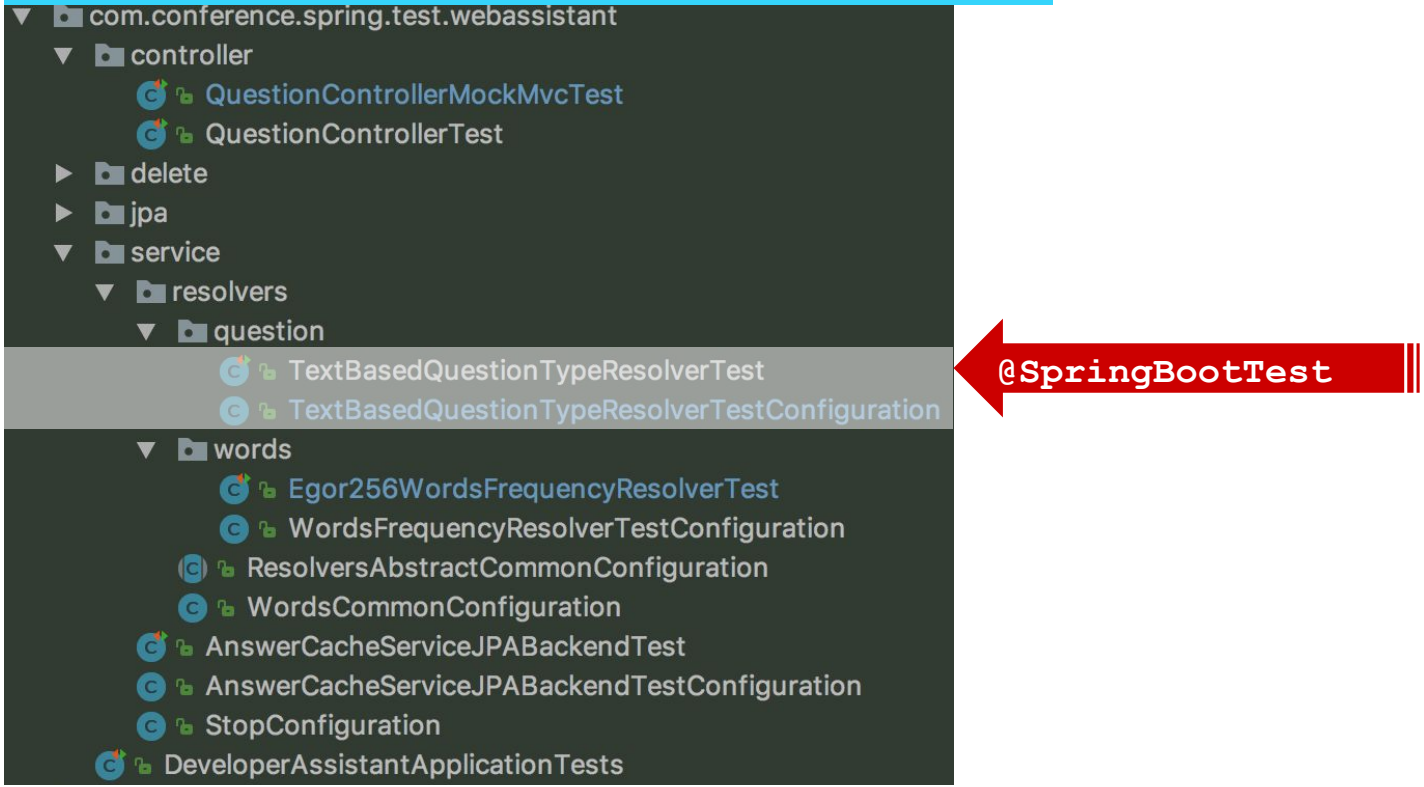
Два процесса сканирования

1. `@SpringBootTest` сканирование
2. `@SpringBootApplication` (`@ComponentScan`)

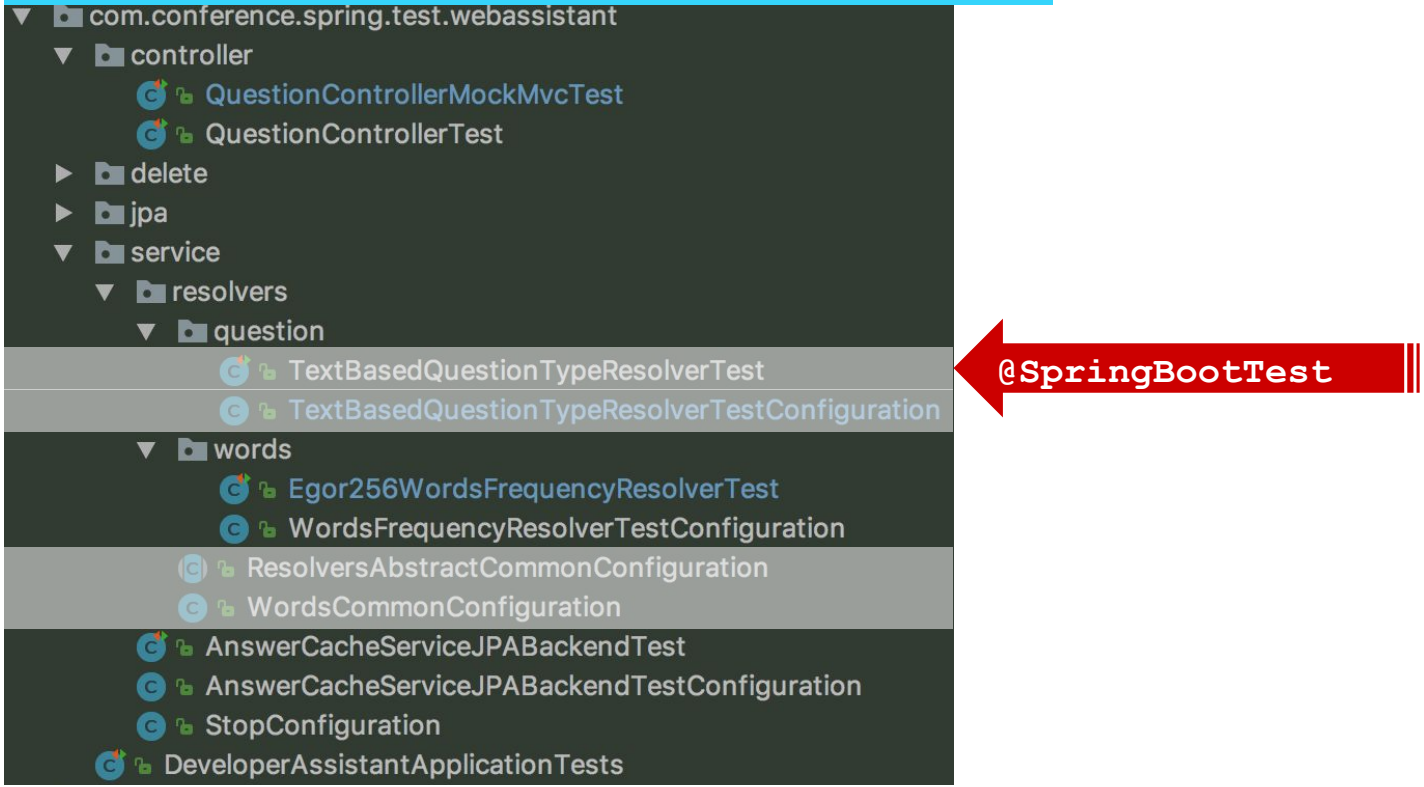
Два процесса сканирования



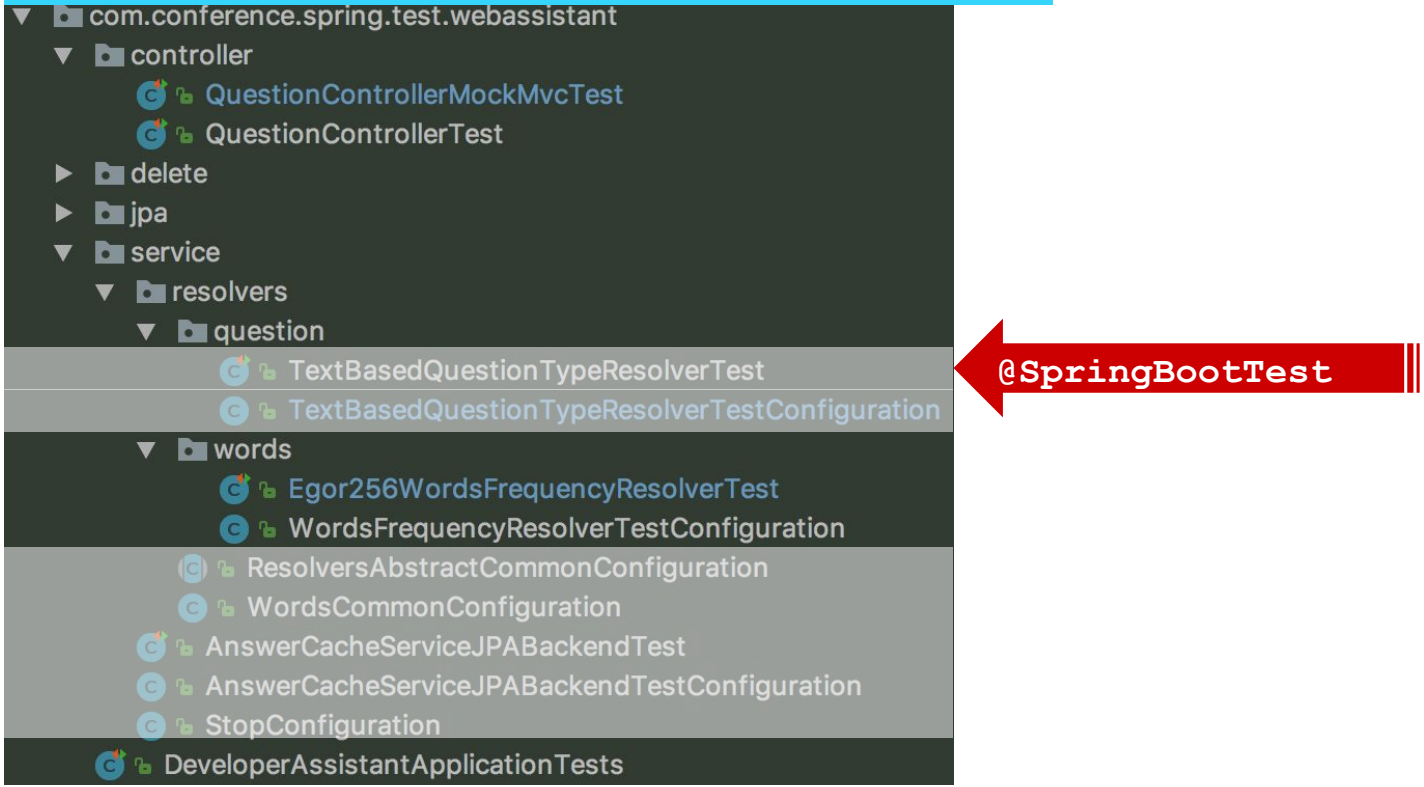
Два процесса сканирования



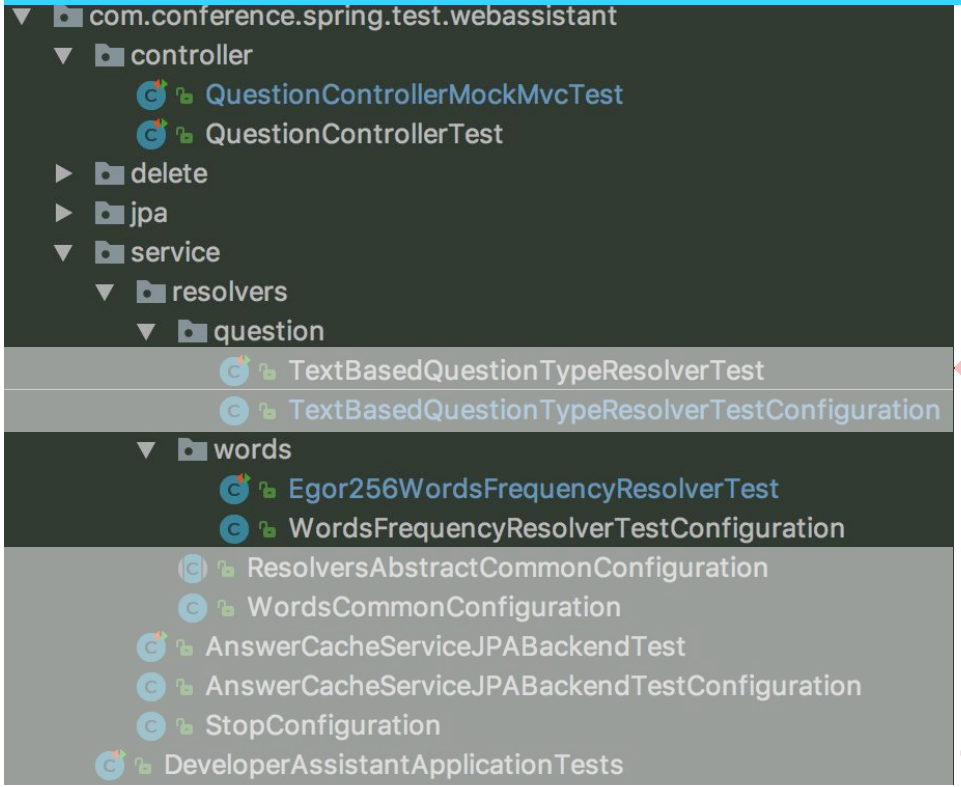
Два процесса сканирования



Два процесса сканирования



Два процесса сканирования

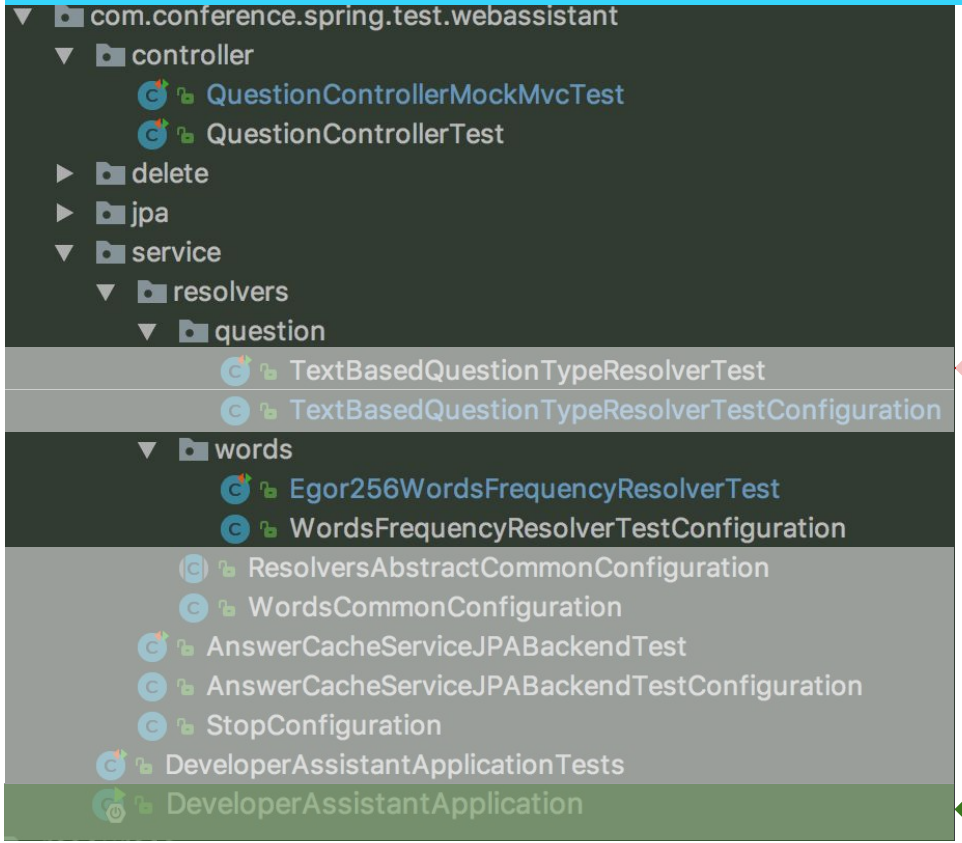


`@SpringBootTest`

test classpath extends **main classpath**


Я знаю ещё один путь...
Сусанин, ты уверен?

Два процесса сканирования



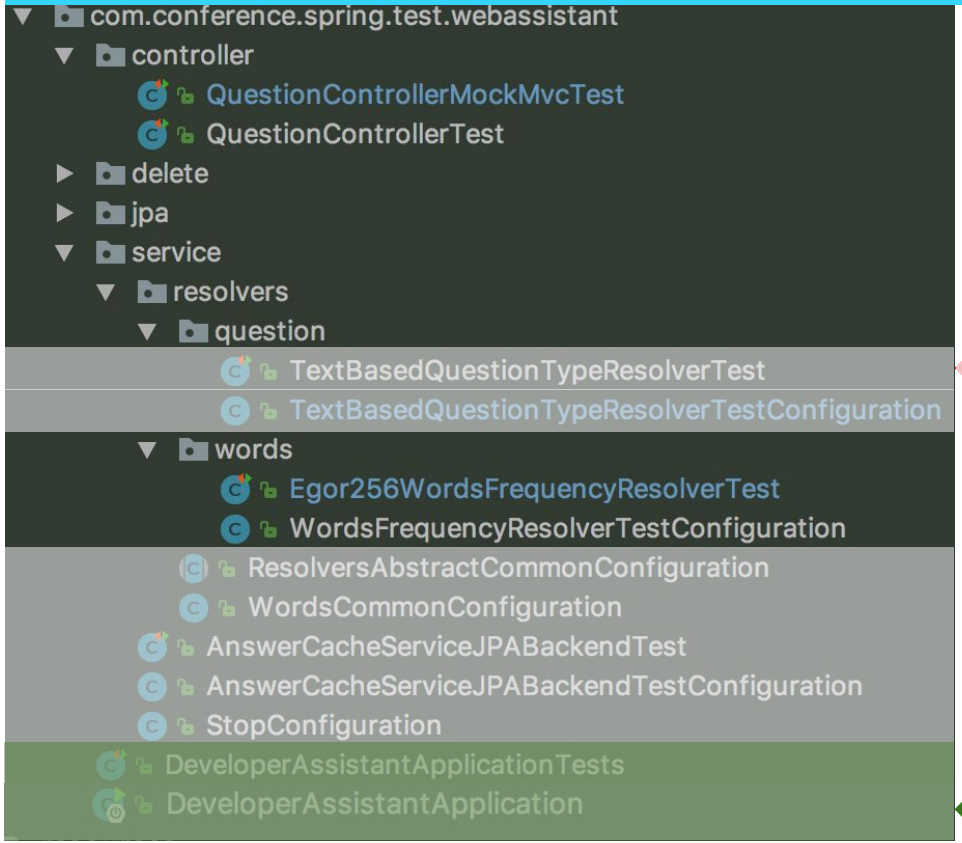
`@SpringBootTest`

test classpath extends **main classpath**

`src/main` будет так же просканирован*

`@SpringBootApplication`

Два процесса сканирования



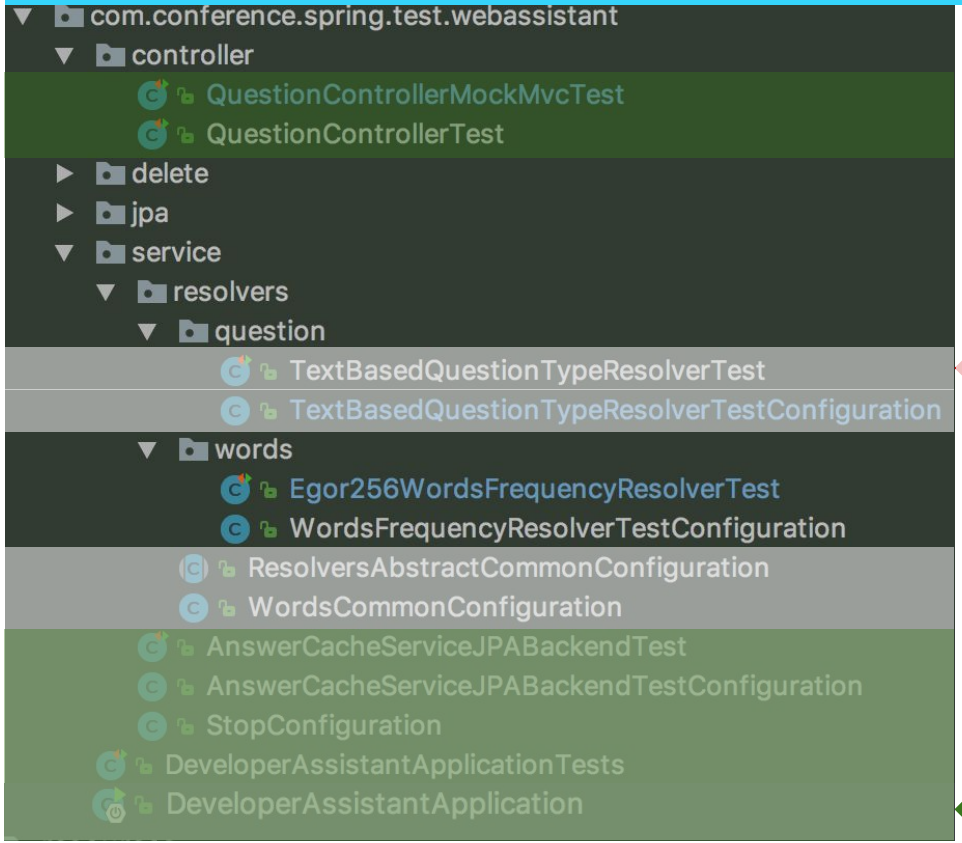
`@SpringBootTest`

test classpath extends **main classpath**

`src/main` будет так же просканирован*

`@SpringBootApplication`

Два процесса сканирования



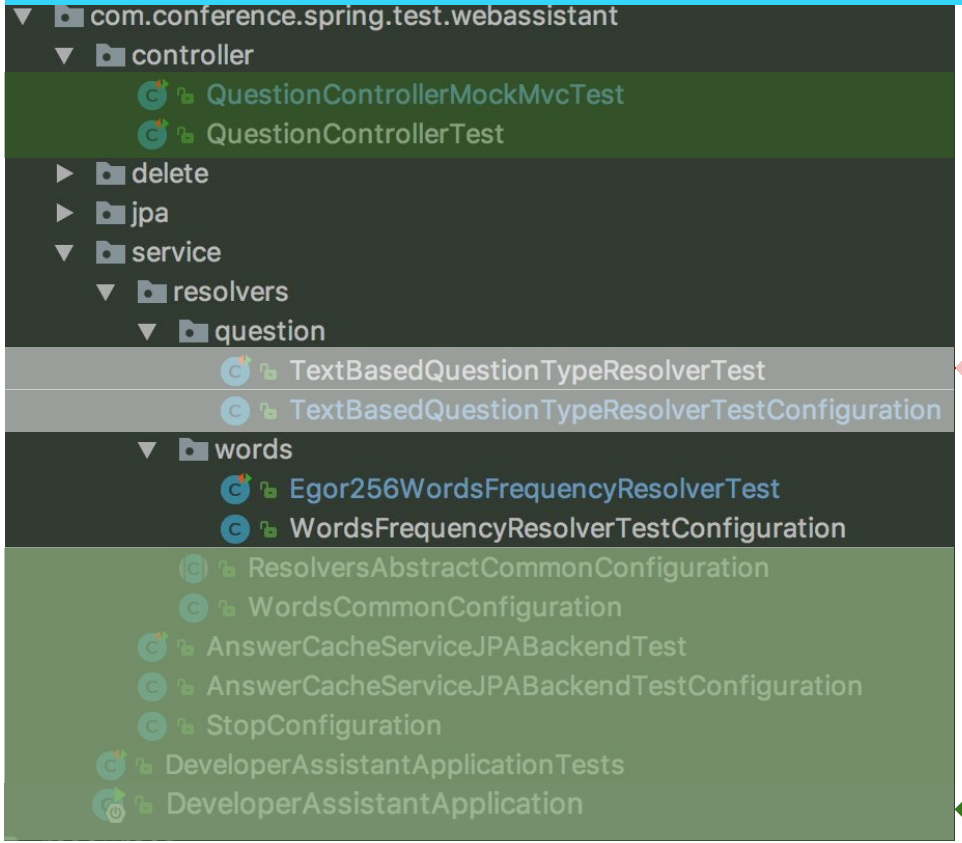
`@SpringBootTest`

test classpath extends **main classpath**

`src/main` будет так же просканирован*

`@SpringBootApplication`

Два процесса сканирования



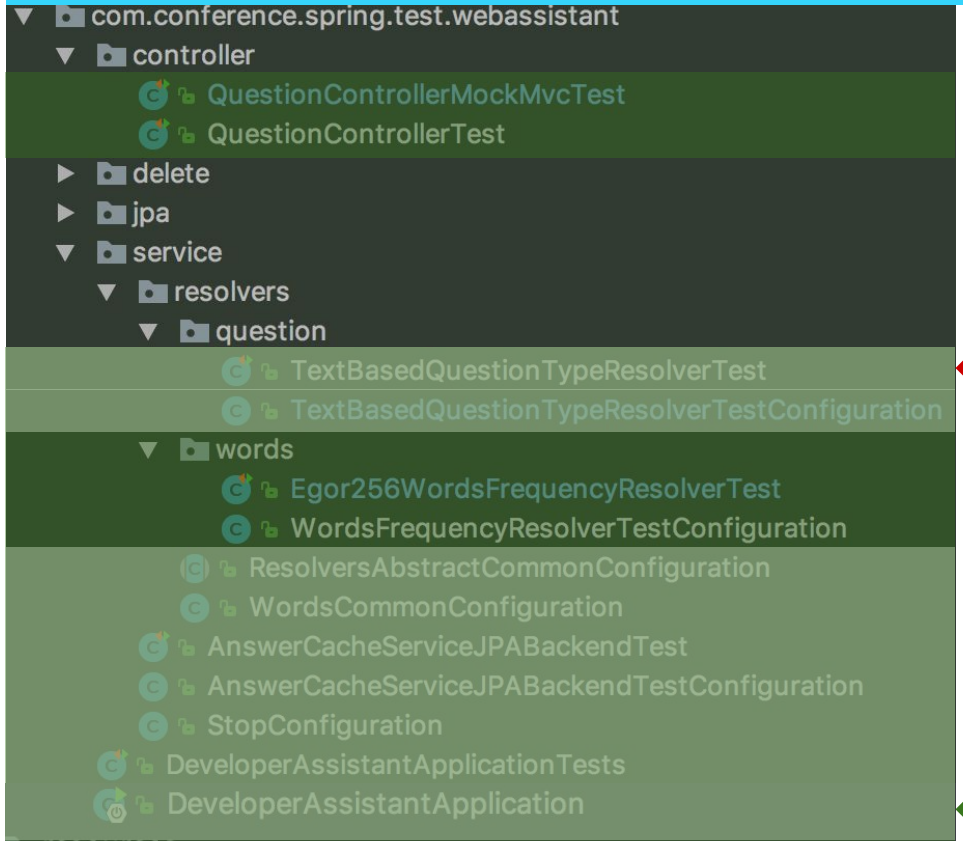
`@SpringBootTest`

test classpath extends **main classpath**

`src/main` будет так же просканирован*

`@SpringBootApplication`

Два процесса сканирования



`@SpringBootTest`

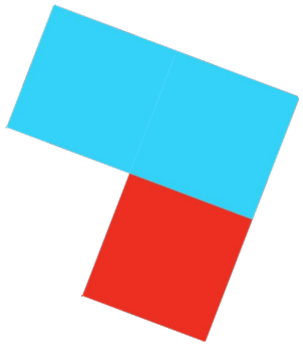
test classpath extends **main classpath**

`src/main` будет так же просканирован*

`@SpringBootApplication`

Тоже и с

`src/main/**`



Как чинить

```
@SpringBootApplication
@EnableFeignClients
@EnableConfigurationProperties(AssistantProperties.class)
public class DeveloperAssistantApplication {

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

}
```

Как чинить

```
@SpringBootApplication
@EnableFeignClients
@EnableConfigurationProperties(AssistantProperties.class)
public class DeveloperAssistantApplication {

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

}
```

Как чинить

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
 @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
 @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

Как чинить

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
 @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
 @Filter(type = FilterType.CUSTOM, classes =
AutoConfigurationExcludeFilter.class) })
public @interface SpringBootApplication {
```

Как чинить

```
/**
 * @author Phillip Webb
 * @since 1.4.0
 */
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Configuration
public @interface SpringBootConfiguration {

}
```

Все ли хорошо?

1. Запустим все тесты
2. `DeveloperAssistantApplicationTests.contextLoad` падает
3. Загрузил бины из другого теста!
4. `@TestConfiguration`!
5. `DeveloperAssistantApplicationTests.contextLoad` работает
6. А `AnswerCacheServiceJPABackendTest` перестал
7. Загрузил бины из другого теста!
8. `@SpringBootConfiguration` остановит сканирование

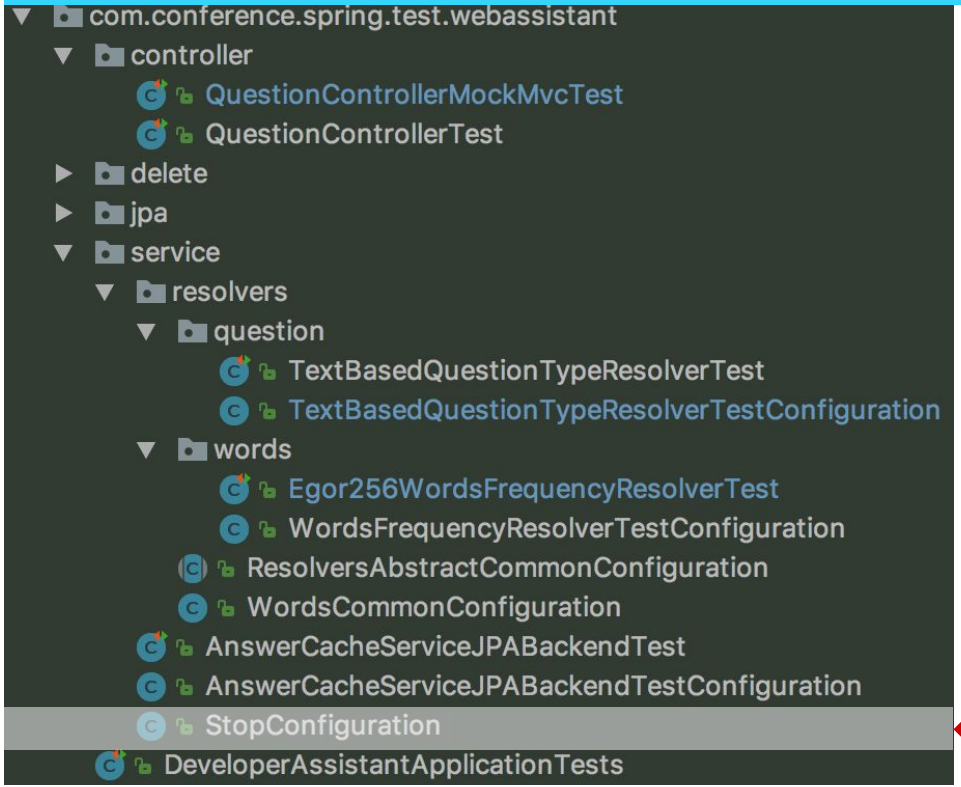
Чиним

```
@SpringBootConfiguration
public class StopConfiguration {

}
```

В нужном пакете!

Нужный пакет для остановки




```
@SpringBootConfiguration
```

Component Tests

Spring Boot обновки

1. `@SpringBootTest`
2. `@TestConfiguration`
3. `@MockBean && @SpyBean`
4. **`@DataJpaTest`**
5. `@MockMvcTest`

@DataJpaTest

1. сканирует все репозитории

@DataJpaTest

1. сканирует все репозитории
2. конфигурирует **`EntityManager`**
3. загружает другие конфигурации

`@DataJpaTest`

1. сканирует все репозитории
2. конфигурирует **`EntityManager`**
3. загружает другие конфигурации
4. фильтрует все не относящееся к Data/JPA

Применим знания

Тестируем `DefaultAssistantJpaBackendTest`

1. `@DataJpaTest` не загружает компоненты Spring*

Тестируем `DefaultAssistantJpaBackendTest`

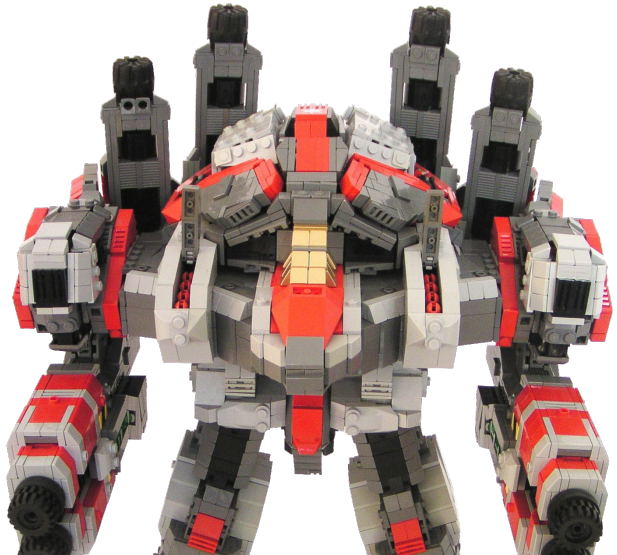
1. `@DataJpaTest` не загружает компоненты Spring*
2. Делаем конфигурацию, загружаем недостающее

Тестируем `DefaultAssistantJpaBackendTest`

1. `@DataJpaTest` не загружает компоненты Spring
2. Делаем конфигурацию, загружаем недостающее
3. Ничего не работает, из за `@SpringBootConfiguration`

Тестируем `DefaultAssistantJpaBackendTest`

1. `@DataJpaTest` не загружает компоненты Spring*
2. Делаем конфигурацию, загружаем недостающее
3. Ничего не работает, из за `@SpringBootConfiguration`
4. Переносим в новый package – все `@*Test` тесты должны быть изолированы



@WebMvcTest

1. Не грузит компоненты спринга

@WebMvcTest

1. Не грузит компоненты спринга
2. Грузит только то что относится к Web

@WebMvcTest

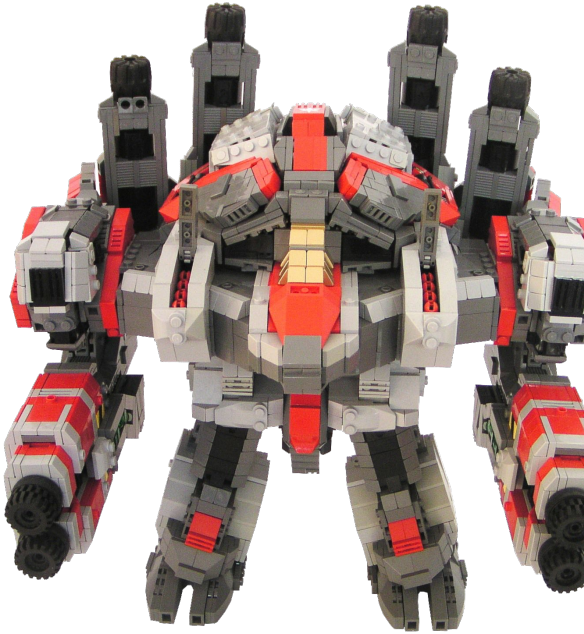
1. Не грузит компоненты спринга
2. Грузит только то что относится к Web
3. Сразу изолируем в отдельный пакет

Получаем суперспособность:

```
@Autowired
MockMvc mockMvc;
```

Где настраивать @MockBean

1. В `@*Configuration` –
   если мок нужен на этапе создания контекста
2. В тесте (`@Before/setup/etc`)
   если мок нужен только на этапе выполнения теста



Что же делает `@SpringBootTest`

1. Без `classes`
   a. сканирует со своего пакета “вверх” в поисках `@SpringBootConfiguration`
      i. игнорирует остальных
   b. падает если не находит или находит несколько в **одном** пакете
2. `classes=~@Configuration`
   a. поднимет только указанные конфигурации
3. `classes=~@TestConfiguration`
   a. поднимет указанный контекст и продолжит сканирование. см пункт 1

Зачем нужен `@SpringBootTest`

1. Полный тест на весь контекст
2. Изменение properties
3. Тесты с определенным скоупом – пакет/конфигурация/автоскан

Зачем нужен `@TestConfiguration`

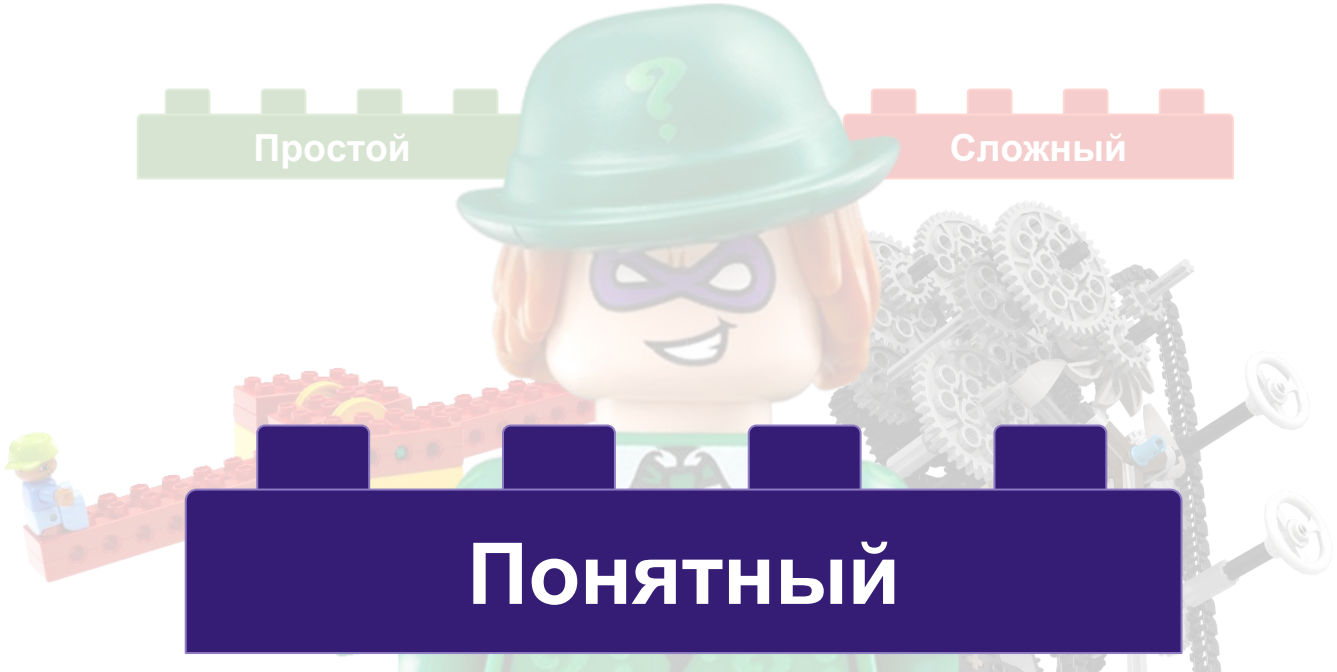
1. Если нужно не прерывать сканирование `@SpringBootTest`
2. Изолированные тесты (игнорируется при сканировании)

Зачем нужен `@SpringBootConfiguration`

1. Прерывать сканирование инициированное `@SpringBootTest`

Есть два типа тестов
Какой сам выберешь,
а какой разработчику оставишь?

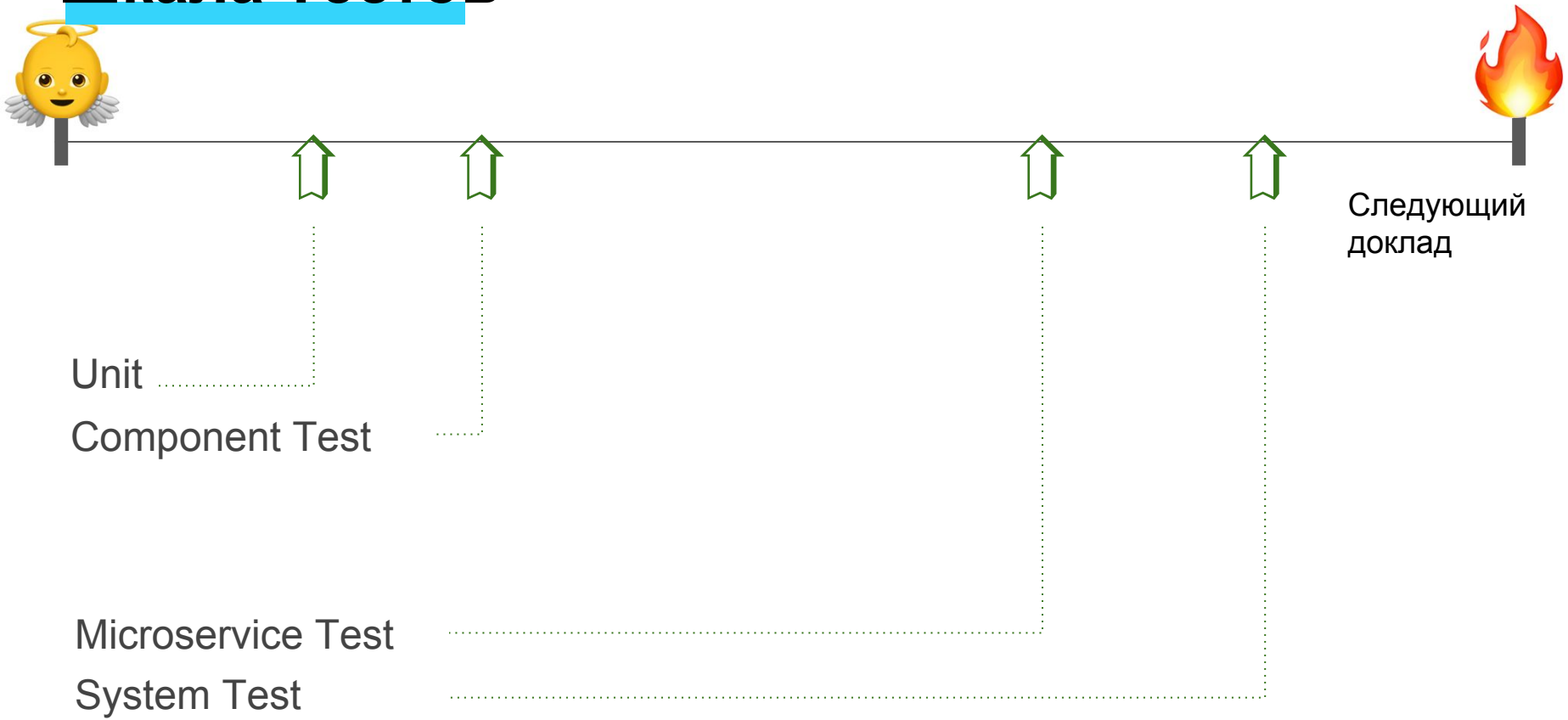
Есть два типа тестов
Какой сам выберешь,
а какой разработчику оставишь?



Выводы

1. Не боимся залезать в кишки приложения
2. Spring Boot богат на инструменты для тестирования
3. Но вносит свои ограничения – структура тестов

Шкала Тестов



	Unit	Component	Microservice
Что нужно	Junit/Mockito	`@ContextConfiguration`	`@SpringBootTest`
Кто управляет	new	`Spring`	`Spring Boot`

QA

Дополнительно

1. `@ComponentScan` > `@TestConfiguration` > `@Configuratin`
   ! `@ComponentScan` находит даже `@TestConfiguration`
2. `@DataJpaTest` > `@SpringBootTest`
3. `@DataJpaTest` и `@WebMvcTest` должны быть в отдельных пакетах

Если есть сомнения – смотри автора! **Juergen Hoeller***

Замечания

1. Spring для Unit тестирования может быть быстрым

Замечания

1. Spring для Unit тестирования может быть быстрым
2. **Кэш контекстов – хрупкая штука**

Замечания

1. Spring для Unit тестирования может быть быстрым
2. Кэш контекстов – хрупкая штука
3. **Для тестов – только `@TestConfiguration`**

Замечания

1. Spring для Unit тестирования может быть быстрым
2. Кэш контекстов – хрупкая штука
3. Для тестов – только `@TestConfiguration`
4. **Изолировать группы тестов с помощью**

Замечания

1. Spring для Unit тестирования может быть быстрым
2. Кэш контекстов – хрупкая штука
3. Для тестов – только `@TestConfiguration`
4. **Изолировать группы тестов с помощью**
   a. выделения в пакеты
   b. `@SpringBootConfiguration`

Замечания

1. Spring для Unit тестирования может быть быстрым
2. Кэш контекстов – хрупкая штука
3. Для тестов – только `@TestConfiguration`
4. Изолировать группы тестов с помощью
   a. выделения в пакеты (особенно для `@*Test`)
   b. `@SpringBootConfiguration`
5. **SpringBootTest надо в основном использовать для микросервис тестов**

Дополнительно

1. Spring для Unit тестирования может быть быстрым
2. Кэш контекстов – хрупкая штука
3. Для тестов – только `@TestConfiguration`
4. Изолировать группы тестов с помощью
   a. выделения в пакеты
   b. `@SpringBootConfiguration`
5. SpringBootTest надо в основном использовать для микросервис тестов
6. **Если есть `DirtiesContext` – стоит задуматься :)**

Ссылки

1. Demo Source with Spring Boot 2.1 and Gradle — https://github.com/lavcraft/spring-boot-curse
2. Old Demo Source with Spring Boot 1.5 and Maven — https://github.com/lavcraft/conference-test-with-spring-boot-test
3. Spring Test Reference Guide
4. Spring Boot Test Reference Guide
5. Spring 1.4 Test Improvements
6. Custom Test Slice with Spring Boot