

Новый HttpClient

JSP & Servlets
10 уровень, 0 лекция

ОТКРЫТА

1.1 Знакомство с HttpClient

Начиная с JDK 11 разработчики платформы Java добавили в JDK новый мощный инструмент для выполнения http-запросов — пакет `java.net.http`. Он содержит четыре ключевых класса:

- `HttpClient`
- `HttpRequest`
- `HttpResponse`
- `WebSocket`

Это очень мощные классы, которые позволяют выполнять все возможные виды запросов по протоколам `HTTP`, `HTTP/2` и `WebSocket`.

Кроме того, с помощью этих классов можно выполнять как синхронные, так и асинхронные http-запросы.

Выполнение http-запроса состоит из трех частей:

1. Создание объекта `HttpClient`
2. Создание объекта `HttpRequest`
3. Отправка запроса с помощью метода `send()` или `sendAsync()`
4. Обработка ответа `HttpResponse`

Пример такого запроса:

```
1  HttpClient client = HttpClient.newBuilder()
2      .version(Version.HTTP_1_1)
3      .followRedirects(Redirect.NORMAL)
4      .connectTimeout(Duration.ofSeconds(20))
5      .proxy(ProxySelector.of(new InetSocketAddress("proxy.example.com", 80)))
6      .authenticator(Authenticator.getDefault())
7      .build();
8
9  HttpResponse<String> response = client.send(request, BodyHandlers.ofString());
10 System.out.println(response.statusCode());
11 System.out.println(response.body());
```

1.2 Декларативный подход

В примере выше ты наблюдаешь пример так называемого декларативного подхода к написанию кода. Давай разберем первую часть примера:

```
1  HttpClient client = HttpClient.newBuilder()
2      .version(Version.HTTP_1_1)
3      .followRedirects(Redirect.NORMAL)
4      .connectTimeout(Duration.ofSeconds(20))
5      .proxy(ProxySelector.of(new InetSocketAddress("proxy.example.com", 80)))
6
```

7	<code>.authenticator(Authenticator.getDefault())</code> <code>.build();</code>
---	---

Как бы выглядел этот код, написанный в классическом стиле:

1	<code>HttpClient client = HttpClient.new();</code>
2	<code>client.setVersion(Version.HTTP_1_1);</code>
3	<code>client.setFollowRedirects(Redirect.NORMAL);</code>
4	<code>client.setConnectTimeout(Duration.ofSeconds(20));</code>
5	<code>client.setProxy(ProxySelector.of(new InetSocketAddress("proxy.example.com", 80)));</code>
6	<code>client.setAuthenticator(Authenticator.getDefault());</code>

При использовании декларативного подхода в коде меняются две вещи. **Во-первых**, все методы класса `HttpClient` **возвращают свой же объект**, что позволяет организовать код в виде цепочек.

Классический код:	
1	<code>HttpClient client = HttpClient.new();</code>
2	<code>client.setVersion(Version.HTTP_1_1);</code>
3	<code>client.setFollowRedirects(Redirect.NORMAL);</code>
4	<code>client.setConnectTimeout(Duration.ofSeconds(20));</code>
5	<code>client.setAuthenticator(Authenticator.getDefault());</code>
В виде цепочки:	
1	<code>HttpClient client = HttpClient.new() .setVersion(Version.HTTP_1_1) .setFollowRedirects(Redirect.NOR</code>
Переносим каждый метод на отдельную строку (это один длинный statement)	
1	<code>HttpClient client = HttpClient.new()</code>
2	<code>.setVersion(Version.HTTP_1_1)</code>
3	<code>.setFollowRedirects(Redirect.NORMAL)</code>
4	<code>.setConnectTimeout(Duration.ofSeconds(20))</code>
5	<code>.setAuthenticator(Authenticator.getDefault());</code>

Во-вторых, у методов убирают префикс `set`, что позволяет писать код еще компактнее:

Было	
1	<code>HttpClient client = HttpClient.new()</code>
2	<code>.setVersion(Version.HTTP_1_1)</code>
3	<code>.setFollowRedirects(Redirect.NORMAL)</code>
4	<code>.setConnectTimeout(Duration.ofSeconds(20))</code>
5	<code>.setAuthenticator(Authenticator.getDefault());</code>
Стало	
1	<code>HttpClient client = HttpClient.new()</code>
2	<code>.version(Version.HTTP_1_1)</code>
3	<code>.followRedirects(Redirect.NORMAL)</code>
4	
5	

```
.connectTimeout(Duration.ofSeconds(20))  
.authenticator(Authenticator.getDefault());
```

Такой код проще читать, хотя сложнее писать.

И еще один важный момент. В этом примере использовался шаблон (pattern) Builder. Бывают сценарии, когда создание объекта — это сложный процесс. Поэтому его предпочитают формализовать: он начинается с вызова условного метода `begin()` и заканчивается вызовом условного метода `end()`.

В примере, который мы разбирали, метод `HttpClient.newBuilder()` возвращает объект `HttpClient.Builder` (это внутренний служебный класс у класса `HttpClient`). Все методы типа `version()` вызываются как раз у этого служебного объекта. Ну а вызов метода `build()` обозначает окончание построения объекта и возвращает объект `HttpClient`.

[← Предыдущая лекция](#)

[Следующая лекция →](#)

+9

Комментарии (3) + 1

популярные новые старые

JavaCoder

Введите текст комментария

kv0ut Уровень 51

27 декабря 2022, 11:10

То ли я не понял, то ли опечатка:
"Выполнение http-запроса состоит из ТРЕХ частей:"
после чего перечислены ЧЕТЫРЕ пункта

Ответить

0

Oleg Уровень 41

7 ноября 2022, 10:07

Здесь скорее паттерн билдер объясняется, чем http клиент)

Ответить

0

YakovlevPA Уровень 81

13 июля 2022, 11:31

В последнем стало вторая строка видимо лишняя)

Ответить

+5

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

