Лекция 6. HTTP сервер

Здесь мы попробуем прикоснуться к огромной области программирования, связанной с поддержкой различных интернет сервисов. Естественно, нам, в основном, придется ограничиться упоминанием некоторых технологий и рассмотреть самые примитивные подходы.

Общая задача — обеспечить передачу информационных материалов с одной сетевой точки (информационной базы данных) на другие сетевые точки (клиентские станции) по запросам от этих клиентов, и также отобразить эти результаты в удобной для человека форме.

В силу общности и важности этой задачи для ее решения было разработано множество технологий, которые продолжают развиваться и в настоящее время. Упомянем некоторые из них.

TCP — базовый транспортный протокол. Этот протокол естественным образом был положен в основу необходимых реализаций, поскольку он обеспечивает надежность передачи данных.

HTML — HyperText Markup Language. Язык гипертекстовой разметки. Был создан для того, чтобы размечать компоненты передаваемой информации (web-странички) с целью их подходящего (правильного, красивого, удобного и т.п.) отображения для пользователя на клиентской машине. За время своего существования разросся до огромных объемов и породил множество дополнительных концепций и технологий, связанных с обработкой и представлением данных, передаваемых на клиентские машины.

HTTP — HyperText Transfer Protocol. Это протокол, предназначенный изначально специально для пересылки и обработки HTML документов. Интернет браузеры (их много разных) собственно и занимаются передачей и приемом HTTP пакетов, которые поддерживают стандартную идеологию работы клиентов и сервера, т.е. клиент отправляет серверу HTTP запрос на некоторые данные и получает от сервера HTTP ответ, который в общем случае содержит затребованный HTML документ. Затем браузер отображает этот документ в своих окнах в соответствии с указанной разметкой.

Этих трех компонент уже хватает, чтобы построить простейший Web сервис с нуля и “на коленке”, хотя в реальной практике, конечно, используются разнообразные инструментальные среды и типовые решения.

Помимо этого с развитием информационных и интернет технологий появились и другие инструменты для поддержки решения данной задачи. Можно упомянуть, например, Javascript, CSS, JSON и другие, о которых мы скажем здесь, разве что, пару слов.

Рассмотрим теперь эти понятия чуть более подробно.

URL — Uniform Resource Locator

Рассматриваемая нами общая задача преподагает получение информации от некоторого интернет ресурса. Таким образом, возникает проблема идентификации этого ресурса. Для этой цели было введено понятие URL как некоторой унифицированной записи, которая показывает как и откуда можно получить требуемые данные (т.е. местоположение и способ получения). В общем виде конструкцию URL можно записать в виде

```
<протокол>://<логин>:<пароль>@<хост>:<порт>/<URL-путь>?<параметры>\#<якорь>
```

Здесь

`<протокол>` — протокол, который дает возможность получить ресурс;

`<логин>:<пароль>` — идентицикация пользователя и его прав, если она предусмотрена для доступа к ресурсу;

`<хост>:<порт>` — сетевая идентификация станции, отвечающей за данный ресурс;

`<URL-путь>` — относительный путь к файлу, обеспечивающему ресурс на данной станции (например, к html файлу, или к программе, запуск которой предоставит нам ресурс);

<параметры> — параметры, которые могут быть необходимы для получения ресурса, они задаются в виде последовательности записей <имя>=<значение>;

<якорь> — метка, указывающая на отдельную часть ресурса, если он поделен на части.

Обычно не все из этих полей необходимы, к том смысле, что если они не указаны, то при работе используется некоторое значение по умолчанию. Например, для протокола значение по умолчанию — http, для пользователя — anonimous с пустым паролем, т.е. общедоступные данные, для порта — тот порт, который назначен данному сервису по умолчанию (например, 80 для http), для URL-пути — файл index.html и т.д. Тут надо понимать, что подставлять значения по умолчанию должна та программа, которая такой URL обрабатывает, т.е. если мы реализуем свой собственный HTTP сервер, то это будет нашей задачей.

HTTP — HyperText Transfer Protocol

HTTP протокол формирует пакеты двух типов — запрос (от клиента к серверу) и ответ (от сервера к клиенту). В обоих случаях данные, которые требуется переслать, снабжаются специальным заголовком, и эта пара (заголовок+данные) пересылаются средствами TCP протокола.

Заголовок HTTP пакета представляет собой набор текстовых строк, которые отделеные от последующих данных пустой строкой. Таким образом, заголовок можно легко прочитать, если его распечатать, так как он содержит обычный текст.

Первая строка в заголовке запроса указывает на то, что и как хочется получить клиенту, последующие строки задают рекомендации серверу или возможную дополнительную информацию, которую сервер мог бы использовать для формирования ответа.

Первая строка в заголовке ответа сервера содержит информацию об успешности обработки запроса, последующие строки заголовка также содержат дополнительную информацию или рекомендации, которые сервер хочет сообщить клиенту.

Данные, идущие после заголовка в запросе или ответе, могут присутсвовать или отсутствовать в зависимости от смысла выполняемых действий.

По историческим причинам текстовые строки заголовка завершаются в стиле Windows, т.е. содержат два байта — возврат каретки и перевод строки, которые мы будем обозначать <CR> <LF> (carridge return, line feed, или в обозначениях языков C/C++ как $\backslash r\backslash n$).

HTTP, формат запроса

```
метод  URL  версия <CR> <LF>                          --- request line
название заголовка : значение <CR> <LF>               --- header line
.......                                               --- header line
<CR> <LF>
entity body                                           --- данные, если надо
```

Метод — это “команда”, которую должен выполнить сервер. В нашем случае мы рассматриваем два возможных метода — GET и POST. Остальные строки задают параметры, которые сервер может принять во внимание при формировании наиболее подходящего ответа (а может и не принять и поступить по-своему). В методе GET главные параметры запроса должны быть указаны в поле URL. В этом случае длина поля URL не должна превышать 1 килобайта. Если же данные для запроса, отправляемого на сервер, занимают много места, то можно использовать метод POST, в котором эти данные передаются через поле entity body. Например, если мы ходим получить от сервера файл, расположенный по относительному пути /dir/file.html, то можно сформировать такой запрос

```
GET  /dir/file.html HTTP/1.1 <CR> <LF>                --- GET по протоколу HTTP/1.1
Host:  localhost:5555 <CR> <LF>                       --- к кому обращаемся
```

```
Connection : keep-alive <CR> <LF>                          --- или close
User-agent : Mozilla Firefox 27.0.1 <CR> <LF>              --- это наш браузер
Accept : text/html  image/gif image/jpeg <CR> <LF>         --- что ожидаем в ответе
Accept-language : ru <CR> <LF>                             --- желательный тип кодировки
<CR> <LF>
```

На самом деле подобный запрос формирует браузер, и задача сервера — правильно на него ответить.

HTTP, формат отклика

```
версия код-отклика сообщение <CR> <LF>              --- status line
название заголовка : значение <CR> <LF>             --- header line
.......                                             --- header line
<CR> <LF>
entity body                                         --- пересылаемые данные
```

Здесь полная аналогия с запросом с точностью до первой строчки. Например, ответ на предыдузий запрос мог выглядеть как-то так

```
HTTP/1.1  200 OK <CR> <LF>                                  --- все в порядке
Connection : close <CR> <LF>                                --- мы закрыли сооединение
Date : Wed, 05 Apr 2024  06:32:25 GMT <CR> <LF>             --- когда ответили
Server : Apache/1.3.0 (Unix) <CR> <LF>                      --- кто ответил
Last-modified : Mon, 27 Jan 2010  11:12:42 GMT <CR> <LF>--- когда изменялся этот файл
Content-length : 27631 <CR> <LF>                            --- длина файла в байтах
Content-type : text/html <CR> <LF>                          --- что в файле
<CR> <LF>
<содержимое файла file.html>
```

HTTP, коды отклика

Статусная строка ответа содержит версию протокола и код отклика с его текстовой расшифровкой Протокол определяет 5 категорий откликов, которые различатся первой цифрой кода

1xx — information
2xx — success
3xx — redirection
4xx — client error
5xx — server error

Вот некоторые примеры откликов

200 OK
301 Moved permanently
400 Bad request
404 Not found
505 HTTP version not supported

Соответственно клиент должен проанализировать этот код и далее поступать в соответствеии с логикой своей работы.

Здесь мы более не будем погружаться в глубины HTTP.

Понятие о HTML

Исторически праролителем 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 ввода, 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, местоположение 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 отравлен 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.