МДК.02.01 Программное обеспечение компьютерных сетей

3-курс

Занятие 16

Установка WEB-сервера

Как устроен и работает веб-сервер

Что такое веб-сервер?

С точки зрения обывателя – это некий черный ящик, который обрабатывает запросы браузера и выдает в ответ веб-страницы.

Технический специалист засыплет вас массой малопонятных терминов.

В итоге начинающим администраторам веб-серверов бывает порой трудно разобраться во всем многообразии терминов и технологий.

Действительно, область веб-разработки динамично развивается, но в основе многих современных решений лежат базовые технологии и принципы.

Как устроен и работает веб-сервер

Если не знаешь с чего начать, то начинать надо сначала.

Чтобы не запутаться во всем многообразии современных веб-технологий нужно обратиться к истории.

Это необходимо для того, чтобы понять:

- с чего начинался современный интернет,
- как развивались современные технологии и
- как они совершенствовались.

HTTP-сервер

На заре развития интернета сайты представляли собой простое хранилище специальным образом размеченных документов и некоторых связанных с ними данных:

- файлов,
- изображений и т.п.

Для того, чтобы документы могли ссылаться друг на друга и связанные данные был предложен специальный язык гипертекстовой разметки HTML, а для доступа к таким документам посредством сети интернет протокол HTTP.

И язык, и протокол, развиваясь и совершенствуясь, дожили до наших дней без существенных изменений.

HTTP-сервер

И только начавший приходить на смену принятому в 1999 году протоколу HTTP/1.1 протокол HTTP/2 несет кардинальные изменения с учетом требований современной сети.

Протокол HTTP реализован по клиент-серверной технологии и работает по принципу запрос-ответ без сохранения состояния.

Целью запроса служит некий ресурс, который определяется единым идентификатором ресурса – URI (Uniform Resource Identifier).

HTTP использует одну из разновидностей URI - URL (Uniform Resource Locator) – универсальный указатель ресурса, который помимо сведений о ресурсе определяет также его физическое местоположение

HTTP-сервер

Задача HTTP-сервера обработать запрос клиента и либо выдать ему требуемый ресурс, либо сообщить о невозможности это сделать.

Рассмотрим следующую схему:

HTTP-сервер

Пользователь посредством HTTP-клиента, чаще всего это браузер, запрашивает у HTTP-сервера некий URL, сервер проверяет и отдает соответствующий этому URL-файл, обычно это HTML-страница.

Полученный документ может содержать ссылки на связанные ресурсы, например, изображения.

Если их нужно отображать на странице, то клиент последовательно запрашивает их у сервера, кроме изображений также могут быть запрошены таблицы стилей, скрипты, исполняемые на стороне клиента и т.д.

Получив все необходимые ресурсы браузер обработает их согласно кода HTML-документа и выдаст пользователю готовую страницу.

HTTP-сервер

Как уже многие догадались, под именем HTTP-сервера в данной схеме находится сущность, которая более известна сегодня под названием **веб-сервер**.

Основная цель и задача веб-сервера - обработка HTTP-запросов и возврат пользователю их результатов.

Веб-сервер не умеет самостоятельно генерировать контент и работает только со статическим содержимым.

Это актуально и для современных веб-серверов, несмотря на все богатство их возможностей.

HTTP-сервер

Долгое время одного веб-сервера было достаточно для реализации полноценного сайта.

Но по мере роста сети интернет возможностей статического HTML стало остро не хватать.

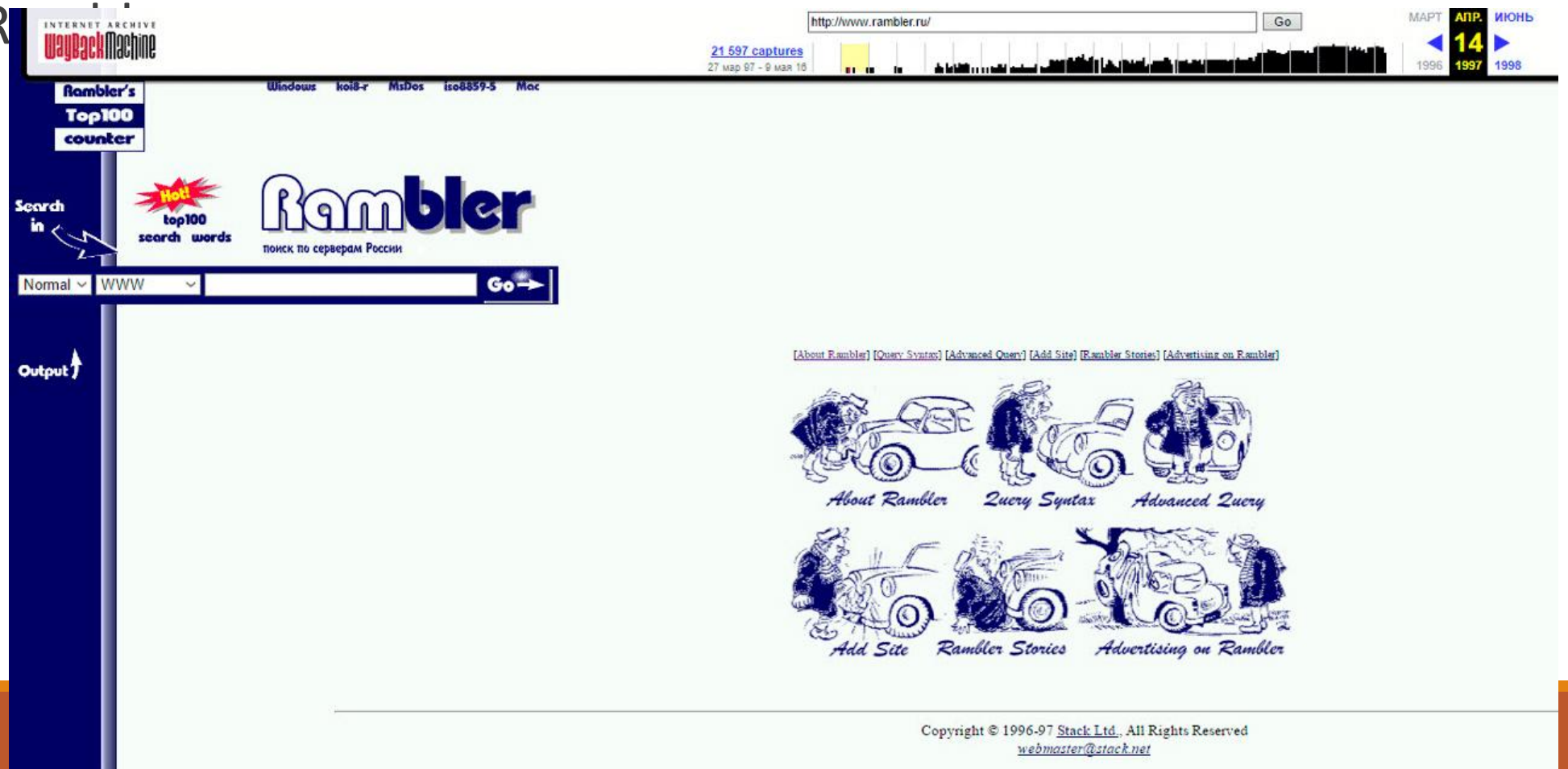
Простой пример.

Каждая статическая страница самодостаточна и должна содержать ссылки на все связанные с ней ресурсы.

При добавлении новых страниц ссылки на них потребуется добавить на уже существующие страницы, иначе пользователь никогда не сможет на них попасть.

HTTP-сервер

Сайты того времени вообще мало походили на современные, например, ниже показан вид одного из пионеров русскоязычного интернета, сайт компании R



HTTP-сервер

А переход по любой из ссылок вообще может привести современного пользователя в недоумение, вернуться назад с такой страницы не представляется возможным, кроме как через нажатие одноименной кнопки в браузере.

Query Syntax

Multiple Words
You can use several words in one query, separated by a space.
deep purple

Logical Words
Words may be separated by the logical words **'AND'** and **'OR'** to indicate the logical operation. The symbols **'&'** and '|' can be used instead.
deep and purple & fireball

Shift
Each word can contain capital and small letters (case insensitive).
Deep Or Purple

Partial Words
You can use the metasymbols '*' and '?' to indicate any part of the word (*) or unknown letter (?).
Deep and Purp*

Word Groups
Words can be grouped by using the symbols '(' and ')'.
(Deep and Purp*) or Beatles

HTTP-сервер

Попытка создать что-то более-менее похожее на современный сайт очень скоро превращалась в нарастающий объем работ по внесению изменений в уже существующие страницы.

Ведь если мы что-то поменяли в общей части сайта, например, логотип в шапке, то нам нужно внести это изменение на все существующие страницы.

А если мы изменили путь к одной из страниц или удалили ее, то нам надо будет найти все ссылки на нее и изменить или удалить их.

Поэтому следующим шагом в развитии веб-серверов стала поддержка технологии **включения на стороне сервера** – **SSI (Server Side Includes)**.

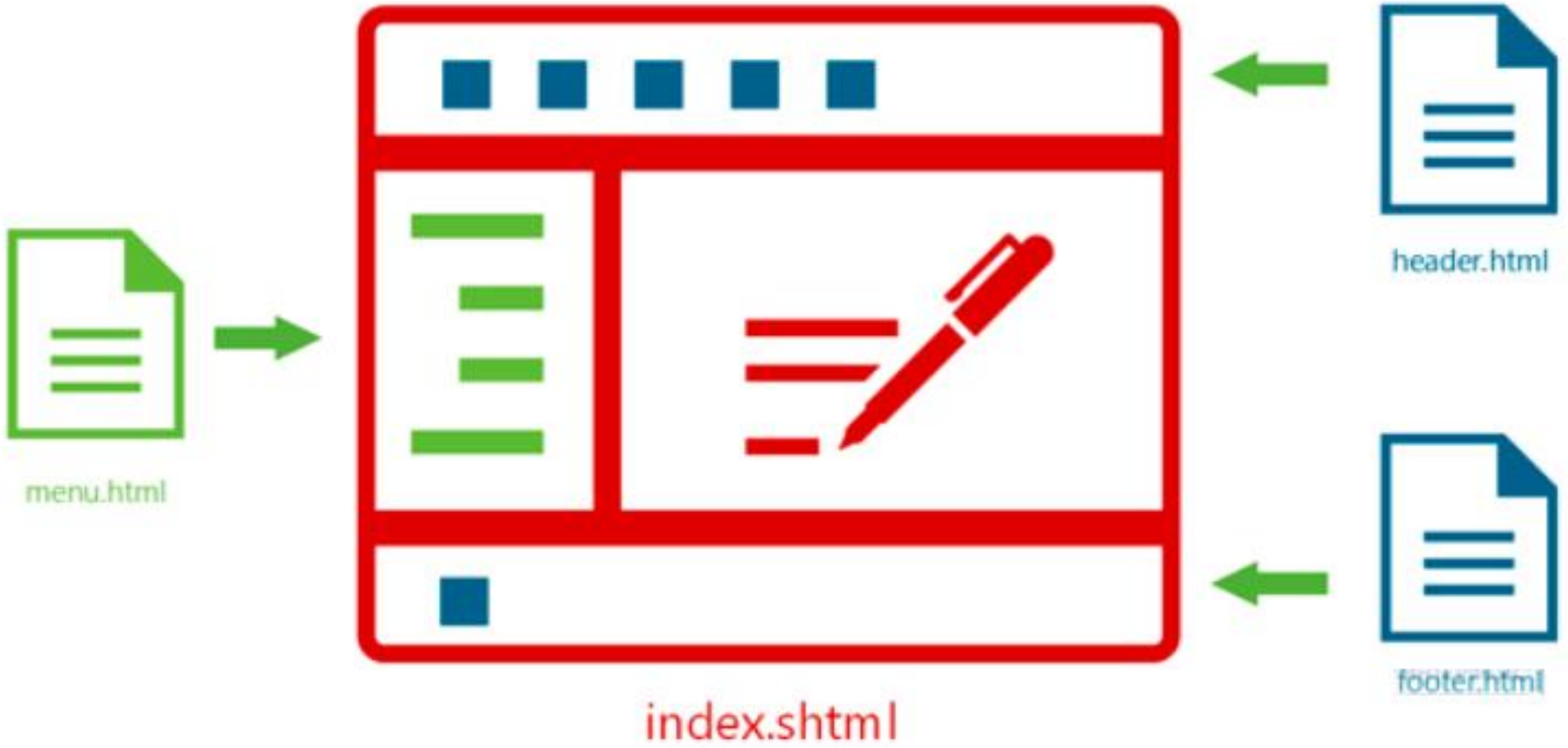
HTTP-сервер

Она позволяла включать в код страницы содержимое иных файлов, что давало возможность вынести повторяющиеся элементы, такие как:

- шапка,
- подвал,
- меню и
- многие другие

в отдельные файлы и просто подключать при окончательной сборке страницы.

HTTP-сервер



HTTP-сервер

Теперь, чтобы изменить логотип или пункт меню изменения надо будет внести всего лишь в один файл, вместо правки всех существующих страниц.

Кроме того, SSI позволял выводить на страницы некоторое динамическое содержимое, например, актуальную дату и выполнять несложные условия и работать с переменными.

Это был значительный шаг вперед, облегчавший труд вебмастеров и повышавший удобство пользователей.

Однако реализовать по-настоящему динамический сайт данные технологии все еще не позволяли.

Стоит отметить, что SSI активно применяется и сегодня, там, где в код страницы нужно вставить некий статический контент, прежде всего благодаря простоте и нетребовательности к ресурсам.

CGI

Следующим шагом в развитии веб-технологии стало появление специальных программ (скриптов) выполняющих обработку запроса пользователей на стороне сервера.

Чаще всего они пишутся на скриптовых языках, первоначально это был Perl, сегодня пальму лидерства удерживает PHP.

Постепенно возник целый класс программ – системы управления контентом – CMS (Content management system), которые представляют полноценные веб-приложения способные обеспечить динамическую обработку запросов пользователя.

CGI

Рассмотрим один важный момент.

Веб-серверы не умели и не умеют выполнять скрипты.

Их задача – это выдача статического содержимого.

Здесь на первый план выходит новая сущность – сервер приложений.

Он представляет собой интерпретатор скриптовых языков и с помощью которого работают написанные на них веб-приложения.

Для хранения данных обычно используются СУБД (система управлениями базами данных).

CGI

Её существование обусловлено необходимостью доступа к большому количеству взаимосвязанной информации.

Однако сервер приложений не умеет работать с протоколом HTTP и обрабатывать пользовательские запросы, так как это задача веб-сервера.

Чтобы обеспечить их взаимодействие был разработан **общий интерфейс шлюза** – **CGI** (*Common Gateway Interface*).

CGI

CGI

Следует четко понимать, CGI – это не программа и не протокол, это именно интерфейс, т.е. совокупность способов взаимодействия между приложениями.

Также не следует путать термин CGI с понятием CGI-приложения или CGI-скрипта, которыми обозначают программу (скрипт) поддерживающую работу через интерфейс CGI.

Для передачи данных используются стандартные потоки ввода-вывода, от веб-сервера к CGI-приложению данные передаются через **stdin**, принимаются назад через **stdout**, для передачи сообщений об ошибках используется **stderr**.

CGI

Рассмотрим процесс работы такой системы подробнее.

Получив запрос от браузера пользователя веб-сервер определяет, что запрошено динамическое содержимое и формирует специальный запрос, которой через интерфейс CGI направляет веб-приложению.

При его получении приложение запускается и выполняет запрос, результатом которого служит HTML-код динамически сформированной страницы, который передается назад веб-серверу, после чего приложение завершает свою работу.

CGI

Еще одно важное отличие динамического сайта - его страницы физически не существуют в том виде, который отдается пользователю.

Фактически имеется веб-приложение, т.е. набор скриптов и шаблонов, и база данных, которая хранит материалы сайта и служебную информацию, отдельно располагается статическое содержимое:

- картинки,
- java-скрипты,
- файлы.

CGI

Получив запрос веб-приложение извлекает данные из БД и заполняет ими указанный в запросе шаблон.

Результат отдается веб-серверу, который дополняет сформированную таким образом страницу статическим содержимым:

- изображениями,
- скриптами,
- стилями

и отдает ее браузеру пользователя.

CGI

Сама страница при этом нигде не сохраняется, разве что в кэше, и при получении нового запроса произойдет повторная генерация страницы.

К достоинствам CGI можно отнести:

- языковую независимость и
- архитектурную независимость.

CGI-приложение может быть написано на любом языке и одинаково хорошо работать с любым веб-сервером.

Учитывая простоту и открытость стандарта это привело к бурному развитию веб-приложений.

CGI

Однако, кроме достоинств, CGI обладает и существенными недостатками.

Основной из них: высокие накладные расходы на запуск и остановку процесса.

Это влечет за собой повышенные требования к аппаратным ресурсам и невысокую производительность.

А использование стандартных потоков ввода-вывода ограничивает возможности масштабирования и обеспечения высокой доступности, так как требует, чтобы веб-сервер и сервер приложений находились в пределах одной системы.

На текущий момент CGI практически не применяется, так как ему на смену пришли более совершенные технологии.

FastCGI

Как следует из названия, основной целью разработки данной технологии было повышение производительности CGI.

Являясь ее дальнейшим развитием FastCGI представляет собой клиент-серверный протокол для взаимодействия веб-сервера и сервера приложений, обеспечивающий:

- высокую производительность и
- более высокую безопасность.

FastCGI

FastCGI устраняет основную проблему CGI – повторный запуск процесса веб-приложения на каждый запрос, FastCGI процессы запущены постоянно, что позволяет существенно экономить время и ресурсы.

Для передачи данных вместо стандартных потоков используются UNIX-сокеты или TCP/IP, что позволяет размещать веб-сервер и сервера приложений на разных хостах.

Таким образом обеспечивается масштабирование и/или высокая доступность системы.

FastCGI

FastCGI

Также мы можем запустить на одном компьютере несколько FastCGI процессов, которые могут обрабатывать запросы параллельно, либо иметь различные настройки или версии скриптового языка.

Например, можно одновременно иметь несколько версий PHP для разных сайтов, направляя их запросы разным FastCGI процессам.

Для управления FastCGI процессами и распределением нагрузки служат менеджеры процессов, они могут быть как частью веб-сервера, так и отдельными приложениями.

Популярные веб-сервера Apache и Lighttpd имеют встроенные менеджеры FastCGI процессов, в то время как Nginx требует для своей работы с FastCGI внешний менеджер.

PHP-FPM и spawn-fcgi

FastCGI устраняет основную проблему CGI – повторный запуск процесса веб-приложения на каждый запрос, FastCGI процессы запущены постоянно, что позволяет существенно экономить время и ресурсы.

Для передачи данных вместо стандартных потоков используются UNIX-сокеты или TCP/IP, что позволяет размещать веб-сервер и сервера приложений на разных хостах.

Таким образом обеспечивается масштабирование и/или высокая доступность системы.

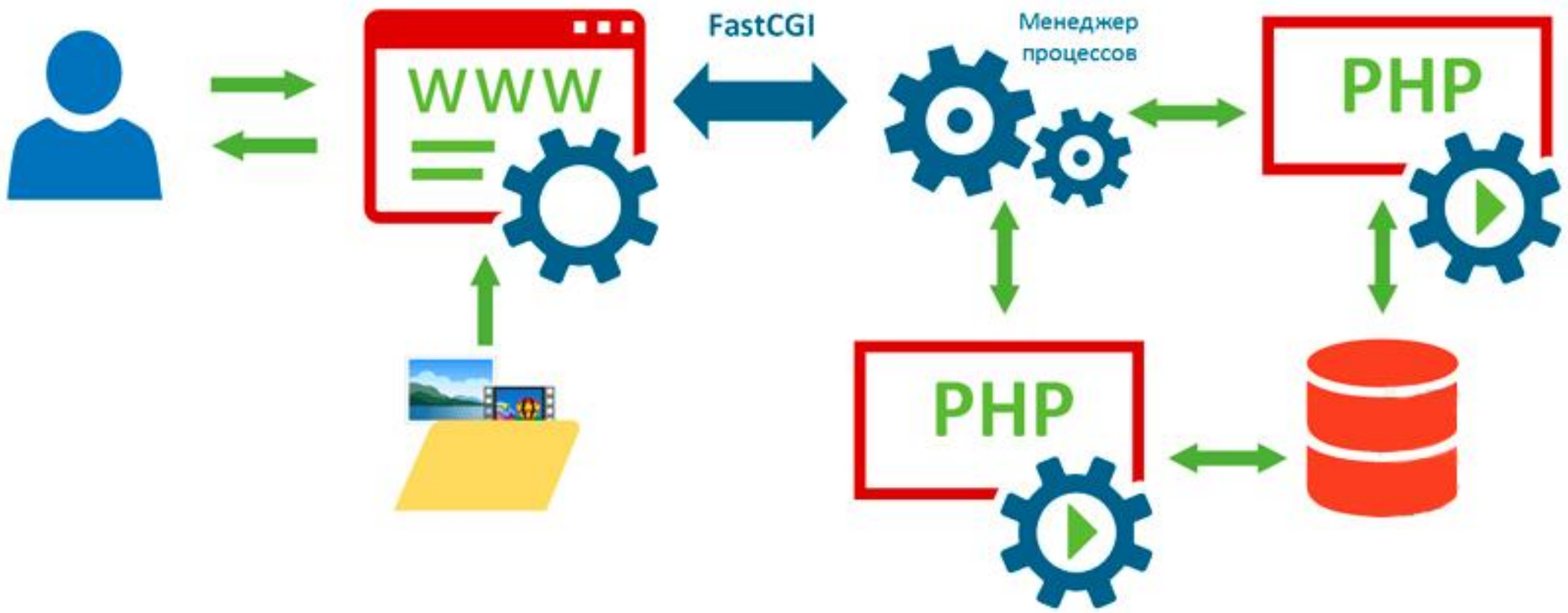
PHP-FPM и spawn-fcgi

Spawn-fcgi является частью проекта Lighttpd, но в состав одноименного веб-сервера не входит, по умолчанию Lighttpd использует собственный, более простой, менеджер процессов.

Разработчики рекомендуют использовать его в случаях, когда вам нужно управлять FastCGI процессами расположенными на другом хосте, либо требуются расширенные настройки безопасности.

Внешние менеджеры позволяют изолировать каждый FastCGI процесс в своем chroot (смена корневого каталога приложения без возможности доступа за его пределы), отличном как от chroot иных процессов, так и от chroot веб-сервера.

PHP-FPM и spawn-fcgi



PHP-FPM и spawn-fcgi

И, как мы уже говорили, позволяют работать с FastCGI приложениями расположенными на других серверах через TCP/IP, в случае локального доступа следует выбирать доступ через UNIX-сокет, как быстрый тип соединения.

Если снова посмотреть на схему, то мы увидим, что у нас появился новый элемент - менеджер процессов, который является посредником между веб-сервером и серверами приложений.

Это несколько усложняет схему, так как настраивать и сопровождать приходится большее количество служб, но в тоже время открывает более широкие возможности, позволяя настроить каждый элемент сервера четко под свои задачи.

PHP-FPM и spawn-fcgi

На практике, выбирая между встроенным менеджером и внешним здраво оцените ситуацию и выбирайте именно тот инструмент, который наиболее подходит вашим запросам.

Например, создавая простой сервер для нескольких сайтов на типовых движках применение внешнего менеджера будет явно излишним.

Хотя никто не навязывает вам своей точки зрения.

Linux тем и хорош, что каждый может, как из конструктора, собрать именно то, что ему надо.

SCGI, PCGI, PSGI, WSGI и прочие

Погружаясь в тему веб-разработки, вы непременно будете встречаться с упоминанием различных CGI-технологий, наиболее популярные из которых мы перечислили в заголовке.

От такого многообразия можно и растеряться, но если вы внимательно прочитали начало нашей статьи, то знаете, как работает CGI и FastCGI, а, следовательно, разобраться с любой из этих технологий не составит для вас труда.

Несмотря на различия в реализациях того или иного решения базовые принципы остаются общими.

SCGI, PCGI, PSGI, WSGI и прочие

Все эти технологии предоставляют интерфейс шлюза (Gateway Interface) для взаимодействия веб-сервера с сервером приложений.

Шлюзы позволяют развязать между собой среды веб-сервера и веб-приложения, позволяя использовать любые сочетания без оглядки на возможную несовместимость.

Проще говоря, неважно, поддерживает ли ваш веб-сервер конкретную технологию или скриптовый язык, главное, чтобы он умел работать с нужным типом шлюза.

И раз уж мы перечислили в заголовке целый набор аббревиатур, то пройдем по ним более подробно.

SCGI, PCGI, PSGI, WSGI и прочие

SCGI (*Simple Common Gateway Interface*) - **простой общий интерфейс шлюза** - разработан как альтернатива CGI и во многом аналогичен FastCGI, но более прост в реализации.

Все, о чем мы рассказывали применительно к FastGCI справедливо и для SCGI.

PCGI (*Perl Common Gateway Interface*) - библиотека Perl для работы с интерфейсом CGI, долгое время являлась основным вариантом работы с Perl-приложениями через CGI, отличается хорошей производительностью (насколько это применимо к CGI) при скромных потребностях в ресурсах и неплохой защиты от перегрузки.

SCGI, PCGI, PSGI, WSGI и прочие

PSGI (*Perl Web Server Gateway Interface*) - технология взаимодействия веб-сервера и сервера приложений для Perl.

Если PCGI представляет собой инструмент для работы с классическим CGI интерфейсом, то PSGI более напоминает FastCGI.

PSGI-сервер представляет среду для выполнения Perl-приложений которая постоянно запущена в качестве службы и может взаимодействовать с веб-сервером через TCP/IP или UNIX-сокеты и предоставляет Perl-приложениям те же преимущества, что и FastCGI.

SCGI, PCGI, PSGI, WSGI и прочие

WSGI (*Web Server Gateway Interface*) - еще один специфичный интерфейс шлюза, предназначенный для взаимодействия веб-сервера с сервером приложений для программ, написанных на языке Phyton.

Как несложно заметить, все перечисленные нами технологии являются в той или иной степени аналогами CGI/FastCGI, но для специфичных областей применения.

Приведенных нами данных будет вполне достаточно для общего понимания принципа и механизмов их работы, а более глубокое их изучение имеет смысл только при серьезной работе с указанными технологиями и языками.

Сервер приложений как модуль Apache

Если раньше мы говорили о некоем абстрактном веб-сервере, то теперь речь пойдет о конкретном решении и дело здесь не в наших предпочтениях.

Среди веб-серверов **Apache** занимает особое место, в большинстве случаев, когда говорят о веб-сервере на платформе **Linux**, да и о веб-сервере вообще, то подразумеваться будет именно **Apache**.

Можно сказать, что это своего рода веб-сервер «по умолчанию».

Возьмите любой массовый хостинг – там окажется **Apache**, возьмите любое веб-приложение – настройки по умолчанию выполнены под **Apache**.

Сервер приложений как модуль Apache

Да, с технологической точки зрения Apache не является венцом технологий, но именно он представляет золотую середину, прост, понятен, гибок в настройках, универсален.

Если вы делаете первые шаги в сайтостроении, то Apache ваш выбор.

Здесь нас могут упрекнуть, что **Apache** уже давно неактуален, все уже поставили **Nginx** и т.д. и т.п., поэтому поясним данный момент более подробно.

Все популярные CMS из коробки сконфигурированы для использования совместно с Apache, это позволяет сосредоточить все внимание на работу именно с веб-приложением, исключив из возможного источника проблем веб-сервер.

Сервер приложений как модуль Apache

Все популярные среди новичков форумы тоже подразумевают в качестве веб-сервера Apache и большинство советов и рекомендаций будут относиться именно к нему.

В тоже время альтернативные веб-сервера как правило требуют более тонкой и тщательной настройки, как со стороны веб-сервера, так и со стороны веб-приложения.

При этом пользователи данных продуктов обычно гораздо более опытны и типовые проблемы новичков в их среде не обсуждаются.

В итоге может сложиться ситуация, когда ничего не работает и спросить не у кого.

С Apache такого гарантированно не произойдет.

Сервер приложений как модуль Apache

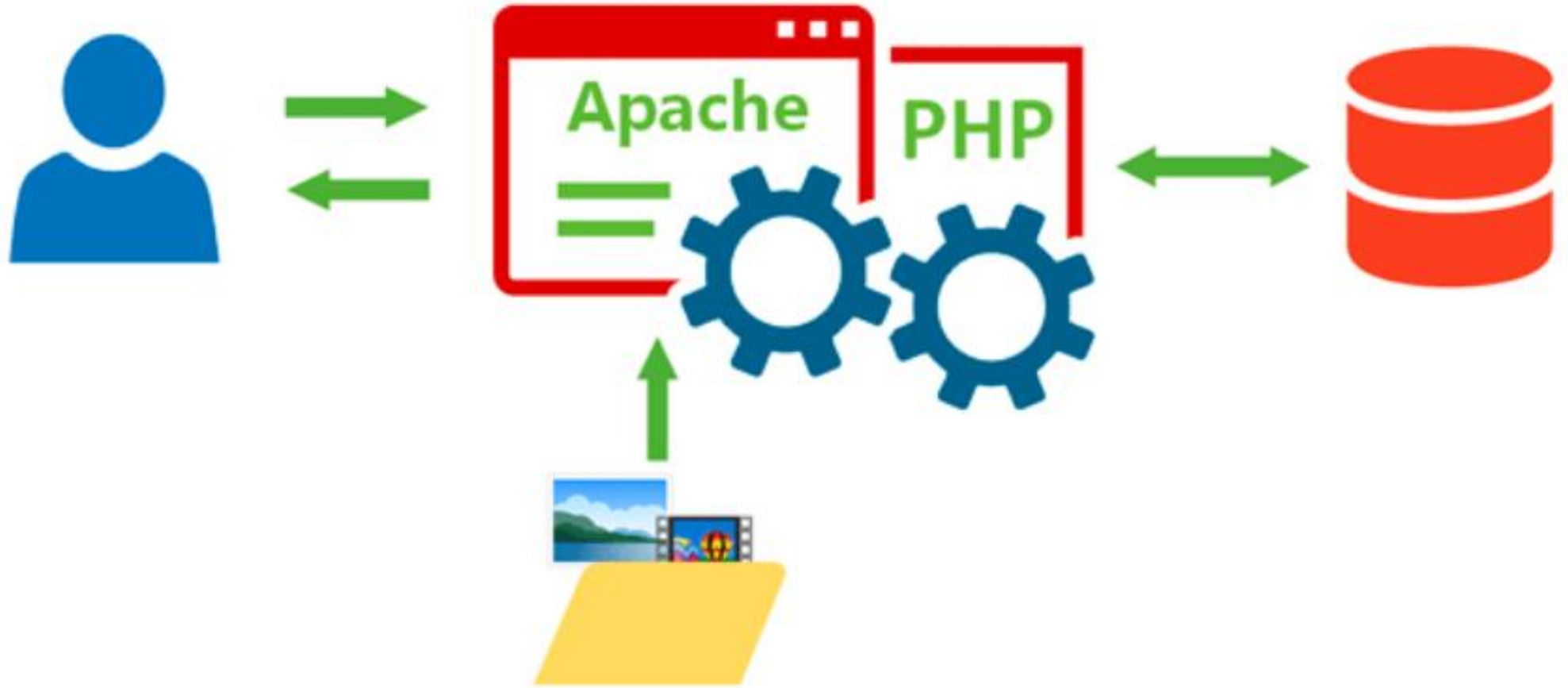
Собственно, что такого сделали разработчики Apache, что позволило их детищу занять особое место?

Ответ достаточно прост: они пошли своим путем.

В то время как CGI предлагал абстрагироваться от конкретных решений, сосредоточившись на универсальном шлюзе, в Apache поступили по-другому – максимально интегрировали веб-сервер и сервер приложений.

Действительно, если запустить сервер приложений как модуль веб-сервера в общем адресном пространстве, то мы получим гораздо более простую схему:

Сервер приложений как модуль Apache



Сервер приложений как модуль Apache

Какие преимущества это дает?

Первое преимущество.

Чем проще схема и меньше в ней элементов, тем проще и дешевле сопровождать ее и обслуживать, тем меньше в ней точек отказа.

Если для единичного сервера это может быть не так важно, то в рамках хостинга это весьма значительный фактор.

Сервер приложений как модуль Apache

Второе преимущество – производительность.

Снова огорчим поклонников **Nginx**, благодаря работе в едином адресном пространстве, по производительности сервера приложений Apache + mod_php всегда будет на 10-20% быстрее любого иного веб-сервера + FastCGI (или иное CGI решение).

Но также следует помнить, что скорость работы сайта обусловлена не только производительностью сервера приложений, но и рядом иных условий, в которых альтернативные веб-сервера могут показывать значительно лучший результат.

Сервер приложений как модуль Apache

Но есть еще одно, достаточно серьезное преимущество (третье преимущество) – это возможность настройки сервера приложений на уровне отдельного сайта или пользователя.

Давайте вернемся немного назад: в FastCGI/CGI схемах сервер приложений – это отдельная служба, со своими, отдельными, настройками, которая даже может работать от имени другого пользователя или на другом хосте.

С точки зрения администратора одиночного сервера или какого-нибудь крупного проекта – это отлично, но для пользователей и администраторов хостинга – не очень.

Сервер приложений как модуль Apache

Развитие интернета привело к тому, что количество возможных веб-приложений:

- CMS,
- скриптов,
- фреймворков и т.п.

стало очень велико, а низкий порог вхождения привлек к сайтостроению большое количество людей без специальных технических знаний.

В тоже время разные веб-приложения могли требовать различной настройки сервера приложений.

Как быть? Каждый раз обращаться в поддержку?

Сервер приложений как модуль Apache

Решение нашлось довольно просто.

Так как сервер-приложений теперь часть веб-сервера, то можно поручить последнему управлять его настройками.

Традиционно для управления настройками Apache помимо конфигурационных файлов применялись файлы httaccess, которые позволяли пользователям писать туда свои директивы и применять их к той директории, где расположен данный файл и ниже, если там настройки не перекрываются своим файлом httaccess.

В режиме mod_php данные файлы позволяют также изменять многие опции PHP для отдельного сайта или директории.

Сервер приложений как модуль Apache

Для принятия изменений не требуется перезапуск веб-сервера и в случае ошибки перестанет работать только этот сайт (или его часть).

Кроме того, внести изменения в простой текстовый файл и положить его в папку на сайте под силу даже неподготовленным пользователям и безопасно для сервера в целом.

Сочетание всех этих преимуществ и обеспечило Apache столь широкое применение и статус универсального веб-сервера.

Другие решения могут быть быстрее, экономичнее, лучше, но они всегда требуют настройки под задачу, поэтому применяются в основном в целевых проектах, в массовом сегменте безальтернативно доминирует Apache.

Сервер приложений как модуль Apache

Поговорив о достоинствах, перейдем к недостаткам.

Некоторые из них просто являются обратной стороной медали.

Тот факт, что сервер приложений является частью веб-сервера дает плюсы в производительности и простоте настройки, но в тоже время ограничивает нас как с точки зрения безопасности.

Сервер приложений всегда работает от имени веб-сервера, так и в гибкости системы, мы не можем разнести веб-сервер и сервер приложений на разные хосты.

Кроме того мы не можем использовать серверы с разными версиями скриптового языка или разными настройками.

Сервер приложений как модуль Apache

Второй минус – это более высокое потребление ресурсов.

В схеме с CGI сервер приложений генерирует страницу и отдает ее веб-серверу, освобождая ресурсы, связка Apache + mod_php держит ресурсы сервера приложений занятыми до тех пор, пока веб-сервер не отдаст содержимое страницы клиенту.

Если клиент медленный, то ресурсы будут заняты на все время его обслуживания.

Именно поэтому перед Apache часто ставят Nginx, который играет роль быстрого клиента, это позволяет Apache быстро отдать страницу и освободить ресурсы, переложив взаимодействие с клиентом на более экономичный Nginx.

Заключение

Охватить коротко весь спектр современных технологий невозможно.

Поэтому мы сосредоточились только на основных из них, некоторые вещи умышленно оставив за кадром, а также прибегли к существенным упрощениям.

Несомненно, начав работать в этой области вам потребуется более глубокое изучение темы, но для того, чтобы воспринимать новые знания нужен определенный теоретический фундамент, который хотелось заложить данным материалом.

Список литературы:

1. Олифер В. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд., СПб: Питер, 2015 г.
2. https://interface31.ru/tech_it/2016/03/kak-ustroen-i-rabotaet-web-server.html

Благодарю за внимание!

Преподаватель: Солодухин Андрей Геннадьевич

Электронная почта: asoloduhin@kait20.ru