УТВЕРЖДАЮ

Генеральный директор
ООО «ИЭЭФ»

_____ / Алешина А.В. /

М.П.

«02» сентября 2024 г.

РАБОЧАЯ ПРОГРАММА - ОНЛАЙН-ЛЕКЦИЯ (ВЕБИНАР)
по учебной программе
Дополнительной профессиональной образовательной программе –
программе повышения квалификации «ВЕБ-ДИЗАЙН И РАЗРАБОТКА»
Тема 3: «Веб-дизайн и программирование»

Тип: Онлайн-лекция (вебинар)
Объем часов: 24 ч.

П Л А Н

1. ВВЕДЕНИЕ В WEB-КОНСТРУИРОВАНИЕ
2. ПРОГРАММИРОВАНИЕ НА PHP.
3. СТИЛЕВОЕ ОФОРМЛЕНИЕ HTML-ДОКУМЕНТОВ
4. СЦЕНАРИИ JAVASCRIPT
5. WEB-СЕРВЕРЫ
6. БЕЗОПАСНОСТЬ И ШИФРОВАНИЕ

1. ВВЕДЕНИЕ В WEB-КОНСТРУИРОВАНИЕ

При реализации проектных творческих работ и коллективного общения (в том числе удаленного) является создание собственных web-сайтов. При этом важно не забывать, что web-технология позволяет не только представить результаты своей работы на всеобщее обозрение в сети Интернет, но и создать такой сайт в рамках локальной сети или подготовить свои медиа материалы в виде локальной копии сайта, распространяемой на любом носителе и просматриваемой на компьютере без подключения к Интернету. В том числе в форме такой «локальной копии сайта» может быть разработана интерактивная интерфейсная (пользовательская) оболочка, например, существенно повышающая удобство работы с компакт-диском, на котором учитель или учащиеся размещают разработанные ими учебные, демонстрационные или дополнительные материалы.

Для создания сайта или «интерфейсной оболочки могут использоваться различные инструментальные средства web-редактирования, прежде всего — специальные редактирующие программы и программные пакеты, реализующие принципы «визуального редактирования» web-страниц и сайтов. Кроме того, современные версии офисного пакета

Microsoft Office и ряд других прикладных программ позволяют сохранять обрабатываемые в них документы в web-совместимом формате, позволяющем размещать их на web-сайте и просматривать через сети Интернет. Однако для понимания

принципов web-конструирования по-прежнему необходимо знание языка гипертекстовой разметки HTML — как основы основ web-представления текстовых и медиа материалов.

Данная информация призвана предназначена для обучения основным приемам создания web-сайтов с использованием языка HTML, а также сопутствующих web-технологий (стилевые таблицы, слои, фреймы, скриптовое программирование и др.).

Предполагается, что слушатели обладают навыками работы в среде операционной системы Windows, работы в многоуровневой файловой системе, а также ввода и редактирования текста хотя бы в простейших текстовых редакторах (Блокнот).

Основные определения

Язык HTML (HyperText Markup Language — «язык гипертекстовой разметки») — набор команд (тегов), вставляемых в текст web-страницы и определяющих форматирование абзацев, вид шрифта, ссылки на внешние файлы, другие web-страницы или части той же web-страницы.

Гипертекст — «многомерный» текстовый документ (либо объединение нескольких текстовых документов), построенный по принципу «нелинейного» структурирования материала за счет применения гипертекстовых ссылок (как внутренних, в пределах данного документа, так и перекрестных, указывающих на другие документы, в том числе размещенные на других компьютерах сети, или на их фрагменты), позволяющих одним щелчком мыши перемещаться из одной «смысловой точки» гипертекстового документа в другую.

Контейнер — конструкция из парных «открывающего» и «закрывающего» тегов. При этом действие открывающего тега и его параметров распространяется на весь текстовый фрагмент, заключенный внутри контейнера.

Метатеги (теги, начинающиеся со слова META) — размещаются внутри контейнера HEAD и служат для указания различных дополнительных сведений о содержимом страницы. На экран браузера текст метатегов не выводится.

Среди метатегов можно выделить по крайней мере три группы:

-команды управления отображением страницы — прежде всего указание кодовой таблицы кириллицы;

-команды, предназначенные для автономных поисковых программ, рассылаемых по сети поисковыми серверами для сбора сведений о новых web-сайтах;

-теги, предоставляющие дополнительную информацию о содержании страницы

Типовая структура метатега обычно включает в себя два параметра: NAME (или HTTP-EQUIV) и CONTENT, первый из которых определяет название (тип) метатега, определяющее его назначение, а второй— собственно значение метатега.

Web-страница (интернет-страница, WWW-страница), web-документ — обособленный документ, хранящийся в отдельном файле на диске и включающий в себя текст, отображаемый на экране во время просмотра в браузере, а также теги языка HTML, дополненный хранящимися в отдельных файлах и подгружаемыми дополнительно по размещенным в тексте страницы ссылкам мультимедиа-иллюстрациями (рисунками, видео, аудио- и пр.).

Сайт, web-сайт — набор web-страниц, составляющих единую подборку и связанных между собой перекрестными ссылками. Одна из этих страниц является основной (головной, индексной, стартовой) и автоматически выдается на просмотр пользователю, указавшему в браузере только адрес сайта, тогда как все остальные страницы, как правило, вызываются из основной с помощью гиперссылок.

Сервер — обособленный компьютер, подключенный к сети Интернет (чаще всего круглосуточно) и имеющий собственный адрес (URL), на диске которого расположены один или несколько сайтов. Серверы могут быть служебными (пример — поисковые системы), принадлежащими организации или частному лицу (обычно такой сервер

содержит только один сайт, и тогда эти два понятия можно считать идентичными) или общедоступными для размещения сайтов всех желающих (например, сервер www.narod.ru).

Еще одно значение понятия сервер предполагает кроме самого компьютера как узла сети также установленное на нем специализированное программное обеспечение, поддерживающее информационный обмен с пользователями (web-сервер, FTP-сервер, почтовый сервер и т. д.).

Локальный компьютер (терминал) — компьютер пользователя, работающего в Интернете. При посещении несколькими пользователями одного и того же сервера размещенные на его диске web-страницы и другие необходимые файлы данных (графика, звук, цифровое видео и пр.) пересылаются (копируются) по сети на диски локальных компьютеров (локальные диски) и отображаются на экранах, запущенных на локальных компьютерах браузеров.

«Домашняя страница» (HomePage) — своего рода «наследие» прежних времен, когда предполагалось создание частными пользователями лишь простейших одностраничных web-страниц — своего рода «сетевых визитных карточек» их владельцев. В настоящее время этот термин используется для обозначения указанной в настройках браузера некоторой (назначенной по умолчанию или самим пользователем) «изначальной» web-страницы, автоматически загружаемой при запуске браузера (это может быть любая, чаще всего головная страница какого-либо сайта в сети Интернет, произвольный HTML-документ на жестком диске локального компьютера либо «пустая страница» — web-документ с белым фоном, не содержащий никакого текста).

Браузер (web-браузер) — программа, запускаемая на локальном компьютере для просмотра web-страниц, их сохраненных на локальном диске копий, а также любых документов, созданных с использованием языка HTML. Сегодня наиболее популярными являются браузеры Microsoft Edge, Opera, Mozilla, Firefox и др.

Фрейм — «вложенное» окно («подокно», поле), создаваемое в окне браузера при помощи специальной конструкции HTML-тегов, содержимое которого (отдельная web-страница, рисунок и пр.) полностью независимо от содержимого остальных (соседних) фреймов.

Каскадная стилевая разметка (CSS, Cascading StyleSheet — «каскадная стилевая таблица») — технология, позволяющая разделить содержание и оформление (форматирование) web-страницы путем создания «шаблонных» (типовых) наборов параметров оформления тех или иных фрагментов текста (заголовков, ссылок, таблиц и пр.) — стилей. При указании для любого абзаца того или иного стиля этот абзац автоматически приобретает все указанное в данном стиле оформление, а любое изменение стиля немедленно отражается на оформлении всех абзацев, которым назначен этот стиль. Набор стилей называется стилевой таблицей. В HTML можно создавать разные уровни («каскады») стилевых таблиц — как общие, так и «частные», содержащие общие стили для всего сайта, стили для отдельных его разделов, особые стили для отдельных страниц. При этом значения однотипных параметров более «частных» стилей перекрывают (замещают) значения более общих, а локальное форматирование при помощи обычных тегов — перекрывает однотипное стилевое оформление.

Слой — составной элемент web-страницы с «прозрачным» фоном, содержащий какие-либо объекты (абзацы текста, таблицы, списки, иллюстрации и пр., — все, что может содержаться на обычной web-странице), который может быть «наложен» поверх «основного» содержимого web-страницы и/или поверх других слоев, аналогично наложению текста и рисунков на прозрачной пленке поверх изображения на листе бумаги. При этом можно управлять как взаимным расположением по высоте наложенных друг на друга слоев, так и расположением каждого слоя (понимаемого как некая прямоугольная область с заданным содержимым) по горизонтали и вертикали относительно верхнего левого угла окна браузера, создавая тем самым различные визуальные и даже (при помощи скриптового программирования) анимационные эффекты.

Скриптовое программирование — возможность создавать динамические и/или интерактивные web-страницы (технология Dynamic HTML, или DHTML), содержимое

которых может меняться согласно заданному алгоритму и/или в зависимости от действий пользователя, просматривающего web-страницу. Эта технология реализуется путем написания скрипта (текста программы на скриптовом языке программирования), внедряемого непосредственно в HTML-код web-документа: при просмотре web-страницы браузер выделяет этот скрипт и выполняет записанную в нем программу. Среди скриптовых языков программирования наиболее часто используются языки JavaScript и VBScript, представляющие собой упрощенные разновидности языков программирования Java и Visual Basic соответственно.

«Активное содержимое» web-страницы — условный термин, используемый, в частности, в браузерах Microsoft Edge последних версий и обозначающий любые элементы web-страницы, реализующие динамические, визуальные эффекты, интерактивность и пр., например, скрипты, фильтры, «всплывающие» окна, содержащие рекламу, и т. д. Как правило, настройки безопасности в современных браузерах предписывают автоматически блокировать «активное содержимое» просматриваемых web-страниц, поэтому при выполнении ряда соответствующих заданий, предлагаемых в данной книге, необходимо заблаговременно изменить настройки безопасности в браузере, либо каждый раз вручную разрешать отображение активного содержимого web-страницы, щелкая мышью на соответствующей ссылке в верхней части окна браузера

Структура HTML-документа

Документ HTML представляет собой обычный текстовый файл, который содержит конструкции языка HTML. Поэтому этот документ можно создавать в обычных текстовых редакторах, например в программе Блокнот, а затем сохранять созданные файлы с расширением .htm или .html.

Суть языка HTML - в разметке текста с помощью управляющих символов - тегов, которые располагаются в угловых скобках.

Большинство тегов парные, т.е. имеют открывающий элемент <> и закрывающий элемент </>.

Между ними и находятся коды, которые распознает браузер.

HTML-документ всегда должен начинаться отрывающим тегом <html> и заканчиваться закрывающим </html>.

Внутри расположены: блок заголовка <head> </head> и тело <body> </body>, в котором размещаются тексты, рисунки, аудио и видеофрагменты.

В блоке заголовка размещается тег <title> текст </title>. Текст, указанный в этом теге, отображается в заголовке окна браузера.

В HTML-документе можно разместить комментарии, которые браузером не отображаются: <! Комментарии>.

Пример 1. Создать простой HTML-документ в Блокноте. Сохранить созданный файл prim1.htm. Открыть созданный документ в браузере.

Структура такого документа представляется следующим образом:

<html> <!Открытие HTML документа>
<head>
<title> Заголовок </title>
</head>
<body> <!Содержание (тело) документа> Текст первой странички
</body>
</html>

Оформление HTML-документа

Теги языка HTML могут содержать атрибуты, которые являются параметрами или свойствами элементов разметки документа.

Правило записи атрибутов в теге следующее:

<тег атрибут1=значение атрибут2=значение …>

Тег <body> определяет внешний вид всей веб-станицы, в то время как отдельные ее элементы, например, заголовки, таблицы могут иметь свое особенное оформление.

Выбор цвета страницы, фонового рисунка и цвета текста на ней является весьма важным, так как от этого зависит визуальное восприятие всего сайта.

Цвет страницы задается атрибутом bgcolor, а цвет текста - text. Значением атрибутов является цвет, который задается своим названием на английском языке или его шестнадцатеричным кодом.

Работа с редактором визуального конструирования FrontPage. Основные элементы интерфейса FrontPage.

Редактор Microsoft FrontPage относится к редакторам визуального веб-конструирования. При работе с веб-редактором FrontPage можно обойтись без знания языка разметки гипертекстовых документов HTML. Веб-страница просто конструируется на экране и сохраняется в формате .html. Выполняемые при этом действия напоминают работу в текстовом редакторе Word.

Интерфейс редактора FrontPage достаточно прост, чтобы пользователь смог быстро освоить основные приемы работы. После запуска FrontPage открывается окно.Панели Стандартная, Форматирование и Рисование по своим возможностям напоминают аналогичные панели текстового редактора Word.

Вид Рабочей области зависит от выбранного режима работы. В режиме Конструктор веб-страница конструируется из текстовых блоков и графических объектов. При этом автоматически генерируется ее HTML-код, который можно просматривать и редактировать в режиме Код. Режим с разделением является комбинацией этих двух режимов. Режим просмотр позволяет просматривать созданные страницы.

С помощью меню вид можно выбрать и другие полезные при конструировании сайта режимы. Режим страница предназначен для создания и редактирования веб-страниц. В режиме Папки просматривается структура папок сайта, выполняются файловые операции. Заметим, что любые изменения файлов и папок сайта необходимо производить из среды разработки, так как это позволяет отследить изменения в структуре взаимосвязанных веб-страниц сайта. Режим Переходы отображает структуру связей между страницами и позволяет ее изменять. Режим Гиперссылки наглядно отображает систему ссылок, обеспечивает их проверку и редактирование.

Создание веб-сайта в редакторе FrontPage

Рассмотрим этапы и приемы работы в редакторе FrontPage на примере разработки веб-сайта «Беларускія пісьменнікі».

Для создания этого веб-сайта необходимо уяснить его структуру, и предполагаемое содержание, т.е. подготовить проект сайта. Фрагмент структуры сайта для наглядности изобразим в виде двухуровневой схемы.

На первой странице (верхний уровень) размещается заголовок сайта «Беларускія пісьменнікі», текстовые гиперссылки и гиперссылки с изображением писателей и поэтов, позволяющие открыть страницы второго уровня.

На втором уровне разместим три страницы одинаковой структуры.

После просмотра любой страницы второго уровня внизу расположим гиперссылки для возврата на главную (первую) страницу сайта и две гиперссылки для движения вперед и назад по сайту.

При проектировании сайта «Беларускія пісьменнікі» мы будем использовать методику построения сверху вниз, то есть конструировать от верхнего уровня к нижнему уровню. При создании веб-страниц сайта с помощью редактора FrontPage мы будем для удобства создавать страницы снизу-вверх, то есть начнем создавать страницы нижнего уровня.

Файлы всех веб-страниц сайта размещаются в отдельной папке. Это в дальнейшем поможет упростить размещение сайта в сети Интернет. Файлу главной веб-страницы принято давать имя index или main с расширением .htm, хотя это требование не является обязательным.

В рамках каждого уровня страницы сайта имеют одинаковую структуру. Все страницы одного уровня обычно оформляются в едином стиле. Стиль оформления веб-страницы определяется стилем текста и графических элементов.

Под стилем текста понимается тип шрифта, размер, начертания; отступы, выравнивание, межстрочный интервал и др. Стиль графических элементов задается формой, размером, цветом, фактурой материала и т. д.

В редакторе FrontPage для оформления документов разработаны шаблоны – Темы.

Темой называется специально разработанный компьютерным дизайнером набор элементов оформления и цветовых схем документа.

С помощью меню Файл → Создать в появившемся окне Шаблоны веб-узлов выберем Одностраничный веб-узел.

Для облегчения подбора элементов оформления и цветовых схем используем готовый шаблон оформления – тему.

С помощью меню Формат → Тема в появившемся диалоговом окне выберем, например, тему Горизонт и укажем применить как тему по умолчанию.

Тему можно применять к отдельным страницам, а также ко всему сайту. В последнем случае при создании каждой новой страницы тема устанавливается автоматически.

Форматирование текста и вставка изображений в редакторе FrontPage

Форматирования текста в редакторе FrontPage осуществляется с помощью панели инструментов Форматирование, а также окон Формат и Абзац.

Выделив текст в редакторе FrontPage (выполняется аналогично Word) в окне Шрифт можно выбрать его имя, начертание и указать размер, а в окне Абзац указать выравнивание, отступы и межстрочный интервал. Важно помнить, что отображение веб-страницы на экране зависит от разрешения монитора и настройки браузера. Поэтому размер шрифта принято задавать в условных единицах от 1 до 7. Если размер шрифта не указан, то по умолчанию он принимается равным 3.

Для привлечения внимания к информации создается бегущая строка с помощью цепочки действий: Вставка → Веб-компонент → Бегущая строка. В окне свойства бегущей строки вводится текст, задается направление и скорость движения, время задержки и другие параметры.

Размещение изображений на веб-странице производится аналогично тому, как это выполнялось в редакторе Word (команда меню Вставка→Рисунок.)

Задавать параметры изображения на странице можно в окне положение, которое активизируется с помощью меню Формат → Положение. Можно задать размер изображения (в пикселях или процентах), выравнивание, толщину границы, обтекание текстом и другие параметры.

Важно подчеркнуть, что все изображения, которые мы видим на веб-страницах, хранятся в отдельных файлах, а на самой странице имеются лишь ссылки на соответствующие файлы.

В зависимости от обтекания текстом вставляемые изображения могут располагаться различными способами. Регулирование позиции изображения относительно других элементов веб-страницы, например текста, осуществляется в диалоговом окне, которое активизируется с помощью меню Формат → Положение.

В редакторе FrontPage предусмотрена возможность простейшей обработки изображений (коррекция яркости и контрастности, повороты, обрезка и т.д.). Для этого используется панель Рисунки аналогичная текстовому редактору Word.

На веб-странице в редакторе FrontPage вставляются таблицы, которые часто используются для структурирования информации.

С помощью таблиц можно выполнять верстку в несколько колонок, применять эффекты состыковки картинки и фона, размещать тонкие линии на всю ширину или высоту странички и т.д.

Для вставки таблицы можно воспользоваться стандартной панелью инструментов, однако наиболее полно команды работы с таблицами представлены в меню таблица. В диалоговом окне вставка таблицы можно указать не только количество строк и столбцов таблицы, но и способ выравнивания, цвет фона ячеек, цвет и толщину границ, интервал между ячейками и другие параметры.

Создание гиперссылок в редакторе FrontPage

Для создания гиперссылки требуется выделить ее в документе (определить элемент привязки) и задать адрес перехода по данной ссылке. В качестве элементов привязки могут выступать слова, группы слов, изображения.

Текстовые ссылки обычно отмечаются цветом и подчеркиваются, а графические - иногда обводятся рамкой. Благодаря этому они отличаются от обычного текста и других элементов страницы.

Ссылки могут обеспечивать переход к веб-страницам или иным документам, например, рисункам, презентациям, видеофрагментам, расположенным как на данном сайте (внутренние ссылки), так и на других сайтах (внешние ссылки). Переходы внутри документа выполняют внутристраничные гиперссылки. Они облегчают навигацию по длинной странице, например, быстрый переход из конца страницы в начало.

Перед созданием внутристраничных ссылок нужно сначала расставить закладки. Для этого курсор устанавливается на нужное место веб-страницы. С помощью меню Вставка→Закладка вызывается диалоговое окно Закладка, в котором вводится имя закладки, например, вниз.

Для организации гиперссылки необходимо выделить элемент привязки (текст или рисунок), с помощью кнопки на Стандартной панели инструментов или меню Вставка →Гиперссылка вызвать диалоговое окно, в котором указать адрес перехода (адрес веб-страницы или электронной почты, имя файла или документа, а в случае внутренней ссылки имя закладки)

По умолчанию новый документ открывается в текущем окне браузера, однако из окна Добавление гиперссылки можно щелкнуть по кнопке Выбор рамки и в открывшемся окне Конечная рамка задать открытие нового документа в новом окне.

Заметим, что в редакторе FrontPage создание гиперссылок выполняется в режиме конструктор, а проверить работу гиперссылок можно в режиме просмотр.

Публикация сайта

Публикацией сайта называют его размещение на сервере или локальном компьютере с возможностью вызова из сети (глобальной или локальной).

Некоторые серверы бесплатно предоставляют дисковое пространство под сайт, например, www.narod.ru. Процесс публикации сайта заключается в переносе файлов вашего сайта на этот сервер. Адрес перенесенного сайта может иметь вид, например, www.belpisateli.narod.ru.

Для публикации сайта, подготовленного в редакторе FrontPage, необходимо с помощью меню Файл → Опубликовать узел вызвать диалоговое окно Свойства удаленного веб-узла.

Для размещения созданного сайта на сервере narod.ru в строке Расположение удаленного веб-узла следует указать адрес ftp-узла сервиса narod.ru, который для всех пользователей является одинаковым: ftp://ftp.narod.ru.

В дальнейшем пользователю необходимо ввести имя и пароль в окне требуется имя и пароль.

В открывшемся новом окне слева будут размещены файлы и папки созданного локального веб-узла, а справа файлы и папки удаленного веб-узла, например, странички сайта «Беларускія пісьменнікі».

После указания режима все файлы локального веб-узла копировать на сервер необходимо нажать кнопку опубликовать веб-узел.

Когда копирование всех файлов завершится, пользователь может просмотреть размещенный в сети сайт. Для этого в строке браузера вводится адрес, например, www.belpisateli.narod.ru.

2. ПРОГРАММИРОВАНИЕ НА PHP

История PHP

PHP, как всем известно, на самом деле преемник продукта под названием PHP/FI. Созданное в 1994 году Расмусом Лердорфом, самое первое воплощение PHP было простым набором CGI-скриптов, написанных на языке программирования Си. Изначально используя

их для отслеживания посещений своего веб-резюме, он назвал этот набор скриптов "Personal Homepages Tools" ("Инструменты для персональных домашних страниц"), но более часто упоминалось название "PHP Tools". Со временем требовалось все больше улучшений функциональности, и Расмус переписал PHP Tools, создав более крупную и богатую реализацию. Эта новая реализация была способна взаимодействовать с базами данных и многое другое, что создавало фреймворк, с помощью которого пользователи могли создавать простые динамические веб-приложения, такие как гостевые книги. В июне 1995 года Расмус » открыл исходный код PHP Tools общественности, что позволило разработчикам использовать его по своему усмотрению. Это также дало возможность пользователям исправлять ошибки в коде и улучшать его.

В сентябре того же года, Расмус расширил PHP и на короткое время убрал из названия PHP. Теперь в виде инструмента FI (сокращение от "Интерпретатор Форм"), новая реализация включала в себя некоторые основные функциональные возможности того PHP, который мы знаем сегодня. Она имела Perl-подобные переменные, автоматическую интерпретацию форм и встраиваемый в HTML синтаксис. Синтаксис языка был похож на Perl, хотя и был гораздо более ограниченным, простым, и в некоторой степени противоречивым. Для того, чтобы вставлять код в HTML-файл, разработчикам пришлось использовать HTML комментарии. Хотя этот метод был не совсем хорошо принят, FI по-прежнему набирал популярность в качестве CGI-инструмента, но все-таки не в качестве языка. Однако, перемены начались в следующем месяце, когда в октябре 1995 года Расмус выпустил полностью переписанный код. С вернувшимся именем PHP, но уже сокращенным от "Personal Home Page Construction Kit", это был первый релиз, который мог похвастаться расширенным интерфейсом сценариев. Язык намеренно напоминал Си по структуре, что делал его легким для восприятия разработчиками, знакомыми с Си, Perl и подобными языками. Будучи все еще ограниченными UNIX и POSIX-совместимыми системами, был изучен вопрос для реализации языка в Windows NT.

Код получил еще одно существенное преобразование в апреле 1996 года. Объединив названия предыдущих версий, Расмус представил PHP/FI. Реализации второго поколения начали по-настоящему развивать PHP из набора инструментов в самостоятельный язык программирования. PHP включал в себя встроенную поддержку для DBM, mSQL и Postgres95 баз данных, cookies, поддержку определяемых пользователем функций и многое другое. В июне PHP/FI была присвоена версия 2.0. Интересно, однако, что существовала только одна версия PHP 2.0. Когда она, наконец, в ноябре 1997 года преодолела статус бета-версии, парсер языка был уже полностью переписан.

Несмотря на короткую историю разработки, PHP/FI продолжал набирать популярность в молодом мире веб-разработки. В 1997 и 1998, PHP/FI стал культом для нескольких тысяч пользователей по всему миру. Исследования Netcraft в мае 1998 года показали, что почти 60 тысяч доменов передавали заголовки, содержащие "PHP". Это число равнялось примерно 1% от всех доменов в Интернете в то время. Несмотря на эти впечатляющие цифры, развитие PHP/FI было ограничено: несмотря на нескольких второстепенных участников, в целом он по-прежнему разрабатывался одним человеком.

PHP 3.0 был первой версией, напоминающей PHP, каким мы знаем его сегодня. Посчитав PHP/FI 2.0 все еще неэффективным и недостаточно функциональным для использования в коммерческих приложениях, разрабатываемых для их университетского проекта, Энди Гутманс и Зив Сураски из Тель-Авива начали еще раз заново переписывать парсер в 1997 году. Связавшись с Расмусом, они обсудили различные аспекты текущей реализации и их новой разработки PHP. Для улучшения движка и использования уже существующей базы пользователей PHP/FI, Энди, Расмус и Зив решили сотрудничать в развитии нового, независимого языка программирования. Этот совершенно новый язык был выпущен под новым именем, без упоминания о персональном использовании, как в PHP/FI 2.0. Он был назван просто "PHP" - аббревиатура, означающая рекурсивный акроним - PHP: Hypertext Preprocessor.

Одной из сильнейших сторон PHP 3.0 была возможность расширения ядра. Кроме обеспечения пользователей надежной инфраструктурой из множества различных баз данных,

протоколов и API, расширяемость PHP 3.0 привлекла к нему множество сторонних разработчиков, желающих добавить к языку свои модули. Возможно, это и был главный ключ к успеху, но стоит добавить, что немаловажным шагом оказалась поддержка ООП синтаксиса и намного более мощного и последовательного синтаксиса самого языка.

В июне 1998 года, со множеством новых разработчиков со всего мира присоединившихся к проекту, PHP 3.0 был представлен новой командой разработчиков, как официальный преемник PHP/FI. Активное развитие PHP/FI 2.0, фактически прекратившееся в ноябре прошлого года, теперь официально закончилось. Примерно через девять месяцев открытого публичного тестирования, при объявлении официального выпуска PHP 3.0, он уже был установлен на более чем 70000 доменах по всему миру, и уже не ограничивается POSIX-совместимыми операционными системами. Относительно небольшая доля доменов с установленным PHP была размещена на серверах под управлением Windows 95, 98 и NT, а также Macintosh. На пике своего развития, PHP 3.0 был установлен приблизительно на 10% веб-серверов в Интернете.

К зиме 1998 года, практически сразу после официального выхода PHP 3.0, Энди Гутманс и Зив Сураски начали переработку ядра PHP. В задачи входило увеличение производительности сложных приложений и улучшение модульности кодовой базы PHP. PHP 3.0 дал возможность подобным приложениям успешно работать с набором баз данных и поддерживать большое количество различных API и протоколов, но PHP 3.0 не имел качественной поддержки модулей и приложения работали неэффективно.

Новый движок, названный 'Zend Engine' (от имен создателей: Zeev и Andi), успешно справлялся с поставленными задачами и впервые был представлен в середине 1999 года. PHP 4.0, основанный на этом движке и принесший с собой набор дополнительных функций, официально вышел в мае 2000 года, почти через два года после выхода своего предшественника. Кроме значительного улучшения производительности, PHP 4.0 имел еще несколько ключевых нововведений, таких как поддержка намного большего числа веб-серверов, поддержка HTTP сессий, буферизация вывода, более безопасные способы обработки вводимой пользователем информации и несколько новых языковых конструкций.

После долгой разработки и нескольких пре-релизов в июле 2004 был выпущен PHP 5. В основном он управляется ядром Zend Engine 2.0 с новой объектной моделью и множеством различных других нововведений. Команда разработчиков PHP включает в себя десятки разработчиков, а также десятки других организаций, работающих над связанными с PHP и его поддержкой проектами, такими как PEAR, PECL и документацией, а также базовую инфраструктуру сети более чем из ста серверов на шести из семи континентах мира. Основываясь на статистике прошлых лет, можно с уверенностью предположить, что PHP теперь установлен на десятки или даже, возможно, сотни миллионов доменов по всему миру.

Константы

Константы определяются в PHP-программе с помощью функции define( ). Например: define("PI", 3.1415927);

После определения константа не может быть изменена. В имени константы обычно используются только заглавные буквы.

PHP имеет ряд предопределенных констант. Например,:

- _FILE_ содержит имя файла, включая полный путь, который в данный момент читает PHP.
- _LINE_ содержит номер строки этого файла.
- __DIR__ представляет только путь к файлу.
- __CLASS__ представляет имя текущего класса.
- __FUNCTION__ представляет имя текущей функции.
- __METHOD__ представляет имя текущего метода.
- __NAMESPACE__ представляет имя текущего пространства имен.

Переменные

Имя любой переменной в PHP начинается со знака $. Имена переменных чувствительны к регистру символов.

Тип переменной не требуется задавать специально. Конкретный тип переменной устанавливается и меняется в ходе выполнения программы.

PHP поддерживает восемь типов данных:

- логический (принимает значения true или false);
- целое число;
- вещественное число с плавающей точкой;
- строка;
- объект;
- массив;
- ресурс (специальный тип);
- null (специальный тип).

Тип переменной можно проверить с помощью функции gettype().

Приведение типа переменной осуществляется с помощью операторов:

- (bool) - к логическому типу;
- (int) - к целому числу;
- (double) - к вещественному числу;
- (string) - к строке;
- (array) - к массиву;
- (object) - к объекту.

Отличие от изменения типа с помощью функции settype() состоит в том, что оператор приведения создает временную копию нового типа, оставляя саму переменную без изменений.

Иногда для упрощения логики программы удобнее использовать переменные имена переменных. PHP предоставляет такую возможность в виде динамических переменных. Динамической называют переменную, имя которой хранится в ней самой.

В PHP возможно обращение к одной и той же переменной с использованием различных имен. Для реализации этого используются ссылки. Ссылки позволяют двум или большему количеству переменных ссылаться на одну и ту же область памяти.

Операторы

Операторы PHP напоминают общеизвестные операторы языка Си.

Унарные операторы

- Изменение знака на противоположный

! Дополнение. Используется для реверсирования значения логических переменных

++ Увеличение значения переменной. Может применяться и как префикс, и как суффикс

-- Уменьшение значения переменной. Может применяться и как префикс, и как суффикс

Арифметические операторы

- Вычитание

+ Сложение

* Умножение

/ Деление

% Остаток от деления

Оператор конкатенации

Оператор конкатенации присоединяет правую строку к левой.

Операторы присваивания

= Присваивание

+= Сложение ($n += 777; аналогично $n = $n + 777;)

-= Вычитание ($n -= 777; аналогично $n = $n - 777;)

*= Умножение

/= Деление

%= Остаток от деления

.= Конкатенация ($n .= "777"; аналогично $n = $n."777";)

Битовые операторы

Битовые операторы позволяют изменять отдельные биты целых чисел.

& И

| ИЛИ

^ Исключающее ИЛИ

~ Инверсия

>> Сдвиг вправо

<< Сдвиг влево

Операторы сравнения

> Больше (Больше ли первое значение, чем второе?)

>= Больше или равно (Верно ли, что первое значение не меньше второго?)

< Меньше (Меньше ли первое значение, чем второе?)

<= Меньше или равно (Верно ли, что первое значение не больше второго?)

== Равно (Равнозначны ли значения двух переменных?)

=== Идентично (Одинаковы ли как значения, так и типы двух переменных?)

!= , <> Не равно (Не равны ли значения двух переменных?)

!== Не идентично (Не одинаковы ли значения или типы данных двух переменных?)

Логические операторы

Логические операторы отличаются от битовых тем, что работают не с числами, а с логическими значениями: TRUE и FALSE.

and И

or ИЛИ

xor Исключающее ИЛИ

! Инверсия

>> Сдвиг вправо

<< Сдвиг влево

&& И

|| ИЛИ

Логические операторы "И" и "ИЛИ" имеют два формата. Это не синонимы. Дело в том, что оператор or имеет приоритет ниже, чем ||, а and - ниже, чем &&. Таким образом, при построении сложных условных выражений можно обойтись без скобок. Однако, в таком способе указания порядка проще и запутаться.

Проверка содержимого переменной

Иногда необходимо проверить, существует ли переменная или какое она имеет значение. Ниже приведены функции, позволяющие выполнить такие действия. isset($имя_переменной) #Истина, если переменная объявлена даже без присваивания значения.

empty($имя_переменной) #Истина, если значение переменной равно нулю или пустой строке, либо переменная не объявлена.

PHP также позволяет проверить тип переменной. Например, для того чтобы проверить, является ли переменная целочисленной, следует воспользоваться функцией is_int($number).

Результатом выполнения этой функции является TRUE, если переменная $number имеет тип integer. Рассмотрим подобные функции.

is_array ($var2) проверяет, является ли переменная $var2 массивом.

is_f loat ($number) проверяет, является ли переменная $number числом с плавающей точкой.

is_null ($varl) проверяет, равно ли значение переменной $varl нулю

is_numeric($string) проверяет, является ли переменная $string числовой строкой.

is_string ($string) проверяет, является ли переменная $string строкой.

Для проверки обратных условий следует воспользоваться символом восклицания (!). Например, при обработке следующего выражения будет получено значение TRUE, если переменная не объявлена:! isset($имя_переменной)

3. СТИЛЕВОЕ ОФОРМЛЕНИЕ HTML-ДОКУМЕНТОВ

Сегодня мы будем знакомиться со стилевыми таблицами. HTML код задает базовую структуру документа, а с помощью стилей можно задавать различные варианты оформления веб-страницы. Разберемся в чем разница между внутренними, внешними и глобальными стилями. Подробно рассмотрим, как они применяются и узнаем об иерархии каскадных приоритетов. Также рассмотрим правила написания стилей и познакомимся с определением классов. В следующем шаге мы научимся форматировать текст с помощью стилей и узнаем основные стилевые свойства. А также уделим внимание способам оформления ссылок, меняющим их внешний вид.

Стилевые таблицы

Как вы уже знаете, HTML разрабатывался как язык разметки страниц, задающий базовую структуру документа (абзацы, заголовки, таблицы итд). При этом в HTML уже содержались такие оформительские элементы, как bgcolor, font size или align. В далеком 1996 году консорциум W3C впервые сформулировал идею использования каскадных стилевых таблиц (Cascading Style Sheets — CSS) для форматирования HTML документов. Данная рекомендация, более детализированная в 1998 году, позволяла разработчикам разделить коды структуры веб-страниц и коды оформления документа.

Стилевые таблицы — это набор правил, определяющих способ применения стилей к тегам HTML документа. В данных рекомендациях описывались перечисленные ниже три типа стилевых таблиц.

Внутренние стили

Содержатся в теле веб-страницы в качестве атрибутов HTML элементов. Каждому тегу назначаются свои собственные стили по мере их определения.

Глобальные стили

Свойства стиля описываются (внутри тега <style>) в верхней части HTML документа. Стиль, назначенный какому-то тегу, применяется ко всем прочим тегам этого же типа в пределах одной веб-страницы.

Внешние стили

Информация о свойствах стилей сохраняется в отдельном файле. Этот файл может быть подключен к любому HTML документу с помощью тега <link>, помещаемого внутри тега <head>.

Даже если не учитывать преимущества, предоставляемые стилевыми таблицами при форматировании документов, их использование, вне всяких сомнений, позволит вам, как веб-дизайнеру, сэкономить массу рабочего времени и усилий. Поскольку уже созданные стилевые таблицы можно применять к произвольному количеству HTML документов, последующее внесение корректирующих изменений будет выполняться в течение минут, а не часов или целых дней.

Если не применять стилевые таблицы, то для изменения внешнего вида какого-то отдельного тега веб-сайта пришлось бы открывать каждый документ, находить в нем этот тег, вносить в него изменения, сохранять документ, переходить к следующему документу и т.д. С другой стороны, можно изменить этот тег только в одном документе, где описаны стилевые таблицы, и эти изменения сразу же вступят в силу для всех страниц, к которым подключен документ.

Определение правил

Правила стилевых таблиц состоят из селекторов (тегов HTML, для которых создается стиль) и описаний (свойств стилевых таблиц и их значений). В приведенном далее примере селектором является тег <body>, а описание состоит из стилевого свойства (background) и его значения (black). В данном случае для всего документа в качестве фонового устанавливается черный цвет. body {background:black}

Как видите, в стилевой таблице тег HTML не заключается в угловые скобки (как это происходит обычно при добавлении его в HTML документ), а описание окружают фигурные скобки. В описании может содержаться более одного свойства. Так, в следующем примере

цвет текста для данной страницы устанавливается белым. Обратите внимание, что в описании одно свойство от другого отделяется точкой с запятой. body {background:black; color:white}

Если необходимо применить одни и те же правила сразу к нескольким тегам HTML, можете сделать это одновременно, как показано ниже.

```
body, td, hi {
background:black;
color:white }
```

Определение классов

Как известно, правила существуют для того, чтобы их нарушать. Что, если вы не хотите, чтобы все добавляемые в документе заголовки первого уровня, задаваемого с помощью тега <h1>, отображались белым цветом на черном фоне? Может необходимо, чтобы каждый второй заголовок этого уровня отображался синим цветом на белом фоне? Что ж, справиться с данной задачей можно с помощью атрибута class. Этот атрибут можно задавать практически для любого HTML тега, и результат такого действия сравним с созданием собственных тегов.

Например, нужно сделать, чтобы одна таблица была с голубым фоном и выравниванием по левому краю, а другая таблица — с желтым фоном и выравниванием по центру. Тогда в теге <style> в шапке документа прописываются сразу два стиля. За именем HTML тега table следует точка (.) и далее имя класса (nav или rest).

```
<style type=text/css>
table.nav {background:aqua}
table.rest {background:yellow; text-align:center; color:black}
</style>
```

При добавлении таблицы в теле документа необходимо задать значение атрибута class, чтобы сообщить браузеру, какие стилевые свойства должны быть к этой таблице применены. Коды добавления на веб-страницу обеих таблиц приведены ниже. Обратите внимание, что названия классов указываются в кавычках, подобно значениям любых других атрибутов (например, как значения используемого здесь же атрибута width).

```
<table class="nav" width="100%">
<table class="rest" width=50%>
```

Применение стилей

Прежде чем двигаться далее, вкратце рассмотрим процедуру применения стилевых свойств к элементам HTML документов. Как вы уже знаете, существуют три метода добавления стилевых таблиц: внутренние, глобальные и внешние стили. Остановимся на каждом из них в отдельности.

Глобальные стили

Все стили данного типа определяются в верхней части HTML документа внутри тега <head>, поскольку они содержат описания, применяемые ко всей странице. Действие определяемых таким способом стилей ограничивается пределами текущего документа. Если необходимо использовать эти же стили в каком-то другом HTML документе, вам придется непосредственно добавить их определения в этот документ. Тег <style> почти всегда включает в себя атрибут text=и text/css. Запомните это и не забывайте его добавлять.

```
<head>
<style type="text/css">
table.nav {background:aqua}
table.rest {background:yellow; text-align:center; color:black}
a:link {color:red; text-decoration:none}
</style>
</head>
```

Внешние стили

При использовании внешних стилевых таблиц все описания стилей содержатся в отдельном файле. Этот файл подключается ко всем HTML документам, где заданные в нем стилевые свойства должны использоваться.

```
<head>
<link rel="stylesheet" href="mystyles.css" type="text/css">
```

</head>

В данном примере был создан отдельный файл для каскадных стилевых таблиц, содержащий описания всех стилевых свойств. Для него было задано имя mystyles.css. Обратите внимание, что здесь также присутствует атрибут type=»text/css». Далее приведено содержание файла mystyles.css. Как видите, здесь описаны те же самые стили, что и в предыдущем примере с глобальными стилевыми таблицами, однако в данном случае они вынесены в отдельный текстовый файл.

table.nav {background:aqua}

table.rest {background:yellow; text-align:center; color:black}

a:link {color:red; text-decoration:none}

Внутренние стили

При использовании данного метода стилевые свойства применяются к тем тегам HTML, внутри которых они определяются. Например, если вы хотите с помощью этого метода применить один и тот же стиль оформления ко всем заголовкам первого уровня, необходимо набрать их описание внутри всех тегов <h1> данного документа. Рассмотрим это на следующем примере. В данном случае используются те же стилевые свойства, что и в предыдущих примерах, однако теперь каждый стиль описывается внутри тега своей таблицы.

<table style="background:aqua" width="100%">

<table style="background:yellow; text-align:center; color:black" width=50%>

При использовании метода внутритекстового определения стилей тег <style> превращается в атрибут style. Разные стилевые свойства по-прежнему отделяются друг от друга точкой с запятой, однако вся группа свойств, относящихся к какому-то конкретному тегу, описывается внутри него. Этот метод удобен в тех случаях, когда требуется применить какие-то специальные стили оформления к одному-двум элементам веб-страницы, однако вы вряд ли захотите его использовать для добавления множества стилей по всему документу.

Каскадный приоритет

Прежде чем двигаться дальше, необходимо прояснить еще один момент. Для браузера стили этих трех типов не эквивалентны друг другу. Браузер устанавливает более высокий приоритет для стиля, который является ближайшим к конкретному тегу. Таким образом, внутренний стиль (определяемый как атрибут внутри самого тега) всегда имеет наивысший приоритет. Следующими применяются глобальные стили (определяемые в верхней части HTML документа); и, наконец, внешние стили (вообще задаваемые в отдельном файле) применяются в последнюю очередь.

Форматирование текста с помощью стилей

Текст является наиболее важным компонентом любой веб-страницы. Если на веб-странице вообще не будет никакой текстовой информации, посетители вряд ли смогут извлечь из нее какую-либо пользу. Текст в HTML документы добавляется с помощью тегов <body>, <p>, <td>, <tr>, <th>, <h1>…<h6>, <li> и др. Для текста каждого из этих тегов можно задавать собственные параметры форматирования, используя стилевые свойства, перечисленные ниже.

Если вы сами или посетители веб-страницы не измените заданных по умолчанию настроек браузера, обычный текст HTML документа будет воспроизводиться шрифтом Times New Roman размером 12 пунктов.

background — Задает фоновый цвет текста;

color — Задает цвет самого текста;

font-family — Задает шрифт;

font-size — Указывает размер шрифта;

font-style — Определяет стиль начертания текста; может принимать значения normal (по умолчанию) или italic;

font-weight — Принимает значения от extra-light до extra-bold;

text-align — Определяет способ выравнивания текста; может принимать значения left, right, center или justify;

text-indent — Задает отступ текста; величина отступа может быть задана как фиксированное

значение или в процентном выражении;
text-decoration — Принимает значения underline, overline, strikethrough и none.

Стили оформления ссылок

Наверное, вы уже привыкли к тому, что на веб-страницах ссылки обычно оформляются как подчеркнутый текст синего цвета. Далее перечислены предусмотренные для стилевых таблиц селекторы, позволяющие изменить внешний вид ссылок.

a:link — Задает стиль оформления еще не использованных ссылок;
a:visited — Задает стиль оформления уже использованных ссылок;
a:active — Задает стиль оформления активных в данный момент ссылок;
a:hover — Задает стиль оформления ссылки, на которую наведен указатель мыши.

Для ссылок, добавляемых на веб-страницы можно использовать следующие стилевые свойства: background-color (цвет фона для ссылки), color, font-family, text-decoration. В последнее время один из наиболее популярных эффектов стилевых таблиц состоит в удалении для ссылок линии подчеркивания. Чтобы сделать это, просто добавьте декларацию text-decoration:none к стилям a, как показано на примере следующего кода.

a:link {color:red; text-decoration:none}

Если вы хотите видеть свои ссылки подчеркнутыми, никаких специальных действий выполнять не нужно. Подчеркивание и так по умолчанию задано для всех стилей a.

4. СЦЕНАРИИ JAVASCRIPT

JavaScript является языком сценариев (скриптов), который применяют в основном для создания на Web-страницах интерактивных элементов. Его можно использовать для построения меню, проверки правильности заполнения форм, смены изображений или для чего-то еще, что можно сделать на Web-странице. Если взглянуть на Google Maps или службу GMail компании Google, то можно понять, на что способен сегодня язык JavaScript.

Так как JavaScript является в настоящее время единственным языком сценариев, который поддерживают все основные браузеры Web (Firefox, Netscape, Safari, Opera, Camino и т.д.), то он используется очень широко.

Код JavaScript обычно выполняется Web-браузером клиента, и в этом случае он называется сценарием на стороне клиента. Но код JavaScript можно выполнять также на Web-сервере для формирования документов HTML, воплощая тем самым сценарий на стороне сервера. Хотя использование JavaScript обычно ограничивается сценариями на стороне клиента, он является также очень мощным серверным языком.

При создании кода JavaScript требуется фактически только текстовый редактор и Web-браузер. Знание HTML и CSS будет играть определенно положительную роль, и если вы захотите использовать навыки JavaScript на Web-сайте, то понадобится также Web-сайт. Если у вас уже есть Web-сайт, то отлично! Если нет, то существует множество бесплатных серверов, которые можно использовать для размещения своих страниц.

Что касается текстового редактора, то в Windows имеется редактор NotePad. Хотя этого будет достаточно для редактирования JavaScript, HTML и CSS, более мощный редактор, такой, например, как EditPlus или другой, может оказаться более удобным.

Ну, а теперь можно перейти к созданию сценария JavaScript!

Прежде всего, необходимо узнать, как добавить сценарий JavaScript на страницу HTML. Это можно сделать одним из двух способов: поместить теги Script на Web-странице и расположить код JavaScript внутри этих тегов, или поместить весь код JavaScript в отдельный файл и связаться с ним с помощью тега Script.

Любой из этих методов вполне допустим, но они имеют разное назначение. Если имеется небольшой код, который будет использоваться только на одной странице, то размещение его между тегами Script будет хорошим решением. Если, однако, имеется большой фрагмент кода, который будет использоваться на нескольких страницах, то, наверно, лучше поместить этот код JavaScript в отдельный файл и соединиться с ним. Это делается для того, чтобы не нужно было загружать этот код всякий раз при посещении

различных страниц. Код загружается один раз, и браузер сохраняет его для последующего использования. Это похоже на то, как используются каскадные таблицы стилей (CSS).

Ниже приведены примеры двух способов подключения кода JavaScript:

```
<script type="text/javascript"></script>
<script type="text/javascript" src="scripts/JavaScriptFile.js"></script>
```

В первом примере, код JavaScript помещается между символами > и <, прямо перед </script>. Если вы совершенно не знаете, как работает Web-страница, то вот пример того, как устроена страница HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE> Здесь располагается заголовок страницы </TITLE>
<META NAME="Generator" CONTENT="EditPlus">
<META NAME="Author" CONTENT="Имя автора">
<SCRIPT TYPE="text/javascript">
</SCRIPT>
</HEAD>
<BODY>
Здесь располагается основная содержательная часть Web-страницы (тело).
</BODY>
</HTML>
```

Сохраните этот файл где-нибудь на своем компьютере с расширением *.html*, так, чтобы полное имя файла было, например, таким: *JavaScript_Lecture_1.html*. После сохранения файла сделайте на нем двойной щелчок мышью, чтобы открыть в используемом по умолчанию браузере.

Почти любой язык программирования в мире имеет дело с объектами, называемыми "переменными", и *JavaScript* не является исключением. Переменная является просто элементом данных с присоединенным к нему именем. Она может содержать число, слово или предложение (называемые строками - *String*) или объект (*Object*), о которых будет рассказано позже. Если коду необходимо сообщить, что имеется *5* яблок, то можно создать переменную с именем *apples* и задать ей значение *5*. Давайте сейчас это сделаем. В *JavaScript* для определения переменной используется ключевое слово *var*. Отметим, что *JavaScript* различает регистр символов, поэтому *var* означает не то же самое, что *VAR* или *Var*.

```
var apples = 5;
```

Необходимо сделать два важных замечания в отношении этого небольшого фрагмента кода. Первое: требуется помнить о том, что *JavaScript* является "слабо типизированным" языком. Это означает, что при определении переменных не требуется указывать, какого они типа: будут ли они числами, строками, объектами, и т.д. Во многих других языках необходимо делать это различие.

Второе: обратите внимание на точку с запятой (*;*) в конце строки. Это говорит интерпретатору *JavaScript*, что вы закончили делать то, что делали в данный момент, - в нашем случае это задание значения *5* переменной *apples*. Хотя точка с запятой не является обязательной в *JavaScript*, лучше привыкнуть ее использовать.

Итак, мы определили в коде, что имеется *5* яблок. Что дальше? Информация о яблоках имеется в коде, но никто об этом не знает. Надо сообщить о яблоках! Одним из наиболее распространенных методов вывода простого сообщения пользователю является отправка уведомления (*alert*):

```
var apples = 5;
alert('Имеется ' + apples + ' яблок!');
```

Если протестировать этот сценарий, то на экране появится окно с сообщением "*Имеется 5 яблок!*" Сейчас подходящее время, чтобы ввести строки (*String*) и так называемую конкатенацию или соединение строк. Строка является просто небольшим фрагментом текста и может содержать любой текст. В *JavaScript* мы сообщаем коду, что

имеется строка, заключая ее в одиночные или двойные кавычки (*"* или *'*). Можно использовать любой тип кавычек. Знаки плюс (+) в примере выше сообщают коду, что мы соединяем строку с предыдущей строкой.

Итак мы имеем строку "*Имеется*" за которой следует переменная *apples* (равная *5*), за которой следует еще одна строка "*яблок!*". Соединим их вместе и получим "*Имеется 5 яблок!*". Команда *alert* получает то, что ей передается (то, что находится между скобок) и открывает окно с текстом.

Что если мы хотим предложить пользователю съесть яблоко? Можно, например, спросить, сколько яблок он хотел бы съесть:

```
var apples = 5;
alert('Имеется ' + apples + ' яблок!');
var eat = prompt('Сколько яблок вы хотите съесть?', '1');
```

prompt является другой встроенной функцией, аналогичной *alert*. Однако вместо простого вывода информации она также получает ввод от пользователя. В данном случае мы спрашиваем у пользователя, сколько яблок он хотел бы съесть. *'1'* в коде сообщает функции *prompt*, что значением по умолчанию для количества яблок будет *1*, так как люди обычно едят только одно яблоко за раз. Однако пользователь может изменить это значение на любое другое. Когда пользователь щелкнет на кнопке *OK*, переменной *eat* будет задано значение этого ввода. Поэтому если пользователь введет *2*, то *eat* будет равно *2*.

Поэтому, если пользователь съел *2* яблока, то останется *3*, так? Поэтому выполним несколько простых математических операций и покажем результат.

```
var apples = 5;
alert('Имеется ' + apples + ' яблок!');
var eat = prompt('Сколько яблок вы хотите съесть?', '1');
apples -= parseInt(eat);
alert('А теперь имеется только' + apples + ' яблок!');
```

Здесь мы видим два новых элемента. Прежде всего, обращение к функции *parseInt*, которая получает строку и возвращает число. Так как для выполнения математических операций требуются числа, то это гарантирует, что мы имеем число. Если пользователь введет в поле *2*, то *parseInt* превратит это в число *2*.

Затем, оператор -=, который означает вычитание из левой части оператора значения правой части. Поэтому значение переменной *eat* вычитается из переменной *apples*. Можно также записать эту строку следующим образом:

```
apples = apples - parseInt(eat);
```

Это означает в точности то же самое и может быть немного легче для понимания. Теперь, когда известно, сколько осталось яблок, мы еще раз сообщаем пользователю эту информацию.

Существуют другие операторы, подобные -=, которые делают похожие вещи. Всего имеется 8 обычных арифметических операторов: +, -, /, *, +=, -=, /=, *=

5. WEB-СЕРВЕРЫ

Современные Web-приложения применяются для создания информационных систем в сетях Интернет/интранет и обычно строятся как многозвенные (многоуровневые) приложения, публикующие БД.
При большом числе клиентских компьютеров многоуровневая архитектура позволяет уменьшить нагрузку на сервер баз данных и линии связи. В этой лекции описываются принципы функционирования и структура Web-приложений, особенности Web-приложений, публикующих БД в сетях Интернет/интранет, а также рассматриваются характеристики популярных Web-серверов: Microsoft Internet Information Server, Apache, Netscape Enterprise.

Основные функции web-серверов

Изначально, Web была создана для распространения гипертекстовых документов [3]. Со временем функциональная часть росла, возможностей web становилось больше. Сейчас

web обладает дюжинной функциональностью, и в первую очередь этим обеспечивают web-сервера.

Основные функции, выполняемые web-серверами:

–Прием запросов от web-браузеров по протоколу HTTP с использованием сетевых протоколов TPC/IP;

–Выполнение поиска и отсылки файлов с гипертекстом и/или каких-либо документов в web-обозреватель по протоколу HTTP;

–Обслуживание и обработка запросов, типа: mailto, telnet, ftp и другие;

–Запуск прикладных программ на web-сервере с передачей и возвратом параметров обработки через интерфейс CGI;

–Работа и обслуживание навигационных карт изображений (image map);

–Загрузка java-приложений;

–Администрирование и оперативное управление сервером;

–Авторизация пользователей и их аутентификация;

–Ведения регистрационного журнала обращения пользователей к различным ресурсам;

–Поддержка динамических страниц;

–Поддержка защищенного протокола HTTPS для защищенных соединений с пользователями.

Web-браузеры и Web-серверы сообщаются через протокол HTTP, это простой протокол запросов и ответов для пересылки информации с использованием TCP/IP. Работает это следующим образом: web-сервер получает запрос, находит файл, посылает его браузеру и потом разрывает соединение с ним. Затем web-браузер выводит загруженный из сети HTML-документ на экран пользователя.

Web-сервер может хранить любые данные, это может быть текст, изображения, видео и тому подобное. В совокупности, на web-сервере могут работать прикладные программы, например процессоры поиска и средства связи с базами данных. Для их создания, зачастую, используются такие стандарты, как общий шлюзовой интерфейс (CGI - CommonGatewayInterface), языки сценариев, (например JavaScript), а также полноценные языки программирования(Java, Basic и др.).

Критерии выбора web-серверов

Критериями для выбора web-серверов могут послужить различные факторы. В первую очередь это то, для чего будет использоваться web-сервер. Так же немаловажное значение имеют характеристики сервера: насколько легко он устанавливается, настройка его конфигурации, управление содержанием, администрирование и управление самим сервером, защита информации, а так же производительность.

Установка большинства web-серверов происходит довольно легко. Наибольшую трудоемкость составляет дальнейшая настройка конфигурации сервера.

Что касается средств локального управления - они позволяют управлять сервером с его консоли, а средства дистанционного управления - с другого сетевого компьютера.

Средства управления содержательным материалом – это общая организация узла web, они содержат инструменты наподобие HTML-редакторов и преобразователей форматов документов, инструменты проверки правильности внутренних и внешних гипертекстовых связей. Администраторы сервера должны сами выбрать, где будут храниться файлы, и каким образом к ним будет получать доступ конечный пользователь. Для этого необходимо установить соответствие между логическими URL и физическими каталогами.

Так как в наши дни web-сервера активно используются в интрасетях, а так же используются для коммерческих целей в сети Internet, то это задает вопрос о безопасности. Часто системы web-сервера предоставляют избыточный, или недостаточный уровень безопасности для сегодняшних интрасетей. Для организации доступа внутри компании у вас есть выбор: использовать незашифрованные пароли, с передачей их по каналам связи, или же использовать SSL(англ. secure sockets layer — уровень защищённых сокетов), сложного и, относительно, медленного метода, используемого для шифровки паролей и данных. Так же существует различные комплексы защиты сетей, например, игнорирование длинных и

сложных запросов, которые могут загрузить сервер так, что он перестанет отвечать на простые запросы пользователей.

Создание прикладных программ - одна из самых важных функций Web-сервера, одновременно самая незаметная. Среда разработки программ и инструменты подключения к базам данных критически важны для расширения возможностей Web-сервера. Этим характеристикам нелегко дать оценку, так как они зависят от абстрактных и отличающихся своеобразными деталями API, особенностей языков сценариев и личных предпочтений программистов.

Web-серверы обслуживают любые системы от небольшой интрасети подразделения до крупных информационных центров Web, рассылающих HTML-страницы миллионам пользователей.

Для подразделенческих интрасетей, лучше других подойдет пакет Microsoft IIS. Он отличается простотой инсталляции и настройки конфигурации; он хорошо интегрирован со средствами управления доступом, программой контроля функционирования PerformanceMonitor и утилитой просмотра журнала событий EventViewer, имеющейся в ОС WindowsNT; для динамической передачи информации из баз данных в ней предлагается ряд инструментов. IIS характеризуется весьма высоким быстродействием.

Инструменты управления содержательным материалом поставляются вместе с несколькими Web-серверами, чтобы облегчить создание информационных центров Web. Помимо HTML-редакторов и преобразователей форматов документов одними из самых полезных являются средства контроля URL, гарантирующие действительность всех гипертекстовых связей вашего Web-узла.

Все приведенные ниже web-сервера обладают стандартной, на сегодняшний день, функциональностью и, с точки зрения конечного пользователя, отличаются, пожалуй, лишь средствами управления.

Обзор Web-серверов

В настоящий момент развитие технологий Интернета идет очень быстрыми темпами. Появляется много производителей программного обеспечения для Интернета, в том числе Web-серверов. Дадим краткую характеристику наиболее распространенным Web-серверам, используемым в корпоративных сетях и в "домашних" компьютерах, которые могут применяться для публикации информации из БД. Для начала определим само понятие Web-сервера.

Web-сервер — это программное средство, установленное на Web-узле глобальной или корпоративной сети и позволяющее пользователям сети получать доступ к гипертекстовым документам, расположенным на этом Web-узле. Иногда под Web-сервером понимают программное обеспечение Web-сервера вместе с его аппаратной частью, т. е. компьютером, на котором Web-сервер установлен.

В общем случае программное обеспечение Web-сервера может устанавливаться на компьютеры общего назначения, предназначенные для решения различных задач, не обязательно связанных с технологиями Интернета. Поэтому более корректно использовать понятие Web-сервера для обозначения только программного обеспечения, а компьютер с операционной системой и сетевой структурой называть средой работы Web-сервера, или платформой.

Отметим, что Web-серверы используются для следующих целей:

- создания корпоративных сетей интранет на основе принципов Интернета, многоуровневой архитектуры и клиент-серверных технологий;
- подключения корпоративных сетей интранет к Интернету для доступа к пре доставляемым в нем услугам;
- публикации информации из корпоративных сетей интранет, в том числе и содержимого БД из информационных систем, функционирующих в среде интранет;
- распространения собственной информации, находящейся на домашнем компьютере, создания собственного сайта.

В настоящее время в Интернете функционирует большое число типов серверов,

используемых для обеспечения различных функций. Кроме того, существует многочисленные однотипные серверы, разработанные различными производителями. В Интернете поддерживается ряд сайтов с информацией о серверах, содержащих ежемесячные обзоры всего, что касается данной темы. Например, один из наиболее популярных источников информации о статистике использования Web-серверов находится на узле компании Netcraft по адресу http://www.netcraft.com /survey, который помимо самой свежей информации о серверах и платформах содержит также список адресов узлов, где можно найти сведения об основных Web-серверах.
На выбор сервера большое влияние оказывает платформа, на которой работает Web-сервер. Анализ различных источников показывает, что в качестве узлов Web могут работать компьютеры любых типов, имеющие достаточные ресурсы и производительность. Активно используемых типов операционных систем намного меньше, чем типов компьютеров. Статистика показывает, что для высокопроизводительных объемных узлов наиболее часто используется операционная система UNIX (около 80% Web-серверов работают под ее управлением), а для средне- и низкопроизводительных узлов чаще всего используется Windows NT (менее 20% Web-серверов).

Операционные системы Web-серверов

В сети Интернет в основном используется несколько операционных систем. Как уже сказано выше, самыми распространенными среди Web-серверов являются UNIX-подобные операционные системы.

Операционная система UNIX и ее клоны получили наибольшее распространение в среде Интернет по следующим причинам:

UNIX может работать на значительно большем количестве платформ по сравнению с другими операционными системами. Она распространяется в исходных кодах, поэтому легко может быть перекомпилирована для любой аппаратной платформы.

UNIX раньше других начала применяться в Интернете.

UNIX поддерживает большое количество услуг, ориентирована на работу с большим количеством процессоров, а также адресов IP.

UNIX устойчиво работает в условиях даже весьма загруженных сетей.

Естественно, за все надо платить, и в результате система UNIX является самой сложной для изучения и конфигурирования из всех существующих операцион-ных систем.

Статистические данные

По данным сайта Netcraft, в рейтинге активных, мировых, сайтов Apache является несомненным лидером и прочно держит свои позиции не первый год [4]. Его доля составляет 50.72%от рынка (89.8 миллионов хостов). На втором месте держится nginx с долей в 14.82%. На третьем месте IIS, его доля 10.55%, что весьма достойный показатель в виду причин, указанных ниже. При сужении выборки до миллиона крупнейших web-сайтов доли Apache, nginx и IIS составляют 49.57%, 21.09% и 12.27%, соответственно.

Что касается русской части Интернета, то согласно представленным данным, наибольшей популярностью пользуется web-сервер nginx (53,2%) . На второй позиции фигурирует Apache с 36,5% установок, так же в тройке лидеров Microsoft IIS (6,1%).

Сервер *Apache*

Под управлением Apache работают более 6 миллионов серверов Интернета. Они довольно хорошо отлажены и протестированы разработчиками и пользователями. Группа разработчиков Apache придерживается строгих стандартов в отношении выпуска новых версий. Когда обнаруживаются ошибки в работе сервера, то компания Apache Development Group выпускает корректирующие файлы или новые версии продукта. Эта компания является международной организацией добровольцев, разработавших данный программный продукт для некоммерческого распространения среди широкого круга пользователей. Созданный под покровительством компании Apache Digital Corporation, проект Web-сервера Apache развивался как ветвь NCSA httpd проекта — одного из самых первых и наиболее эффективных из существующих серверов сети Интернет.
Само слово Apache звучит похоже на A PAtCHy server ("лоскутный" сервер). Одновременно это — просто красивое название, связанное с американским индейским племенем Apache,

известным своим военным мастерством и неутомимостью.
По сравнению с другими серверами Apache показал себя более устойчивым, более быстрым, имеющим более широкий набор функций и возможностей. Кроме того, Web-сервер Apache характеризуется открытой архитектурой, заключающейся в том, что этот сервер распространяется в исходных кодах и позволяет легко наращивать дополнительные возможности.
Его безусловное доминирование на рынке объясняется тем, что сервер был разработан для самой популярной платформы UNIX. Хотя сервер Apache распространяется бесплатно, но организация, разработавшая и обслуживающая этот мощный пакет (см. сервер www.apache.org), обеспечивает свое функционирование с помощью пользователей, которые спонсируют его развитие и сопровождение.
Используя открытый код Apache, разработчик может создавать собственные конфигурации сервера, компилируя внесенные в код изменения. Apache имеет модульную структуру, т. е. в его состав входит набор модулей, которые служат для обеспечения требуемых функций сервера и могут быть динамически включены в конфигурацию даже во время активной работы сервера. Сервер Apache позволяет использовать CGI-сценарии, написанные на Perl или PHP.
Перечислим основные особенности функционирования сервера Apache:

- является мощным, гибким, HTTP/1.1 -совместимым сервером;
- поддерживает современные протоколы;
- имеет легко перестраиваемую конфигурацию с возможностью установления дополнительных функций (модулей) от сторонних производителей;
- может быть сконфигурирован с использованием модулей API;
- снабжается полным исходным текстом и поступает с бесплатной лицензией на использование без ограничений;
- работает под управлением популярных операционных систем Windows NT/9x, Netware 5.x, OS/2 и большинства версий UNIX;
- поддерживает ведение отчетной документации об ошибках и файлы коррекции.

Сервер Apache поддерживает следующие функции:

- доступ к базам данных, используемым для аутентификации, т. е. возможность установки защищенных паролем страниц с огромным кфшчеством уполномоченных пользователей без перегрузки сервера;
- настройку реакции сервера на ошибки и сбои, заключающуюся в возможности устанавливать файлы или даже сценарии CGI, используемые сервером при возникновении ошибки (например, установка сценариев, позволяющих обрабатывать около 500 ошибок сервера, вести непрерывную диагностику и устранять неполадки по желанию пользователя);
- автоматическая обработка HTML-данных с изменяющейся структурой и мо дификация их для удобного представления информации клиенту;
- поддержка виртуальных хостов, заключающаяся в возможности настройки нескольких "домашних хостов", что позволяет серверу различать запросы, сделанные по различным IP-адресам; Apache также предоставляет возможность динамически настраивать функции "главного" виртуального хоста;
- генерация информации о настройках в удобном для пользователя формате;
- формирование на большинстве архитектур UNIX Apache так называемых журналов учета (log-файлов).

Отметим, что для сервера Apache отсутствуют официальное техническое обслуживание и поддержка. Тем не менее для этой программы можно найти большое количество информации или советов. Например, список известных сбоев можно найти на Web-узле, а со службой поддержки от независимых разработчиков можно связаться через список рассылки Apache comp.infosystems.www.servers.unix, через коммерческие службы, например, Cygnus (http://www.cygnus.com/product /idk/apache) или ежемесячный дайджест от разработчиков Apache. Еще одним неплохим источником технической информации является Apache Week.
Приведем краткое описание некоторых модулей Apache.

mod_access — отвечает за доступ к каталогам и файлам веб-сервера и переопределение ряда параметров веб-сервера для заданных каталогов и файлов.

mod_alias — отвечает за переадресацию и использование псевдонимов, позволяет перенаправлять запросы к физическим каталогам по их псевдонимам (алиасам). Такое перенаправление используется, например, для каталога с CGI-скриптами.

mod_asis позволяет отдавать клиенту запрошенный ресурс «как есть», без какой-либо обработки сервером.

Модули из семейства mod_auth отвечают за аутентификацию пользователей. Различные модули из этого семейства реализуют разные способы аутентификации. Подробности — в документации на модули mod_auth, mod_auth_dbm, mod_auth_digest и подобных.

mod_autoindex предназначен для автоматической генерации индексных файлов. Это может быть очень удобно при работе, например, с файловым архивом, когда нужно просто поместить на индексной странице названия файлов. С помощью директив этого модуля можно сортировать файлы, добавлять разным типам файлов свои иконки, отображать или скрывать файлы с заданными расширениями и так далее.

mod_deflate позволяет сжимать файлы в формат GZIP перед отправкой пользователю для ускорения загрузки.

mod_status позволяет администратору контролировать работу веб-сервера. Система будет сама записывать в файл все запросы, время перезагрузок и остановок сервера, загрузку процессора компьютера и другую информацию.

mod_proxy позволяет использовать Apache в качестве прямого или обратного прокси-сервера. В первом случае он управляет доступом во внешнюю сеть из ЛВС, во втором — напротив, предоставляет доступ к узлам ЛВС, которые не видны «извне».

mod_rewrite отвечает за перенаправление запросов и позволяет скрывать параметры скриптов. Например, с помощью этого модуля можно настроить преобразование клиентского запроса вида:

http://example.com/news/2009/05/03

к фактическому виду:

http://example.com/news.php?date=20090503

mod_perl — загружаемый (а не вызываемый, как в CGI) интерпретатор Perl.

mod_include — набор функций препроцессинга SSI (Server Side Includes), позволяющих «собирать» веб-страницу на стороне сервера из отдельных файлов.

Это далеко не полный перечень модулей Apache, но даже он позволяет дать представление о гибкости этого самого распространенного веб-сервера.

ISAPI (Internet Server Application Programming Interface) — это набор интерфейсов, предоставляемых веб-сервером MS IIS (Internet Information Services) для написания приложений, взаимодействующих с этим сервером и расширяющих его возможности. Приложения ISAPI представляют собой динамически подключаемые библиотеки (Dynamic Link Library, DLL), напрямую взаимодействующие с API IIS. Приложения ISAPI загружаются и выполняются в адресном пространстве IIS, поэтому серверу не нужно создавать новый процесс при каждом HTTP-запросе. Поскольку Windows загружает динамически подключаемую библиотеку один раз при первом вызове функции в DLL, то приложение ISAPI остается загруженным и не удаляется, пока не будет остановлен/выключен веб-сервер (если включено кэширование ISAPI), либо приложение не будет выгружено явным образом (если кэширование выключено).

Приложения ISAPI могут быть оформлены или в виде расширений, или в виде фильтров.

ISAPI-расширение — это приложение IIS, которое является адресатом запроса, оно выполняет действия, которые веб-сервер не может выполнять сам (например, обращение к базе данных). Расширение не влияет на параметры запроса, а использует их как входные данные. В этом ISAPI-расширение напоминает CGI-приложение. ISAPI-расширение может быть вызвано как явно (через запрос вида http://<URL/path>/isapiext.dll?paramstring), так и неявно (через карту расширений, в которой указаны обработчики для зарегистрированных

типов файлов (mapping), или при вызове через фильтр). Расширения ISAPI — наиболее частый способ применения ISAPI.

ISAPI-фильтр, в отличие от расширения, является своего рода посредником в обработке пользовательского запроса с момента его получения веб-сервером и до момента отправки ответа. Фильтр может модифицировать запрос или ответ, вызвать специфичный для конкретного запроса обработчик и т.п., при этом сам фильтр не является конечным обработчиком. Фильтры ISAPI довольно сложны в разработке и сфера их использования ограничена, как правило, решением таких задач, как шифрование, журналирование, аутентификация, сжатие данных.

Основное преимущество ISAPI — в скорости выполнения операций. ISAPI-приложение работает быстрее, чем обычная CGI-программа (которая должна всякий раз загружаться и выгружаться) или скрипт (который сначала должен быть загружен, потом обработан интерпретатором, которому, в общем случае, тоже нужно время на загрузку).

Среди недостатков — трудности при отладке ISAPI-приложений. Незамеченная ошибка может привести к нарушению работоспособности не только самой библиотеки dll, но и всего веб-сервера. При разработке ISAPI-приложений на универсальных языках программирования, таких как C++ или Object Pascal еще одной проблемой становится неконтролируемый код. Т.е. разработчик должен сам заботиться об управлении памятью, «сборке мусора» и прочих аспектах системного уровня. Еще один недостаток, пожалуй, самый главный — непереносимость. Несмотря на то, что ISAPI — открытая спецификация и поддержка технологии реализована, например, в модуле mod_isapi.dll для Apache for Win32, нет никакой возможности применять ISAPI на платформах, отличающихся от MS Windows.

6. БЕЗОПАСНОСТЬ И ШИФРОВАНИЕ

Шифрование — способ защиты данных от неавторизованных пользователей путем обработки и представления данных в искаженном зашифрованном виде. Неавторизованным считается пользователь, у которого нет ключа для расшифровки данных.

Таким образом, шифрование можно определить, как обработку данных, осуществляемую путем выполнения определенных математических действий, называемых алгоритмом шифрования, предназначенную для перевода данных в формат, недоступный для общего понимания. Перевод информации в исходное состояние осуществляется путем выполнения обратных преобразований, которые именуются расшифровкой. Ключом к шифру называется генерируемый в ходе шифрования набор символов, который является частью математической функции, заложенной в алгоритме шифрования. Именно благодаря ключу авторизованные пользователи, выполняя расшифровку данных, могут получить их в первоначальном виде. Наука, которая описывает способы сокрытия информации методами шифрования, называется криптографией

Благодаря шифрованию обеспечивается три главных аспекта безопасности информации:

– конфиденциальность (полезная информация скрывается от пользователей, не обладающих правами доступа к ней);

– целостность (при передаче и хранении данных полезная информация защищена от редактирования и внесения каких-либо изменений);

– идентифицируемость (источник или отправитель информации может быть аутентифицирован, что исключит возможность отказа отправителя от факта передачи данной информации. Алгоритмы шифрования делятся на три основных типа: бесключевые, одноключевые и двухключевые.

Первые основаны на общей математической функции, не включающей в себя дополнительных параметров. В других методах для усложнения алгоритма шифрования и расшифровки подключается дополнительный параметр — ключ, один или несколько, в зависимости от типа шифрования.

Шифрование методом AES.

Алгоритм шифрования AES основан на использования для кодирования и раскодировки информации одного и того же секретного ключа. Таким образом, отправитель шифрует информацию, а получатель, который заведомо владеет секретным ключом, расшифровывает. Алгоритм основан на выполнении определенного порядка простейших математических операций: подстановках, перестановках и линейных преобразованиях. Эти операции выполняются некоторое количество раз, называемое раундами. Данные делятся на блоки по 16 байтов, что позволяет получить ряд преимуществ перед традиционным потоковым шифрованием, поскольку изменение каждого бита в ключе или блоке открытого текста приведет к получению совершенно нового блока зашифрованных данных. Различают три основных типа AES шифрования, определенных по длине ключа:

– AES-128;

– AES-192;

– AES-256.

Длина ключей в данных типах шифрования равно 128, 192 и 256 бит соответственно. Алгоритм AES отличается достаточно высокой скоростью шифрования данных, а также сравнительно высокой степенью надежности. Попытка подбора AES ключа на современных ЭВМ по приблизительным расчетам ученых заняла бы период времени, сравнимый с предполагаемым возрастом Вселенной.

Шифрование методом RSA.

Алгоритм шифрования RSA относится к ассиметричным системами шифрованию, что означает использование для шифрования и расшифровки не одного ключа, а пары ключей: открытого, доступного всем пользователям и не требующего какой-либо защиты, и закрытого. Данные ключи работают в паре. Это означает, что сообщение, зашифрованное одним из ключей, может быть расшифровано только при наличии второго. Алгоритм основан на одной из главных математических проблем: факторизации целого числа. Данные, которые необходимо зашифровать, рассматриваются как одно большое целое число. В процессе шифрования это число увеличивается до степени ключа и делится с остатком на произведение двух заведомо выбранных простых чисел. Выполняя данный процесс с другим ключом, исходный текст можно получить вновь. Однако на основе описанного алгоритма можно сделать вывод, что данный шифр может быть взломан путем факторизации продукта, используемого при делении. Однако на данный момент времени вычислить необходимые коэффициенты для чисел, код числа которых превышает 768 бит, невозможно.

Кто понимает, что такое DES? AES? TLS? Биноминальное отображение? Мы поговорим о том, что такое криптографические примитивы, простые штучки, из которых впоследствии можно строить более сложные вещи, протоколы.
Мы будем говорить о трех примитивах: симметричном шифровании, аутентификации сообщений и асимметричном шифровании. Из них вырастает очень много протоколов. Сегодня мы попробуем поговорить про то, как вырабатываются ключи. В общем виде поговорим о том, как отправить защищенное сообщение, используя криптопримитивы, которые у нас есть, от одного пользователя другому.
Когда люди говорят про криптографию вообще, есть несколько фундаментальных принципов. Один из них — принцип Керкгоффса, который говорит, что open source в криптографии очень важен. Если точнее, он дает общее знание об устройстве протоколов. Смысл очень простой: криптографические алгоритмы, которые используются в той или иной системе, не должны быть секретом, обеспечивающим ее устойчивость. В идеале необходимо строить системы так, чтобы их криптографическая сторона была полностью известна атакующему и единственным секретом являлся криптографический ключ, который в данной системе используется.
Современные и коммерчески доступные системы шифрования — все или почти все или лучшие из них — построены из компонент, устройство и принцип работы которых хорошо известны. Единственная секретная вещь в них — ключ шифрования. Есть только одно известное значимое исключение — набор секретных криптографических протоколов для всевозможных государственных организаций. В США это называется NSA suite B, а в России это странные секретные алгоритмы шифрования, которые до определенной степени

используются военными и государственными органами.
Не сказать, что такие алгоритмы приносят им большую пользу, за исключением того, что это примерно как атомная физика. Можно попытаться по пониманию дизайна протокола понять направление мысли людей, которые его разработали, и неким образом обогнать другую сторону. Не знаю, насколько такой принцип актуален по нынешним меркам, но люди, знающие поступают именно так.
В каждом коммерческом протоколе, с которым вы столкнетесь, ситуация обстоит иначе. Там везде используется открытая система, все придерживаются этого принципа.
Первый криптографический примитив — симметричные шифры. Они очень простые. У нас есть какой-то алгоритм, на вход которого поступает открытый текст и нечто, называемое ключом, какое-то значение. На выходе получается зашифрованное сообщение. Когда мы хотим его дешифровать, важно, чтобы мы брали тот же самый ключ шифрования. И, применяя его к другому алгоритму, алгоритму расшифровки, мы из шифротекста получаем наш открытый текст назад.

Какие здесь важные нюансы? В большинстве распространенных алгоритмов симметричного шифрования, с которыми можно столкнуться, размер шифротекста всегда равен размеру открытого текста. Современные алгоритмы шифрования оперируют размерами ключей. Размер ключей измеряется в битах. Современный размер — от 128 до 256 бит для алгоритмов симметричного шифрования. Об остальном, в том числе о размере блока, мы поговорим позже.

Исторически, в условном IV веке до нашей эры, существовало два метода дизайна шифров: шифры подстановки и перестановки. Шифры подстановки — алгоритм, где в те времена заменяли одну букву сообщения на другую по какому-то принципу. Простой шифр подстановки — по таблице: берем таблицу, где написано, что А меняем на Я, Б на Ю и т. д. Дальше по этой таблице шифруем, по ней же дешифруем.
Как вы считаете, с точки зрения размера ключа насколько это сложный алгоритм? Сколько вариантов ключей существует? Порядок факториала длины алфавита. Мы берем таблицу. Как мы ее строим? Допустим, есть таблица на 26 символов. Букву А можем заменить на любой из них, букву Б — на любой из оставшихся 25, С — на любой из оставшихся 24… Получаем 26*25*24*… — то есть факториал от 26. Факториал размерности алфавита. Если взять $\log_2 26!$, это будет очень много. Вы точно получите в районе 100 бит длины ключа, а то и больше. Оказалось, что с точки зрения формального представления стойкости указанный алгоритм шифрования — довольно неплохой. 100 бит — приемлемо. При этом все, наверное, в детстве или юности, когда сталкивались с кодировками, видели, что такие алгоритмы дешифруются тривиально. Проблем с расшифровкой нет.
Долго существовали всякие алгоритмы подстановки в разных конструкциях. Одним из них, еще более примитивным, является шифр Цезаря, где таблица формируется не случайной перестановкой символов, а сдвигом на три символа: А меняется на D, В на Е и т. д. Понятно, что шифр Цезаря вместе со всеми его вариантами перебрать очень легко: в отличие от табличной подстановки, в ключе Цезаря всего 25 вариантов при 26 буквах в алфавите — не считая тривиального шифрования самого в себя. И его как раз можно перебрать полным перебором. Здесь есть некоторая сложность.
Почему шифр табличной подстановки такой простой? Откуда возникает проблема, при которой мы можем легко, даже не зная ничего про криптографию, расшифровать табличную подстановку? Дело в частотном анализе. Есть самые распространенные буквы — какая-нибудь И или Е. Их распространенность велика, гласные встречаются намного чаще, чем согласные, и существуют негативные пары, никогда не встречающиеся в естественных языках, — что-то вроде ЬЪ.

В чем проблема? Надо статистику распределения букв исказить, чтобы распространенные буквы не так светились в зашифрованном тексте. Очевидный способ: давайте будем шифровать самые часто встречающиеся буквы не в один символ, а в пять разных, например. Если буква встречается в среднем в пять раз чаще, то давайте по очереди — сначала в первый символ будем зашифровывать, потом во второй, в третий и т. д. Далее у нас получится маппинг букв не 1 к 1, а, условно, 26 к 50. Статистика, таким образом,

нарушится. Перед нами первый пример полиалфавитного шифра, который как-то работал. Однако с ним есть довольно много проблем, а главное, очень неудобно работать с таблицей. Дальше придумали: давайте не будем шифровать такими таблицами, а попробуем брать шифр Цезаря и для каждой следующей буквы изменять сдвиг. Результат — шифр Виженера. Берем в качестве ключа слово ВАСЯ. Берем сообщение МАША. Задействуем шифр Цезаря, но отсчитывая от этих букв. Например, В — третья буква в алфавите. Мы должны сдвинуть на три буквы соответствующую букву в открытом тексте. М сдвигается в П. А в А. Ш — на 16, перескочим букву А, получим, условно, Д. Я сдвинет А в Я. ПАДЯ.
Что удобно в получившемся шифре? Здесь было две одинаковых буквы, но в результате они зашифровались в разные. Это хорошо, потому что размывает статистику. Метод хорошо работал, пока где-то в XIX веке, буквально недавно на фоне истории криптографии, не придумали, как его ломать. Если посмотреть на сообщение из нескольких десятков слов, а ключ довольно короткий, то вся конструкция выглядит как несколько шифров Цезаря. Мы говорим: хорошо, давайте каждую четвертую букву — первую, пятую, девятую — рассматривать как шифр Цезаря. И поищем среди них статистические закономерности. Мы обязательно их найдем. Потом возьмем вторую, шестую, десятую и так далее. Опять найдем. Тем самым мы восстановим ключ. Единственная проблема — понять, какой он длины. Это не очень сложно, какой он может быть длины? 4,10 символов. Перебрать 6 вариантов от 4 до 10 не сложно. Простая атака — она была доступна и без компьютеров, просто за счет ручки и листа бумаги.
Как из этого сделать невзламываемый шифр? Взять ключ размера текста. Персонаж по имени Клод Шэннон в XX веке, в 1946 году, написал классическую первую работу по криптографии как по разделу математики, где сформулировал теорему. Длина ключа равна длине сообщения — он использовал XOR вместо сложения по модулю, равному длине алфавита, но в данной ситуации — это не очень принципиально. Ключ сгенерирован случайным образом, является последовательностью случайных бит, и на выходе тоже получится случайная последовательность бит. Теорема: если у нас есть такой ключ, то подобная конструкция является абсолютно стойкой. Доказательство не очень сложное, но сейчас не будем про него говорить.
Важно, что можно создать невзламываемый шифр, но у него есть недостатки. Во-первых, ключ должен быть абсолютно случайным. Во-вторых, он никогда не должен использоваться повторно. В-третьих, длина ключа должна быть равна длине сообщения. Почему нельзя использовать один и тот же ключ для шифровки разных сообщений? Потому что, перехватив этот ключ в следующий раз, можно будет расшифровать все сообщения? Нет. В первых символах будет виден шифр Цезаря? Нет.
Возьмем два сообщения: МАША, зашифрованная ключом ВАСЯ, и другое слово, у которого ключ тоже был ВАСЯ, — ВЕРА. Получим примерно следующее: ЗЕШЯ. Сложим два полученных сообщения, причем так, чтобы два ключа взаимно удалились. В итоге получим лишь разницу между осмысленным шифротекстом и осмысленным шифротекстом. На XOR это делается удобнее, чем на сложении по длине алфавита, но разницы практически никакой. Если мы получили разницу между двумя осмысленными шифротекстами, то дальше, как правило, становится намного легче, поскольку у текстов на естественном языке высокая избыточность. Зачастую мы можем догадаться, что происходит, делая разные предположения, гипотезы. А главное, что каждая верная гипотеза будет раскрывать нам кусочек ключа, а значит и кусочки двух шифротекстов. Как-то так. Поэтому плохо.
Помимо шифров подстановки, были еще шифры перестановки. С ними тоже все довольно просто. Берем сообщение ВАСЯИ, записываем его в блок какой-то длины, например в ДИДОМ, и считываем результат так же. Переберем все возможные варианты перестановок. Тут их не очень много. Берем длину блока, подбираем и восстанавливаем.
В качестве следующей итерации был выбран такой способ: возьмем все то же самое, а сверху напишем какой-нибудь ключ — СИМОН. Переставим столбцы так, чтобы буквы оказались в алфавитном порядке. В итоге получим новую перестановку по ключу. Она уже намного лучше старой, поскольку количество перестановок намного больше и подобрать ее не всегда легко.

Каждый современный шифр тем или иным способом базируется на этих двух принципах — подстановки и перестановки. Сейчас их использование намного более сложное, но сами базовые принципы остались прежними.

Если говорить про современные шифры, они делятся на две категории: поточные и блочные. Поточный шифр устроен так, что фактически представляет собой генератор случайных чисел, выход которого мы складываем по модулю 2, «ксорим», с нашим шифротекстом. Ранее уже говорилось: если длина получившегося ключевого потока — она же ключ — абсолютно случайная, никогда повторно не используется и ее длина равна длине сообщения, то у нас получился абсолютно стойкий шифр, невзламываемый. Возникает вопрос: как сгенерировать на такой шифр случайный, длинный и вечный Ключ? Как вообще работают поточные шифры? По сути, они представляют собой генератор случайного числа на основе какого-то начального значения. Начальное значение и является ключом шифра, ответом.

Из этой истории есть одно занятное исключение — шифроблокноты. Речь идет о настоящей шпионской истории про настоящий шпионаж. Некие люди, которым нужна абсолютно устойчивая коммуникация, генерируют случайные числа — например, буквальным бросанием кубика или буквальным вниманием шаров из барабана, как в лото. Создают два листа, где печатают эти случайные числа. Один лист отдают получателю, а второй оставляют у отправителя. При желании пообщаться они используют этот поток случайных чисел в качестве ключевого потока. История взята не из совсем далекого прошлого. У меня есть настоящий радиоперехват от 15 октября 2014 года: 7 2 6, 7 2 6, 7 2 6. Это позывной. 4 8 3, 4 8 3, 4 8 3. Это номер шифроблокнота. 5 0, 5 0, 5 0. Это количество слов. 8 4 4 7 9 8 4 4 7 9 2 0 5 1 4 2 0 5 1 4 и т. д. 50 таких числовых групп. Неизвестно где, где-то не в России сидел какой-нибудь человек с ручкой и карандашом у обычного радиоприемника и записывал эти цифры. Записав их, он достал похожую вещь, сложил их по модулю 10 и получил свое сообщение. Другими словами, это действительно работает, и подобное сообщение нельзя взломать. Если действительно были сгенерированы хорошие случайные числа и он впоследствии сжег бумажку с ключом, то осуществить взлом нельзя никак, совсем.

Но тут есть довольно много проблем. Первая — как нагенерировать по-настоящему хорошие случайные числа. Мир вокруг нас детерминирован, и если мы говорим про компьютеры, они детерминированы полностью.

Во-вторых, доставлять ключи такого размера. Если мы говорим про передачу сообщений из 55 цифровых групп, то проделать подобное не очень сложно, а вот передать несколько гигабайт текста — уже серьезная проблема. Следовательно, нужны какие-нибудь алгоритмы, которые, по сути, генерируют псевдослучайные числа на основе какого-нибудь небольшого начального значения и которые могли бы использоваться в качестве таких потоковых алгоритмов.

Самый исторически распространенный алгоритм подобного рода называется RC4. Он был разработан Роном Ривестом лет 25 назад и активно использовался очень долго, был самым распространенным алгоритмом для TLS, всех его различных вариантов, включая HTTPS. Но в последнее время RC4 начал показывать свой возраст. Для него существует некоторое количество атак. Он активно используется в WEP.

RC4 устроен несложно. Есть внутренний байтовый стейт из 256 байт. На каждом шаге этого стейта есть два числа, два указателя на разные байты в стейте. И на каждом шаге происходит сложение между этими числами — они помещаются в некоторое место стейта. Полученный оттуда байт является следующим байтом в числовой последовательности. Вращая эту ручку таким образом, выполняя подобное действие на каждом шаге, мы получаем каждый следующий байт. Мы можем получать следующий байт числовой последовательности вечно, потоком.

Большое достоинство RC4 — в том, что он целиком внутрибайтовый, а значит, его программная реализация работает довольно быстро — сильно быстрее, в разы, если не в десятки раз быстрее, чем сравнимый и существовавший примерно в одно время с ним шифр DES. Поэтому RC4 и получил такое распространение. Он долго был коммерческим секретом компании RSA, но потом, где-то в районе 90-х годов, некие люди анонимно опубликовали

исходники его устройства в списке рассылки cypherpunks. В результате возникло много драмы, как же так, какие-то неприличные люди украли интеллектуальную собственность компании RSA и опубликовали ее. RSA начала грозить всем патентами, всевозможными юридическими преследованиями. Чтобы их избежать, все реализации алгоритма, которые находятся в опенсорсе, называются не RC4, а ARC4 или ARCFOUR. A — alleged. Речь идет о шифре, который на всех тестовых кейсах совпадает с RC4, но технически вроде как им не является.

Если вы конфигурируете какой-нибудь SSH или OpenSSL, вы в нем не найдете упоминания RC4, а найдете ARC4 или что-то подобное. Несложная конструкция, он уже старенький, на него сейчас есть атаки, и он не очень рекомендуется к использованию.

Было несколько попыток его заменить. Наверное, самым успешным стал шифр Salsa20 и несколько его последователей от широко известного в узких кругах персонажа Дэна Берштайна. Линуксоидам он обычно известен как автор qmail.

Salsa20 устроен сложнее, чем DES. Его блок-схема сложная, но он обладает несколькими интересными и классными свойствами. Для начала, он всегда выполняется за конечное время, каждый его раунд, что немаловажно для защиты от тайминг-атак. Это такие атаки, где атакующий наблюдает поведение системы шифрования, скармливая ей разные шифротексты или разные ключи за этим черным ящиком. И, понимая изменения во времени ответа или в энергопотреблении системы, он может делать выводы о том, какие именно процессы произошли внутри. Если вы думаете, что атака сильно надуманная, это не так. Очень широко распространены атаки подобного рода на смарт-карты — очень удобные, поскольку у атакующего есть полный доступ к коробке. Единственное, что он, как правило, не может в ней сделать, — прочитать сам ключ. Это сложно, а делать все остальное он может — подавать туда разные сообщения и пытаться их расшифровать.

Salsa20 устроен так, чтобы он всегда выполнялся за константное одинаковое время. Внутри он состоит всего из трех примитивов: это сдвиг на константное время, а также сложение по модулю 2 и по модулю 32, 32-битных слов. Скорость Salsa20 еще выше, чем у RC4. Он пока что не получил такого широкого распространения в общепринятой криптографии — у нас нет cipher suite для TLS, использующих Salsa20, — но все равно потихоньку становится популярным. Указанный шифр стал одним из победителей конкурса eSTREAM по выбору лучшего поточного шифра. Их там было четыре, и Salsa — один из них. Он потихоньку начинает появляться во всяких опенсорс-продуктах. Возможно, скоро — может, через пару лет — появятся даже cipher suite в TLS с Salsa20.

На него имеется некоторое количество криптоанализа, есть даже атаки. Снаружи он выглядит как поточный, генерируя на основе ключа последовательность почти произвольной длины, 2^{64}. Зато внутри он работает как блочный. В алгоритме есть место, куда можно подставить номер блока, и он выдаст указанный блок.

Какая проблема с поточными шифрами? Если у вас есть поток данных, передаваемый по сети, поточный шифр для него удобен. К вам влетел пакет, вы его зашифровали и передали. Влетел следующий — приложили эту гамму и передали. Первый байт, второй, третий по сети идут. Удобно.

Если данные, например гигабайтный файл целиком, зашифрованы на диске поточным шифром, то чтобы прочитать последние 10 байт, вам нужно будет сначала сгенерировать гаммы потока шифра на 1 гигабайт, и уже из него взять последние 10 байт. Очень неудобно. В Salsa указанная проблема решена, поскольку в нем на вход поступает в том числе и номер блока, который надо сгенерировать. Дальше к номеру блока 20 раз применяется алгоритм. 20 раундов — и мы получаем 512 бит выходного потока.

Самая успешная атака — в 8 раундов. Сам он 256-битный, а сложность атаки в 8 раундов — 250 или 251 бит. Считается, что он очень устойчивый, хороший. Публичный криптоанализ на него есть. Несмотря на всю одиозность личности Берштайна в этом аспекте, вещь хорошая и у нее большее будущее.

Исторически поточных шифров было много. Они первые не только в коммерческом шифровании, но и в военном. Там использовалось то, что называлось линейными регистрами сдвига.

Какие тут проблемы? Первая: в классических поточных шифрах, не в Salsa, чтобы расшифровать последнее значение гигабайтного файла, последний байт, вам нужно сначала сгенерировать последовательность на гигабайт. От нее вы задействуете только последний байт. Очень неудобно.
Поточные шифры плохо пригодны для систем с непоследовательным доступом, самый распространенный пример которых — жесткий диск.
Есть и еще одна проблема, о ней мы поговорим дальше. Она очень ярко проявляется в поточных шифрах. Две проблемы в совокупности привели к тому, что здорово было бы использовать какой-нибудь другой механизм.
Другой механизм для симметричного шифрования называется блочным шифром. Он устроен чуть по-другому. Он не генерирует этот ключевой поток, который надо ксорить с нашим шифротекстом, а работает похоже — как таблица подстановок. Берет блок текста фиксированной длины, на выходе дает такой же длины блок текста, и всё.
Размер блока в современных шифрах — как правило, 128 бит. Бывают разные вариации, но как правило, речь идет про 128 или 256 бит, не больше и не меньше. Размер ключа — точно такой же, как для поточных алгоритмов: 128 или 256 бит в современных реализациях, от и до.
Из всех широко распространенных блочных шифров сейчас можно назвать два — DES и AES. DES очень старый шифр, ровесник RC4. У DES сейчас размер блока — 64 бита, а размер ключа — 56 бит. Создан он был в компании IBM под именем Люцифер. Когда в IBM его дизайном занимался Хорст Фейстель, они предложили выбрать 128 бит в качестве размера блока. А размер ключа был изменяемый, от 124 до 192 бит.
Когда DES начал проходит стандартизацию, его подали на проверку в том числе и в АНБ. Оттуда он вернулся с уменьшенным до 64 бит размером блока и уменьшенным до 56 бит размером ключа.

20 лет назад вся эта история наделала много шума. Все говорили — наверняка они туда встроили закладку, ужасно, подобрали такой размер блока, чтобы получить возможность атаковать. Однако большое достоинство DES в том, что это первый шифр, который был стандартизован и стал тогда основой коммерческой криптографии.
Его очень много атаковали и очень много исследовали. Есть большое количество всевозможных атак. Но ни одной практически реализуемой атаки до сих пор нет, несмотря на его довольно почтенный возраст. Единственное, размер ключа в 56 бит сейчас просто неприемлемый и можно атаковать полным перебором.
Как устроен DES? Фейстель сделал хорошую вещь, которую называют сетью Фейстеля. Она оперирует блоками. Каждый блок, попадающий на вход, делится на две части: левую и правую. Левая часть становится правой без изменений. Правая часть ксорится с результатом вычисления некой функции, на вход которой подается левая часть и ключ. После данного преобразования правая часть становится левой.

У нее есть несколько интересных достоинств. Первое важное достоинство: функция F может быть любой. Она не должна обладать свойствами обратимости, она может и не быть линейной или нелинейной. Все равно шифр остается симметричным.
Второе очень удобное свойство: расшифровка устроена так же, как шифрование. Если нужно расшифровать данную сеть, вы в прежний механизм вместо открытого текста засовываете шифротекст и на выходе вновь получаете открытый текст.
Почему это удобно? 30 лет назад удобство являлось следствием того, что шифраторы были аппаратными и заниматься дизайном отдельного набора микросхем для шифрования и для расшифровки было трудоемко. А в такой конструкции все очень здорово, фактически мы можем один блок использовать для разных задач.
В реальной ситуации такая конструкция — один раунд блочного шифра, то есть в реальном шифре она выполняется 16 раз с разными ключами. На каждом 16 раунде генерируется отдельный ключ и 16 раундовых подключей, каждый из которых применяется на каждом раунде для функции F.
Раунд тоже выглядит довольно несложно — он состоит всего из двух-трех операций. Первая операция: размер попавшегося полублока становится равен 32 бита, полубок проходит

функцию расширения, на вход попадает 32 бита. Дальше мы по специальной несекретной таблице немного добавляем к 32 битам, превращая их в 48: некоторые биты дублируются и переставляются, такая гребеночка.
Потом мы его ксорим с раундовым ключом, размер которого — тоже 48 бит, и получаем 48-битное значение.
Затем оно попадает в набор функций, которые называются S-боксы и преобразуют каждый бит входа в четыре бита выхода. Следовательно, на выходе мы из 48 бит снова получаем 32 бита.
И наконец, окончательная перестановка P. Она опять перемешивает 32 бита между собой.
Все очень несложно, раундовая функция максимально простая.
Самое интересное ее свойство заключается в указанных S-боксах: задумано очень сложное превращение 6 бит в 4. Если посмотреть на всю конструкцию, видно, что она состоит из XOR и пары перестановок. Если бы S-боксы были простыми, весь DES фактически представлял бы собой некоторый набор линейных преобразований. Его можно было бы представить, как матрицу, на которую мы умножаем наш открытый текст, получая шифротекст. И тогда атака на DES была бы тривиальной: требовалось бы просто подобрать матрицу.

Вся нелинейность сосредоточена в S-боксах, подобранных специальным образом. Существуют разные анекдоты о том, как именно они подбирались. В частности, примерно через 10 лет после того, как DES был опубликован и стандартизован, криптографы нашли новый тип атак — дифференциальный криптоанализ. Суть атаки очень простая: мы делаем мелкие изменения в открытом тексте — меняя, к примеру, значение одного бита с 0 на 1 — и смотрим, что происходит с шифротекстом. Выяснилось, что в идеальном шифре изменение одного бита с 0 на 1 должно приводить к изменению ровно половины бит шифротекста. Выяснилось, что DES, хоть он и был сделан перед тем, как открыли дифференциальный криптоанализ, оказался устойчивым к этому типу атак. В итоге в свое время возникла очередная волна паранойи: АНБ еще за 10 лет до открытых криптографов знало про существование дифференциального криптоанализа, и вы представляете себе, что оно может знать сейчас.
Анализу устройства S-боксов посвящена не одна сотня статей. Есть хорошие статьи, которые называются примерно так: особенности статистического распределения выходных бит в четвертом S-боксе. Потому что шифру много лет, он досконально исследован в разных местах и остается достаточно устойчивым даже по нынешним меркам.
56 бит сейчас уже можно просто перебрать на кластере машин общего назначения — может, даже на одном. И это плохо. Что можно предпринять?
Просто сдвинуть размер ключа нельзя: вся конструкция завязана на его длину. Triple DES.
Очевидный ответ был таким: давайте мы будем шифровать наш блок несколько раз, устроим несколько последовательных шифрований. И здесь всё не слишком тривиально.
Допустим, мы берем и шифруем два раза. Для начала нужно доказать, что для шифрований k1 и k2 на двух разных ключах не существует такого шифрования на ключе k3, что выполнение двух указанных функций окажется одинаковым. Здесь вступает в силу свойство, что DES не является группой. Тому существует доказательство, пусть и не очень тривиальное.
Хорошо, 56 бит. Давайте возьмем два — k1 и k2. 56 + 56 = 112 бит. 112 бит даже по нынешним меркам — вполне приемлемая длина ключа. Можно считать нормальным всё, что превышает 100 бит. Так почему нельзя использовать два шифрования, 112 бит?
Одно шифрование DES состоит из 16 раундов. Сеть применяется 16 раз. Изменения слева направо происходят 16 раз. И он — не группа. Есть доказательство того, что не существует такого ключа k3, которым мы могли бы расшифровать текст, последовательно зашифрованный выбранными нами ключами k1 и k2.
Есть атака. Давайте зашифруем все возможные тексты на каком-нибудь ключе, возьмем шифротекст и попытаемся его расшифровать на всех произвольных ключах. И здесь, и здесь получим 2^{56} вариантов. И где-то они сойдутся. То есть за два раза по 2^{56} вариантов — плюс память для хранения всех расшифровок — мы найдем такую комбинацию k1 и k2, при

которых атака окажется осуществимой.
Эффективная стойкость алгоритма — не 112 бит, а 57, если у нас достаточно памяти. Нужно довольно много памяти, но тем не менее. Поэтому решили — так работать нельзя, давайте будем шифровать три раза: k1, k2, k3. Конструкция называется Triple DES. Технически она может быть устроена по-разному. Поскольку в DES шифрование и дешифрование — одно и то же, реальные алгоритмы иногда выглядят так: зашифровать, расшифровать и снова расшифровать — чтобы выполнять операции в аппаратных реализациях было проще. Наша обратная реализация Triple DES превратится в аппаратную реализацию DES. Это может быть очень удобно в разных ситуациях для задачи обратной совместимости.
Где применялся DES? Вообще везде. Его до сих пор иногда можно пронаблюдать для TLS, существуют cipher suite для TLS, использующие Triple DES и DES. Но там он активно отмирает, поскольку речь идет про софт. Софт легко апдейтится.
А вот в банкоматах он отмирал очень долго, и нет уверенности, что окончательно пропал. Неизвестно, нужна ли отдельная лекция о том, как указанная конструкция устроена в банкоматах. Если коротко, клавиатура, где вы вводите PIN, — самодостаточная вещь в себе. В нее загружены ключи, и наружу она выдает не PIN, а конструкцию PIN-блок. Конструкция зашифрована — например, через DES. Поскольку банкоматов огромное количество, то среди них много старых и до сих пор можно встретить банкомат, где внутри коробки реализован даже не Triple DES, а обычный DES.
Однажды DES стал показывать свой возраст, с ним стало тяжело, и люди решили придумать нечто поновее. Американская контора по стандартизации, которая называется NIST, говорит: давайте проведем конкурс и выберем новый шифр. Им стал AES.
DES расшифровывается как digital encrypted standard. AES — advanced encrypted standard. Размер блока в AES — 128 бит, а не 64. Это важно с точки зрения криптографии. Размер ключа у AES — 128, 192 или 256 бит. В AES не используется сеть Фейстеля, но он тоже многораундовый, в нем тоже несколько раз повторяются относительно примитивные операции. Для 128 бит используется 10 раундов, для 256 — 14.
Сейчас рассмотрим, как устроен каждый раунд. Первый и последний раунды чуть отличаются от стандартной схемы — тому есть причины.
Как и в DES, в каждом раунде AES есть свои раундовые ключи. Все они генерируются из ключа шифрования для алгоритма. В этом месте AES работает так же, как DES. Берется 128-битный ключ, из него генерируется 10 подключей для 10 раундов. Каждый подключ, как и в DES, применяется на каждом конкретном раунде.
Каждый раунд состоит из четырех довольно простых операций. Первый раунд — подстановка по специальной таблице.
В AES мы строим байтовую матрицу размером 4 на 4. Каждый элемент матрицы — байт. Всего получается 16 байт или 128 бит. Они и составляют блок AES целиком.
Вторая операция — побайтовый сдвиг.
Устроен он несложно, примитивно. Мы берем матрицу 4 на 4. Первый ряд остается без изменений, второй ряд сдвигается на 1 байт влево, третий — на 2 байта, четвертый — на 3, циклично.
Далее производим перемешивание внутри колонок. Это тоже очень несложная операция. Она фактически переставляет биты внутри каждой колонки, больше ничего не происходит. Можно считать ее умножением на специальную функцию.
Четвертая, вновь очень простая операция — XOR каждого байта в каждой колонке с соответствующим байтом ключа. Получается результат.
В первом раунде лишь складываются ключи, а три других операции не используются. В последнем раунде не происходит подобного перемешивания столбцов:
Дело в том, что это не добавило бы никакой криптографической стойкости и мы всегда можем обратить последний раунд. Решили не затормаживать конструкцию лишней операцией.
Мы повторяем 4 описанных шага 10 раз, и на выходе из 128-битного блока снова получаем 128-битный блок.
Какие достоинства у AES? Он оперирует байтами, а не битами, как DES. AES намного

быстрее в софтовых реализациях. Если сравнить скорость выполнения AES и DES на современной машине, AES окажется в разы быстрее, даже если говорить о реализации исключительно в программном коде.
Производители современных процессоров, Intel и AMD, уже разработали ассемблерные инструкции для реализации AES внутри чипа, потому что стандарт довольно несложный. Как итог — AES еще быстрее. Если через DES на современной машинке мы можем зашифровать, например, 1-2 гигабита, то 10-гигабитный AES-шифратор находится рядом и коммерчески доступен обычным компаниям.
Блочный алгоритм шифрует блок в блок. Он берет блок на 128 или 64 бита и превращает его в блок на 128 или 64 бита.
А что мы будем делать, если потребуется больше, чем 16 байт?
Первое, что приходит в голову, — попытаться разбить исходное сообщение на блоки, а блок, который останется неполным, дополнить стандартной, известной и фиксированной последовательностью данных.
Да, очевидно, побьем всё на блоки по 16 байт и зашифруем. Такое шифрование называется ECB — electronic code boot, когда каждый из блоков по 16 байт в случае AES или по 8 байт в случае DES шифруется независимо.

Шифруем каждый блок, получаем шифротекст, складываем шифротексты и получаем полный результат.

Примерно так выглядит картинка, зашифрованная в режиме ECB. Даже если мы представим себе, что шифр полностью надежен, кажется, что результат менее чем удовлетворительный. В чем проблема? В том, что это биективное отображение. Для одинакового входа всегда получится одинаковый выход, и наоборот — для одинакового шифротекста всегда получится одинаковый открытый текст.
Надо как-нибудь исхитриться и сделать так, чтобы результат на выходе все время получался разным, в зависимости от местонахождения блока — несмотря на то, что на вход подаются одинаковые блоки шифротекста. Первым способом решения стал режим CBC.

Мы не только берем ключ и открытый текст, но и генерируем случайное число, которое не является секретным. Оно размером с блок. Называется оно инициализационным вектором.
При шифровании первого блока мы берем инициализационный вектор, складываем его по модулю 2 с открытым текстом и шифруем. На выходе — шифротекст. Дальше складываем полученный шифротекст по модулю 2 со вторым блоком и шифруем. На выходе — второй блок шифротекста. Складываем его по модулю 2 с третьим блоком открытого текста и шифруем. На выходе получаем третий блок шифротекста. Здесь видно сцепление: мы каждый следующий блок сцепляем с предыдущим.
В результате получится картинка, где всё, начиная со второго блока, равномерно размазано, а первый блок каждый раз зависит от инициализационного вектора. И она будет абсолютно перемешана.