

# Полезные аннотации в JUnit

JSP & Servlets  
3 уровень, 3 лекция

ОТКРЫТА

## 4.1 @Disabled

Теперь разберем несколько очень полезных и популярных аннотаций фреймворка JUnit. Первая аннотация позволяет выключить определенный тест, чтобы JUnit его не вызывал. Она нужна в случаях, если ты заметишь, что тест работает неверно, или поменяешь код и тест случайно сломается.

Как я писал раньше, 99% тестов никто не поддерживает, поэтому все они рано или поздно оказываются отключены. Поэтому эта аннотация первая в списке полезных.

Рассмотрим ее пример:

```
1 public class AppTest {
2
3     @Disabled("Тест временно отключен. Правда, правда")
4     @Test
5     void testOnDev(){
6         System.setProperty("ENV", "DEV");
7         Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
8     }
9
10
11     @Test
12     void testOnProd(){
13         System.setProperty("ENV", "PROD");
14         Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
15     }
16 }
```

В примере выше метод `testOnDev()` вызываться не будет. Кстати, аннотацию `@Disabled` можно написать сразу перед объявлением класса, тогда все его методы будут проигнорированы.

```
1 @Disabled("Временно отключили тест, починим к маю 2001 года")
2 public class AppTest {
3     @Test
4     void testOnDev(){
5         System.setProperty("ENV", "DEV");
6         Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
7     }
8
9     @Test
10    void testOnProd(){
11        System.setProperty("ENV", "PROD");
12        Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
13    }
14 }
```

## 4.2 @Nested

JUnit позволяет вызывать тестовые методы у вложенных классов. Имеется в виду вложенные тестовые классы. Не факт ,что ты будешь часто сталкиваться с ними, но вероятность такая есть, поэтому нужно понимать, что это такое.

Чтобы вызвать методы вложенного класса перед его объявлением, нужно написать аннотацию `@Nested` . Пример:

```
1 public class AppTest {
2     @Nested
3     public class DevStagingEnvironment {
4         @Test
5         void testOnDev(){
6             System.setProperty("ENV", "DEV");
7             Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
8         }
9     }
10
11     @Nested
12     public class ProductionEnvironment {
13         @Test
14         void testOnProd(){
15             System.setProperty("ENV", "PROD");
16             Assumptions.assumeFalse("DEV".equals(System.getProperty("ENV")));
17         }
18     }
19 }
```

Более подробно можно почитать в официальной [.документации](#)

### 4.3 @ExtendWith

Другая полезная аннотация — `@ExtendWith` . Скорее всего ты будешь встречать ее очень часто, так что рассмотрим ее подробнее.

JUnit — это мощный фреймворк, который позволяет писать различные плагины (расширения) для гибкой настройки своей работы. Некоторые расширения могут собирать статистику о тестах, другие — эмулировать файловую систему в памяти, третьи — эмулировать работу внутри веб-сервера, и так далее.

Если твой код работает внутри какого-нибудь фреймворка (например Spring), **то почти всегда этот фреймворк управляет созданием и настройкой объектов твоего кода.** Поэтому без специального тестового плагина не обойтись. Примеры:

**Пример 1.** Расширение `WebServerExtension` передает в вызываемый тестовый метод `URL` для корректной работы.

```
1 @Test
2 @ExtendWith(WebServerExtension.class)
3 void getProductList(@WebServerUrl String serverUrl) {
4     WebClient webClient = new WebClient();
5     // Use WebClient to connect to web server using serverUrl and verify response
6     assertEquals(200, webClient.get(serverUrl + "/products").getResponseStatus());
7 }
```

Вот так обычно начинаются тесты для тестирования кода, который работает с фреймворком Spring:

```
1 @ExtendWith(SpringExtension.class)
2 @ExtendWith(MockitoExtension.class)
3 class TestServiceTest {
4
5     @MockBean
6     TestService service;
```

```
7
8     @Test
9     void test() {
10         assertNotNull(service); // Test succeeds
11     }
12 }
```

**SpringExtension** создает тестовый вариант фреймворка Spring, а **MockitoExtention** позволяет создавать фейковые объекты. Фейковые объекты — тема очень интересная, мы обязательно ее затронем, но немного позже.

## 4.4 @Timeout

Закончим эту лекцию небольшой и интересной аннотацией **@Timeout**. Она позволяет задать время на выполнение теста. Если выполнение теста заняло больше времени, чем указано в аннотации, то он считается проваленным.

```
1 class TimeoutDemo {
2     @Test
3     @Timeout(value = 100, unit = TimeUnit.MILLISECONDS)
4     void failsIfExecutionTimeExceeds100Milliseconds() {
5         // тест упадет, если займет более 100 миллисекунд
6     }
7 }
```

На этом закончим нашу лекцию.



### Комментарии (8)

популярные новые старые

JavaCoder

Введите текст комментария

Inna Lapina Java Developer 12 ноября, 00:12

"99% тестов никто не поддерживает, поэтому все они рано или поздно оказываются отключены" - это полная неправда, если у вас такой опыт, это еще не значит, что везде так. По-хорошему, на проекте должно быть покрыто тестами минимум 80% кода и при любых изменениях тесты переписываются или добавляются для нового кода.

Ответить 0

Сергей Student в EPAM 17 июня, 16:16

@Timeout не работает. Если верно показывает IDE - то тест выполняется явно дольше чем указанное в аннотации. Почему так?



Ответить 0

Stas S Уровень 75, Гродно, Беларусь 2 сентября, 13:32

попробуй через Maven запустить фазу тестов ;)   
Еще как вариант Thread.sleep , на моем старом ноуте срабатывает даже с чуть меньшим

значением, чем указано в тайм-ауте.

Ответить

0

Musa Muradzade

Уровень 20, Санкт-Петербург, Россия

17 июня, 03:35

...

@Disabled для всего класса не работает ... мб кто сталкивался с подобной проблемой?

Ответить

+1

Сергей

Student в EPAM

17 июня, 16:18

...

У меня работает

Ответить

0

Musa Muradzade

Уровень 20, Санкт-Петербург, Россия

18 июня, 00:28

...

А Вы не могли бы рот файл скинуть?

Ответить

0

Сергей

Student в EPAM

18 июня, 13:38

...

Скинул в приват

Ответить

+1

Pavlo Plynko

Java-разработчик в JavaRush

EXPERT

22 июля, 11:14

...

Если запускать напрямую отключенный тест, то он выполнится. А если запускать много тестовых классов сразу, например все в директории, то тестовый класс должно пропустить.

Ответить

+4

ОБУЧЕНИЕ

- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

СООБЩЕСТВО

- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

КОМПАНИЯ

- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

 Русский

▼

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ



