HTTP сервер

Мы рассмотрим HTTP протокол на примере передачи простейших Web страниц. Как это, собственно, и предполагалось в самом начале развития WWW. Со временем оказалось, что HTTP протокол весьма удобен и для передачи других данных, но это уже отдельный разговор.

Основу WWW страницы составляет HTML документ, поэтому нам придется познакомиться с этим понятием хотя бы в самых общих чертах.

Понятие о HTML

Исторически прародителем HTML является SGML — Standard Generalized Markup Language — стандартный обобщённый язык разметки. Это довольно громоздкая конструкция, из которой позднее выделилось более удобное подмножество XML — eXtensive Markup Language — расширяемый язык разметки, используемый для структуризации и разметки данных, если связь между элементами данных чем-то напоминает дерево (хотя это и не обязательно).

Для простейшего понимания достаточно двух понятий:

тег — элемент разметки, состояший из открывающей и закрывающей частей: <имя тега> ... </имя тега>, некоторые теги могут не иметь закрывающей части

атрибуты — поименованные “параметры” тега, например, <tag attr=“abc”>

XML вводит правила для использования тегов и атрибутов, которые формируют “окружение” каких-либо данных, текстов и т.п. Сами имена тегов и атрибутов не входят в спецификацию XML, а конкретизируются в зависимости от прикладных задач. Таким образом, для каждой прикладной задачи мы можем определить свой набор тегов и атрибутов и соответственно обрабатывать их, исходя из того, что нам требуется.

Язык HTML определяет свой набор тегов и атрибутов для решения задачи структурирования текстового документа и обеспечения необходимых форм и стилей его представления на экране браузера.

HTML вводит теги для разметки абзацев текста, заголовков разных уровней, списков, таблиц, структурных блоков, ссылок на другие ресурсы, изображения, видео и аудиофайлы, поля ввола, кнопки и т.д. Теги для установки типа, цвета и размера шрифтов, цвета заливок, фонов, рамок и других изобразительных стилей представления документа

Описание тегов HTML и их атрибутов можно найти в справочной литературе или в интернете.

Есть неколько версий HTML, а также расширений для разных браузеров. Есть проблема совместимости и различия результатов отображения. Т.е. каждый конкретный браузер интерпретирует теги, пытаясь следовать устоявшимся “стандартам”, но может также и действовать по своему разумению.

Вот пример простейшего HTML документа (с пояснениями).

```
<!DOCTYPE html>                          --- рекомендуемый тег типа документа
<html>                                   --- основной тег
<head>                                   --- заголовочный раздел
   <meta charset ="cp1251">              --- всякие параметры отображения
   <title>Учебный сервер</title>         --- подпись вверху браузера
</head>                                  --- конец заголовка
<body>                                   --- тело документа
   <h2>МГУ им М.В.Ломоносова</h2>        --- заголовок 2 уровня
   <p>                                   --- начало абзаца
       Просто абзац текста
   </p>                                  --- конец абзаца
</body>                                  --- конец тела документа
</html>                                  --- конец html документа
```

Можно сохранить этот текст в файл, и потом открыть в браузере и посмотреть, что и как будет выведено.

В браузерах есть пункт меню типа “исходный код страницы” (обычно под пунктом “web разработка” или типа того). Можно посмотреть код отображаемой страницы. Однако это не всегда продуктивно, так как сейчас большинство страниц генерируется специализированным программным обеспечением и обычно оказывается весьма громоздким и запутанным, поэтому разобраться в тексте очень сложно. Но для простых (своих) примеров это может оказаться весьма полезно, особенно если браузер отображает не совсем то, что вы задумали.

Довольно давно стало понятно, что сама по себе разметка HTML не позволяет решить многие насущные задачи или сильно усложняет их решение. В частности, особенности отображения при разных условиях (разные браузеры, разная аппаратура) и добавление интерактивности, когда пользователь может активно управлять отображением на стороне клиента. Решение этих проблем вылилось в создание Javascript и CSS.

Javascript — это язык программирования, который может исполняться на стороне сервера или на стороне клиента с тем, чтобы что-то изменить в исходном коде страницы. Код Javascript (или ссылки на файлы с кодом) записывается непосредственно в текст HTML и выполняется в разные моменты времени в соответствии с указанным сценарием работы. Объектами Javascript являются как обычные объекты типа числовых переменных или строк, так и структурные элементы текста, имеющие атрибут name в своем теге (по этому имени к ним можно и обратиться из скриптов). При выполнении код Javascript может изменить атрибуты объектов, изменить код HTML, произвести разнообразные вычисления и т.д. В результате получится другой документ, который и будет в результате отображен. Кроме всего прочего, Javascript является кодом, управляемым событиями, т.е. выполнение отдельных функций привязано к возникновению тех или иных событий, например, перемещению мышки, нажатию кнопок, загрузке или выгрузке документа или его частей и т.п. Таким образом, при перемещении курсора мыши или нажатии на элементы текста мы видим изменения (появляние новых объектов, смена стиля отображения и т.п.).

CSS — Cascading Style Sheets — каскадные таблицы стилей — это технология, состоящая в отделении стилевых параметров отображения (цвет, форма, размер, взаимное расположение и т.д.) размеченных объектов от их информационного содержания. Если в начальной идеологии HTML программист мог указывать стилевые параметры непосредственно в атрибутах тега, то CSS позволяет создать “таблицы стилей”, которые затем можно применять в целом к указанным тегам или группам тегов. Таким образом, при необходимости изменить стиль отображения ранее нам надо было переписывать все атрибуты во всех размеченных элементах текста, а с CSS будет достаточно просто заменить таблицы стилей (или проще — дать ссылки на другие таблицы). CSS определяет иерархическую систему подобных таблиц, что позволяет достаточно эффективно и разнообразно применять стилевые параметры в разных ситуациях.

Простейший пример

```
<style>
p { max-width:50em; background:#00ff00; }
</style>
```

Такое описание задает параметры тегу абзаца текста <p>, теперь любой абзац будет отображаться не более заданной ширины и на зеленом фоне.

JSON — язык задания параметров. В первом приближении JSON позволяет задавать набор параметров в виде множества записей вида `<имя>=<значение>`. Ситуация часто состоит в том, что финальный код HTML документа очень велик, хотя по существу он определяется относительно небольшим количеством содержательных данных. То есть большую часть текста занимают именно конструкции HTML разметки, которые, например, многократно повторяются как в случае больших списков или таблиц. Передача подобного файла по сети может занимать много времени. Поэтому можно передать необходимые данные (в форме JSON) и код генератора текста (например, Javascript). Теперь на стороне клиента можно запустить скрипт и сгенерировать необходимый HTML текст, а сетевой траффик при этом сокращается.

Все эти технологии развивались годами и весьма объемны. Поэтому мы о них здесь больше говорить не будем, а перейдем к простейшему модельному примеру.

модельный HTTP сервер

Задача — реализовать примитивный HTTP сервер, к которому можно было бы обращаться от обычного интернет-браузера. Таким образом, сервер должен передавать браузеру некоторый HTML текст, предваряя его соответствующим HTTP заголовком. В свою очередь, браузер, выполняя функции клиента, должен каким-то образом получать от пользователя требуемы данные и также формировать нужный HTTP запрос к серверу.

Проблемы:

1. Сложность всесторонней поддержки HTTP протокола, множество технических тонкостей.

2. Представление результата в форме HTML документа. Опять же, HTML имеет множество разноообразных средств для представления данных.

3. Логика работы конкретного браузера с HTTP протоколом (параллельные соединения, упреждающие запросы, служебные запросы и т.д.)

К счастью, сам HTTP устроен так, что общение возможно и при самой примитивной реализации сервера. Основой этого является концепция “рекомендаций” и самостоятельного принятия решений, если что-то оказалось непонятным.

Задача сервера — понять, что от него хочет клиент (браузер).

Задача клиента — принять данные для запроса от пользователя и отправить их на сервер.

Начнем с клиента.

Клиентская часть

На стороне клиента (браузера) мы имеем только отображаемый HTML документ. Всю поддержку HTTP выполняет браузер, и нам надо только воспользоваться имеющимися в нашем распоряжении возможностями.

Для отправки запроса на сервер у нас есть две основных возможности:

1. Адресная строка браузера. В эту строку мы можем записать URL, который далее будет отправлен как составная часть запроса по методу GET на сервер. Например, мы можем там написать

```
localhost:5555/index.html
```

Тогда на сервер, запущенные на TCP порт 5555, придет запрос, начинающийся примерно так

```
GET  /index.html HTTP/1.1
Host:  localhost:5555
.....
```

Если набрать в адресной строке

```
localhost:5555
```

То получим запрос

```
GET  / HTTP/1.1
Host:  localhost:5555
.....
```

и т.п. Можно распечатывать на сервере полученные запросы и смотреть как они выглядят в разных ситуациях.

2. Использование специального тега form, которые содержит поля ввода и отправляет на сервер значения этих полей при возникновении события submit (например,

при нажатии соответствующей кнопки). Событие submit формирует HTTP запрос и отправляет его в соответствии с атирибутами action и method, описанными в данном теге form. Напомним, что метод get передает параметры в первой строке запроса, а метод post отправляет параметры из формы в рамках entity body запроса.

Далее мы обсуждаем конкретные примеры, построенные на основе уже хорошо знакомого TCP сервера на базе select. Пока, не вдаваясь в логику построения сервера, можно просто посмотреть как он работает. Запустите сервер, откройте браузер, и наберите в адресной строке `localhost:5555`. Сервер пришлет вот такой HTML текст

```
<!DOCTYPE html>
<html>
<head>
   <meta charset ="cp1251">
   <title>Учебный сервер</title>
</head>
<body>
   <h2>МГУ им М.В.Ломоносова</h2>
   <form name="myform" action="http://localhost:5555"  method="get">
     <input name="request" type="text" width="20" value=""> request <br>
     <input name="comment" type="text" width="20" value="some text"> comment <br>
     1 <input name="rb1" type="radio" value="check1">
     2 <input name="rb1" type="radio" value="check2" checked>
     3 <input name="rb2" type="radio" value="check3">
     4 <input name="rb2" type="radio" value="check4">
     <br>
     A <input name="ch1" type="checkbox" value="checkA">
     B <input name="ch2" type="checkbox" checked>
     C <input name="ch3" type="checkbox">
     <br>
     <input name="sub" type="submit" value="Send">
    </form>
</body>
</html>
```

Этот текст содержит форму с несколькими полями ввода разных типов. Набирайте в полях разные данные. Отмечайте разные кнопочки и нажимайте кнопку Send. Как видно из текста, кнопка Send порождается здесь последним полем ввода, имеющим тип submit. Таким образом, нажатие этой кнопки создает событие submit, и на сервер отправляется запрос с данными, соответствующими полям данной формы и имеющимися там в данным момент значениями. Внимательно рассматривая полученную сервером строку запроса, можно увидеть, что введенные значения составляют список параметров запроса, а именно, мы видим записи типа

request=.....
comment=.....
rb1=check2
ch2=on

и т.п. Слова в левой части присваивания — это имена соответствующих элементов формы, а в правой части стоят значения этих элементов. Таким образом, если мы что-то набрали в полях формы, то эти данные придут на сервер в строке запроса, и их надо оттуда выделить, чтобы потом использовать.

Некоторые проблемы. Значения в строке запроса "кодируются". Так, пробелы заменяются на символ `+`, русские буквы представляются в текущей кодировке вашей системы в виде `%xx`, где `xx` — шестнадцатиричный код байта кодировки символа. Напомним, что в кодировке Windows cp1251 каждый русский символ кодируется одним байтом, а кодировке Linux unicode UTF-8 русские буквы кодируются двумя байтами. Кроме этого некоторые служебные символы (например, тот же `+`) так же кодируются шестнадцатиричным кодом в форме `%xx`. Соответственно, для декодирования

строки параметров нужно написать процедуры перекодировки, а какие символы как представляются можно узнать просто вводя их в поля формы и наблюдая как они выглядят в запросе.

Немного об элементах тега form. Подробности как всегда можно узнать из справочных пособий. Здесь мы использовали следующее.

```
тег       атрибут
form      action    --- URL куда посылать запрос с данными из формы
form      methog    --- get или post (какой запрос формировать)
input     type      --- поле ввода и его тип
                        есть разные типы - text, radio, checkbox, button и т.д.
                        соответственно для ввода текста, выбора одного из нескольких,
                        выбора нескольких, формирования кнопки и т.д.
input     name      --- имя, которое появится в строке запроса со своим значением
input     value     --- значение или состояние, которое имеет данный элемент
input     type=submit  --- специальная кнопка, которая инициирует отправление
                        запроса на сервер
input     checked   --- отметка, что данный элемент выбран
```

Серверная часть

Здесь идейно повторяется все как было раньше — сервер получает от клиента текст запроса и отправлет ему текст ответа. Особенносью является только то, что текст запроса и текст ответа должны быть пакетами, удовлетворяющими спецификациям HTTP, т.е. в общем случае имеют заголовок и область с содержательными данными.

Таким образом, сервер должен сначала выделить из запроса содержательные параметры, принять их во внимание, а потом сформировать ответ с корректным HTTP заголовком и требуемой информационной частью.

Рассмотрим сначала анализ запроса. Во-первых, браузер может запрашивать данные, о которых мы не имеем понятия (мы не погружаемся в глубины HTTP). В таких случаях нам достаточно сформировать ответ `HTTP/1.1 404 Not found`, а дальше браузер уж пусть сам разбирается как поступать при отсутствии таких данных. Ярким примерм такого поведения является напрос на получения файла favicon.ico. Эта иконка используется для отображения в закладке браузера, назначенной нашей страничке. Конечно, вы можете создать такую иконку (в интернете есть для этого генераторы), но можно и отказать браузеру, тогда он нарисует что-то по умолчанию. Браузер может попросить и что-нибудь другое.

Отделить содержательный с нашей точки зрения запрос от бессодержательного можно, например, при наличии в запросе слова request, которое является именем первого поля ввода в нашей форме. Но, кроме того, может появиться и пустой запрос типа `GET / HTTP/1.1`, на который мы тоже должны как-то отреагировать, например, послать страничку по умолчанию.

Нужно ли анализировать другие строки заголовка? В данном примере не надо. Единственное исключение — это строка Connection, которая может содержать два из возможных значений — close или keep alive. Т.е. браузер просит либо закрыть соединение после обработки запроса, либо сохранить его. Обычно мы имеем Connection : keep alive. Соответственно такая же строка должа появиться и в ответе.

После того как мы разобрались с запросом, надо сформировать ответ. Ответ состоит из двух частей — HTML текста и HTTP заголовка. В нашем примере мы формируем эти части ответа с помощью класса stringstream, который позволяет последовательно наращивать текст добавлением новых строк или слов. Необходимость накапливать текст и заголовок по отдельность вызвана тем, что в заголовке мы должны указать длину текста (строка Content-length). Поэтому заголовок удобнее формировать уже после формирования всего текста.

Итак, работа сервера состоит из следующих шагов:

— получили запрос

— проанализировали запрос и выбрали параметры запроса

— для непонятного запроса сформировали ответ 404 Not found и отправили его

— сформировали ответ HTML: содержательный текст, соответствующий параметрам запроса

— вычислили длину текста HTML

— сформировали заголовок HTTP с указанием длины Content-length, также можем добавить другие понятные нам строки в заголовок

— отправили заголовок HTTP

— отправили текст HTML

Приведенные примеры содержать практически одинаковые реализации под Windows и Linux.