# Разработка HTTP сервера

*Т.В. Бондаренко, Е.А. Федотов, А.В. Бондаренко*

*Белгородский государственный технологический университет им. В.Г. Шухова, Белгород*

**Аннотация:** В рамках данной статьи описывается разработка программы для обслуживания HTTP-клиентов в автоматическом режиме. Программа позволяет клиентам получать, регистрировать, перезаписывать и удалять ресурсы. На вход подается адрес конфигурационного файла, в котором указаны основные параметры и каталог ресурсов. В процессе работы сервер принимает входящее соединение, определяет запрошенный ресурс, его приемлемую версию и, проверив допустимость выполнения данного метода, осуществляет соответствующую операцию. Процесс работы сопровождается выводом информации об обработанных запросах: идентификатор обработавшего потока, адрес клиента, метод, идентификатор ресурса, код состояния.

**Ключевые слова:** HTTP сервер, протокол HTTP, веб-сервер, клиент-серверная технология.

Протокол HTTP реализует клиент-серверную технологию, которая предполагает наличие множества клиентов, инициирующих соединение и посылающих запрос, а также множества серверов, получающих запросы, выполняющих требуемые действия и возвращающих клиентам результат [1].

Первоначальное назначение протокола HTTP заключалось в передаче данных в текстовом формат, при этом передаваемый текст мог включать гиперссылки. В протоколе HTTP главным объектом обработки является ресурс, который в клиентском запросе записан в URI (Uniform Resource Identifier). В качестве ресурса выступают файлы, хранящиеся на сервере. Протокол HTTP позволяет определить в запросе и ответе способ представления ресурса по различным параметрам [2].

Веб-сервер — сервер, работающий по протоколу HTTP и выдающий своим клиентам HTTP-ответы, обычно вместе с запрашиваемой веб-страницей. Для доступа в Интернет используются специальные приложения, веб-браузеры, которые представляют содержимое веб-страницы, запрашивают встроенные в страницу ресурсы и сценарии [3]. Набор серверов

Microsoft Internet Information Services реализует протокол HTTP. Основным компонентом IIS является веб-сервер — служба WWW, которая предоставляет клиентам доступ к сайтам по протоколам HTTP. Сервер IIS способен выполнять обслуживание несколько различных сайтов, каждый из которых имеет атрибуты: IP-адрес сайта; TCP-порт, на котором служба WWW ожидает подключений к сайту [4].

Преимущества протокола HTTP: простота, расширяемость, распространённость, документация на различных языках. Недостатки протокола HTTP: отсутствие "навигации", отсутствие поддержки распределенных действий.

В рамках данной статьи рассматривается разработка алгоритма, который анализирует запросы на выдачу страницы от Интернет-браузера, формирует и отправляет ответ. Алгоритм должен поддерживать методы GET, PUT и HEAD; отправлять в ответ не только HTML текст, но и изображение.

Сервер реализует методы OPTIONS, HEAD, GET, PUT, POST и DELETE. Ресурсы, которыми может манипулировать сервер, указываются в конфигурационном файле. Ресурсы с одним и тем же идентификатором могут храниться в нескольких версиях с разными MIME-типами, языками и алгоритмами сжатия. Для каждой версии указываются заголовки типа Content-*, а также специальные собственные заголовки — расположение файла и номер секции следующей версии ресурса. Возможно внедрение собственных заголовков, которые будут передаваться клиентам в GET- и HEAD-запросах. В этом же файле находятся параметры конфигурации сервера. Когда запрос приходит на сервер, главный поток выбирает свободный поток из пула и передает ему дескриптор сокета. При OPTIONS-, HEAD- и GET-запросах, сервер просматривает список версий ресурсов с заданным URI и выбирает из них первую, соответствующую заголовкам Accept, Accept- Language и Accept-Encoding запроса. Если таковая не

найдена, клиенту отправляется ответ с кодом состояния 503 Not Acceptable. В ответ включается код состояния и заголовок Allow; если это HEAD-запрос, то включаются заголовки Content-Type, Content-Language, Content-Encoding, Content-Length, а если это GET-запрос, то и содержимое файла. При PUT-запросах создается новая версия ресурса и удаляются все те, которые являются ее подмножеством. При POST-запросе в каталоге регистрируется новый ресурс; если ресурс с таким URI уже существует, отправляется ответ 409 Conflict. При DELETE-запросах удаляются все версии ресурса [4]. Конфигурационный файл имеет следующий синтаксис:

```
"[gencfg]"
["port=" номер-порта-сервера]
["threads=" количество-потоков-в-пуле]
["backlog=" длина-очереди-соединений]
["timeout=" время-в-секундах-после-которого-соединение-автоматически-закрывается]
"dumpdir=" каталог-файловой-системы-в-который-будут-записываться-новые-файлы
{
        "[номер-раздела]"
        "_type=head"
        "_urn=" URI-ресурса
        "_next=" номер-раздела-первой-версии-ресурса
        {
                "[номер-раздела]"
                ("_next=" номер-раздела-следующей-версии-ресурса) | ("_next=null")
                "_path=" путь-к-файлу-в-файловой-системе
                ["Allow=" список-допустимых-методов]
                ["Content-Type=" MIME-тип-данных]
                ["Content-Language=" язык-данных-файла]
                ["Content-Encoding=" алгоритм-сжатия-данных-в-файле]
                ""
        }
}
```

Каталог разделяется между всеми потоками, доступ к нему разграничен по времени [6].

Основные алгоритмы HTTP сервера:

– OPTIONS: найти первую подходящую версию ресурса, отправить заголовок Allow.

– HEAD: отправить заголовки Allow, Content-Type, Content-Language, Content-Encoding, Content-Length, если HEAD содержится в Allow.

– GET: включить в ответ содержимое соответствующего файла, если GET

содержится в Allow.

– POST: если ресурс с таким URI отсутствует, создать запись в каталоге ресурсов с одной версией. Создать файл со случайным именем в специально указанном каталоге файловой системы, в него записать тело запроса.

– PUT: удалить все версии ресурса, которые может заменить собой новая.

– DELETE: удалить все версии ресурса.

Распознавание запроса происходит при помощи потокового считывания данных из сокета до символов-разделителей и последующего сравнения запросов со строками. Для хранения заголовков используется тип данных map<string, string> [7]. Алгоритм подпрограммы, разбирающей запрос и возвращающей ответ:

1. Записывать в строку "метод" символы из сокета до пробела
2. Если строка пустая, выйти из процедуры
3. Записывать в строку "URI" символы из сокета до пробела
4. Записывать в строку "версия" символы из сокета до пробела
5. Принять все заголовки
6. Выделить заголовок Connection и передать через параметр-ссылку в вызвавшую подпрограмму.
7. Если в строке "версия" первые 5 символов не равны "HTTP/" или следующие за ними символы не образуют число, меньшее или равное 1.1, отправить сообщение с кодом состояния 505 HTTP Version Not Supported.
8. Если строка "метод" равна "POST" или "PUT", принять тело запроса. Если в запросе не указан способ определения его длины, отправить ошибку 400 Bad Request.
9. Если запрос не OPTIONS *, то проверить наличие ресурса с заданным в строке "URI" идентификатором. Если ресурс не существует и метод не равен "POST", отправить сообщение 404 Not Found.
10. Если метод реализован, вызвать соответствующую функцию,

передав URI, заголовки и тело запроса, вернуть результат.

11. Если метод не реализован, но описан в стандарте, отправить 501 Not Implemented.

12. Если метод не описан в стандарте, отправить 400 Bad Request [8].

Выбор допустимой версии ресурса осуществляется перебором версий в каталоге ресурсов и сравнением при помощи функций отдельного модуля.

В функциях, обрабатывающих методы POST, PUT и DELETE, после внесения изменений в каталог ресурсов, вызывается процедура преобразования каталога в текстовую форму и записи в файл конфигурации. Тело запросов POST и PUT записывается в файл со случайно сгенерированным именем.

Функции, формирующей HTTP ответ, передается код состояния с пояснением, набор заголовков и тело ответа. Последние два аргумента могут отсутствовать [9]. Строка ответа составляется при помощи конкатенации:

ОТВЕТ(КОД_СОСТОЯНИЯ, [<НАЗВ, ЗНАЧ>] ЗАГОЛОВКИ, ТЕЛО)

1. СТРОКА_ОТВЕТА: = "HTTP/1.1 " + КОД_СОСТОЯНИЯ + "\r\n"
2. Для каждого заголовка: СТРОКА_ОТВЕТА += НАЗВ + ": " + ЗНАЧ + "\r\n"
3. СТРОКА_ОТВЕТА += "\r\n" + ТЕЛО
4. Вернуть СТРОКА_ОТВЕТА

Ответ в случае ошибки дополняется HTML-кодом в качестве тела ответа:

ОШИБКА(КОД_СОСТОЯНИЯ, [<НАЗВ, ЗНАЧ>] ЗАГОЛОВКИ, ДОП_ИНФ)

1. HTML: = "<h1 align=\"center\">" + КОД_СОСТОЯНИЯ + "</h1>"
2. Если есть доп. информация: HTML += "<p align=\"center\">" + ДОП_ИНФ +"</p>"
3. HTML += "<hr><p align=\"center\">" + НАЗВАНИЕ_СЕРВЕРА + "</p>"
4. Вернуть ОТВЕТ(КОД_СОСТОЯНИЯ, ЗАГОЛОВКИ, HTML)

Функции преобразования наборов заголовков:

1. Фильтрация перед ответом на запрос GET и HEAD: исключаются заголовки _type, _next, _urn и _path.

2. Фильтрация перед ответом на запрос OPTIONS: возвращается заголовок Allow.

3. Фильтрация перед сохранением в каталог ресурсов: исключаются все заголовки, кроме Content-Type, Content-Language и Content-Encoding [9].

Цикл обработки запросов:

```
ПОВТОРЯТЬ
    ОТВЕТИТЬ (СОКЕТ, РАЗБОР_ЗАПРОСА (СОКЕТ, &CONNECTION));
    ЕСЛИ CONNECTION != ""
        ПОСЛЕДНИЙ_ЗАПРОС := ТЕКУЩЕЕ_ВРЕМЯ ( );
ПОКА
    CONNECTION != "close" &&
    ТЕКУЩЕЕ_ВРЕМЯ ( ) - ПОСЛЕДНИЙ_ЗАПРОС < TIMEOUT_S &&
    ! СОКЕТ → СОЕДИНЕНИЕ_ЗАКРЫТО ( );
```

Модуль содержит два класса: конфигурация и сеанс работы. Класс конфигурации объявлен дружественным по отношению к классу сеанса работы с конфигурацией; работа с ним возможна только через класс сеанса. Класс конфигурации содержит блокирующую Конфигурация представлена типом данных map<string, vector<map<string, string>>>:

```
typedef map<string /*name*/, string /*value*/> version_headers;
typedef vector<version_headers> resource_versions;
typedef map<string /*URI*/, resource_versions> resources;
```

В структуре всегда храниться псевдо-ресурс с единственной версией, заголовки которой задают параметры сервера. Разбор конфигурационного файла производится по следующему алгоритму:

1. Инициализация переменных:
   map<string, vector<map<string, string>>> A
   map<string, map<string, string>> B
2. Открыть конфигурационный файл
3. Для каждой строки файла: если строка начинается символом '[', присвоить ее подстроку, начиная со 2-го символа и заканчивая предпоследним, строке "секция". Иначе, если строка содержит знак равенства, то присвоить ее подстроку до знака равенства строке "ключ", а подстроку после знака равенства — строке "значение", и занести тройку <"секция", <"ключ", "значение">> в B. Иначе, пропустить строку.
4. Закрыть конфигурационный файл.

В модуль управления версиями входят два класса: ContentSpec с параметрами ресурса и AcceptSpec с приемлемыми для клиента значениями

параметров по запросам GET, HEAD, PUT (Accept-*). Сравнение двух версий ресурса осуществляется методом "isSubsetOf". Один и тот же параметр двух версий сравнивается так: если параметр первой версии равен "*" или равен параметру второй версии, то вторая версия является частным случаем второй. Такое сравнение производится над параметрами Content-Language и Content-Encoding. Content-Type сравниваются в два этапа: если первый тип равен "*" или совпадает со вторым типом, то второй тип является подмножеством первого, если первый подтип равен "*" или равен второму подтипу. Проверка приемлемости ресурса реализуется методом "acceptable" класса AcceptSpec. Конструктор этого класса принимает на вход набор заголовков запроса, выделяет из них Accept, Accept-Language и Accept-Encoding. Метод acceptable проверяет соответствие фактического значения параметров значениям, приемлемым для клиента.

Пул потоков представлен одним классом. Конструктор пула принимает на вход количество потоков. С каждым потоком пул связывает структуру, в которой содержится указатель на признак незанятости потока и указатель на сокет, с которым он будет работать. Поток выполняется с процедуры:

```
DWORD WINAPI serverThread(void *param)
{
        for(;;) {
                *((ThreadTask *)param)->freeFlag = true;
                SuspendThread(GetCurrentThread());
                serverProc(((ThreadTask *)param)->currSocket);
        delete ((ThreadTask *)param)->currSocket;
        }
}
```

В качестве параметра в процедуру передается адрес соответствующей структуры. Сервер, принимая новое соединение, вызывается метод task:

```
void SvrThreadPool::task(TCPSocket *sock)
{
        while(!freeFlags[nextThread = (nextThread + 1) % numThreads]);

        threadTasks[nextThread].currSocket = sock;
        freeFlags[nextThread] = false;
        ResumeThread(threadHandles[nextThread]);
```

}

Он проходит циклически по признакам незанятости потоков и выбирает первый свободный поток. После выполнения обслуживания клиента сокет закрывается обслужившим его потоком.

Основная подпрограмма создает объект класса конфигурации, устанавливает значения параметров, создает объект пула потоков, открывает прослушивающий сокет, устанавливает его в режим прослушивания входящих соединений. Далее в бесконечном цикле принимаются соединения, сокеты которых отправляются в пул потоков. Строки символов и приходящие по сети данные также интерпретируются как потоки:

− Абстрактный класс Stream и его наследники NetStream и StringStream, реализуют виртуальный метод getChar: извлекает очередной символ из потока, при отсутствии — исключение StreamError.

− Прием данных из потока до обнаружения одной из перечисленных последовательностей символов.

− Разделение строки на подстроки.

− Чтение всего файла в строку.

− Запись всей строки в файл.

− Подпрограмма, генерирующая случайную строку при записи нового ресурса на диск.

− Подпрограмма, считывающая тело запроса.

− Прием заголовков запроса и разбор их в ассоциативный массив.

− Прием из сокета заданного числа символов.

− Отправка ответа клиенту.

Управление сокетами представляет класс, инкапсулирующий дескриптор TCP-сокета. Если создается первый сокет, конструктор инициализирует библиотеку Windows Sockets. Конструктор вызывает функции socket, bind, WSAStartup, если это единственный сокет. Деструктор

вызывает closesocket и WSACleanup. Методы snd, rcv, lsn, acp вызывают функции send, recv, listen, accept, не требуя явно передавать в них дескриптор сокета [10].

Разработанное приложение позволяет обслуживать HTTP клиентов, позволяя получить, зарегистрировать, перезаписать и удалить ресурс. Логическая структура каталога ресурсов не связана со структурой файловой системы, что придает серверу гибкость. Пул потоков помогает избежать потерь времени, связанных с созданием и уничтожением потоков. Выбор версии ресурса по параметрам, указанным в заголовках запроса, позволяет отправить клиенту веб-содержимое в желательной для него форме.

## Литература

1. Натальченко И.А. Анализ механизмов передачи крупных массивов данных через сеть интернет с помощью технологии веб-сервиса // Инженерный вестник Дона, 2008, №4 URL: ivdon.ru/ru/magazine/archive/n4y2008/98

2. HTTP: The Protocol Every Web Developer Must Know. URL: code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1-net-31177 (дата обращения: 11.05.2018)

3. HTTP серверы. URL: apachedev.ru/2006/03/12/the-apache-modeling-project-glava-2-chast-1/ (дата обращения: 11.05.2018)

4. Кришнамурти Б., Рэксфорд Дж. Web-протоколы. Теория и практика. М.: Изд-во БИНОМ, 2002 г. 292 с.

5. Станек У. Р. Internet Information Services (IIS) 7.0. Справочник администратора. СПб.: Русская редакция, 2009. 528 с.

6. Коды ответа HTTP URL: developer.mozilla.org/ru/docs/Web/HTTP/Status (дата обращения: 11.05.2018)

7. Java. HTTP протокол и работа с WEB. URL: javaportal.ru

/java/articles/java_http_web/article01.html#intro (дата обращения: 11.05.2018)

8. Федотов Е.А., Бондаренко Т.В., Федотова В.Н. Исследование протоколов обмена сообщениями в режиме реального времени // Вестник магистратуры. 2016. № 5-2 (56). С. 64-66.

9. Федотов Е.А., Солодилов М.А. Использование протокола ipx для передачи данных // В сборнике: Международная научно-техническая конференция молодых ученых БГТУ им. В.Г. Шухова. 2016. С. 3585-3590.

10. Сироткин А.В., Брачун Т.А., Бархатов Н.И. Моделирование приоритетного управления информационными потоками с использованием сокетов // Инженерный вестник Дона, 2012, №4 (1) URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1192

11. Pavan Podila HTTP: The Protocol Every Web Developer Must Know. 2013. URL: code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177 (date of access 11.05.2018)

## References

1. Natalchenko I.A. Inženernyj vestnik Dona (Rus), 2008, №4. URL: ivdon.ru/ru/magazine/archive/n4y2008/98

2. HTTP: The Protocol Every Web Developer Must Know. URL: code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177 (date of access 11.05.2018)

3. HTTP serveryi [HTTP servers] URL: apachedev.ru/2006/03/12/the-apache-modeling-project-glava-2-chast-1/ (accessed 11.05.2018)

4. Krishnamurti B., Reksford Dzh.. Web-protokolyi. Teoriya i praktika [Web-based protocols. Theory and practice]. M.: Izd-vo BINOM, 2002. 292p.

5. Stanek U.R. Internet Information Services (IIS) 7.0. Spravochnik administratora [Directory administrator]. SPb.: Russkaya redaktsiya, 2009. 528p.

6. Kodyi otveta HTTP [HTTP response codes] URL:

developer.mozilla.org/ru/docs/ Web/HTTP/Status (accessed 11.05.2018)

7. Java. HTTP protokol i rabota s WEB [Java. HTTP protocol and work with WEB] URL: javaportal.ru/java/articles/java_http_web/ article01.html#intro (accessed 11.05.2018)

8. Fedotov E.A., Bondarenko T.V., Fedotova V.N. Vestnik magistraturyi , 2016, № 5-2 (56). pp. 64-66.

9. Fedotov E.A., Solodilov M.A. Mezhdunarodnaya nauchno-tehnicheskaya konferentsiya molodyih uchenyih BGTU im. V.G. Shuhova, 2016. pp. 3585-3590.

10. Sirotkin A.V., Brachun T.A., Barhatov N.I. Inženernyj vestnik Dona (Rus), 2012, №4 (1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1192

11. Pavan Podila HTTP: The Protocol Every Web Developer Must Know. 2013. URL: code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177 (date of access 11.05.2018)