Титульный лист материалов по дисциплине

ДИСЦИПЛИНА **Разработка клиент-серверных приложений**

полное название дисциплины без аббревиатуры

ИНСТИТУТ **Информационных технологий**

КАФЕДРА **Инструментального и прикладного программного**

полное название кафедры **обеспечения**

ГРУППА/Ы **ИКБО-01/02/03/12/13/16-18**

номер групп/ы, для которых предназначены материалы

ВИД УЧЕБНОГО МАТЕРИАЛА **Материал к практическим занятиям**

лекция; материал к практическим занятиям; контрольно-измерительные материалы к практическим занятиям; руководство к КР/КП, практикам

ПРЕПОДАВАТЕЛЬ **Строганкова Наталья Владимировна**

фамилия, имя, отчество

СЕМЕСТР **5**

указать номер семестра обучения

ПРАКТИЧЕСКАЯ РАБОТА № 2

1 Цель работы

Целью данной работы является знакомство с протоколом передачи файлов прикладного уровня HTTP, а также научиться работать с GET- и POST-запросами.

2 Теоретическая часть

2.1 Hyper Text Transfer Protocol (HTTP)

Для создания клиент-серверных приложений чаще используют протокол прикладного уровня (модели OSI), который готовит компьютеры к обмену информацией: Hypertext Transfer Protocol (HTTP).

HTTP – протокол прикладного уровня поверх коммуникационного протокола TCP/IP. Изначально HTTP рассматривался как протокол передачи гипертекста, сейчас он широко используется и для передачи файлов.

HTTP оперирует разными запросами, однако, на сайтах чаще всего встречается только два основных запроса:

– **GET** — запрос документа. Наиболее часто употребляемый метод (в HTTP/0.9 он был единственным);

– **POST** — этот метод применяется для передачи данных CGI-скриптам. Сами данные идут в следующих строках запроса в виде параметров.

У всех запросов (Request) и ответов (Response) используется одна и та же структура: строка запроса/ответа, заголовки, пустая разделительная строка и тело сообщения. При этом обязательной является только строка запроса.

2.1.1 Структура HTTP-запросов и ответов

В таблицах 1 и 2 представлена структура входящего GET-запроса на сервер.

Таблица 1 – Пример GET-запроса страницы сайта yandex.ru

Строка запроса (Request Line)	`1. GET/index.html HTTP/1.1`
Заголовки (Message Headers)	`2. Host: yandex.ru`
	`3. Connection: keep-alive`

	4. Cache-Control: max-age=0
	5. Accept: text/html, application/xhtml+xml, application/xml; q=0.9, image/webp,*/*; q=0.8
	6. User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Safari/537.36
Заголовки (Message Headers)	7. Referer: http://yandex.ru/…
	8. Accept-Encoding: gzip, deflate, sdch
	9. Accept-Language: ru-RU, ru; q=0.8, en-US; q=0.6, en; q=0.4
	10. Cookie: z=…
Пустая строка разделитель	
Тело сообщения (EntityBody) (*необязательный параметр)	Например: файл

Таблица 2 – Пример GET-ответа сайта yandex.ru

Строка ответа (Response Line)	1. HTTP/1.1 200 OK
Заголовки (Message Headers)	2. Server: nginx
	3. Date: Mon, 29 Sep 2014 08:04:48 GMT
	4. Connection: close
	5. Cache-Control: no-cache, no-store, max-age=0, must-revalidate
	6. Content-Length: 5073
	7. Content-Type: text/html; charset=utf-8
	8. Content-Encoding: gzip
Пустая строка разделитель	
Тело сообщения (EntityBody) (*необязательный параметр)	HTML-документ

Строка запроса (Request Line)

Указывает метод передачи, URL-адрес, к которому нужно обратиться, и версию протокола HTTP. В примере используется метод GET, адрес index.html, что позволяет серверу определить требуемый ресурс клиенту и версию HTTP.

Строка ответа (Response Line)

Указывает версию HTTP и код состояния запроса (Таблица 3).

Таблица 3 – Коды состояния запроса HTTP

№ кода	Значение состояния запроса
200	Запрос был получен и обработан
301	Ресурс перемещен постоянно
302	Ресурс перемещен временно
400	Неверный запрос — сообщение с запросом имеет некорректный формат
401	Несанкционированный доступ — у пользователя нет прав для доступа к запрошенному документу
402	Ресурс доступен за плату
408	Тайм-аут запроса
500	Внутренняя ошибка сервера — ошибка помешала HTTP-серверу обработать запрос

2.1.2 Заголовки HTTP-запросов и -ответов

Заголовки (Message Headers)

Host — имя хоста, на который производится запрос; необходимо в ситуациях, когда на сервере имеется несколько виртуальных серверов под одним IP-адресом. В этом случае имя виртуального сервера определяется по этому полю.

Connection — может принимать значения Keep-Alive и close.

Keep-Alive («*оставить в живых*») означает, что после выдачи данного документа соединение с сервером не разрывается, и можно выдавать еще запросы. Большинство браузеров работает именно в режиме Keep-Alive, так как он позволяет за одно соединение с сервером скачать html-страницу и рисунки к ней. Будучи однажды установленным, режим Keep-Alive сохраняется до первой ошибки или до явного указания в очередном запросе Connection: close.

close («*закрыть*») — соединение закрывается после ответа на данный запрос.

Cache-Control — срок годности содержимого в секундах:

– *no-cache*. Сервер не должен использовать кэшированный ответ;

– *no-store*. Ответ на этот запрос не должен кэшироваться;

– *max-age*=delta-seconds. Клиент допускает кэшированный ответ, если его возраст не превышает delta-seconds секунд; клиент не требует его валидации;

– *max-stale*=delta-seconds. Клиент допускает кэшированный ответ, если его возраст не превышает delta-seconds секунд;

– *min-fresh*=delta-seconds. Клиент допускает кэшированный ответ, если он будет оставаться действительным не менее deltaseconds секунд от момента запроса;

– *no-transform*. К запрашиваемому документу не должны применяться преобразования;

– *only-if-cached*. Допускается только кэшированный ответ. Если подходящего ответа нет в кэше, то не нужна ни валидация старого ответа, ни получение нового.

Accept — список поддерживаемых браузером типов содержимого в порядке их предпочтения данным браузером, например, для Google Chrome она будет выглядеть следующим образом: text/html, application/xhtml+xml, application/xml; q=0.9,

image/webp,*/*; q=0.8

Эти данные необходимы тогда, когда сервер может выдавать один и тот же документ в разных форматах.

User-Agent — значением является «кодовое обозначение» браузера.

Пример: Mozilla/5.0 (Windows NT 6.3; WOW64)

AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.124 Saf ari/537.36

Этот заголовок несет в себе несколько видов информации:

– название браузера и его версию;

– название операционной системы и ее версию;

– язык системы по умолчанию.

Referer — позволяет клиенту указать адрес (URL) ресурса, из которого получен запрашиваемый URL. Этот заголовок дает возможность серверу сгенерировать список обратных ссылок на ресурсы для будущего анализа, регистрации, оптимизированного кэширования и т.д. Он также позволяет прослеживать с целью последующего исправления устаревшие или введенные с ошибками ссылки.

Accept-Encoding — большинство современных браузеров поддерживает метод сжатия Gzip, о чем они и сообщают в заголовке AcceptEncoding. Сервер может отправить страницу в сжатом виде. При этом объем трафика может уменьшиться до 80%, что довольно неплохо разгружает интернет-канал сайтов.

Server — информация о программном обеспечении сервера, отвечающего на запрос (это может быть как веб-, так и прокси-сервер).

Date — дата и время формирования сообщения.

Accept-Language — поддерживаемый язык. Имеет значение для сервера, который может выдавать один и тот же документ в разных языковых версиях.

Content-Type — MIME-тип возвращаемого документа.

Content-Length — размер возвращаемого документа.

Пустая строка-разделитель необходима для разделения частей сообщения заголовков и тела сообщения.

Тело сообщения (*EntityBody*) может содержать в себе любую информацию, которую мы хотим передать на сервер, например, изображение, аудио, видео, файл, html и др.

2.2 Управление потоками

В настоящее время количество пользователей интернета невероятно велико, и очевидно, что к серверу одновременно обращается большое количество клиентов. Для решения данной задачи используются потоки, которые позволяют параллельно обрабатывать нескольких клиентов.

Потоки — это независимые последовательности операций. Существует два пути создания потока. Первый — наследование от класса java.lang.Thread и переопределение его метода run(). Второй — реализация интерфейса java.lang.Runnable и создание потока на основе этой реализации. В принципе, эти методы эквивалентны, разница в деталях.

Наследование от java.lang.Thread делает его единственным родителем класса, что не всегда удобно. Таким образом, простейший поток может быть реализован так:

```
1. public class SampleThread extends Thread {
```

```
2. Integer counter = 0;
3. public SampleThread (Integer integer) {
4. counter=integer;
5. }
6. @Override
7. public void run () {
8. Integer result = counter*counter;
9. System.out.println (result);
10. }
11. public static void main (String [] args) {
12. for (int i = 10; i > 0; i —) {
13. new Thread (new SampleThread (i)).start ();
14. }
15. }
16. }
```

При этом результат всегда будет выводиться в разной очередности, например: 81;100;64;49;36;4;16;9;25;1.

3 Порядок выполнения работы

3.1 Технические требования к разрабатываемому серверу

Технические требования к разрабатываемому серверу следующие:

1) порт, на котором запускается сервер, 8080;

2) количество одновременно обрабатываемых клиентских запросов не ограничено (создание многопоточности);

3) сервер должен распознавать метод запроса и реагировать только на метод GET. Реакция на все остальные методы (POST, PUT, DELETE и др.) не оговаривается и может быть реализована по желанию, но при этом, если будет реализован только один метод GET, на другие методы ваш сервер не должен срабатывать;

4) в заголовке выдаваемого ответа нужно указать корректный тип (text/html) и длину тела сформированного сообщения;

5) в ответ на любой запрошенный ресурс сервер должен выдавать HTML-страницу с возможностью отображения русских слов.

3.2 Дополнительные требования к выполнению работы

Дополнительные требования к выполнению работы следующие:

1) написать процедуру распознавания окончания запроса (наличия пустой строки, в которой есть символ возврата каретки, но нет ни одного значащего текстового символа);

2) по окончании запроса провести анализ полученного метода. Если это GET — перейти к формированию корректного ответа клиенту;

3) сформировать ответ и выдать его клиенту. Корректно завершить соединение с клиентом. При следующем запросе клиент установит новое соединение;

4) должен быть предусмотрен механизм обработки кодировки входящего/исходящего потока данных.

3.3 Результат выполнения практической работы

В данной практической работе каждый студент работает с тем индивидуальным заданием, которое было у него в Практической работе №1.

1) В ответ на любой запрос к серверу ответ должен содержать одну и ту же HTML-страницу, состоящую из надписи (пример):

Работу выполнил: Иванов Иван Иванович

Номер группы: ИКБО-01-18

Номер индивидуального задания: 3

Текст индивидуального задания: «Создание калькулятора…»

2) Сервер должен работать по принципу echo-сервера, то есть то, что мы запрашиваем в адресной строке, то и отображаем на странице. Ответ должен приходить в виде HTML-страницы и должен содержать заполненные данные предыдущего этапа работы и сформированный ответ адресной строки, при этом ответ не должен содержать адреса сервера. Пример сформированного ответа адресной строки:

Запрос: http://localhost:8080/display_the_entered_value

Ответ: HTML-страница: <h1>display the entered value</h1>

3) Сервер должен принимать данные и выполнять вычисления в соответствии с индивидуальным заданием (практическая работа №1), и отправлять результат клиенту в виде HTML-страницы. HTML-страница должна содержать ранее выполненные этапы практической работы, а также информацию о произведенных вычислениях. Пример: при выполнении практической работы было выбрано индивидуальное задание №1 «Найти максимальное значение среди множества чисел…», значит, у нас в адресную строку в качестве ресурса должна быть введена цепочка чисел через разделительный символ: «11, 32, 1, 22, 14», сервер должен разбить строку, произвести вычисления и вернуть результат клиенту (браузеру).

4) Произвести сборку HTTP-сервера в jar-файл и проверку работоспособности jar-файла.