электронный журнал
МОЛОДЕЖНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
Издатель Общероссийская общественная организация "Академия инженерных наук им. А.М. Прохорова"
ISSN 2307-0609

 # 01, март 2018

УДК 004.415.25

Современные способы разработки http клиента на Java для интеграции с внешними API

***Саневич Е.Г.**, студент*
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Информационные системы и телекоммуникации»
sanevichj@gmail.com

Научный руководитель: Сидякин И.М., доцент, к.т.н.
Россия, 105005, г. Москва, МГТУ им. Н.Э. Баумана,
кафедра «Информационные системы и телекоммуникации»
ivan.sidyakin@gmail.com

Аннотация: В статье рассмотрены современные способы реализации http клиентов. Описаны основы стандарта HTTP, основные виды запросов и ответов и их наполнение. Проведен анализ поддержки REST стандарта различными клиентами. Представлен обзор трёх библиотек с функциями http клиентов. Описана методика разработки интеграции информационной системы с системой управления версиями GitLab с помощью его API.

Ключевые слова: http клиент (http client), разработка программного обеспечения (software development), язык программирования Java (Java programming language).

Введение

Понятие http клиента. Протокол HTTP [1] работает поверх TCP/IP. Фактически же это означает, что клиент открывает сокет до сервера, пишет туда HTTP запрос (request), сервер читает запрос, обрабатывает его и посылает результат обработки (response) обратно клиенту.

Любой HTTP запрос, как и любой ответ по этому протоколу состоит из двух блоков: заголовок и собственно данные. Заголовок отделён от данных двойным символом переноса строки (в Java это будет "\n\n", хотя допускается и "\r\n\r\n" для платформы Windows).

Так как HTTP был изначально ориентирован на пересылку прежде всего текстовой информации, то HTTP заголовок является полностью текстовым, все символы, передающиеся в нём, являются печатными (прежде всего цифры и литеры латинского алфавита A-Z, a-z, а также набор других отображаемых символов + символ переноса строки "\n" или "\r\n"). При передаче в HTTP заголовке других символов, будет выдана ошибка "400 Bad request".

Таким образом, http клиент – это программа, которая может посылать http запросы на сервер и обрабатывать ответ сервера.

1. Структура http запроса

Разберём подробнее HTTP запрос клиента. Он может выглядеть, например, так:

```
POST http://localhost/ HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Language: ru
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
Host: localhost
Proxy-Connection: Keep-Alive

param1=1&param2=2
```

Взглянув на пример, можно заметить, что запрос начинается со слова "POST". Это слово означает метод передачи данных на сервер, в котором дополнительные данные запроса (строка "param1=1&param2=2") передаются после заголовка.

Разберём по строкам HTTP заголовок запроса.

Первая строка, первое слово - имя метода запроса. Это слово может быть одно из следующих:

- OPTIONS
- GET
- HEAD
- POST
- PUT
- DELETE
- TRACE
- CONNECT

Метод GET передает запрос на сервер с параметрами в URI, тело запроса при этом пустое.

Метод POST предназначен для запроса, при котором веб-сервер принимает данные, заключённые в тело сообщения, для хранения. Он часто используется для загрузки файла или представления заполненной веб-формы.

Метод HEAD по действию практически идентичен методу GET с одним отличием - в ответе на метод HEAD сервер выдаёт только HTTP заголовок, не выдавая содержимого документа.

Метод PUT по действию идентичен методу POST, но, как и HEAD, выдаёт только заголовок HTTP.

Метод OPTIONS выдаёт все действия, которые можно совершить с документом.

Метод DELETE указывает серверу, чтобы он предпринял попытку к удалению документа. Возможным ответом является ошибка политики безопасности ("403 Forbidden").

Методом TRACE можно получить путь запроса до сервера, список узловых точек, гейтов (Gate), путь через прокси-сервера.

Метод CONNECT возвращает есть ли связь с сервером и поддерживает ли сервер HTTP протокол.

Сразу после ключевого слова, определяющего метод, идёт символ пробела и указан URI документа, запрашиваемого с сервера. После URI документа идёт ещё один символ пробела и название протокола (строка "HTTP/1.1").

URI расшифровывается как Uniform Resource Identifier (формат записи индитефикатора ресурса), полностью он описан в RFC 2396. Для HTTP существует разновидность стандарта URI, называемая URL (Uniform Resource Location - формат записи нахождения ресурса), к примеру

```
http://bmstu.ru:80/javalinks/catalog.php3?name=java&cat=2#section1
```

Из приведённого примера URL можно выделить логические части:

1) [http://]
2) [devresource.org:80/javalinks/catalog.php3]
3) [?name=java&cat=2]
4) [#section1]

Часть (1) указывает на протокол доступа к документу, часть (2) - можно разбить на три части - [devresource.org], [:80], [/javalinks/catalog.php3] – это имя хоста (вместо имени devresource.org может стоять и IP адрес), порт сервера через символ ":" и путь (path) до документа от корня (root) сервера. Стандартным портом для HTTP сервера является порт 80 и, поэтому, его можно не указывать. Внимание! Если порт сервера отличается от 80, то в URL его нужно обязательно указать.

Третья часть URL - это GET часть запроса, отделена от документа символом "?".

И, наконец, последняя часть URI - "секция", отделённая символом "#". При HTML форматировании в документе можно положить закладку (по-другому – установить якорь, он же anchor, отсюда и название HTML тега <A>). Если в URI ресурса указана секция, то в HTML ищется одноимённая закладка, а браузер при отображении документа показывает текст, отмеченный якорем.

2. Параметры http запроса

Далее приведены основные параметры для HTTP заголовка. Каждая строка, содержащая параметр начинается с ключевого слова (например, "Host"), потом идёт символ двоеточие, пробел, значение параметра и символ переноса строки. Приведённые параметры соответствуют стандарту RFC 2616 (HTTP/1.1). Здесь приводится не весь список возможных полей запроса и их значений.

```
Host: localhost
```

Этот параметр содержит имя хоста, например, "localhost" или "localhost:80" (если порт 80, то его можно не указывать, если порт отличается от 80, то его нужно обязательно указать). Это второй обязательный параметр HTTP заголовка (первый - HTTP метод и имя протокола).

```
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
```

В этом параметре через запятую указаны MIME типы документов, которые способен обработать браузер. Так же в MIME типе указываются доступные к обработке кодировки документов. Подробнее о стандарте MIME смотрите тут - RFC 2045. Символ "*" в указании типа означает, что браузер может обработать весь класс документов. К примеру, image/* означает, что браузер может обработать и image/gif, и image/x-xbitmap, и image/jpeg, и image/pjpeg и вообще любые документы изображений, а заключительный тип - */* указывает, что браузер обработает любые документы, присланные сервером. На деле это обычно означает, что если MIME тип присланного сервером документа браузеру неизвестен, то он предложит сохранить его на диск.

```
Accept-Language: ru, en
Accept-Charset: windows-1251, KOI8-R
```

Эти параметры отвечают за языки. В первом - через запятую указываются предпочтительные языки для сервера. В частности, Google.com, обработав этот параметр, перенаправит вас на русскоязычную страничку. Во втором - кодировка, в которой закодированы символы в CGI запросе. Также через запятую могут быть указаны предпочитаемые кодировки для ответа сервера.

```
Accept-Encoding: compressed, gzip
```

Для запроса к серверу, обязательными являются лишь два параметра: первая строка, уточняющая метод запроса и несущая адрес ресурса и параметр "Host", содержащий имя хоста и порт сервера.

3. Разбор http ответа

Стартовая строка ответа имеет следующую структуру:

```
HTTP/Версия Код состояния Пояснение
```

Версия протокола здесь задаётся так же, как в запросе.

Код состояния (Status Code) — три цифры (первая из которых указывает на класс состояния), которые определяют результат совершения запроса. Например, в случае, если был использован метод GET, и сервер предоставляет ресурс с указанным идентификатором, то такое состояние задаётся с помощью кода 200. Если сервер сообщает о том, что такого ресурса не существует — 404. Если сервер сообщает о том, что не может предоставить доступ к данному ресурсу по причине отсутствия необходимых привилегий у клиента, то используется код 403. Спецификация HTTP 1.1 определяет 40 различных кодов HTTP, а также допускается расширение протокола и использование дополнительных кодов состояний.

Пояснение к коду состояния (Reason Phrase) — текстовое (но не включающее символы CR и LF) пояснение к коду ответа, предназначено для упрощения чтения ответа человеком. Пояснение может не учитываться клиентским программным обеспечением, а также может отличаться от стандартного в некоторых реализациях серверного ПО.

После стартовой строки следуют заголовки, а также тело ответа. Например:

```
HTTP/1.1 200 OK
Server: nginx/1.2.1
Date: Sat, 08 Mar 2014 22:53:46 GMT
Content-Type: application/octet-stream
Content-Length: 7
Last-Modified: Sat, 08 Mar 2014 22:53:30 GMT
Connection: keep-alive
Accept-Ranges: bytes

Wisdom
```

Тело ответа следует через два переноса строки после последнего заголовка. Для определения окончания тела ответа используется значение заголовка Content-Length (в данном случае ответ содержит 7 восьмеричных байтов: слово «Wisdom» и символ переноса строки).

4. Реализации http клиентов

Http клиент должен уметь формировать запрос и обрабатывать ответ в соответствии с протоколом http, частично описанным выше. Ясно, что, в самом простом случае запрос можно формировать вручную, как простую строку, соответствующую стандарту запроса, и ответ сервера тоже рассматривать как строку. Однако такой подход затруднителен в разработке и может привести к различным ошибкам и опечаткам при формировании запросов. Поэтому существует большое количество готовых http клиентов, которые скрывают в себе тонкости формирования запроса и получения ответа, концентрируя внимание разработчика не на технических особенностях протокола, а на реализации бизнес функций системы.

Рассмотрим 3 http клиента: Apache HttpClient [2], RestTemplate в Spring [3] и новейший HttpClient из Java 9 [4].

5. Apache HttpClient

Одна из старейших реализаций http клиента, обладает богатым набором возможностей для конфигурации запросов, однако имеет несколько громоздкий синтаксис. На листинге ниже показан пример формирования POST запроса.

```
public static void main(String[] args) throws IOException {
    HttpClient client = new DefaultHttpClient();
    HttpPost post = new HttpPost("http://restUrl");
    List nameValuePairs = new ArrayList(1);
    nameValuePairs.add(new BasicNameValuePair("name", "value"));
    post.setEntity(new UrlEncodedFormEntity(nameValuePairs));
    HttpResponse response = client.execute(post);
    BufferedReader rd = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
    String line = "";
    while ((line = rd.readLine()) != null) {
        System.out.println(line);
    }
}
```

6. HttpClient из Java 9

Этот клиент встроен в стандартную библиотеку JDK, начиная с 9 версии. До сих пор является экспериментальной функцией языка и находится в модуле incubator, поэтому для использования требуется указать флаг при старте приложения
--add-modules jdk.incubator.httpclient

На листинге ниже показан пример формирован POST запроса, по структуре такого же как в описании Apache HttpClient.

```
public static void main(String[] args) throws Exception {
   HttpClient client = HttpClient.newHttpClient();
   BodyProcessor bodyProcessor = BodyProcessor.fromString("{\"name\":\"value\"}");

   HttpRequest request = newBuilder()
         .header("Content-Type", "application/json")
         .uri(new URI("http://restUrl"))
         .POST(bodyProcessor)
         .build();

   HttpResponse<String> response = client.send(request, HttpResponse.BodyHandler.asString());
   System.out.println(response.body());
}
```

Видно насколько более просто и явно описывается запрос в этом клиенте, в отличие от реализации Apache.

Еще одна особенность HttpClient – очень лаконичная поддержка асинхронных запросов, как показано на листинге ниже.

```
client.sendAsync(request, HttpResponse.BodyHandler.asString())
      .completeOnTimeout(errorResponse, 10, TimeUnit.SECONDS)
      .thenAccept(response -> ...);
```

Кстати, completeOnTimeout – тоже нововведение в Java 9.

Работа над клиентом по-прежнему ведется. Несмотря на значительные успехи, пока еще нет поддержки для application/x-www-form-urlencoded, multipart/form-data, JSON обработчиков тела запроса и т.д.

7. Spring RestTemplate

Предыдущие 2 http клиента обладают достоинствами и недостатками, в частности Apache HttpClient поддерживает все возможные особенности http протокола, но выглядит весьма громоздко и тяжел в разработке. HttpClient из Java 9 крайне удобен для разработки, однако не поддерживает все необходимые для крупного проекта функции, и вообще является экспериментальным, поэтому в настоящий момент не подходит для включения в стек разработки настоящего крупного приложения.

Хорошей альтернативой, включающей в себя преимущества обоих, описанных выше клиентов является RestTemplate и фреймворка Spring. С одной стороны, он полностью поддерживает REST стандарт, а с другой – имеет весьма лаконичный код, подобный клиенту из Java 9.

На листинге ниже показан пример формирован POST запроса.

```
public static void main(String[] args) throws Exception {
   RestTemplate restTemplate = new RestTemplate();
   HttpHeaders headers = new HttpHeaders();

   headers.set("Content-Type", MediaType.APPLICATION_JSON_VALUE);

   RequestBody body = RequestBody.builder()
      .name(("value")
      .build();

   HttpEntity<RequestBody> req = new HttpEntity<>(body, headers);
   ResponseEntity<String> response =
            restTemplate.postForEntity("http://restUrl", req, String.class);
   System.out.println(response.getBody());
}
```

RestTemplate имеет специализированные методы для выполнения разных запросов, как показано в таблице 1. Кроме того, метод exchange() можно выполнять для любого типа запросов.

Таблица 1

Методы для http запросов

DELETE	*delete(java.lang.String, java.lang.Object...)*
GET	*getForObject(java.lang.String, java.lang.Class<T>, java.lang.Object...)*
	getForEntity(java.lang.String, java.lang.Class<T>, java.lang.Object...)
HEAD	*headForHeaders(java.lang.String, java.lang.Object...)*
OPTIONS	*optionsForAllow(java.lang.String, java.lang.Object...)*
POST	*postForLocation(java.lang.String, java.lang.Object, java.lang.Object...)*
	postForObject(java.lang.String, java.lang.Object, java.lang.Class<T>, java.lang.Object...)
PUT	*put(java.lang.String, java.lang.Object, java.lang.Object...)*
any	*exchange(java.lang.String, org.springframework.http.HttpMethod, org.springframework.http.HttpEntity<?>, java.lang.Class<T>, java.lang.Object...)*

8. Реализация интеграции с Gitlab API

Цель использования http клиента в проекте – реализация интеграции разрабатываемого сервиса с какой-либо внешней системой. В моей системе управления обучением [6] необходимо сделать интеграцию с системой управления версиями – Gitlab. Необходимо иметь возможность автоматически производить регистрацию студентов в Gitlab, давать им права, делать форки проектов и тп.

Для этого у GitLab есть свой API [7] – описание интерфейса взаимодействия через http протокол по REST. Для реализации клиента было принято решение использовать возможности RestTemplate.

На листинге ниже показан настоящий метод из исходного кода системы, который уменьшает права пользователя в проекте после форка:

```
void editProject(User user, long forkedProjectId) {
    String token = user.getGitAccessToken();
    HttpHeaders headers = new HttpHeaders();
    headers.set("Accept", MediaType.APPLICATION_JSON_VALUE);
    headers.set("Authorization", "Bearer " + token);

    RequestEditProject body = RequestEditProject.builder()
            .id(forkedProjectId)
            .issuesEnabled(false)
            .mergeRequestsEnabled(false)
            .visibilityLevel(0)
            .build();

    HttpEntity<RequestEditProject> reqEditProject = new HttpEntity<>(body, headers);

    String urlProjects = GITLAB_BASE_URL + "/api/v3/projects/";
    logger.info("Step: Edit permissions for forked project");
    logger.info("Send PUT request to {} ...", urlProjects);
    logger.info("Body of PUT: {}", body);
    restTemplate.put(urlProjects + forkedProjectId, reqEditProject);

    logger.info("Send GET request to {} ...", urlProjects);
    ResponseEntity<String> dataEditedProject = restTemplate.exchange(urlProjects + forkedProjectId,
HttpMethod.GET, new HttpEntity<>(headers), String.class);

    logger.info("Project {} for user {} was edited, rights reduced, response: {}",
            forkedProjectId,
            user.getEmail(),
            dataEditedProject.getBody());
}
```

В данном методе используется общий метод exchange. Это обусловлено его широким применением, и легкостью к дальнейшим изменениям. К тому же сейчас формат парсинга ответа – String, который можно будет заменить на класс, в который ответ сможет десериализоваться.

Благодаря использованию RestTemplate код получился легким для восприятия и последующей модификации.

Заключение

В статье описан процесс разработки http клиента – от объяснения видов и полей http запросов, до особенностей высокоуровневой реализации библиотек http клиентов. Задача разработки http клиента является стандартной задачей в области web разработки, поэтому сегодня существует большое количество способов упростить её решение в виде готовых модулей, специальных классов и т.п. Это безусловно упрощает разработку, однако не освобождает разработчика от необходимости глубинного понимания всего процесса внутренней коммуникации современных информационных систем.

Список литературы

[1]. RFC-2616. [Электронный ресурс]. – Режим доступа: https://tools.ietf.org/html/rfc2616 (Дата обращения: 26.01.18).

[2]. Apache HttpClient [Электронный ресурс]. – Режим доступа https://hc.apache.org/httpcomponents-client-ga/index.html (дата обращения 26.01.18)

[3]. Spring RestTemplate [Электронный ресурс]. – Режим доступа https://docs.spring.io/spring/docs/current/javadocapi/org/springframework/web/client/RestTemplate.html (дата обращения 26.01.18)

[4]. JDK 9 HttpClient [Электронный ресурс]. – Режим доступа https://docs.oracle.com/javase/9/docs/api/jdk/incubator/http/HttpClient.html (дата обращения 26.01.18)

[5]. Apache HttpClient [Электронный ресурс]. – Режим доступа https://hc.apache.org/httpcomponents-client-ga/index.html (дата обращения 26.01.18)

[6]. Саневич Е.Г., Сидякин И.М. Интеграция платформы GitLab с сайтом кафедры «Информационные системы и телекоммуникации» МГТУ им. Н.Э. Баумана для обучения программированию // Молодежный научно-технический вестник. 2017. №5. Режим доступа http://sntbul.bmstu.ru/doc/857361.html (дата обращения 26.01.18)

[7]. GitLab API [Электронный ресурс]. – Режим доступа https://docs.gitlab.com/ee/api/README.html (дата обращения 26.01.18)