ИСТОРИЯ WWW

1989 — работники CERN Тим БЛ и Роберт К предложили проект «Всемирная паутина» (World Wide Web): публикация гипертекстовых документов (документов, связанных между собой ссылками).

Основа сервиса: URL, HTTP, HTML.

Первый веб-сервер: httpd.

Первый браузер: WorldWideWeb (под NeXTStep).

Первый веб-сайт: http://info.cern.ch (6.08.1991)

Tim Berners-Lee

Robert Cailliau

HTTP = HyperText Transfer Protocol

HTTP v.0.9 - январь 1992 г. (1-я реализация — 1990)

HTTP v.1.0 - RFC 1945 (май 1996 г.)

HTTP v.1.1 - RFC 2068 (январь 1997 г.),
RFC 2616 (июнь 1999 г.)

W3C = World Wide Web Consortium (www.w3c.org)

ОСОБЕННОСТИ HTTP

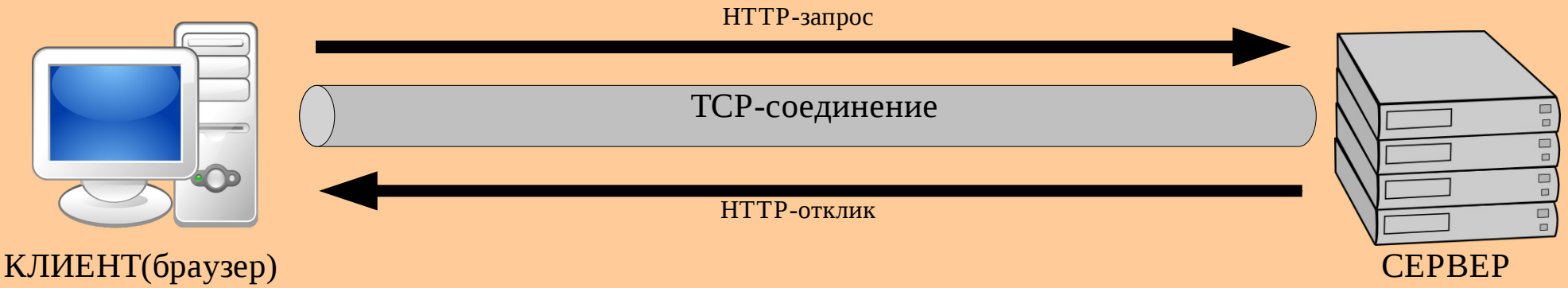
* Для транспорта используется TCP. Порт сервера по умолчанию — 80.

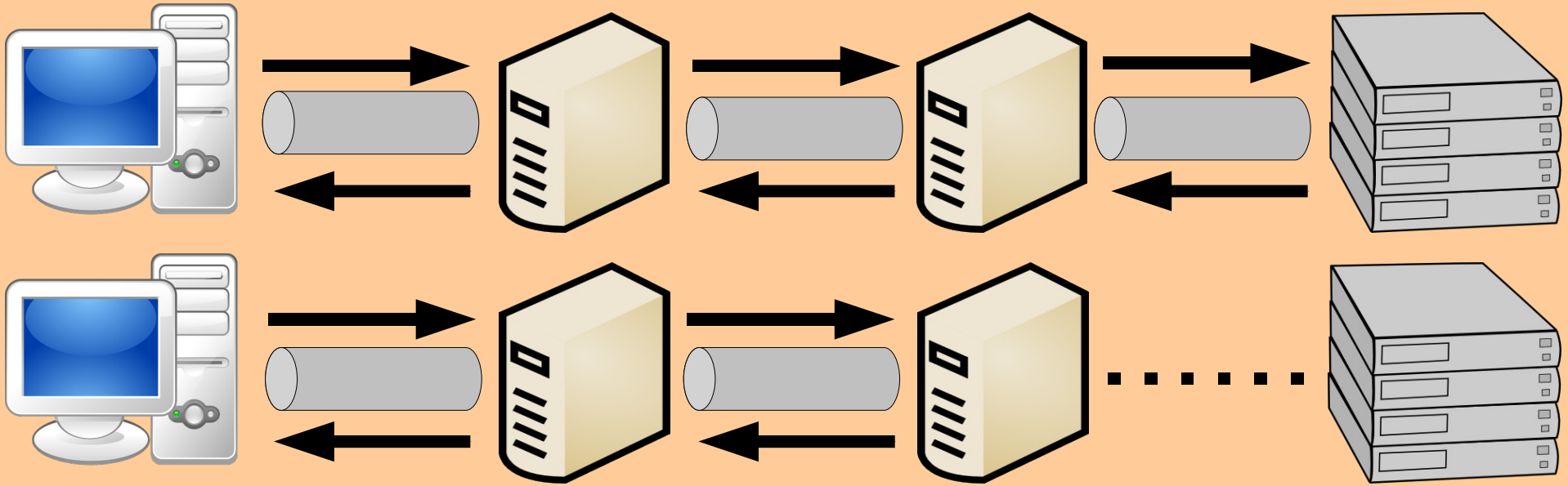
* Протокол ориентирован на транзакции. Одна транзакция состоит из запроса клиента и отклика сервера. Транзакции независимы (stateless protocol). Традиционный подход: одна транзакция — одно соединение.

* Запросы и отклики кодируются в текстовом виде, однако контент может быть в любом формате.

* В своём запросе клиент клиент указывает идентификатор запрашиваемого ресурса (документа) - URI.

ПРОМЕЖУТОЧНЫЕ СИСТЕМЫ





КЛАССЫ ПРОМЕЖУТОЧНЫХ СИСТЕМ

Прокси

Получает HTTP-запрос с абсолютным URI, анализирует (переформатирует) запрос и перенаправляет новый HTTP-запрос на сервер по указанному URI.

Назначение: кэш, контроль доступа, фильтрация контента, ...

Шлюз

Получает HTTP-запрос, анализирует запрос и транслирует в другой запрос (возможен другой протокол).

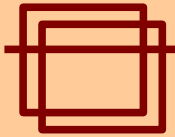
Назначение: использование HTTP в качестве транспорта (туннеля) для другого протокола (ICQ, Kerberos, LDAP, обращение к БД, ...)

Туннель

Ретранслятор. Передаёт HTTP-сообщение, не анализируя.

Назначение: преодоление межсетевого экрана, доступ к сервисам при отсутствии маршрутизации между сетями, ...

URI, URL, URN



URI — Uniform Resource Identifier (RFC 3986 — январь 2005)

Унифицированный идентификатор ресурса (URI): ресурс можно идентифицировать либо по расположению (**Uniform Resource Locator**), либо по названию (**Uniform Resource Name**).

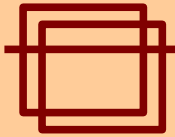
```
scheme : hier-part ? query # fragment
```

Вариант hier-part: `// userinfo @ host : port path`

(необязательные части: query, fragment, userinfo, port, path)

Регистрация схем URI курируется IANA:
http://www.iana.org/assignments/uri-schemes

ПРИМЕРЫ URI



```
ftp://user:password@ftp.is.co.za/movies/example.avi

http://tools.ietf.org/html/rfc3986#section-3.1

ldap://[2001:db8::7]/c=GB?objectClass?one

tel:+1-816-555-1212

mailto:John.Doe@example.com?subject=Invoice

telnet://192.0.2.16:80/

urn:oasis:names:specification:docbook:dtd:xml:4.1.2

http://www.cnn.com&story=breaking_news@10.0.0.1/top_s
tory.htm
```

(семантическая атака)

URL-КОДИРОВАНИЕ

Зарезервированные символы:

```
: / ? # [ ] @
! $ & ' ( ) * + , ; = %
```

Разрешённые символы:

```
A-Z a-z 0-9 - . _ ~
```

Кодирование неразрешённого символа:

%hex

Примеры:

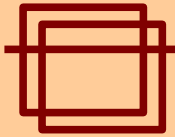
http://www.example.com/test.php?a=%28foo%29&b=35+%25

a=(foo) b=35 %

ftp://download.tv/~user/pub/Some%20Artist%21-%22Song%22.mp3

/~user/pub/Some Artist!-"Song".mp3

URN



URN — Uniform Resource Name (RFC 2141 — май 1997)

Идентификация ресурса, независящая от его расположения.

`urn : `*`NID`*` : `*`NSS`*

NID — идентификатор пространства имён (namespace identifier)
http://www.iana.org/assignments/urn-namespaces
ietf, issn, isbn, oid, oasis, ...

NSS — строка, описывающая ресурс (namespace specific string)

Примеры:
urn:ietf:rfc:3986
urn:ietf:std:50
urn:isbn:978-5-8021-0632-7

СТРУКТУРА HTTP-ТРАНЗАКЦИИ

1. HTTP-клиент устанавливает соединение с сервером и посылает ему сообщение-запрос:

```
method uri HTTP/x.x        - строка запроса
header: value              - может отсутствовать
header: value              - может отсутствовать

body                       - может отсутствовать
```

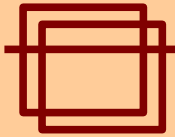
2. В ответ сервер посылает сообщение-отклик, обычно содержащее запрошенный документ:

```
HTTP/x.x 999 reason        - строка статуса
header: value
header: value

body
```

IMF — RFC 5322 (RFC 822)

HTTP-ЗАПРОС



Методы:

OPTIONS	- информация о возможностях (виды запросов, параметры)
GET	- получить ресурс
HEAD	- то же, что GET, но без тела (только заголовки)
POST	- передача информации на сервер (отправка данных формы)
PUT	- передача информации на сервер (отправка ресурса)
DELETE	- удаление ресурса на сервере
TRACE	- «эхо» (посылает обратно запрос клиента)
CONNECT	- туннелирование какого-либо TCP-протокола внутри HTTP

URI: *, абсолютный URI, абсолютный путь, hier-part

Примеры:

```
OPTIONS * HTTP/1.0
GET http://ya.ru HTTP/0.9
HEAD /abc/1.html HTTP/1.0
CONNECT icq.aol.com:443 HTTP/1.0
```

HTTP-ОТКЛИК

Статусы откликов:

1xx — информационный (запрос получен, продолжает обрабатываться)
2xx — успешный (запрос успешно выполнен)
3xx — перенаправление (клиенту необходимо выполнить дополнительный запрос, чтобы получить запрошенный ресурс)
4xx — ошибка клиента (неправильный синтаксис или неподходящие условия запроса)
5xx — ошибка сервера (запрос не выполнен по вине сервера)

Примеры:

HTTP/1.1 100 Continue
HTTP/1.0 200 OK
HTTP/1.0 301 Moved Permanently
HTTP/1.0 400 Bad Request
HTTP/1.0 401 Unauthorized
HTTP/0.9 404 Not Found
HTTP/1.0 500 Internal Server Error

ЗАГОЛОВКИ HTTP-СООБЩЕНИЯ

В запросе могут быть:
Accept (MIME-тип), **Accept-Charset** (кодировка), **Accept-Encoding** (сжатие), **Accept-Language** (язык), **Authorization** (авторизация), **From** (email), **Host** (доменное имя), **If-***xxxx* (предварительное условие), **Proxy-Authorization** (авторизация), **Range** (докачка), **Referer** (обратная ссылка), **TE** (способ кодирования), **User-Agent** (браузер), ...

В отклике могут быть:
Accept-Ranges (поддержка докачки), **Age** (время жизни в сек), **ETag** (отпечаток), **Location** (перенаправление), **Proxy-Authenticate** (аутентификация), **Retry-After** (сек), **Server**, **WWW-Authenticate** (аутентификация), ...

В HTTP/1.0 определены 16 заголовков (нет обязательных), в HTTP/1.1 — 46 заголовков (Host — обязательный).

Если HTTP-сообщение содержит тело, используются MIME-заголовки **Content-Type**, **Content-Length** и т. п.

ИДЕМПОТЕНТНЫЕ МЕТОДЫ

GET, HEAD, PUT, DELETE
OPTIONS, TRACE

Эффект от N идентичных запросов с помощью идемпотентного метода такой же, как от одного такого запроса.
Результат запроса с помощью идемпотентного метода может быть кэширован.

Отправка данных формы — метод GET (идемпотентный):

```
GET /path/script.cgi?name=John+Doe&email=john%40example.com HTTP/1.0
```

Отправка данных формы — метод POST (неидемпотентный):

```
POST /path/script.cgi
Content-Type: application/x-www-form-urlencoded
Content-Length: 38

name=John+Doe&email=john%40example.com
```

ОСОБЕННОСТИ HTTP/1.1

Постоянные соединения (persistent connections)

Во время одного сеанса (TCP-соединения) выполняется несколько транзакций. Управляется заголовком Connection. Если этот заголовок отсутствует, соединение не закрывается. Для завершения сеанса в последней транзакции клиент указывает:

```
Connection: close
```

Поддержка виртуальных хостов (virtual hosts)

Широко распространена практика назначения одному адресу нескольких доменных имён. Один физический сервер обслуживает несколько «виртуальных» доменов. Для идентификации виртуального хоста, к которому относится запрос, используется заголовок Host (обязательный):

```
Host: www.domain.com
```

Информационные статусы 1xx

На медленных соединениях сервер может сгенерировать промежуточный отклик, подтверждающий получение запроса от клиента:

```
HTTP/1.1 100 Continue
```

ОСОБЕННОСТИ HTTP/1.1

Кусочное кодирование (chunked transfer encoding)

При выдаче динамически генерируемых страниц сервер может начать формировать отклик, не зная полного размера страницы, используя схему кусочного кодирования. Тело отклика состоит из «кусков»: размер в hex-виде, CRLF, содержимое куска, CRLF. Последний кусок содержит размер 0, за ним могут следовать дополнительные заголовки.

```
HTTP/1.1 200 OK
Content-Type: text/plain
Transfer-Encoding: chunked

1a; ignore-stuff-here
abcdefghijklmnopqrstuvwxyz
10
1234567890abcdef
0
some-footer: some-value
another-footer: another-value
[пустая строка]
```

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 42
some-footer: some-value
another-footer: another-value

abcdefghijklmnopqrstuvwxyz1234
567890abcdef
```

ОСОБЕННОСТИ HTTP/1.1

Сжатие HTTP-трафика

Клиент, поддерживающий сжатие трафика, указывает в заголовоке Accept-Encoding допустимые типы сжатия (например, gzip, compress, deflate):

```
Accept-Encoding: compress, gzip
```

Сервер в заголовках отклика указывает выбранный способ сжатия:

```
Content-Encoding: gzip
```

Условные запросы (preconditions)

Чтобы не загружать неизменившийся документ, можно использовать заголовки If-Modified-Since, If-Unmodified-Since (условие — дата) или If-Match, If-None-Match (условие — отпечаток):

```
If-Modified-Since: Sun, 22 Aug 2010 10:00:00 GMT
If-None-Match: "4c8fe-1d6-48e6c307db700"
```

Ресурс высылается сервером только если условие истинно. В противном случае генерируется отклик:

```
HTTP/1.1 304 Not Modified
```

УПРАВЛЕНИЕ КЭШИРОВАНИЕМ

Кэширование может осуществлятся:
* на стороне сервера — кэширование динамически сгенерированных страниц;
* на промежуточных узлах (прокси);
* на стороне клиента (запрошенные документы хранятся на диске или в оперативной памяти).

Сервер в HTTP-отклике передаёт информацию, которая может использоваться для управления кэшированием:

`Date` - дата генерации данного HTTP-сообщения
`Last-Modified` - дата последней модификации документа
`Expires` - ожидаемая дата окончания валидности контента
`Age` - ожидаемое время валидности контента
`ETag` - «отпечаток» контента
`Cache-Control` - управление кэшированием (no-cache, max-age, min-fresh, ...)
`Pragma` (HTTP/1.0)

HTML-документ может содержать тег META:

```
<META HTTP-EQUIV="Cache-Control" CONTENT="max-age=400">
```

ВЗАИМОДЕЙСТВИЕ С WEB-ПРИЛОЖЕНИЯМИ

CGI — Common Gateway Interface v.1.1 (RFC 3875 — окт. 2004)

HTTP-сервер, получив запрос, на основе URI определяет приложение (CGI-скрипт) и транслирует HTTP-запрос в CGI-запрос. CGI-отклик приложения транслируется сервером в HTTP-отклик.

CGI-запрос состоит из мета-переменных (переменные среды окружения) и тела сообщения (передаётся на стандартный ввод приложения). Примеры мета-переменных: AUTH_TYPE, CONTENT-LENGTH, CONTENT_TYPE, GATEWAY_INTERFACE, PATH_INFO, QUERY_STRING, REMOTE_ADDR, REQUEST_METHOD, ...

CGI-отклик выдаётся приложением на стандартный вывод и должен быть непустым. Отклик состоит из заголовков, описывающих контент, и собственно контента, отделяемого пустой строкой.

NPH (Non-Parsed Header) скрипты генерируют на стандартном выводе полный HTTP-отклик.

ВЗАИМОДЕЙСТВИЕ С WEB-ПРИЛОЖЕНИЯМИ

Недостатки CGI

* старт нового процесса — существенная задержка
* интерпретируемые языки

Альтернативы

* встраиваемые в веб-сервер модули (скрипт исполняется в контексте процесса веб-сервера);
* между приложением и веб-сервером клиент-серверное взаимодействие (Simple-CGI, FastCGI);
* серверы приложений (Java EE — GlassFish, JBoss, WebSphere, ...)

ПРИМЕР: CGI В ДЕЙСТВИИ


```
<FORM METHOD="GET" action="/cgi-bin/vote.pl“>
<INPUT TYPE="text" NAME="N" VALUE="test">
<INPUT TYPE="submit" NAME="S" VALUE="Vote">
</FORM>
```

HTML-документ

```
GET /cgi-bin/vote.pl?N=test&S=Vote HTTP/1.1
Host: www.server.ru
User-Agent: Mozilla/1.0
Connection: close
[пустая строка]
```

HTTP-запрос

```
set environment (REQUEST_METHOD, ...)
execute "perl vote.pl"
```

Веб-сервер

ПРИМЕР: CGI В ДЕЙСТВИИ

```
#!/usr/bin/perl
if ($ENV{'REQUEST_METHOD'} eq "POST") {
   read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
}
elsif ($ENV{'REQUEST_METHOD'} eq "GET") {
   $buffer = $ENV{'QUERY_STRING'};
} . . .
```

CGI-скрипт

```
Content-type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
 <HEAD>
  <META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
 </HEAD>
<BODY>Thank you . . .</BODY>
</HTML>
```

CGI-отклик

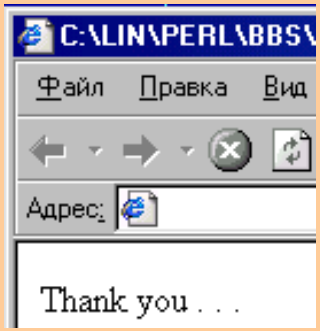
ПРИМЕР: CGI В ДЕЙСТВИИ

```
HTTP/1.1 200 OK
Content-Length: 1354
Content-type: text/html
Connection: close

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">...
```

HTTP-отклик

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
 <HEAD>
  <META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
 </HEAD>
<BODY>Thank you . . .</BODY>
</HTML>
```



HTML-документ

РАСШИРЕНИЯ HTTP

HTTP over TLS — HTTPS (RFC 2818 — май 2000)

* выделенный порт — 443;
* взаимная аутентификация;
* проблема виртуальных хостов;
* альтернатива — S-HTTP (RFC 2660 — август 1999) — не прижилась...

HTTP Authentication (RFC 2617 — июнь 1999)

* схемы Basic и Digest

HTTP State Management Mechanism (RFC 2965 — октябрь 2000)

* куки (заголовки Cookie и Set-Cookie)

Delta encoding (RFC 3229 — январь 2002)

* дельта-кодирование

WebDAV (RFC 4918 — июнь 2007)

* новые методы, обеспечивающие доступ и управление объектами и коллекциями (распределённая ФС, система управления версиями и т.п.)