Выполнение запроса с помощью HttpClient

JSP & Servlets
10 уровень, 3 лекция

ОТКРЫТА

4.1 Метод send(), BodyHandlers

Ты закончил изучать, как формировать **http-запрос**, значит можно переходить к самому главному – отправке этого запроса. В самом простом случае это сделать легко:

```
HttpRequest request = HttpRequest.newBuilder(new URI("https://javarush.com")).build();

   HttpClient client = HttpClient.newBuilder()
        .version(Version.HTTP_1_1)
        .build();

    HttpResponse<String> response = client.send(request, BodyHandlers.ofString());
    System.out.println(response.statusCode() );
    System.out.println(response.body() );
```

А что это за `BodyHandlers` такие? А как ты думаешь? Ты отправил запрос, а значит вам должен прийти ответ – `http response`. И у этого ответа может быть `response body`: строка, файл, массив байт, InputStream.

Да, да, все верно. Так же, как и при формировании запроса, тебе нужно указать тип `response body` у ответа. Всего их может быть 8 штук:

- `BodyHandlers.ofByteArray`
- `BodyHandlers.ofString`
- `BodyHandlers.ofFile`
- `BodyHandlers.discarding`
- `BodyHandlers.replacing`
- `BodyHandlers.ofLines`
- `BodyHandlers.fromLineSubscriber`

В зависимости от того, какой тип `BodyHandlers` ты передал в метод `send()`, такой тип результата он и вернет. Пример:

```
// response body игнорируется
HttpResponse<Void> response = client.send(request, BodyHandlers.discarding());
```

```
// response body это строка
 HttpResponse<String>response = client.send(request, BodyHandlers.ofString());
```

```
// response body это файл
HttpResponse<Path> response = client.send(request, BodyHandlers.ofFile(Paths.get("readme.txt")));
```

```
// response body это InputStream
HttpResponse<InputStream> response = client.send(request, BodyHandlers.ofInputStream());
```

Если в качестве ответа тебе должны прислать файл, то в метод `BodyHandlers.ofFile()` тебе нужно передать имя локального файла, куда он будет сохранен объектом HttpClient.

4.2 Метод followRedirects()

Также при отправке запроса ты можешь указать, что делать HttpClient'у, если сервер в ответ пришлет `301` или `302` (временный или постоянный редирект). Представь, что сервер прислал код `302`, и тебе нужно: отследить эту ситуацию, получить новый URL из ответа и отправить запрос по новому адресу.

Не сильно бы хотелось таким заниматься, особенно учитывая, что эта ситуация встречается часто и уже давно автоматизирована во всех http-клиентах. Также и в этом HttpClient'е тебе нужно просто указать какой режим редиректа ты выбираешь при отправке запроса.

```
1  HttpResponse response = HttpClient.newBuilder()
2    .followRedirects( HttpClient.Redirect.ALWAYS )
3    .build()
4    .send(request, BodyHandlers.ofString());
5
```

Есть всего 3 варианта для редиректа:

- **ALWAYS** – всегда;
- **NEVER** – никогда;
- **NORMAL** – всегда, кроме HTTPS -> HTTP.

Как видишь, вариантов тут немного, но всегда лучше иметь возможность настройки, чем не иметь ее.

4.4 Метод proxy()

Есть еще пара полезных, но не часто используемых опций. Они тебе не нужны ровно до тех пор, пока не станут нужны :)

Первый – это proxy. В обычной жизни ты не часто сталкиваешься с ними, но многие крупные корпорации имеют у себя внутри сложную систему безопасности интернет-трафика, а значит и различные настройки proxy.

Ну, и конечно, твой софт, который будет работать где-то в недрах такой корпорации когда-то столкнется с тем, что ему нужно будет задействовать proxy. Поэтому хорошо, что такая опция тут тоже есть.

Настроить прокси очень просто – пример:

```
1  HttpResponse<String> response = HttpClient.newBuilder()
2    .proxy( ProxySelector.getDefault())
3    .build()
4    .send(request, BodyHandlers.ofString());
```

Тут был выбран proxy по умолчанию, но ты можешь захотеть настроить свой:

```
1  HttpResponse response = HttpClient.newBuilder()
2    .proxy(ProxySelector.of(new InetSocketAddress("proxy.microsoft.com", 80)))
3    .build()
4    .send(request, BodyHandlers.ofString());
```

Как именно работать с proxy мы рассматривать не будем, так как это не входит в рамки данного курса.

4.5 authenticator()

И еще один важный момент. HTTP-протокол поддерживает аутентификацию. Прямо на уровне протокола.

Сейчас таким подходом почти не пользуются, но лет 20 назад он был распространен. Выглядел такой Http-запрос так:

```
http://username@example.com/
```

Или даже так:

```
http://username:password@example.com/
```

Это не почта. Это именно ссылка. И да, тебе не показалось. Логин и даже пароль можно было указать прямо в http-запросе. Да и сейчас можно. Поэтому я и пишу, что сейчас так обычно никто не делает. Но такая возможность есть.

```
Authenticator auth = new Authenticator() {
    @Override
    protected PasswordAuthentication getPasswordAuthentication() {
    return new PasswordAuthentication(
     "username",
        "password".toCharArray());
    }
};

HttpResponse<String> response = HttpClient.newBuilder()
  .authenticator(auth).build()
  .send(request, BodyHandlers.ofString());
```

Это интересно! Знаете почему вместо `“password"` в коде написано `"password".toCharArray()` ?

Потому что второй параметр конструктора `PasswordAuthentication` – не `String`, а `CharArray`.

А почему второй параметр не `String`, а `CharArray`?

Потому что все пароли в целях безопасности запрещается хранить в виде целой строки даже в своем собственном приложении. То есть ваше приложение в своей памяти не должно хранить пароль в виде строки. Чтобы если кто-то сделал damp-памяти, из него нельзя было вытащить пароль…

Но при этом пароль можно передать по незащищенному HTTP-протоколу через полмира :) :) :)

Что ж. Мир не идеален.

Более детально ты можешь ознакомиться с этой темой по ссылкам:

HTTP аутентификация

Understanding HTTP Authentication

‹ Предыдущая лекция

Следующая лекция ›

Введите текст комментария

Andrey Panchenko Моет полы в **Яндекс** 17 сентября 2022, 15:22

Ох попа будет гореть у кого-то на Spring Security без знаний AAA.

Ответить 0

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

RUSH

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

ДОСТУПНО В Google Play

Загрузите в App Store