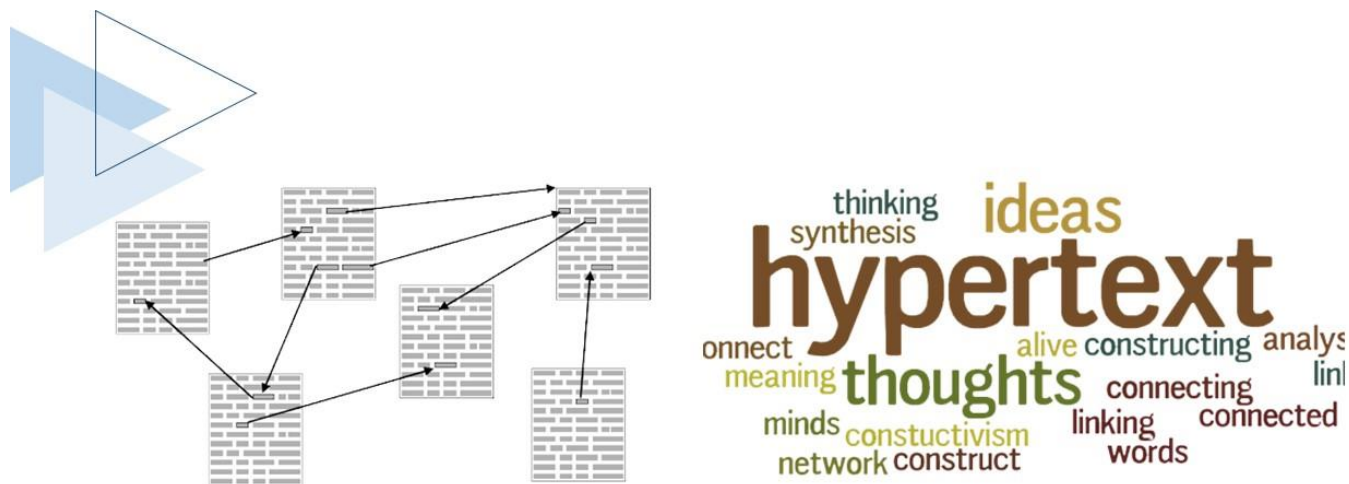


Организация ссылок очень напоминает цитирование и вставку сносок в обычных печатных документах. Видимо, именно это и послужило вдохновением при создании первых гипертекстовых страниц.



Гипертекст – это система текстовых страниц, соединенных между собой ссылками. Практически все сайты в интернете представляют собой гипертексты. Этот термин также означает текст, созданный с помощью языка разметки и рассчитанный на использование гиперссылок.

Из чего состоит гипертекст

Гипертекстом можно назвать любой HTML-документ - HTML – это аббревиатура, которая расшифровывается как HyperText Markup Language или в переводе на русский язык «Язык Разметки Гипертекста». Его содержимое имеет две составляющие – основную информацию, то бишь просто текст, и теги для разметки. С их помощью материал в дальнейшем обретает более структурированный и в некоторой степени аккуратный вид.

А вот как к такому тексту привязываются фото, аудио и видео? Все просто – они хранятся отдельно и просто прикрепляются к документу с помощью специального тега.

Он позволяет пользователю создавать и структурировать разделы, параграфы, заголовки, ссылки и блоки для веб-страниц и приложений.

HTML не является языком программирования, то есть он не имеет возможности создавать динамические функции. Вместо этого он позволяет организовывать и форматировать документы, аналогично Microsoft Word.

При работе с HTML мы используем простые структуры кода (теги и атрибуты), чтобы разметить страницу веб-сайта. Например, мы можем создать абзац, поместив прилагаемый текст в исходный тег **<p>** и закрывающий **</p>**.

<p> Вот как вы добавляете абзац в HTML. **</p>**

**
****<p>** У вас может быть более одного! **</p>****</br>**

В целом, HTML — это язык разметки, который очень прост в освоении даже для начинающих в создании сайтов. Вот что вы узнаете, прочитав эту статью.

Для того, чтобы понять что такое HTML и зачем он нужен, давайте откроем любую страницу в браузере и посмотрим на ее исходный код.

Например, я открою страницу в википедии. И включу девтулз ф12. Во вкладке Elements вы увидите html страницы

Также HTML – это не язык программирования и не отвечает за выполнение логических и программных операций на странице и обработку данных. В этом можно легко убедиться, если вы отключите поддержку клиентского языка веб-программирования в браузере.

Часто новички забывают это правило и пытаются с помощью HTML заставить элемент отображаться тем или иным образом или заставить с помощью HTML выполнять какие-то действия на странице, но это не правильно. За каждое действие над элементом отвечают свои технологии.

Гипертекст.

И, наконец, последнее слово в определении языка HTML – это гиперссылка. На веб-страницах есть элемент, который делает эти страницы особенными и отличает их от обычного текста с картинками. Этот элемент - ссылки.

Ссылка – это такой элемент на странице, который делает возможным открытие другой части текущей страницы или совершенно другой страницы при клике по нему.

Приставка «гипер» означает то, что при клике на ссылку может открываться другой ресурс (страница) в сети интернет, который может располагаться на другом сервере.

HTML – это язык, который разрабатывался специально для того, чтобы создавать страницы, которые содержат гиперссылки.

Подводя итог можно сказать, что HTML – это язык, который создавался для того, чтобы помочь браузеру понять из каких частей состоит веб-страница, и какую смысловую нагрузку эти элементы несут.

Возможности гипертекста

Гипертекст был сделан специально для того, чтобы процесс работы с контентом стал более понятным и удобным, а доступ к данным упростился в разы. И вот, следуя из всего этого, можно вывести следующие возможности гипертекста:

- **Удобство.** Материал на странице логично сконструирован, а также связан с другими страницами, содержащими релевантные данные.
- **Простота.** Гипертекст полезен для пользователей и выглядит вполне себе аккуратно.
- **Возможность вставки медиа.** В материал удобно встраивать видео и аудио, а также фотки. Контент становится менее скучным и однообразным.
- **Уникальный способ запроса данных.** Чтобы получить дополнительную информацию, нужно нажать на выделенный элемент.

Повторюсь – практически все страницы в интернете до сих пор организованы подобным образом, поэтому логика гипертекста будет актуальна всегда.

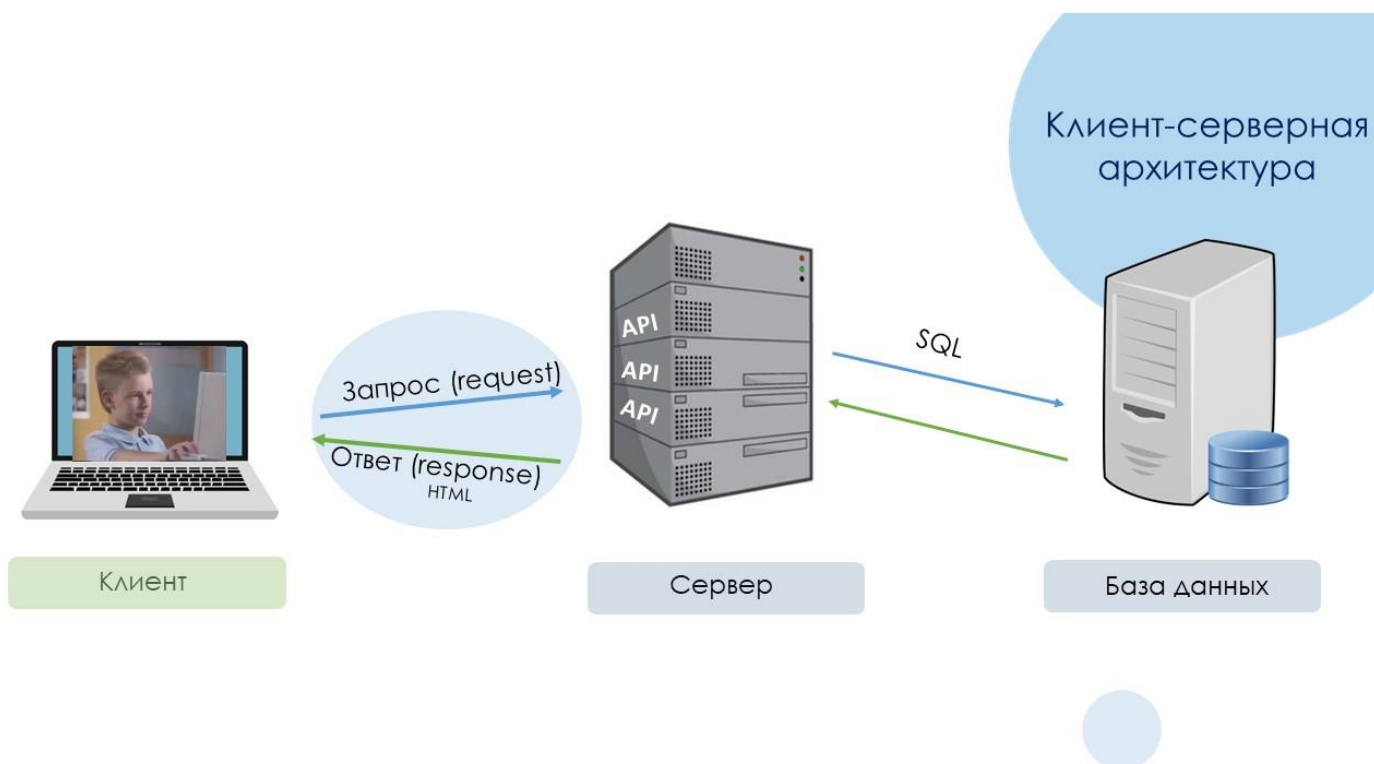
Принцип работы гипердокумента можно схематично описать следующим образом.

- Пункт назначения отмечается анкором (anchor – в слэгне британских моряков «якорь»).
- Ссылка, ведущая к анкору, по сути, представляет собой URL – универсальный локатор, адрес документа в сети интернете.
- Клик по ссылке подает во всемирную сеть команду «искать и перейти».
- После чего запрос поступает на удаленный сервер, на котором хранится запрашиваемый документ.

- Сервер отправляет данные на IP-адрес запрашивающего компьютера.
- Информация в формате HTML-файла обрабатывается в браузере и представляется в виде текст-графического документа на экране компьютера.

Почему переходы по ссылкам между связанными документами происходят так быстро?

Потому что скорость перемещения электромагнитных волн достигает что-то около 300.000 километров в секунду. С такими скоростями земной шарик выглядит совсем малюсеньким.



В соответствии со спецификацией OSI, HTTP является протоколом прикладного (верхнего, 7-го) уровня. Актуальная на данный момент версия протокола, HTTP 1.1, описана в спецификации RFC 2616.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное программное обеспечение обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту. После этого клиентское приложение может продолжить отправлять другие запросы, которые будут обработаны аналогичным образом.

Задача, которая традиционно решается с помощью протокола HTTP — обмен данными между пользовательским приложением, осуществляющим доступ к веб-ресурсам (обычно это веб-браузер) и веб-сервером. На данный момент именно благодаря протоколу HTTP обеспечивается работа Всемирной паутины.

Также HTTP часто используется как протокол передачи информации для других

протоколов прикладного уровня, таких как SOAP, XML-RPC и WebDAV. В таком случае говорят, что протокол HTTP используется как «транспорт».

API многих программных продуктов также подразумевает использование HTTP для передачи данных — сами данные при этом могут иметь любой формат, например, XML или JSON.

Как правило, передача данных по протоколу HTTP осуществляется через TCP/IP-соединения. Серверное программное обеспечение при этом обычно использует TCP-порт 80 (и, если порт не указан явно, то обычно клиентское программное обеспечение по умолчанию использует именно 80-й порт для открываемых HTTP-соединений), хотя может использовать и любой другой.

Как отправить HTTP-запрос?

1



2

```
Командная строка
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\38066>telnet baskino.me 80
```

Как отправить HTTP-запрос?

Самый простой способ разобраться с протоколом HTTP — это попробовать обратиться к какому-нибудь веб-ресурсу вручную. Представьте, что вы браузер, и у вас есть пользователь, который очень хочет посмотреть фильм. Предположим, что он ввёл в адресной строке следующее:

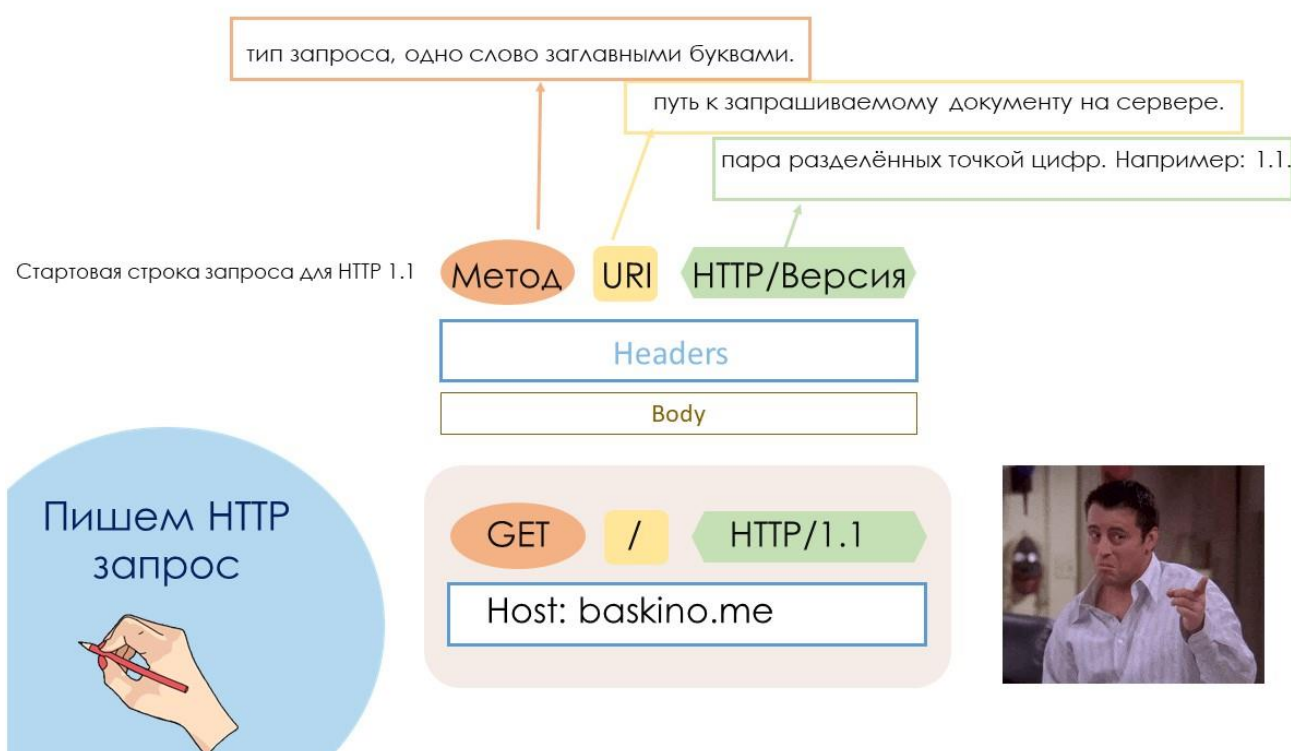
`http://baskino.me/`

Соответственно вам, как веб-браузеру, теперь необходимо подключиться к веб-серверу по адресу `baskino.me`

Для этого вы можете воспользоваться любой подходящей утилитой командной строки. Например, telnet:

```
telnet baskino.me 80
```

Сразу уточню, что если вы вдруг передумаете, то нажмите Ctrl + «]», и затем ввод — это позволит вам закрыть HTTP-соединение. Помимо telnet можете попробовать nc (или ncat) — по вкусу.



После того, как вы подключитесь к серверу, нужно отправить HTTP-запрос. Это, кстати, очень легко — HTTP-запросы могут состоять всего из двух строчек.

Для того, чтобы сформировать HTTP-запрос, необходимо составить стартовую строку, а также задать по крайней мере один заголовок — это заголовок Host, который является обязательным, и должен присутствовать в каждом запросе. Дело в том, что преобразование доменного имени в IP-адрес осуществляется на стороне клиента, и, соответственно, когда вы открываете TCP-соединение, то удалённый сервер не обладает никакой информацией о том, какой именно адрес использовался для соединения: это мог быть, например, адрес адрес film.baskino.me, baskino.me или m.baskino.me — и во всех этих случаях ответ может отличаться. Однако фактически сетевое соединение во всех случаях открывается с узлом 212.24.43.44, и даже если первоначально при открытии соединения был задан не этот IP-адрес, а какое-либо доменное имя, то сервер об этом никак не информируется — и именно поэтому этот адрес необходимо передать в заголовке Host.

Заголовки – это специальные параметры, которые несут определенную служебную информацию о соединении по HTTP. Некоторые заголовки имеют лишь информационный характер для пользователя или для компьютера, другие передают определенные команды, исходя из которых, сервер или клиент будет выполнять какие-то действия.

Стартовая (начальная) строка запроса для HTTP 1.1 составляется по следующей схеме:

Метод URI HTTP/Версия

Например (такая стартовая строка может указывать на то, что запрашивается главная страница сайта):

GET / HTTP/1.1

Метод (в англоязычной тематической литературе используется слово *method*, а также иногда слово *verb* — «глагол») представляет собой последовательность из любых символов, кроме управляющих и разделителей, и определяет операцию, которую нужно осуществить с указанным ресурсом. Спецификация HTTP 1.1 не ограничивает количество разных методов, которые могут быть использованы, однако в целях соответствия общим стандартам и сохранения совместимости с максимально широким спектром программного обеспечения как правило используются лишь некоторые, наиболее стандартные методы, смысл которых однозначно раскрыт в спецификации протокола.

URI (*Uniform Resource Identifier*, унифицированный идентификатор ресурса) — путь до конкретного ресурса (например, документа), над которым необходимо осуществить операцию (например, в случае использования метода GET подразумевается получение ресурса). Некоторые запросы могут не относиться к какому-либо ресурсу, в этом случае вместо URI в стартовую строку может быть добавлена звёздочка (астериск, символ «*»). Например, это может быть запрос, который относится к самому веб-серверу, а не какому-либо конкретному ресурсу. В этом случае стартовая строка может выглядеть так:

OPTIONS * HTTP/1.1

Версия определяет, в соответствии с какой версией стандарта HTTP составлен запрос. Указывается как два числа, разделённых точкой (например **1.1**).

Для того, чтобы обратиться к веб-странице по определённом адресу (в данном случае путь к ресурсу — это «/»), нам следует отправить следующий запрос:
Здесь:

GET / HTTP/1.1
Host: baskino.me

При этом учитывайте, что для переноса строки следует использовать символ возврата каретки (Carriage Return), за которым следует символ перевода строки (Line Feed). После объявления последнего заголовка последовательность символов для переноса строки добавляется дважды.

Впрочем, в спецификации HTTP рекомендуется программировать HTTP-сервер таким образом, чтобы при обработке запросов в качестве межстрочного разделителя воспринимался символ LF, а предшествующий символ CR, при наличии такового, игнорировался. Соответственно, на практике бо́льшая часть серверов корректно обработает и такой запрос, где заголовки отделены символом LF, и он же дважды добавлен после объявления последнего заголовка.

Если вы хотите отправить запрос в точном соответствии со спецификацией, можете воспользоваться управляющими последовательностями \r и \n:

```
echo -en "GET / HTTP/1.1\r\nHost: alizar.habrahabr.ru\r\n\r\n" | ncat alizar.habrahabr.ru 80
```

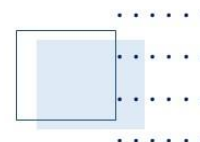
Методы



CRUD

Create - создание
Read - чтение
Update - обновление
Delete - удаление

GET	Метод GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.
HEAD	Запрашивает ресурс так же, как и метод GET, но без тела ответа.
POST	Используется для отправки сущностей к определённому ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.
PUT	Заменяет все текущие представления ресурса данными запроса.
DELETE	Удаляет указанный ресурс.
OPTIONS	Используется для описания параметров соединения с ресурсом.
PATCH	Используется для частичного изменения ресурса.



GET

Читать - Read



GET

Сервер

Пример запроса

GET /blog/?name1=value1&name2=value2 HTTP/1.1

Host: htmlacademy.ru

Метод GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

Метод GET запрашивает представление ресурса. Запросы с использованием этого метода могут только извлекать данные.

Согласно стандарту HTTP, запросы типа GET считаются идемпотентными — многократное повторение одного и того же запроса GET должно приводить к одинаковым результатам (при условии, что сам ресурс не изменился за время между запросами). Это позволяет кэшировать ответы на запросы GET.

HEAD

Читать - Read



HEAD

Сервер

HEAD /text.txt HTTP/1.1
Host: example.com

HEAD запрашивает ресурс так же, как и метод GET, но без тела ответа.

HEAD

Аналогичен методу GET, за исключением того, что в ответе сервера отсутствует тело. Запрос HEAD обычно применяется для извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше копия ресурса помечается как устаревшая.

Каждый сервер обязан поддерживать как минимум методы GET и HEAD. Если сервер не распознал указанный клиентом метод, то он должен вернуть статус 501 (Not Implemented). Если серверу метод известен, но он не применим к конкретному ресурсу, то возвращается сообщение с кодом 405 (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок Allow со списком поддерживаемых методов.

Наиболее востребованными являются методы GET и POST — на человеко-ориентированных ресурсах, POST — роботами поисковых машин и оффлайн-браузерами.

POST /blog/ HTTP/1.1

Host: htmlacademy.ru


name1=value1&name2=value2

POST

CREATE

С

Create - создание
Update - обновление
Delete - удаление




POST используется для отправки сущностей к определённомu ресурсу. Часто вызывает изменение состояния или какие-то побочные эффекты на сервере.

Post применяется для передачи пользовательских данных заданному ресурсу. Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода POST обычно загружаются файлы.

В отличие от метода GET, метод POST не считается идемпотентным, то есть многократное повторение одних и тех же запросов POST может возвращать разные результаты (например, после каждой отправки комментария будет появляться одна копия этого комментария).

При результатах выполнения 200 (Ok) и 204 (No Content) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ 201 (Created) с указанием URI нового ресурса в заголовке Location.

Сообщение ответа сервера на выполнение метода POST не кэшируется.



PUT


UPDATE
U

PUT /text.txt HTTP/1.1
Host: example.com

Title=Новый+заголовок+файла
Text=Новый+текст+файла

PUT вносит изменения в уже имеющуюся на сервере информацию.

PUT может сопоставляться как для создания, так и для обновления в зависимости от наличия URI, используемого с PUT.



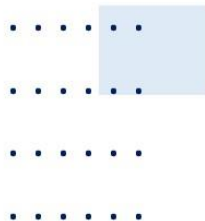


Метод PUT. Этот метод может поместить страницу на сервер. Тело запроса PUT включает размещаемую страницу, которая закодирована по MIME. Это метод требует идентификации клиента.

Фундаментальное различие методов POST и PUT заключается в понимании предназначений URI ресурсов. Метод POST предполагает, что по указанному URI будет производиться обработка передаваемого клиентом содержимого. Используя PUT, клиент предполагает, что загружаемое содержимое соответствует находящемуся по данному URI ресурсу.

Сообщения ответов сервера на метод PUT не кэшируются.

POST это отправка новых данных на сервер. **PUT** вносит изменения в уже имеющуюся на сервере информацию.



DELETE



Удаляет указанный ресурс.

```
DELETE /text.txt HTTP/1.1
Host: example.com
```



DELETE удаляет указанный ресурс.

Все вышеперечисленные методы можно разделить на три группы:

Безопасные для сервера
(GET, HEAD, OPTIONS)

не изменяют данные, их
можно выполнять в любой
последовательности;

Идемпотентные
(GET, HEAD, PUT, DELETE,
OPTIONS)

при повторном выполнении
результаты будут ожидаемо
одинаковыми;

Неидемпотентные
(POST, PATCH) —

при повторном
выполнении результаты
будут разными, если,
например, отправить
POST-запрос на
создание элемента
несколько раз подряд,
то он может создать
несколько элементов с
одинаковыми данными.

Все вышеперечисленные методы можно разделить на три группы:

безопасные (GET, HEAD, OPTIONS) — не изменяют данные, их можно выполнять в любой последовательности;

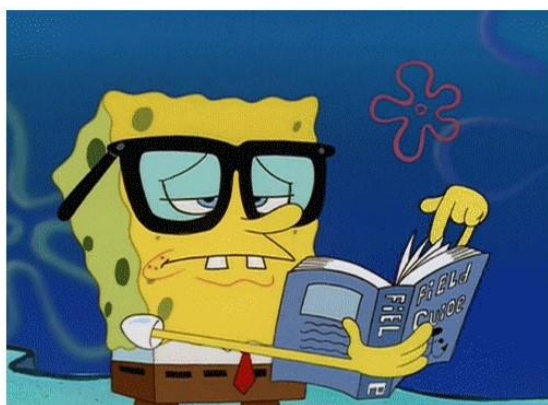
идемпотентные (GET, HEAD, PUT, DELETE, OPTIONS) — при повторном выполнении результаты будут ожидаемо одинаковыми;

неидемпотентные (POST, PATCH) — при повторном выполнении результаты будут разными, если, например, отправить POST-запрос на создание элемента несколько раз подряд, то он может создать несколько элементов с одинаковыми данными.

Если по какой-то причине встроенных методов в спецификации HTTP недостаточно, можно использовать собственные кастомные методы. Для этого нужно чтобы сервер знал об их существовании и понимал как обрабатывать запросы с этими методами.

GET	POST
Кэшируется	Не кэшируется
Тело запроса нет	Тело запроса есть
Остается в истории браузера.	Не остается в истории браузера.
GET запросы могут быть закладками	Запросы POST не могут быть закладками
GET запросы никогда не должны использоваться при работе с конфиденциальными данными	
Да, при отправке данных метод Get добавляет данные в URL-адрес; и длина URL ограничена (максимальная длина URL составляет 2048 символов)	Запросы POST не имеют ограничений по длине данных
GET запросы должны использоваться только для извлечения данных	Обновление, создание, удаление данных
Разрешены только символы ASCII	Никаких ограничений. Двоичные данные также разрешены
Get менее безопасен по сравнению с POST, поскольку отправляемые данные являются частью URL-адреса	POST немного безопаснее, чем Get, поскольку параметры не сохраняются в журнале обозревателя или в журналах веб-сервера
Данные видны всем в URL	Данные не отображаются в URL-адресе
Идемпотентный	Не идемпотентный

Как прочитать ответ?



HTTP/Версия	Код состояния	Пояснение
HTTP/1.1	200	OK

```
Access-Control-Allow-Origin: *
Cache-Control: public, max-age=3600
Content-Length: 515523
Content-Type: text/javascript; charset=utf-8
Expires: Wed, 10 Nov 2021 11:06:57 ora solare FLE
Last-Modified: Wed, 01 Jan 2010 01:00:00 GMT
```

Как прочитать ответ?

Стартовая строка ответа имеет следующую структуру:

HTTP/Версия Код состояния Пояснение

Версия протокола здесь задаётся так же, как в запросе.

Код состояния (*Status Code*) — три цифры (первая из которых указывает на класс состояния), которые определяют результат совершения запроса. Например, в случае, если был использован метод GET, и сервер предоставляет ресурс с указанным идентификатором, то такое состояние задаётся с помощью кода 200. Если сервер сообщает о том, что такого ресурса не существует — 404. Если сервер сообщает о том, что не может предоставить доступ к данному ресурсу по причине отсутствия необходимых привилегий у клиента, то используется код 403. Спецификация HTTP 1.1 определяет 40 различных кодов HTTP, а также допускается расширение протокола и использование дополнительных кодов состояний.

Пояснение к коду состояния (*Reason Phrase*) — текстовое (но не включающее символы CR и LF) пояснение к коду ответа, предназначено для упрощения чтения ответа человеком. Пояснение может не учитываться клиентским программным обеспечением, а также может отличаться от стандартного в некоторых реализациях серверного ПО.

После стартовой строки следуют заголовки, а также тело ответа. Например:

```
Access-Control-Allow-Origin: *
Cache-Control: public, max-age=3600
Content-Length: 515523
Content-Type: text/javascript; charset=utf-8
Expires: Wed, 10 Nov 2021 11:06:57 ora solare FLE
Last-Modified: Wed, 01 Jan 2010 01:00:00 GMT
```

<!DOCTYPE html... (here comes the 29769 bytes of the requested web page)

Заголовок ответа **Access-Control-Allow-Origin** показывает, может ли ответ сервера быть доступен коду, отправляющему запрос с данного источника origin.

* Для запросов без учётных данных. Значение "*" может быть использован как шаблон; значение указывает браузеру разрешить запросы из любых источников. Попытка использовать шаблон с учётными данными приведёт к ошибке.

Общий заголовок **Cache-Control** используется для задания инструкций кеширования как для запросов, так и для ответов. Инструкции кеширования однонаправленные: заданная инструкция в запросе не подразумевает, что такая же инструкция будет указана в ответе

private

Указывает, что ответ предназначен для одного пользователя и не должен помещаться в разделяемый кеш. Частный кеш может хранить ресурс.

max-age=<seconds>

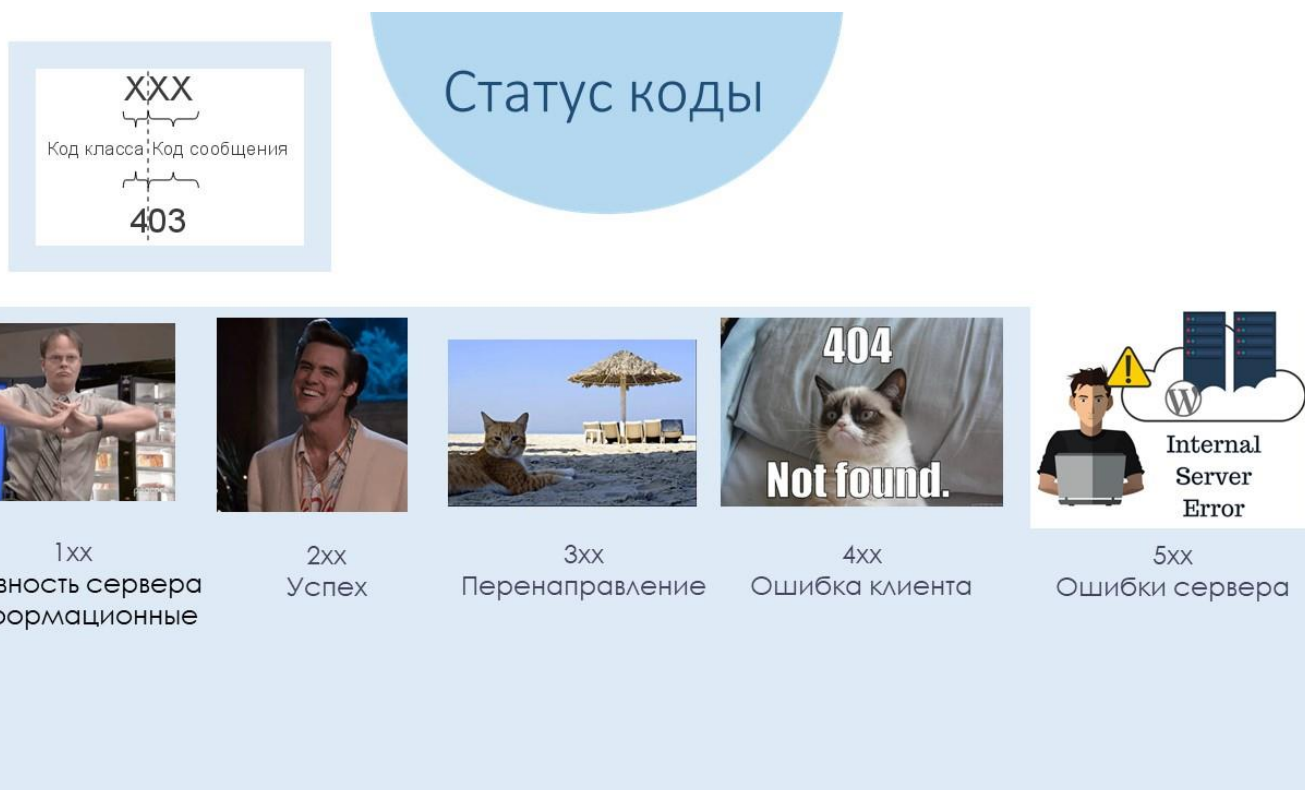
Задаёт максимальное время в течение которого ресурс будет считаться актуальным. В отличие от Expires, данная инструкция является относительной по отношению ко времени запроса.

Заголовок **Content-Length** указывает размер отправленного получателю тела объекта в байтах.

Заголовок-сущность **Content-Type** используется для того, чтобы определить MIME тип ресурса.

В ответах сервера заголовок Content-Type сообщает клиенту, какой будет тип передаваемого контента. В некоторых случаях браузеры пытаются сами определить MIME **Multipurpose Internet Mail Extensions** тип передаваемого контента, но их реакция может быть неадекватной.

- Опционально: тело, содержащее пересылаемый ресурс.



Коды состояния протокола HTTP

Класс кодов	Краткое описание
1xx Informational (Информационный)	<p>В этот класс выделены коды, информирующие о процессе передачи. В HTTP/1.0 сообщения с такими кодами должны игнорироваться. В HTTP/1.1 клиент должен быть готов принять этот класс сообщений как обычный ответ, но ничего отправлять серверу не нужно. Сами сообщения от сервера содержат только стартовую строку ответа и, если требуется, несколько специфичных для ответа полей заголовка. <u>Прокси-сервера</u> подобные сообщения должны отправлять дальше от сервера к клиенту.</p> <p>Примеры ответов сервера: 100 Continue (Продолжать) 101 Switching Protocols (Переключение протоколов)</p>

	102 Processing (Идёт обработка)
2xx Success (Успешно)	<p>Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.</p> <p>Примеры ответов сервера: 200 OK (Успешно). 201 Created (Создано) 202 Accepted (Принято) 204 No Content (Нет содержимого) 206 Partial Content (Частичное содержимое)</p>
3xx Redirection (Перенаправление)	<p>Коды статуса класса 3xx сообщают клиенту, что для успешного выполнения операции нужно произвести следующий запрос к другому URI. В большинстве случаев новый адрес указывается в поле Location заголовка. Клиент в этом случае должен, как правило, произвести автоматический переход (жарг. «редирект»).</p> <p>Обратите внимание, что при обращении к следующему ресурсу можно получить ответ из этого же класса кодов. Может получиться даже длинная цепочка из перенаправлений, которые, если будут производиться автоматически, создадут чрезмерную нагрузку на оборудование. Поэтому разработчики протокола HTTP настоятельно рекомендуют после второго подряд подобного ответа обязательно запрашивать подтверждение на перенаправление у пользователя (раньше рекомендовалось после 5-го). За этим следить обязан клиент, так как текущий сервер может перенаправить клиента на ресурс другого сервера. Клиент также должен предотвратить попадание в круговые перенаправления.</p> <p>Примеры ответов сервера: 300 Multiple Choices (Множественный выбор) 301 Moved Permanently (Перемещено навсегда) 304 Not Modified (Не изменялось)</p>
4xx Client Error (Ошибка клиента)	<p>Класс кодов 4xx предназначен для указания ошибок со стороны клиента. При использовании всех методов, кроме HEAD, сервер должен вернуть в теле сообщения гипертекстовое пояснение для пользователя.</p> <p>Примеры ответов сервера: 401 Unauthorized (Неавторизован) 402 Payment Required (Требуется оплата)</p>

	403 Forbidden (Запрещено) 404 Not Found (Не найдено) 405 Method Not Allowed (Метод не поддерживается) 406 Not Acceptable (Не приемлемо) 407 Proxy Authentication Required (Требуется аутентификация прокси)
5xx Server Error (Ошибка сервера)	<p>Коды 5xx выделены под случаи неудачного выполнения операции по вине сервера. Для всех ситуаций, кроме использования метода HEAD, сервер должен включать в тело сообщения объяснение, которое клиент отобразит пользователю.</p> <p>Примеры ответов сервера:</p> 500 Internal Server Error (Внутренняя ошибка сервера) 502 Bad Gateway (Плохой шлюз) 503 Service Unavailable (Сервис недоступен) 504 Gateway Timeout (Шлюз не отвечает)

Проблема безопасности



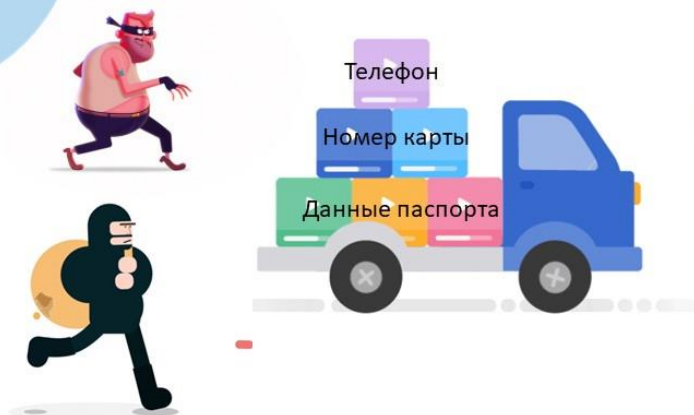
Проблема с HTTP заключается в том, что его изобрели в 1989 году. В то время разработчиков, как и пользователей, не очень беспокоил вопрос интернет-безопасности.



Проблема с HTTP заключается в том, что его изобрели в 1989 году. В то время разработчиков, как и пользователей, не очень беспокоил вопрос интернет-безопасности. Гораздо важнее было разработать унифицированный стандарт, который позволит пользователям со всего мира выходить онлайн и пользоваться нужными сайтами.

Проблемы с HTTP начались уже в 1990-х, одновременно с глобальным распространением интернета. Появилось много популярных сайтов, на некоторых ввели систему онлайн-платежей.

HTTP



Проблемы с HTTP начались уже в 1990-х, одновременно с глобальным распространением интернета. Появилось много популярных сайтов, на некоторых ввели систему онлайн-платежей. Тогда и начали активно действовать злоумышленники.

Тогда и начали активно действовать злоумышленники. Они похищали пользовательские данные либо рушили серверы с сайтами. Киберпреступность процветала, в частности, из-за уязвимости в протоколе HTTP.

Когда интернет-сообщество ввело в употребление протокол HTTP, два основных вопроса были — как осуществить передачу данных без потерь, и как доставить пакеты данных с максимальной скоростью?

При таких критериях программисты договорились, что в интернете все данные будут отправлены и получены «открытым текстом». Представьте, что все почтовые отделения мира отправляют и принимают письма без конвертов. По умолчанию предполагается, что почтальоны достаточно надежные, чтобы пересылать корреспонденцию, не читая её.

Аналогичным образом устроен принцип действия HTTP. Если интернет-магазин использует этот протокол, то при отправке данных платежной карты по проводам в точности будет отправлен номер банковской карты. Злоумышленнику несложно перехватывать данные на промежутке между сайтом, которым пользуется покупатель, и сервером, на котором находится интернет-магазин. Так данные пользователей попадали к мошенникам.

Еще хуже, что хакеры могут перехватить трафик на сайт и добавить небольшие фрагменты кода (сниппеты) к каждому пакету данных. В марте 2015 года по такой схеме хакеры реализовали DDoS-атаку на сайты GreatFire.org и GitHub. Через серверы, которые предположительно имеют связь с «Великим китайским файерволом», хакеры смогли обрушить серверы GitHub. Сайт не работал на протяжении 5 минут, пока системные администраторы подключали резервный сервер. Это крупнейшая, но не единственная атака на сайт, которая стала возможной из-за использования протокола HTTP.

Сейчас, во время повсеместного распространения высокоскоростного интернета, использовать HTTP не нужно даже домохозяйкам, которые ведут небольшой блог о кулинарии. Впрочем, возможная атака на сайт — это не единственная угроза для владельца.

При переходе на сайты с HTTP браузер отображает сообщение о небезопасном подключении.



⚠ Не защищено |



Подключение не защищено

Злоумышленники могут попытаться похитить ваши данные с сайта ВАШ-САЙТ.РФ (например, пароли, сообщения или номера банковских карт). [Подробнее...](#)

NET::ERR_CERT_COMMON_NAME_INVALID

☒ Отправлять в Google URL и контент некоторых посещенных страниц, а также ограниченную информацию о системе для повышения безопасности Chrome. [Поддержка конфиденциальности](#)

[Дополнительные](#)

[Вернуться к безопасной странице](#)

Безопасный протокол гарантирует аутентификацию — попадание пользователей именно на тот ресурс, который необходим, это обеспечивает борьбу с мошенническими вмешательствами.



🔒 https://ksenzov.com

Сервисы Gmail YouTube Maps

Курс тестирования ПО
Вадима Ксендзова

защита: 1 подключение

🔒 Действительный сертификат

При переходе на сайты с HTTP браузер отображает сообщение о небезопасном подключении.

А теперь представьте, что вы хотите купить курсы по QA и заходите на сайт к Вадиму, и видите вот такую штуку в адресной строке – https

Это значит что протокол Безопасный и гарантирует аутентификацию — попадание пользователей именно на тот ресурс, который необходим, это обеспечивает борьбу с мошенническими вмешательствами.



HTTPS

HyperText Transfer Protocol Secure

.....

.....

.....

.....



Принцип действия HTTPS и его отличие от HTTP

Буква «s» в названии протокола HTTPS означает «secure», т.е. защищенный. Веб-браузеры и сайты, которые используют протокол HTTPS, отправляют данные, защищенные криптошифрованием. То есть на пути от компьютера пользователя до сервера веб-приложения информация курсирует в нечитабельном виде. Это случайный набор знаков. Основное достоинство протокола заключается в том, что дешифровка данных происходит в рамках одной веб-сессии. То есть невозможно подобрать универсальный дешифровщик, который позволит приводить любые данные в удобочитаемый вид.

В HTTP отсутствует механизм защиты для шифрования данных, в то время как HTTPS для защиты связи между сервером и клиентом использует цифровой сертификат SSL или TLS.

В HTTP отсутствует механизм защиты для шифрования данных, в то время как HTTPS для защиты связи между сервером и клиентом использует цифровой сертификат SSL или TLS.

SSL (Secure Sockets Layer) уровень защищенных сокетов

SSL или слой защищенных сокетов было оригинальным названием протокола, который разработала компания Netscape в середине 90-х. SSL 1.0 никогда не был публично доступным, а в версии 2.0 были серьезные недостатки. Протокол SSL 3.0, выпущенный в 1996, был полностью переделан и задал тон следующей стадии развития.

TLS (Transport Level Security) безопасность транспортного уровня

Когда следующую версию протокола выпустили в 1999, ее стандартизировала специальная рабочая группа проектирования сети Интернет и дала ей новое название: защита транспортного уровня, или TLS. Как говорится в TLS-документации, «разница между этим протоколом и SSL 3.0 не критичная».

Что такое SSL

Secure Sockets Layer

Проще говоря, сокет - это псевдофайл, представляющий сетевое соединение. Как только сокет был создан (используя надлежащие примитивы и соответствующие параметры для идентификации другого хоста), записи в сокет превращаются в сетевые пакеты, которые отправляются, и данные, полученные из сети, могут считываться из сокета.

С одной стороны, сокеты очень похожи на каналы: они выглядят как файлы для программ, использующих их, но не приводят к чтению или записи на диск; скорее, они позволяют общаться с другой программой (локальной в случае каналов и, возможно, удаленной в случае сокетов). Они также предлагают, как вы упоминаете, двунаправленную связь (так же, как пара правильно соединенных каналов).

Наконец, программы на одном компьютере обычно общаются с использованием стандартных сетевых протоколов, таких как TCP; было бы расточительно пройти весь путь к сетевому оборудованию (если оно есть!), вычислить контрольные суммы и т. д., просто вернуться на тот же хост: вот где появляются сокеты доменов Unix. Это очень похоже на обычные сокеты, за исключением они связывают процессы на одном хосте, а не удаленные процессы, и вообще не пытаются использовать какие-либо сетевые ресурсы. Таким образом, они являются средой межпроцессного взаимодействия.

Как упоминалось в tripleee, в ходе истории BSD трубы были введены раньше, чем сокеты, и были заново реализованы с использованием сокетов, как только они появились. В той же ссылке, «Проектирование и реализация операционной системы FreeBSD», упоминается, что каналы были затем возвращены к реализации без сокетов по соображениям производительности: это, безусловно, подчеркивает тот факт, что каналы имеют сходство.

SSL или слой защищенных сокетов было оригинальным названием протокола, который разработала компания Netscape в середине 90-х. SSL 1.0 никогда не был публично доступным, а в версии 2.0 были серьезные недостатки. Протокол SSL 3.0, выпущенный в 1996, был полностью переделан и задал тон следующей стадии развития.

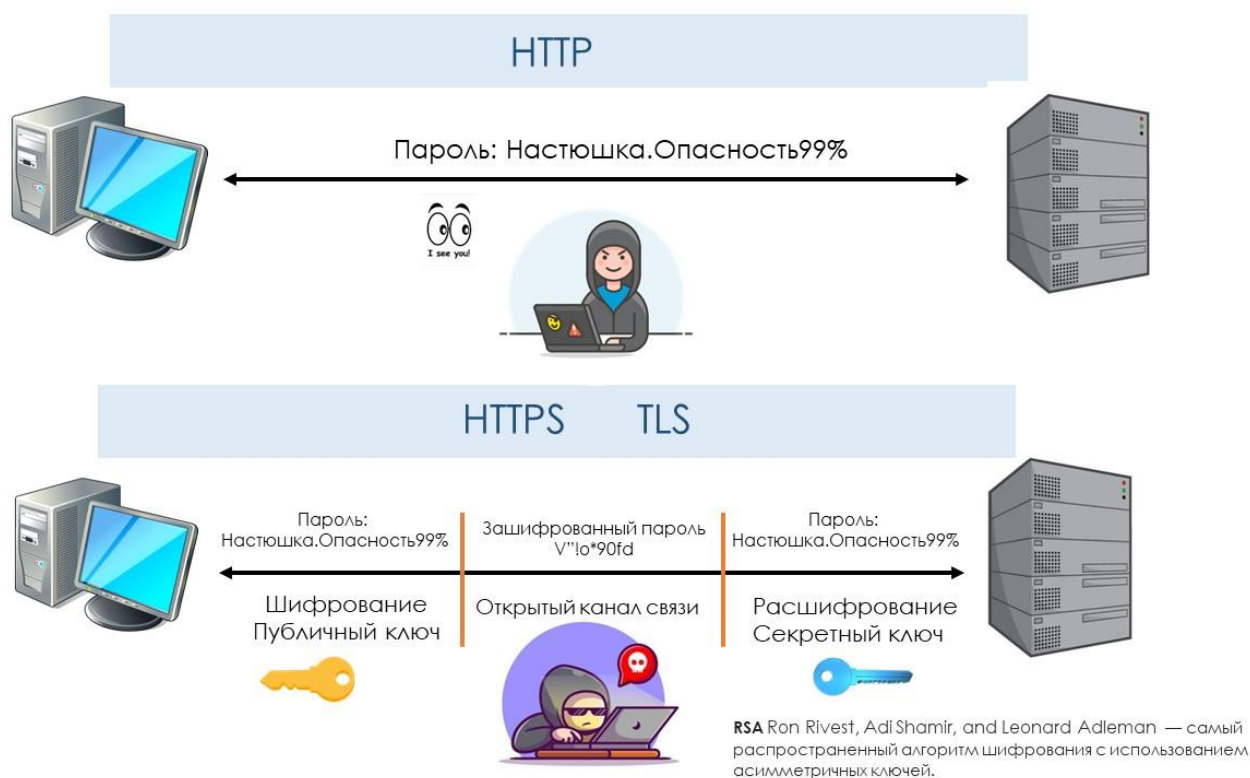
Установите SSL-сертификат и защитите свой сайт!

Что такое TLS

Когда следующую версию протокола выпустили в 1999, ее стандартизировала специальная рабочая группа проектирования сети Интернет и дала ей новое название: защита транспортного уровня, или TLS. Как говорится в TLS-документации, «разница между этим протоколом и SSL 3.0 не критичная». TLS и SSL формируют постоянно обновляемую серию протоколов, и их часто объединяют под названием SSL/TLS.

Протокол TLS шифрует интернет-трафик любого вида. Самый распространенный вид — веб-трафик. Вы знаете, когда ваш браузер устанавливает соединение по TLS — если ссылка в адресной строке начинается с «https».

TLS также используется другими приложениями — например, в почте и системах телеконференций.



Как работает TLS

Шифрование необходимо, чтобы безопасно общаться в интернете. Если ваши данные не шифруются, любой может проанализировать их и прочитать

конфиденциальную информацию. HTTP не шифрует данные, поэтому злоумышленники могут украсть ваши данные.

Самый безопасный метод шифрования — это **асимметричное шифрование**. Для этого требуется 2 ключа, 1 публичный и 1 приватный. Осталось разобраться, почему в названии алгоритма присутствует слово «асимметричный». Самым примечательным в методе является тот факт, что для шифровки и дешифровки данных используются разные ключи.

Механизм сложный, но если попросту, вы можете использовать публичный ключ (как правило, он публикуется в самом сертификате владельца), чтобы зашифровать данные, но вам нужен приватный ключ, чтобы расшифровывать их. Два ключа связаны с помощью сложной математической формулы, которую сложно хакнуть.

Можно представить публичный ключ как информацию о местоположении закрытого почтового ящика с отверстием, и приватный ключ как ключ, который открывает ящик. Любой, кто знает, где находится ящик, может положить туда письмо. Но чтобы прочитать его, человеку нужен ключ, чтобы открыть ящик.

Так как в асимметричном шифровании применяются сложные математические расчеты, нужно много вычислительных ресурсов. TLS решает эту проблему, используя асимметричное шифрование только в начале сессии, чтобы зашифровать общение между сервером и клиентом. Сервер и клиент должны договориться об одном ключе сессии, который они будут вдвоем использовать, чтобы зашифровать пакеты данных.

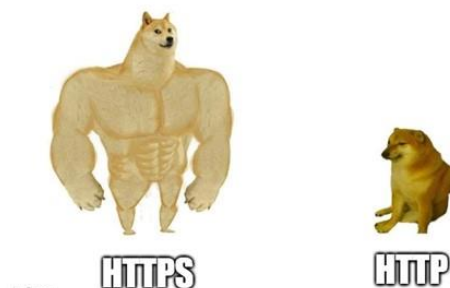
Процесс, согласно которому клиент и сервер договариваются о ключе сессии, называется **рукопожатием**. Это момент, когда 2 общающихся компьютера представляются друг другу.

Любой пользователь может получить открытый ключ по назначению и использовать его для шифрования данных, расшифровать которые может только пользователь, владеющий секретным ключом. (RSA)

Если заголовок шифрует данные, используя свой секретный ключ, каждый может расшифровать данные, используя соответствующий открытый ключ. Именно это является основой для цифровых подписей. (DSA)

RSA Ron Rivest, Adi Shamir, and Leonard Adleman — самый распространенный алгоритм шифрования с использованием асимметричных ключей.

Сайты, которые используют HTTPS, имеют уникальный цифровой сертификат (например, SSL-сертификат), который выдан центром сертификации (Certification authority, CA). Когда интернет-пользователь заходит на такой сайт, сервер передает браузеру данные о сертификате и публичный ключ. Веб-браузер использует публичный ключ, чтобы установить цифровую подпись в сертификате. Затем веб-браузер сравнивает подпись с данными, которые получены от CA. Если сертификат действительный, будет установлено соединение с сайтом, а в адресной строке браузера появится пиктограмма в виде зеленого замка. В случае, если на сайте сертификат отсутствует или он не подтвержден браузером, появится соответствующее уведомление, а в адресной строке появится красный замок.



Почему использование протокола HTTP отрицательно влияет на посещаемость сайта?

Использование протокола HTTP негативно влияет на эффективность веб-сайта. Если интернет-пользователь заходит на страницу, у которой нет протокола безопасности SSL/TLS, десктопные веб-браузеры обязательно покажут уведомление об отсутствии сертификата. Мобильные браузеры, особенно Chrome и Opera, вообще не загрузят сайт.

Рядовые пользователи привыкают к определенной системе. Если у сайта нет сертификата безопасности, — это сигнал. Либо владельцы сайта экономят на безопасности, либо веб-ресурс давно не обновлялся и выданный сертификат устарел.

Для владельцев сайтов отказ от использования протокола HTTPS — это верный путь к утрате трафика. Во-первых, часть пользователей не станет посещать сайт, у которого нет SSL-сертификата. Во-вторых, еще с 2014 года Google использует наличие сертификата как сигнал для повышения сайтов в результатах поиска. Соответственно, веб-ресурс без сертификата рискует откатиться на последние страницы в поисковой выдаче..

ПРЕИМУЩЕСТВА HTTP:

- HTTP может быть реализован на основе другого протокола в Интернете или в других сетях;
- Страницы HTTP хранятся в кэше компьютера и Интернета, поэтому доступ к ним осуществляется быстрее;

- Кроссплатформенность
- Не нуждается в поддержке среды выполнения
- Можно использовать через брандмауэры. Возможны глобальные приложения
- Не ориентирован на подключение; таким образом, отсутствуют накладные расходы на сеть для создания и поддержания состояния сеанса и информации

ПРЕИМУЩЕСТВА HTTPS

- В большинстве случаев сайты, работающие по протоколу HTTPS, будут перенаправлены. Поэтому даже если ввести HTTP://, он перенаправит на https через защищенное соединение
- Это позволяет пользователям выполнять безопасные транзакции электронной коммерции, такие как онлайн-банкинг.
- Технология SSL защищает всех пользователей и создает доверие
- Независимый орган проверяет личность владельца сертификата. Таким образом, каждый SSL-сертификат содержит уникальную аутентифицированную информацию о владельце сертификата.

ОГРАНИЧЕНИЯ HTTP

- Нет защиты информации, так как любой может прослушать и увидеть передаваемый контент
- Обеспечение целостности данных является большой проблемой, поскольку есть возможность изменения содержимого на лету во время передачи.
- Не знаешь кто на противоположной стороне. Любой, кто перехватит запрос, может получить имя пользователя и пароль.

ОГРАНИЧЕНИЯ HTTPS

- Протокол HTTPS не может остановить кражу конфиденциальной информации со страниц, кэшированных в браузере
- Данные SSL могут быть зашифрованы только во время передачи по сети. Поэтому он не может очистить текст в памяти браузера
- HTTPS ввиду вычислений может увеличить задержки во время передачи данных.

Параметр	HTTP	HTTPS
Название	Hypertext Transfer Protocol	Hypertext Transfer Protocol Secure
Безопасность	Менее безопасен. Данные могут быть доступны для злоумышленников	Он предназначен для предотвращения доступа хакеров к критически важной информации. Защищен атак типа Man-in The-Middle.
Порт	По умолчанию – 80	По умолчанию 443
Начинается на	http://	https://
Область применения	Это хорошо подходит для веб-сайтов общего назначения, таких как блоги.*	Если на сайте нужно вводить конфиденциальную информацию, то данный протокол подходит больше
Защита	Нет защиты передаваемой	HTTPS шифрует данные перед передачей их по сети. На стороне

	информации. Любой, кто прослушивает трафик может получить доступ к данным	получателя, данные расшифровываются.
Протокол	Работает с TCP/IP	Нет специального протокола. Работает поверх HTTP, но использует TLS/SSL шифрование.
Проверка названия домена	Сайтам с HTTP не нужен SSL	Для работы с HTTPS нужен SSL сертификат
Шифрование данных	Не использует шифрование	Данные шифруются
Рейтинг поиска	Не влияет на рейтинг поиска	Помогает увеличивать поисковый рейтинг
Скорость	Быстро**	Относительно медленно
Уязвимость	Уязвима для злоумышленников	Лучше защищен, использует шифрование данных.

Источники

<https://netpeak.net/ru/blog/chem-http-otlichayet-sya-ot-https-story/>

<https://netpeak.net/ru/blog/chem-http-otlichayet-sya-ot-https-story/>

<https://wiki.merionet.ru/servernye-resheniya/85/v-chem-raznica-mezhdu-http-i-https/>

<https://www.uplab.ru/blog/http-i-https-sravnenije/>

<https://vebrost.ru/blog/https-ili-http-chto-luchshe-i-v-chem-raznitsa/>

<https://habr.com/ru/post/215117/>

<https://www.4stud.info/web-programming/protocol-http.html>

<https://www.hostinger.ru/rukovodstva/shto-takoe-html/> - html

<https://webkys.info/post/chto-takoe-html-i-zachem-eto-nuzhno>

<https://webonto.ru/protokol-http/>