

Настройка окружения теста в JUnit

JSP & Servlets
3 уровень, 2 лекция

ОТКРЫТА

3.1 Аннотации @BeforeEach, @AfterEach

Обрати внимание, что в предыдущем примере нам в каждом методе приходилось писать код для создания объекта `Calculator` .

Конечно, это всего одна строчка, но если мы будем тестировать реальные системы, то часто будет возникать ситуация, когда нужно создать и сконфигурировать несколько объектов, что может занять несколько десятков строк кода. Пример:

```
1 //Создаем объект HttpClient
2 HttpClient client = HttpClient.newBuilder()
3     .version(Version.HTTP_1_1)
4     .followRedirects(Redirect.NORMAL)
5     .connectTimeout(Duration.ofSeconds(20))
6     .proxy(ProxySelector.of(new InetSocketAddress("proxy.example.com", 80)))
7     .authenticator(Authenticator.getDefault())
8     .build();
9
10 //Создаем объект HttpRequest
11 HttpRequest request = HttpRequest.newBuilder()
12     .uri(new URI("https://javarush.ru"))
13     .headers("Content-Type", " application/octet-stream")
14     .POST( HttpRequest.BodyPublishers.ofInputStream ( () -> is; ))
15     .build();
16
17 //Вызываем метод send()
18 HttpResponse response = client.send(request, BodyHandlers.ofString());
19 System.out.println(response.statusCode());
20 System.out.println(response.body());
```

В примере выше мы создали и сконфигурировали объект `HttpClient` и хотим протестировать работу метода `send()` .

Чтобы каждый раз в тестовом методе не писать создание объекта `HttpClient` , его можно вынести в отдельный метод и поставить ему специальную аннотацию **@BeforeEach**. Тогда JUnit будет вызывать этот метод перед каждым тестовым методом. Пример:

```
1 class HttpClientTest {
2     public HttpClient client;
3
4     @BeforeEach
5     public void init(){
6         client = HttpClient.newBuilder()
7             .version(Version.HTTP_1_1)
8             .followRedirects(Redirect.NORMAL)
9             .connectTimeout(Duration.ofSeconds(20))
10            .proxy(ProxySelector.of(new InetSocketAddress("proxy.example.com", 80)))
11            .authenticator(Authenticator.getDefault())
```

```
12         .build();
13     }
14
15     @Test
16     public void send200() throws Exception {
17         //Создаем объект HttpRequest()
18         HttpRequest request = HttpRequest.newBuilder(new URI("https://javarush.ru")).build();
19
20         //Вызываем метод send()
21         HttpResponse response = client.send(request, BodyHandlers.ofString());
22         assertEquals(200, response.statusCode());
23     }
24
25     @Test
26     public void send404() throws Exception {
27         //Создаем объект HttpRequest()
28         HttpRequest request = HttpRequest.newBuilder(new URI("https://javarush.ru/unknown")).build();
29
30         //Вызываем метод send()
31         HttpResponse response = client.send(request, BodyHandlers.ofString());
32         assertEquals(404, response.statusCode());
33     }
34 }
```

Также можно создать специальный метод, который будет вызываться каждый раз после очередного тестового метода, и **подчищать использованные ресурсы**, писать что-то в лог и т. п. Такой метод нужно пометить аннотацией **@AfterEach**.

Если у тебя есть 3 тестовых метода `test1()`, `test2()` и `test3()`, то порядок вызова будет таким:

- BeforeEach-метод
- test1()
- AfterEach-метод
- BeforeEach-метод
- test2()
- AfterEach-метод
- BeforeEach-метод
- test3()
- AfterEach-метод

3.2 Аннотации @BeforeAll, @AfterAll

JUnit также позволяет добавить метод, который будет вызван **один раз перед всеми тестовыми методами**. Такой метод нужно пометить аннотацией **@BeforeAll**. Для нее так же существует парная аннотация **@AfterAll**. Метод, помеченный ею, JUnit вызовет после всех тестовых методов.

Давай напишем специальный пример, который позволяет лучше понять, как это все работает. За основу возьмем тестирование нашего калькулятора:

```
1  class CalculatorTest {
2      private Calculator calc = new Calculator();
3
4      @BeforeAll
5      public static void init(){
6          System.out.println("BeforeAll init() method called");
7      }
8
9      @BeforeEach
```

```
10      public void initEach(){
11          System.out.println("BeforeEach initEach() method called");
12      }
13
14      @Test
15      public void add(){
16          System.out.println("Testing Addition");
17      }
18
19      @Test
20      public void sub() {
21          System.out.println("Testing Subtraction");
22      }
23
24      @Test
25      public void mul(){
26          System.out.println("Testing Multiplication");
27      }
28
29      @Test
30      public void div() {
31          System.out.println("Testing Division");
32      }
33  }
```

Этот тест напечатает в консоль такой текст:

```
BeforeAll init() method called
BeforeEach initEach() method called
Testing Addition

BeforeEach initEach() method called
Testing Subtraction

BeforeEach initEach() method called
Testing Multiplication

BeforeEach initEach() method called
Testing Division
```

[← Предыдущая лекция](#)

[Следующая лекция →](#)

 +23 

Комментарии (2)

популярные

новые

старые

JavaCoder

Введите текст комментария

eldar516

Работает в **Аэропорт Шереметьево**

20 июня, 01:04

⋮

на 4 и 5 junit методы аннотированные BeforeAll и AfterAll выбрасывают JUnitException, если не аннотировать класс @TestInstance(TestInstance.Lifecycle.PER_CLASS)

Ответить

Ramz

Уровень 35, Нижний Новгород, Russian Federation

8 июля, 18:37

⋮

Либо static

Ответить

−

+2

+

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



