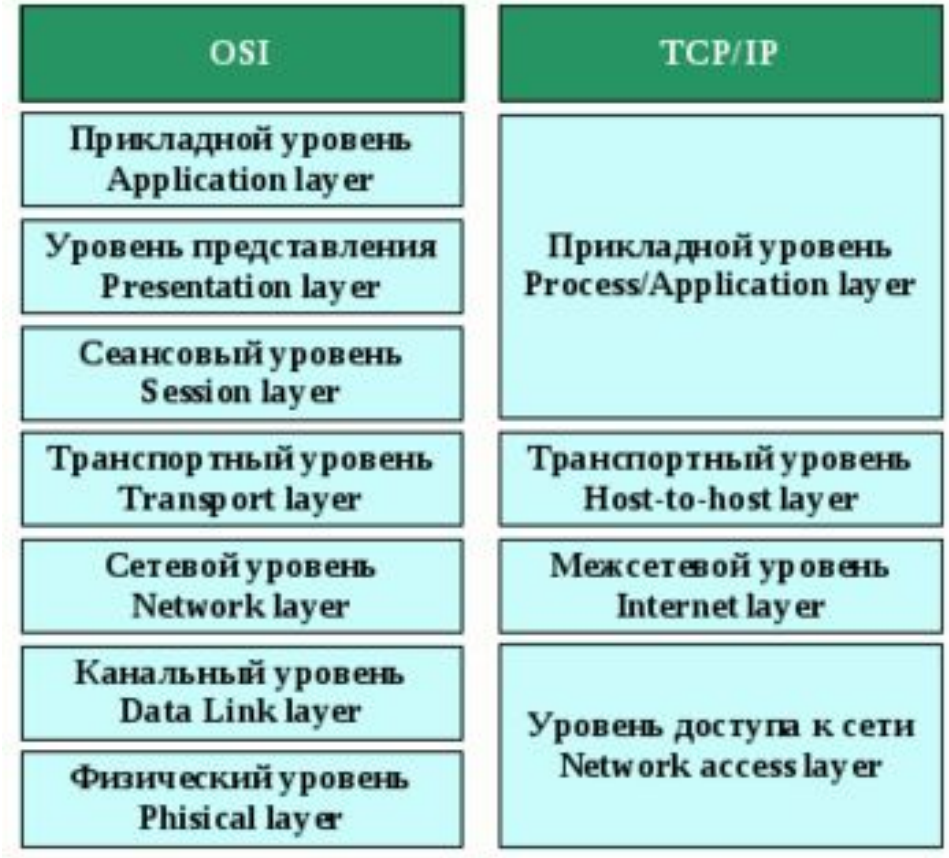
# **Web-**программирование
## Лекция 7. HTTP. Django.

асист. каф. 308 Трутнева Надежда Владимировна

тел: **8-926-880-12-76**

почта: **ntrutn@gmail.com**

# Модели сетевого взаимодействия




Данные

Заголовок + данные HTTP

Заголовок + данные + адресат TCP
(деление на фрагменты)

Заголовок + данные + адресат IP
(адрес устройства)
(деление на фрагменты)

Преобразование к формату среды передачи данных (сигналы)
Wi-fi/Ethernet/FastEthernet

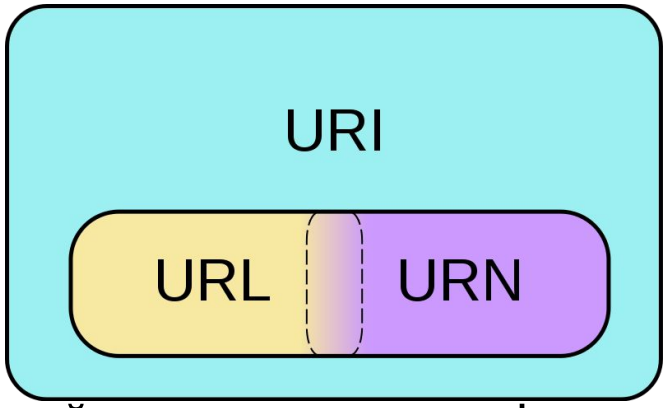
# URI, URL, URN

URI (Uniform Resource Identifier) — унифицированный (единообразный) идентификатор ресурса.
URI — последовательность символов, идентифицирующая абстрактный или физический ресурс. Ранее назывался Universal Resource Identifier — универсальный идентификатор ресурса.

URI — символьная строка, позволяющая идентифицировать какой-либо ресурс:

- документ,
- изображение,
- файл,
- службу,
- ящик электронной почты и т. д.

# URI, URL, URN




**URL** — это URI, который, помимо идентификации ресурса, предоставляет ещё и информацию о местонахождении этого ресурса.

**URN** — это URI, который только идентифицирует ресурс в определённом пространстве имён (и, соответственно, в определённом контексте), но не указывает его местонахождения.

*urn:ISBN:0-395-36341-1* — это URN, который указывает на ресурс (книгу) 0-395-36341-1 в пространстве имён ISBN, но, в отличие от URL, URN не указывает на местонахождение этого ресурса: в нём не сказано, в каком магазине её можно купить или на каком сайте скачать

# структура **URI**

URI = [ схема ":" ] иерархическая-часть [ "?" запрос ] [ "#" фрагмент ]

**схема**
схема обращения к ресурсу (часто указывает на сетевой протокол), например http, ftp, file, ldap, mailto, urn

**иерархическая-часть**
содержит данные, обычно организованные в иерархической форме, которые, совместно с данными в неиерархическом компоненте **запрос**, служат для идентификации ресурса в пределах видимости URI-схемы. Обычно **иер-часть** содержит путь к ресурсу (и, возможно, перед ним, адрес сервера, на котором тот располагается) или идентификатор ресурса (в случае URN).

**запрос**
этот необязательный компонент URI описан выше.

**фрагмент**
(тоже необязательный компонент)

# РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

**Регулярные выражения** (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов. Для поиска используется строка-образец (англ. pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Для манипуляций с текстом дополнительно задаётся строка замены, которая также может содержать в себе специальные символы.

# Структура регулярного выражения

Как правило, регулярное выражение состоит из трёх основных частей:

**1. Якорь** – определяет позицию шаблона в строке текста:

- ^ – якорь, определяющий начало строки;
- $ – якорь, определяющий конец строки.

**2. Набор (последовательность) символов** – для поиска соответствий в заданных позициях строки текста:

- символ "точка" (.) соответствует любому произвольному символу;
- алфавитно-цифровые символы и пробел представляют сами себя;
- прочие символы – интерпретация зависит от диалекта.

**3. Модификатор** – задаёт количество повторов предыдущего символа или набора символов (в зависимости от диалекта):

- * – любое количество повторов символа/набора, в том числе и нулевое;
- ? – соответствует нулю или одному экземпляру символа/набора;
- + – соответствует одному или большему количеству экземпляров символа/набора.

# Структура регулярного выражения

**Пример:**
необходимо найти все директивы определения макроконстант в исходном коде на языке С.

```
grep '^ *#define.*' *.c *.h
```

Здесь учтено, что в начале строки макроопределения может быть вставлено любое количество пробелов или же пробелы отсутствуют. Часть шаблона #define является литеральной, т.е. каждый символ интерпретируется "как есть". Заключительная часть шаблона означает "любые символы в любых количествах".

# Определение диапазонов символов

Если возникает необходимость задать символ из определённой группы, например, только цифровой символ, или только гласную букву нижнего регистра, или только символы пунктуации, то используются квадратные скобки, внутри которых определяются требуемые символы.

[012345789] – соответствует одному цифровому символу из заданного набора;
[аеёиоуыэюя] – соответствует одной из перечисленных гласных букв;
[,.:;] – соответствует одному из символов пунктуации.

# Определение диапазонов символов

Непрерывные диапазоны символов можно записывать в сокращённой форме с использованием дефиса: первый пример удобнее записать в виде [0–9]. Допускаются любые сочетания диапазонов и конкретных символов.

Имеется также возможность исключать заданные наборы символов из поиска:

[^0-9] – соответствует любому символу, кроме цифрового;

[^аеёиоуыэюя] – соответствует любой НЕ гласной букве.

# Модификаторы количества повторений символов

Поиска IP-адреса:
[0-9]*\.[0-9]*\.[0-9]*\.[0-9]
Это приведёт к выводу строк, содержащих элементы типа 2344.5657.11.00000, не являющихся IP-адресами.

Для уточнения количества повторений наборов символов применяется модификатор \{min,max\}.
В каждой части IP-адреса может содержаться от одной до трёх цифр, следует учесть, что значение 0 не используется в качестве первого байта обычных IP-адресов. В итоге получим следующий шаблон поиска:
grep '[1-9][0-9]\{0,2\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}' *.txt

# Запоминание и повторное использование элемента шаблона

В простых регулярных выражениях части шаблона, заключённые внутри конструкции \( \), запоминаются и нумеруются, после чего их можно использовать повторно.

Всего можно запомнить до девяти пронумерованных шаблонов.

**Пример.** Поиск палиндромов (слов, которые одинаково читаются как слева направо, так и справа налево):

\([a-z]\)\([a-z]\)[a-z]\2\1 – для пятибуквенных палиндромов (например, level, rotor, madam и т.д.)

\([a-z]\)\([a-z]\)\([a-z]\)\3\2\1 – для шестибуквенных палиндромов (например, redder, succus, terret и т.д.)

# Полезные регулярные выражения

**Поиска денежной суммы, записываемой в формате "10000 руб. 00 коп."**
[0-9]\{1,\} руб\. [0-9]\{2\} коп\.

Необходимое пояснение: если в модификаторе типа \{min,max\} отсутствует и запятая, и максимальное значение, то такая конструкция задаёт точное количество ожидаемых повторов элемента шаблона. В нашем примере определяются ровно два цифровых символа для обозначения копеек.

# Полезные регулярные выражения

**Поиск URL-строки, соответствующей Web-ресурсу в Интернете:**
http://[a-zA-Z0-9]\{1,\}\.[-a-zA-Z0-9_]\{1,\}/*

Необходимое пояснение: дефис теряет своё специальное значение, если он указан в самой первой позиции сразу после открывающей квадратной скобки в диапазоне. По данному шаблону могут быть найдены и такие "экзотические" URL-строки, как, например, http://my.home-server/
В формате расширенных регулярных выражений этот шаблон можно было бы записать более компактно:
http://[a-zA-Z0-9]+\.[-a-zA-Z0-9_]+/*
Такую запись понимают, например, утилиты egrep и awk.

# Полезные регулярные выражения

**Поиск любого HTML-тэга:**

<[^>]+>

Совпадает с любой последовательностью символов за исключением > в количестве от одного и более, заключённой в угловые скобки. Иными словами, будет найден и односимвольный тэг <p>, и более "многословные" тэги, подобные <hr size=50>.

# Полезные регулярные выражения

**Вариант шаблона для поиска дат**

Расширенные регулярные выражения позволяют написать несколько громоздкий, но тем не менее корректно работающий шаблон для поиска дат, имеющих вид "13 ноября 2009 г.":

```
[12]?[0-9]
(янв|фев|мар|апр|мая|июн|июл|авг|сен|окт|ноя|дек).*
[0-9][0-9][0-9][0-9] г\.
```

Недостаток этого шаблона заключается в том, что с его помощью невозможно найти даты из древней истории, например, "13 ноября 245 г." или 1 января 88 г.", но для работы с современными документами он вполне годится (учитываем контекст поиска!).

# Полезные регулярные выражения

**Поиск палиндромов в любых языков**

\(.\)\(.\)\(.\)\3\2\1

С помощью такого шаблона можно находить шестисимвольные палиндромы не только на английском, но и на русском и на любых других языках, а также последовательности символов, не относящихся к алфавитным, например /*!!*/

# Полезные регулярные выражения

**Поиск ошибок в текстах, как "для для".**

\<\(..*\)\> \<\1\>

Здесь применяются ещё два элемента регулярных выражений: \< для обозначения начальной границы слова и \> для обозначения конечной границы слова. Таким образом, мы запоминаем только отдельные слова, а не любые последовательности символов. Выражение ..* соответствует любому слову, состоящему по крайней мере из одного символа. В результате мы сможем найти такие опечатки-повторения, как "и и", "не не", "для для" и т.п.

# Ограничение размера совподающей части шаблона

"Петров" "охранник"
"Иванов" "отдел снабжения" "экспедитор"
"Сидоров" "администрация" "директор"

```
".*"
  .
".*" ".*"
  .    .
```

```
"[^"]*"
```

Здесь после открывающей кавычки должно следовать любое количество символов, не являющихся кавычками, до тех пор, пока не встретится завершающая эту последовательность кавычка.

# **Python.** Регулярные выражения.

Регулярные выражения компилируются в объекты шаблонов, имеющие методы для различных операций, таких как поиск вхождения шаблона или выполнение замены строки.

```
>>> import re
>>> p = re.compile('ab*')
>>> print p
<_sre.SRE_Pattern object at 0x...>
```

# **Python.** Регулярные выражения.

Объекты шаблонов имеют несколько методов и атрибутов.

| **Метод/ атрибут** | **Цель** |
|---|---|
| match() | Определить, начинается ли совпадение регулярного выражения с начала строки |
| search() | Сканировать всю строку в поисках всех мест совпадений с регулярным выражением |
| findall() | Найти все подстроки совпадений с регулярным выражением и вернуть их в виде списка |
| finditer() | Найти все подстроки совпадений с регулярным выражением и вернуть их в виде итератора |

http://docs.python.org/howto/regex.html

# HTTP-протокол

**HyperText Transfer Protocol (HTTP)** - это протокол высокого уровня, обеспечивающий необходимую скорость передачи данных, требующуюся для распределенных информационных систем гипермедиа. HTTP используется проектом World Wide Web с 1990 года.

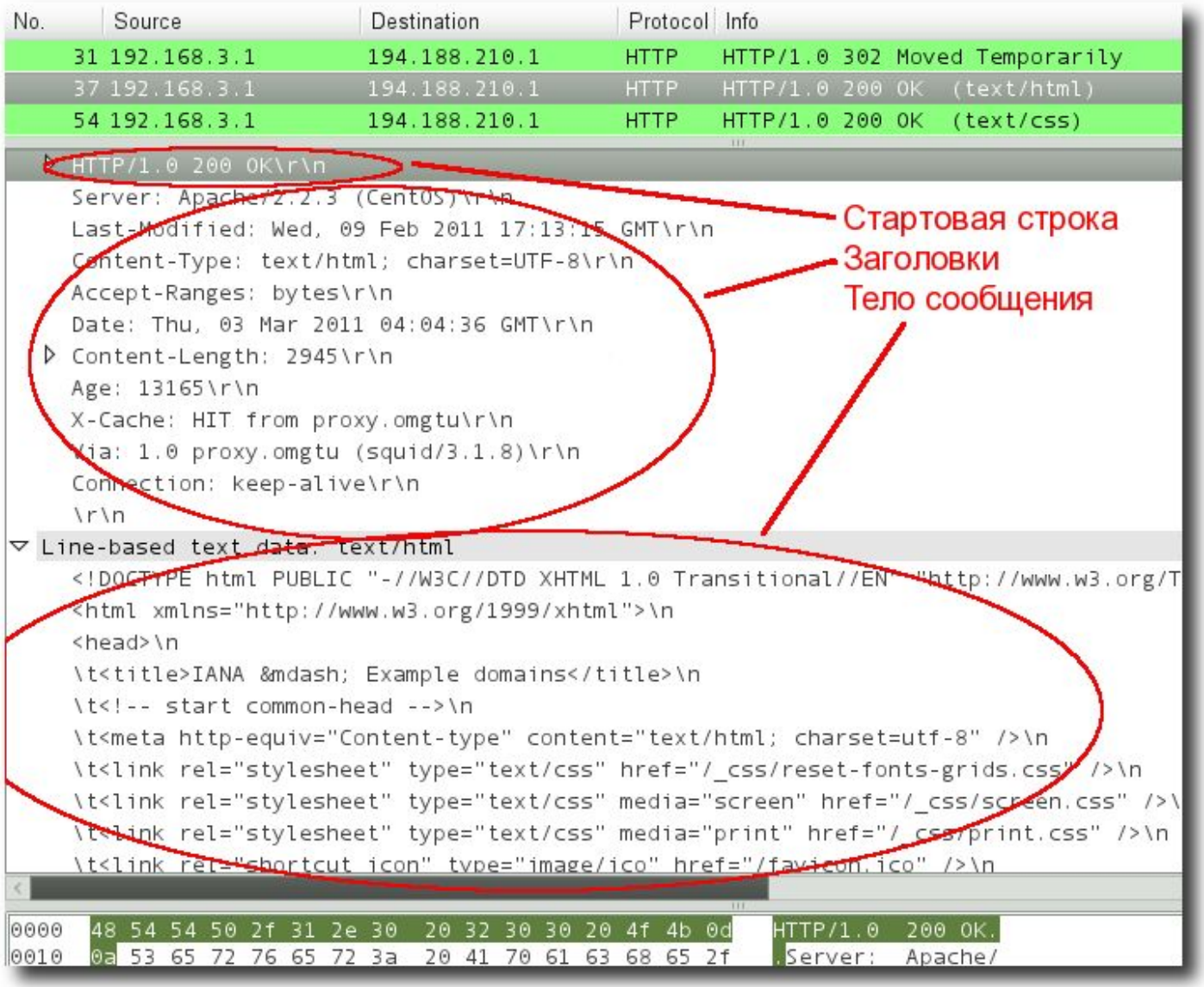
HTTP основывается на парадигме запросов/ответов. Запрашивающая программа (обычно она называется клиент) устанавливает связь с обслуживающей программой-получателем (обычно называется сервер) и посылает запрос серверу в следующей форме: метод запроса, URI, версия протокола, за которой следует MIME-подобное сообщение (MIME-Многоцелевое Расширение Почты Internet), содержащее управляющую информацию запроса, информацию о клиенте и, может быть, тело сообщения. Сервер отвечает сообщением, содержащим строку статуса (включая версию протокола и код статуса - успех или ошибка), за которой следует MIME-подобное сообщение, включающее в себя информацию о сервере, метаинформацию о содержании ответа, и, вероятно, само тело ответа.

# HTTP-протокол

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

- **Стартовая строка (Starting line)** - определяет тип сообщения;
- **Заголовки (Headers)** - разная полезная информация, которая характеризует тело сообщения, параметры передачи и прочие сведения;
- **Тело сообщения (Message Body)** - это сгенерированный html-код, который браузер потом будет отображать. Обязательно должно отделяться от заголовков пустой строкой.

# HTTP-протокол



# HTTP-протокол

Стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать.

Стартовые строки различаются для запроса и ответа. **Строка запроса** выглядит так:

```
Метод URI HTTP/Версия протокола
```

Пример запроса:

```
GET /web-programming/index.html HTTP/1.1
```

Стартовая **строка ответа** сервера имеет следующий формат:

```
HTTP/Версия КодСостояния [Пояснение]
```

Например, на предыдущий наш запрос клиентом данной страницы сервер ответил строкой:

```
HTTP/1.1 200 Ok
```

# HTTP-протокол. Методы.

Метод HTTP (англ. HTTP Method) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Названия метода чувствительны к регистру.

# HTTP-протокол. Методы.

| | |
|---|---|
| **OPTIONS** | Используется для определения возможностей веб-сервера или параметров соединения для конкретного ресурса. Предполагается, что запрос клиента может содержать тело сообщения для указания интересующих его сведений. Формат тела и порядок работы с ним в настоящий момент не определён. Сервер пока должен его игнорировать. Аналогичная ситуация и с телом в ответе сервера.<br><br>Для того чтобы узнать возможности всего сервера, клиент должен указать в URI звёздочку — «*». Запросы «OPTIONS * HTTP/1.1» могут также применяться для проверки работоспособности сервера (аналогично «пингованию») и тестирования на предмет поддержки сервером протокола HTTP версии 1.1.<br><br>Результат выполнения этого метода не кэшируется. |

# HTTP-протокол. Методы.

| | |
|---|---|
| **GET** | Используется для запроса содержимого указанного ресурса. С помощью метода GET можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса. Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «?»: GET /path/resource?param1=value1¶m2=value2 HTTP/1.1<br><br>Согласно стандарту HTTP, запросы типа GET считаются идемпотентными[4] — 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.<br><br>Кроме обычного метода GET, различают ещё условный GET и частичный GET. Условные запросы GET содержат заголовки If-Modified-Since, If-Match, If-Range и подобные. Частичные GET содержат в запросе Range. Порядок выполнения подобных запросов определён стандартами отдельно. |

# HTTP-протокол. Методы.

**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.

**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[4], 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.

# HTTP-протокол. Методы.

**PUT**

Применяется для загрузки содержимого запроса на указанный в запросе URI. Если по заданному URI не существовало ресурса, то сервер создаёт его и возвращает статус 201 (Created). Если же был изменён ресурс, то сервер возвращает 200 (Ok) или 204 (No Content). Сервер не должен игнорировать некорректные заголовки Content-* передаваемые клиентом вместе с сообщением. Если какой-то из этих заголовков не может быть распознан или не допустим при текущих условиях, то необходимо вернуть код ошибки 501 (Not Implemented).

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 соответствуют находящемуся по данному URI ресурсу.

Сообщения ответов сервера на метод PUT не кэшируются.

# HTTP-протокол. Методы.

| | |
|---|---|
| **PATCH** | Аналогично PUT, но применяется только к фрагменту ресурса. |
| **DELETE** | Удаляет указанный ресурс. |
| **TRACE** | Возвращает полученный запрос так, что клиент может увидеть, что промежуточные сервера добавляют или изменяют в запросе. |
| **LINK** | Устанавливает связь указанного ресурса с другими. |
| **UNLINK** | Убирает связь указанного ресурса с другими. |

# HTTP-протокол. Код состояния.

Код состояния информирует клиента о результатах выполнения запроса и определяет его дальнейшее поведение. Набор кодов состояния является стандартом, и все они описаны в соответствующих документах.

Каждый код представляется целым трехзначным числом. Первая цифра указывает на класс состояния, последующие - порядковый номер состояния. За кодом ответа обычно следует краткое описание на английском языке.

# HTTP-протокол. Код состояния.

| | |
|---|---|
| **1xx Informational** (Информационный) | 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. Прокси-сервера подобные сообщения должны отправлять дальше от сервера к клиенту.<br><br>Примеры ответов сервера:<br><br>`100 Continue (Продолжать)`<br>`101 Switching Protocols (Переключение протоколов)`<br>`102 Processing (Идёт обработка)` |
| | Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения. |

# HTTP-протокол. Код состояния.

**2xx Success**
(Успешно)

Сообщения данного класса информируют о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса сервер может ещё передать заголовки и тело сообщения.

Примеры ответов сервера:

```
200 OK (Успешно).
201 Created (Создано)
202 Accepted (Принято)
204 No Content (Нет содержимого)
206 Partial Content (Частичное содержимое)
```

# HTTP-протокол. Код состояния.

**3xx** Redirection (Перенаправление)

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»).

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.

Примеры ответов сервера:

```
300 Multiple Choices (Множественный выбор)
301 Moved Permanently (Перемещено навсегда)
304 Not Modified (Не изменялось)
```

# HTTP-протокол. Код состояния.

**4xx Client Error**
(Ошибка клиента)

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.

Примеры ответов сервера:

```
401 Unauthorized (Неавторизован)
402 Payment Required (Требуется оплата)
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)
```

# HTTP-протокол. Код состояния.

**5xx Server Error**
(Ошибка сервера)

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.

Примеры ответов сервера:

```
500 Internal Server Error (Внутренняя ошибка сервера)
502 Bad Gateway (Плохой шлюз)
503 Service Unavailable (Сервис недоступен)
504 Gateway Timeout (Шлюз не отвечает)
```

# HTTP-протокол. Заголовки.

**Заголовок HTTP (HTTP Header)** — это строка в HTTP-сообщении, содержащая разделённую двоеточием пару вида «параметр-значение». Как правило, браузер и веб-сервер включают в сообщения более чем по одному заголовку. Заголовки должны отправляться раньше тела сообщения и отделяться от него хотя бы одной пустой строкой (CRLF).

```
Server: Apache/2.2.3 (CentOS)
Last-Modified: Wed, 09 Feb 2011 17:13:15 GMT
Content-Type: text/html; charset=UTF-8
Accept-Ranges: bytes
Date: Thu, 03 Mar 2011 04:04:36 GMT
Content-Length: 2945
Age: 51
X-Cache: HIT from proxy.omgtu
Via: 1.0 proxy.omgtu (squid/3.1.8)
Connection: keep-alive

200 OK
```

# HTTP-протокол. Заголовки.

Все HTTP-заголовки разделяются на четыре основных группы:

- General Headers (Основные заголовки) — должны включаться в любое сообщение клиента и сервера.
- Request Headers (Заголовки запроса) — используются только в запросах клиента.
- Response Headers (Заголовки ответа) — присутствуют только в ответах сервера.
- Entity Headers (Заголовки сущности) — сопровождают каждую сущность сообщения.

**Сущности** (entity, в переводах также встречается название "объект") — это полезная информация, передаваемая в запросе или ответе. Сущность состоит из метаинформации (заголовки) и непосредственно содержания (тело сообщения).

# HTTP-протокол. Заголовки.

| Заголовок | Группа | Краткое описание |
|---|---|---|
| Allow | Entity | Список методов, применимых к запрашиваемому ресурсу. |
| Content-Encoding | Entity | Применяется при необходимости перекодировки содержимого (например, gzip/deflated). |
| Content-Language | Entity | Локализация содержимого (язык(и)) |
| Content-Length | Entity | Размер тела сообщения (в октетах) |
| Content-Range | Entity | Диапазон (используется для поддержания многопоточной загрузки или дозагрузки) |
| Content-Type | Entity | Указывает тип содержимого (mime-type, например text/html).Часто включает указание на таблицу символов локали (charset) |
| Expires | Entity | Дата/время, после которой ресурс считается устаревшим. Используется прокси-серверами |
| Last-Modified | Entity | Дата/время последней модификации сущности |
| Cache-Control | General | Определяет директивы управления механизмами кэширования. Для прокси-серверов. |
| Connection | General | Задает параметры, требуемые для конкретного соединения. |
| Date | General | Дата и время формирования сообщения |
| Pragma | General | Используется для специальных указаний, которые могут (опционально) применяется к любому получателю по всей цепочке запросов/ответов (например, pragma: no-cache). |
| Transfer-Encoding | General | Задает тип преобразования, применимого к телу сообщения. В отличие от Content-Encoding этот заголовок распространяется на все сообщение, а не только на сущность. |
| Via | General | Используется шлюзами и прокси для отображения промежуточных протоколов и узлов между клиентом и веб-сервером. |

# HTTP-протокол. Заголовки.

| | | |
|---|---|---|
| Warning | General | Дополнительная информация о текущем статусе, которая не может быть представлена в сообщении. |
| Accept | Request | Определяет применимые типы данных, ожидаемых в ответе. |
| Accept-Charset | Request | Определяет кодировку символов (charset) для данных, ожидаемых в ответе. |
| Accept-Encoding | Request | Определяет применимые форматы кодирования/декодирования содержимого (напр, gzip) |
| Accept-Language | Request | Применимые языки. Используется для согласования передачи. |
| Authorization | Request | Учетные данные клиента, запрашивающего ресурс. |
| From | Request | Электронный адрес отправителя |
| Host | Request | Имя/сетевой адрес [и порт] сервера. Если порт не указан, используется 80. |
| If-Modified-Since | Request | Используется для выполнения условных методов (Если-Изменился...). Если запрашиваемый ресурс изменился, то он передается с сервера, иначе - из кэша. |
| Max-Forwards | Request | Представляет механиз ограничения количества шлюзов и прокси при использовании методов TRACE и OPTIONS. |
| Proxy-Authorization | Request | Используется при запросах, проходящих через прокси, требующие авторизации |

# HTTP-протокол. Заголовки.

| | | |
|---|---|---|
| Referer | Request | Адрес, с которого выполняется запрос. Этот заголовок отсутствует, если переход выполняется из адресной строки или, например, по ссылке из js-скрипта. |
| User-Agent | Request | Информация о пользовательском агенте (клиенте) |
| Location | Response | Адрес перенаправления |
| Proxy-Authenticate | Response | Сообщение о статусе с кодом 407. |
| Server | Response | Информация о программном обеспечении сервера, отвечающего на запрос (это может быть как веб- так и прокси-сервер). |

# HTTP-протокол. Тело сообщения.

Тело HTTP сообщения (*message-body*), если оно присутствует, используется для передачи сущности, связанной с запросом или ответом. *Тело сообщения* (message-body) отличается от *тела сущности* (entity-body) только в том случае, когда при передаче применяется кодирование, указанное в заголовке Transfer-Encoding. В остальных случаях тело сообщения идентично телу сущности.

Заголовок Transfer-Encoding должен отправляться для указания любого кодирования передачи, примененного приложением в целях гарантирования безопасной и правильной передачи сообщения. Transfer-Encoding - это свойство сообщения, а не сущности, и оно может быть добавлено или удалено любым приложением в цепочке запросов/ответов.

Присутствие тела сообщения в запросе отмечается добавлением к заголовкам запроса поля заголовка Content-Length или Transfer-Encoding. Тело сообщения (message-body) может быть добавлено в запрос только когда метод запроса допускает тело объекта (entity-body).

Все ответы содержат тело сообщения, возможно нулевой длины, кроме ответов на запрос методом HEAD и ответов с кодами статуса 1xx (Информационные), 204 (Нет содержимого, No Content), и 304 (Не модифицирован, Not Modified).

Спасибо за внимание !

ВОПРОСЫ???