Федеральное государственное автономное образовательное
учреждение высшего образования
«Санкт-Петербургский государственный университет
аэрокосмического приборостроения»

Основы программной инженерии

Методические указания к выполнению лабораторных работ

Составители:

к.т.н., стар. преп. П.А. Охтилев, асс. А.Э. Зянчурин

Рецензент:

д.т.н., проф. М.Ю. Охтилев

Санкт-Петербург – 2021

Содержание

Требования к оформлению отчетов

Структура и форма отчета о лабораторной работе

Итоговый отчёт по лабораторным работам должен состоять из частей, перечисленных ниже. Отсутствие указанных ниже частей не допускается. Общий объем отчёта должен составлять не менее 15 страниц. Страницы должны быть пронумерованы. Размер шрифта основного текста — 14 пунктов.

1. *Титульный лист* должен соответствовать образцу на сайте ГУАП. При оформлении титульного листа обязательно наличие следующей информации:
   - название дисциплины;
   - ФИО преподавателя, принимающего работу;
   - ФИО обучающихся, выполнивших работу.
   - Отчёты, содержащие неверную информацию на титульном листе, к сдаче не принимаются.
2. *Содержание* с указанием номеров страниц (желательно составленное автоматически).
3. *Подписанное преподавателем задание* на лабораторные работы.
4. *Краткое описание хода выполнения работ*: постановка задачи на каждую работу; описание использованных моделей, алгоритмов, программных компонент. Описание должно быть сопровождено расчетными материалами, снимками с экрана компьютера («скриншотами»), отражающими ход выполнения работы.
5. *Выводы* по результатам выполняемых лабораторных работ.
6. *Список цитируемой и использованной литературы.*

Требования к оформлению отчета о лабораторной работе

Изложение текста и оформление лабораторных работ следует выполнять в соответствии с ГОСТ 2.105-2019 – ЕСКД и общие требования к текстовым документам ГОСТ 7.32 – 2017 – СИБИД, соблюдая следующие требования.

1. *Оформление титульного листа.* Титульный лист следует оформлять на бланке. Бланки для оформления титульных листов учебных работ представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.
2. *Оформление основного текста работы:*
   - следует использовать шрифт Times New Roman размером не менее 12 пт (допускается 14 пт), строчный, без выделения, с выравниванием по ширине;
   - абзацный отступ должен быть одинаковым и равен по всему тексту 1,25 см;
   - строки разделяются полуторным интервалом;

- поля страницы: верхнее и нижнее – 20 мм, левое – 30 мм, правое – 15 мм;
- полужирный шрифт применяется только для заголовков разделов и подразделов. заголовков структурных элементов;
- разрешается использовать компьютерные возможности акцентирования внимания на определенных терминах, формулах, теоремах, применяя шрифты разной гарнитуры;
- наименования структурных элементов работы: «СОДЕРЖАНИЕ», «ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ», «ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ», «ВВЕДЕНИЕ», «ЗАКЛЮЧЕНИЕ», «СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ», «ПРИЛОЖЕНИЕ» следует располагать в середине строки без точки в конце, прописными (заглавными) буквами, не подчеркивая;
- введение и заключение не нумеруются.
- каждый структурный элемент и каждый раздел основной части следует начинать с новой страницы.

3. *Оформление основной части работы следует делить на разделы и подразделы:*
   - разделы и подразделы должны иметь порядковую нумерацию в пределах всего текста, за исключением приложений;
   - нумеровать их следует арабскими цифрами;
   - номер подраздела должен включать номер раздела и порядковый номер подраздела, разделенные точкой;
   - после номера раздела и подраздела в тексте точка не ставится;
   - разделы и подразделы должны иметь заголовки;
   - если заголовок раздела, подраздела или пункта занимает не одну строку, то каждая следующая строка должна начинаться с начала строки, без абзацного отступа;
   - заголовки разделов и подразделов следует печатать с абзацного отступа с прописной буквы, полужирным шрифтом, без точки в конце, не подчеркивая;
   - если заголовок состоит из двух предложений, их разделяют точкой;
   - переносы слов в заголовках не допускаются;
   - обозначение подразделов следует располагать после абзацного отступа, равного двум знакам относительно обозначения разделов;
   - обозначение пунктов приводят после абзацного отступа, равного четырем знакам относительно обозначения разделов;
   - в содержании должны приводиться наименования структурных элементов, после заголовка каждого из них ставят отточие и приводят номер страницы;
   - содержание должно включать введение, наименование всех разделов и подразделов, заключение, список использованных источников и наименование приложений с указанием номеров страниц, с которых начинаются эти элементы работы;

- перечень сокращений и обозначений следует располагать в алфавитном порядке. Если условных обозначений в отчете менее трех, перечень не составляется.

4. *Оформление нумерации страниц:*
   - страницы следует нумеровать арабскими цифрами, соблюдая сквозную нумерацию по всему тексту работы;
   - номер страницы следует проставлять в центре нижней части листа без точки;
   - титульный лист должен включаться в общую нумерацию страниц;
   - номер страницы на титульном листе не проставляется.
5. *Оформление рисунков:*
   - на все рисунки должны быть ссылки: …в соответствии с рисунком 1;
   - рисунки, за исключением рисунков приложений, следует нумеровать арабскими цифрами;
   - рисунки могут иметь наименование и пояснительные данные, которые помещаются в строке над названием рисунка: Рисунок 1 – Детали прибора
   - рисунки каждого приложения должны обозначаться отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения: Рисунок А.3 (третий рисунок приложения А).
6. *Оформление таблиц:*
   - на все таблицы должны быть ссылки, при ссылке следует писать слово «таблица» с указанием ее номера;
   - таблицы, за исключением таблиц приложений, следует нумеровать арабскими цифрами сквозной нумерацией;
   - наименование таблицы следует помещать над таблицей слева, без абзацного отступа: Таблица 1 – Детали прибора
   - таблицы каждого приложения обозначают отдельной нумерацией арабскими цифрами с добавлением перед цифрой обозначения приложения:
     Таблица Б.2 (вторая таблица приложения Б)
   - если таблица переносится на следующую страницу, под заголовком граф должна быть строка с номером колонок, на следующей странице под названием «Продолжение таблицы 1» дается строка с номером колонок.
7. *Оформление приложений:*
   - в тексте отчета на все приложения должны быть ссылки, приложения располагаются в порядке ссылок на них в тексте отчета;
   - каждое приложение следует размещать с новой страницы с указанием в верхней части страницы слова «ПРИЛОЖЕНИЕ»;
   - заголовок приложения записывают с прописной буквы, полужирным шрифтом, отдельной строкой по центру без точки в конце;

- приложения обозначаются прописными буквами кириллического алфавита, начиная с А, за исключением букв Ё, З, Й О, Ч, Ъ, Ы, Ь;
- допускается обозначение приложений буквами латинского алфавита, за исключением I и O;
- в случае полного использования букв кириллического и латинского алфавита допускается обозначать приложения арабскими цифрами;
- приложение следует располагать после списка использованных источников.

8. *Список использованных источников.* Сведения об источниках следует располагать в порядке появления ссылок на источник и в тексте работы и нумеровать арабскими цифрами с точкой и печатать с абзацного отступа. Список использованных источников следует оформлять в соответствии с ГОСТ 7.0.100 – 2018 «Библиографическая запись. Библиографическое описание». Примеры библиографического описания в соответствии с требованиями ГОСТ 7.0.100 – 2018 представлены на сайте ГУАП в разделе «Нормативная документация» для учебного процесса.

Лабораторная работа №3. Применение и конфигурирование общего и системного программного обеспечения. Разработка специального программного обеспечения

Цель работы

Целью работы является формирование практических навыков разработки специального программного обеспечения с учетом специфики клиент-серверной архитектуры.

Задание на лабораторную работу

Разработка программного обеспечения в соответствии с требованиями, предъявленными в первой лабораторной работе и поставленными задачами во второй лабораторной работе. Установка, конфигурирование и применение общего и системного программного обеспечения. Выполнение третьей лабораторной работы предполагает коллективное взаимодействие в группе из трех человек в ролях: «Специалист по дизайну графических пользовательских интерфейсов», «Разработчик Web и мультимедийных приложений», «Администратор баз данных».

Общие рекомендации по выполнению лабораторной работы.

Раздел №1. Назначение клиент-серверной архитектуры

Архитектуры «клиент-сервер» – один из основных принципов работы сети Интернет. Любой веб-сайт, или приложение в Интернет работает на сервере, а его пользователи являются клиентами. Социальные сети (Фейсбук, ВК и пр.), сайты электронной коммерции (Amazon, Озон и др.), мобильные приложения (Instagram и т.д.), устройства Интернета вещей (умные колонки или смарт-часы) работают на основе клиент-серверной архитектуры.

Хорошим примером работы системы «клиент-сервер» является автомобильный навигатор. Приложение навигации на сервере собирает данные с многих смартфонов пользователей, на которых установлены клиенты приложения. Кроме того, приложение навигации использует ещё и данные с сервера базы данных – геоинформационной системы, который предоставляет данные, например, о текущих ремонтах дорог, о появлении новых дорог и пр. Данные со многих клиентов (местоположение, скорость) обрабатывается сервером навигации и выдаётся на смартфоны пользователей в виде

информации о средней скорости движения по тому или иному участку маршрута.

Практически любая корпоративная сеть или ИТ-система предприятия, как правило, строится по архитектуре «клиент-сервер». В небольших сетях (3-5 компьютеров в компании) функции сервера может выполнять один из рабочих компьютеров. Если число машин в организации более 10, то лучше сделать выделенный сервер (почтовый сервер, приложений, баз данных и пр.), который будет заниматься обслуживанием клиентов – компьютеров и телефонов сотрудников организации.

В домашних сетях архитектура «клиент-сервер» тоже используется довольно часто. Например, в домашнюю сеть могут быть объединены компьютеры членов семьи, один из которых выполняет функции сервера. В домашнюю сеть также могут быть включены такие устройства, как умные колонки, умные домашние устройства (пылесосы-роботы, фотоаппараты, DVD-плееры и пр.), а также «умные» счётчики (вода, электричество) и т.д. Тогда в системе управления сервера, будут видны все параметры, данные и медиа-файлы (музыка, видео, фото), а также «умные устройства».

Раздел №2. Характеристика клиент-серверной архитектуры

Среди основных характеристик клиент-серверной архитектуры можно выделить следующие.

- ***Асимметричность протоколов***. Между клиентами и сервером существуют отношения «один ко многим». Инициатором диалога с сервером обычно является клиент.
- ***Инкапсуляция услуг***. После получения запроса на услугу от клиента, сервер решает, как должна быть выполнена данная услуга. Модификация («апгрейд») сервера может производиться без влияния на работу клиентов, поскольку это не влияет на опубликованный интерфейс взаимодействия между ними. Иными словами, максимум, что может при этом почувствовать пользователь – незначительная задержка отклика сервера в течение небольшого времени апгрейда.
- ***Целостность***. Программы и общие данные для сервера управляются централизованно, что снижает стоимость обслуживания и защищает целостность данных. В то же время, данные клиентов остаются персонифицированными и независимыми.
- ***Местная прозрачность***. Сервер – это программный процесс, который может исполняться на той же машине, что и клиент, либо на другой машине, подключенной по сети. Программное обеспечение «клиент-сервер» обычно скрывает местоположение сервера от клиентов, перенаправляя запрос на услуги через сеть.

- ***Обмен на основе сообщений***. Клиенты и сервер являются нежёстко связанными («loosely-coupled») процессами, которые обмениваются сообщениями: запросами на услуги и ответами на них.
- ***Модульный дизайн, способный к расширению***. Модульный дизайн программной платформы «клиент-сервер» придаёт ей устойчивость к отказам, то есть, отказ в каком-то модуле не вызывает отказа всего приложения. В такой системе, один или больше серверов могут отказать без остановки всей системы в целом, до тех пор, пока услуги отказавшего сервера могут быть предоставлены с резервного сервера. Другое преимущество модульности в том, что приложение «клиент-сервер» может автоматически реагировать на повышение или понижение нагрузки на систему, путём добавления или отключения услуг или серверов.
- ***Независимость от платформы***. Идеальное приложение «клиент-сервер» не зависит от платформ оборудования или операционной системы. Клиенты и серверы могут развёртываться на различных аппаратных платформах и разных операционных системах.
- ***Масштабируемость***. Системы «клиент-сервер» могут масштабироваться как горизонтально (по числу серверов и клиентов), так и вертикально (по производительности и спектру услуг).
- ***Разделение функционала***. Система «клиент-сервер» — это соотношение между процессами, работающими на одной или на разных машинах. Сервер – это процесс предоставления услуг. Клиент – это потребитель услуг.
- ***Общее использование ресурсов***. Один сервер может предоставлять услуги множеству клиентов одновременно, и регулировать их доступ к совместно используемым ресурсам.

Раздел №3. Типы клиент-серверной архитектуры

Архитектуру «клиент-сервер» принято разделять на три класса: одно-, двух- и трёхуровневую. Однако, нельзя сказать, что в вопросе о таком разделении в сообществе ИТ-специалистов существует полный консенсус. Многие называют одноуровневую архитектуру двухуровневой и наоборот, то же можно сказать о соотношении двух- и трёхуровневой архитектур.

ОДНОУРОВНЕВАЯ АРХИТЕКТУРА (1-TIER)

Одноуровневая архитектура «клиент-сервер» (1-Tier) – такая, где все прикладные программы рассредоточены по рабочим станциям, которые обращаются к общему серверу баз данных или к общему файловому серверу. Никаких прикладных программ сервер при этом не исполняет, только предоставляет данные.

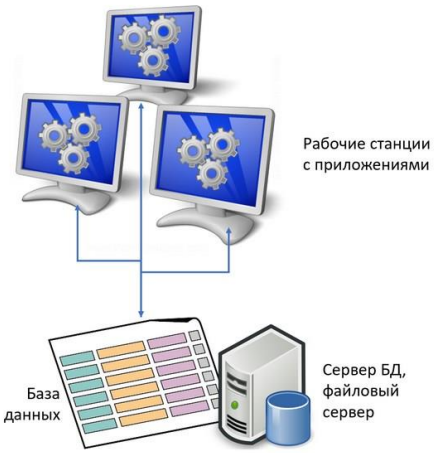

Рис. 1 Одноуровневая архитектура

В целом, такая архитектура очень надёжна, однако, ей сложно управлять, поскольку в каждой рабочей станции данные будут присутствовать в разных вариантах. Поэтому возникает проблема их синхронизации на отдельных машинах. В общем, как можно видеть из рисунка, в этой архитектуре просматривается ещё один уровень – базы данных, что даёт повод во многих случаях называть её двухуровневой.

ДВУХУРОВНЕВАЯ АРХИТЕКТУРА (2-TIER)

К двухуровневой архитектуре «клиент-сервер» следует относить такую, в которой прикладные программы сосредоточены на сервере приложений (Application Server), например, сервере 1С или сервере CRM, а в рабочих станциях находятся программы-клиенты, которые предоставляют для пользователей интерфейс для работы с приложениями на общем сервере.

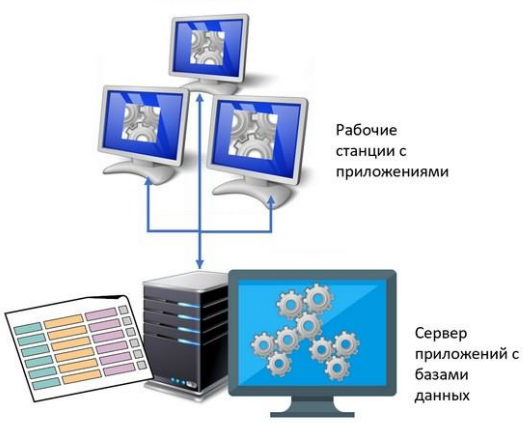

Рис. 2 Двухуровневая архитектура

Такая архитектура представляется наиболее логичной для архитектуры «клиент-сервер». В ней, однако, можно выделить два варианта. Когда общие данные хранятся на сервере, а логика их обработки и бизнес-данные хранятся на клиентской машине, то такая архитектура носит название “fat client thin server” (толстый клиент, тонкий сервер). Когда не только данные, но и логика их обработки и бизнес-данные хранятся на сервере, то это называется “thin

client fat server” (тонкий клиент, толстый сервер). Такая архитектура послужила прообразом облачных вычислений (Cloud Computing).

Преимущества двухуровневой архитектуры:

- Легко конфигурировать и модифицировать приложения;
- Пользователю обычно легко работать в такой среде;
- Хорошая производительность и масштабируемость.

Однако, у двухуровневой архитектуры есть и ограничения:

- Производительность может падать при увеличении числа пользователей;
- Потенциальные проблемы с безопасностью, поскольку все данные и программы находятся на центральном сервере;
- Все клиенты зависимы от базы данных одного производителя.

ТРЁХУРОВНЕВАЯ АРХИТЕКТУРА (3-TIER)

В трёхуровневой архитектуре сервер баз данных, файловый сервер и другие представляют собой отдельный уровень, результаты работы которого использует сервер приложений. Логика данных и бизнес-логика находятся в сервере приложений. Все обращения клиентов к базе данных происходят через промежуточное программное обеспечение (middleware), которое находится на сервере приложений. Вследствие этого, повышается гибкость работы и производительность.

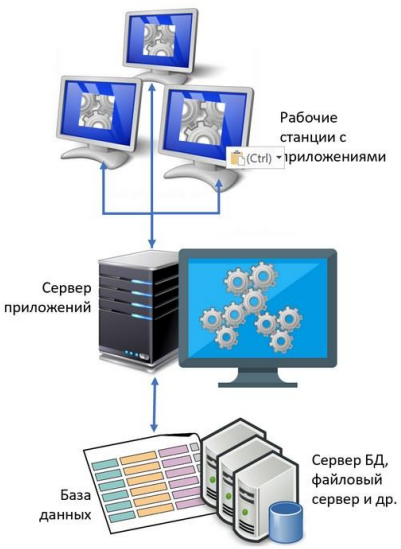

Рис. 3 Трехуровневая архитектура

Преимущества трёхуровневой архитектуры:

- Целостность данных;

– Более высокая безопасность, по сравнению с двухуровневой архитектурой;
– Защищённость базы данных от несанкционированного проникновения.

Ограничения:

– Более сложная структура коммуникаций между клиентов и сервером, поскольку в нём также находится middleware.

МНОГОУРОВНЕВАЯ АРХИТЕКТУРА (N-TIER)

В отдельный класс архитектуры «клиент-сервер» можно вынести многоуровневую архитектуру, в которой несколько серверов приложений используют результаты работы друг друга, а также данные от различных серверов баз данных, файловых серверов и других видов серверов.

По сути, предыдущий вариант, трёхуровневая архитектура – не более, чем частный случай многоуровневой архитектуры.

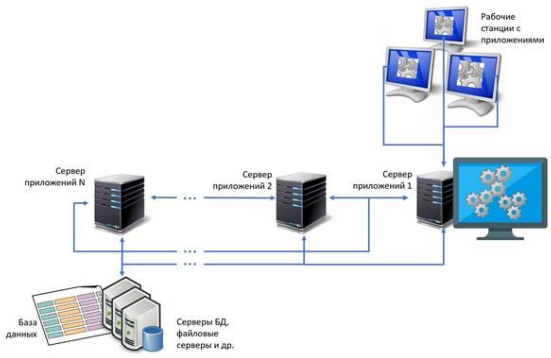

Рис. 4 Многоуровневая архитектура

Преимуществом многоуровневой архитектуры является гибкость предоставления услуг, которые могут являться комбинацией работы различных приложений серверов разных уровней и элементов этих приложений.

Очевидным недостатком является сложность, многокомпонентность такой архитектуры.

Трехзвенная архитектура

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле).

Основная часть приложения, как правило, находится на стороне веб-сервера, который обрабатывает полученные запросы в соответствии с бизнес-логикой продукта и формирует ответ, отправляемый пользователю. На этом

этапе в работу включается браузер, именно он преобразовывает полученный ответ от сервера в графический интерфейс, понятный пользователю.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты (потребители этих функций).

Практические реализации такой архитектуры называются клиент-серверными технологиями.

Двухзвенная архитектура - распределение трех базовых компонентов между двумя узлами (клиентом и сервером). Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме.

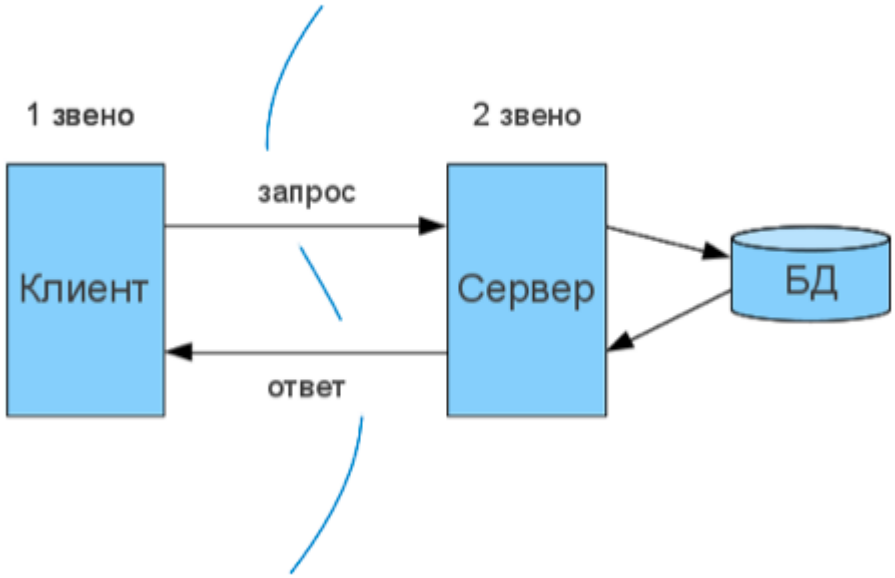

Рис. 5 Двухзвенная архитектура

Расположение компонентов на стороне клиента или сервера определяет следующие основные модели их взаимодействия в рамках двухзвенной архитектуры:

- Сервер терминалов — распределенное представление данных.
- Файл-сервер — доступ к удаленной базе данных и файловым ресурсам.
- Сервер БД — удаленное представление данных.
- Сервер приложений — удаленное приложение.

Клиент – это браузер, но встречаются и исключения (в тех случаях, когда один веб-сервер (ВС1) выполняет запрос к другому (ВС2), роль клиента играет веб-сервер ВС1). В классической ситуации (когда роль клиента выполняет браузер) для того, чтобы пользователь увидел графический интерфейс приложения в окне браузера, последний должен обработать полученный ответ

веб-сервера, в котором будет содержаться информация, реализованная с применением HTML, CSS, JS (самые используемые технологии). Именно эти технологии «дают понять» браузеру, как именно необходимо «отрисовать» все, что он получил в ответе.

Веб-сервер – это сервер, принимающий HTTP-запросы от клиентов и выдающий им HTTP-ответы. Веб-сервером называют как программное обеспечение, выполняющее функции веб-сервера, так и непосредственно компьютер, на котором это программное обеспечение работает. Наиболее распространенными видами ПО веб-серверов являются Apache, IIS и NGINX. На веб-сервере функционирует тестируемое приложение, которое может быть реализовано с применением самых разнообразных языков программирования: PHP, Python, Ruby, Java, Perl и пр.

База данных фактически не является частью веб-сервера, но большинство приложений просто не могут выполнять все возложенные на них функции без нее, так как именно в базе данных хранится вся динамическая информация приложения (учетные, пользовательские данные и пр).

База данных - это информационная модель, позволяющая упорядоченно хранить данные об объекте или группе объектов, обладающих набором свойств, которые можно категоризировать. Базы данных функционируют под управлением так называемых систем управления базами данных (далее – СУБД). Самыми популярными СУБД являются MySQL, MS SQL Server, PostgreSQL, Oracle (все – клиент-серверные).

Трехзвенная архитектура - сетевое приложение разделено на две и более частей, каждая из которых может выполняться на отдельном компьютере. Выделенные части приложения взаимодействуют друг с другом, обмениваясь сообщениями в заранее согласованном формате.

Третьим звеном в трехзвенной архитектуре становится сервер приложений, т.е. компоненты распределяются следующим образом:

- Представление данных — на стороне клиента.
- Прикладной компонент — на выделенном сервере приложений (как вариант, выполняющем функции промежуточного ПО).
- Управление ресурсами — на сервере БД, который и представляет запрашиваемые данные.

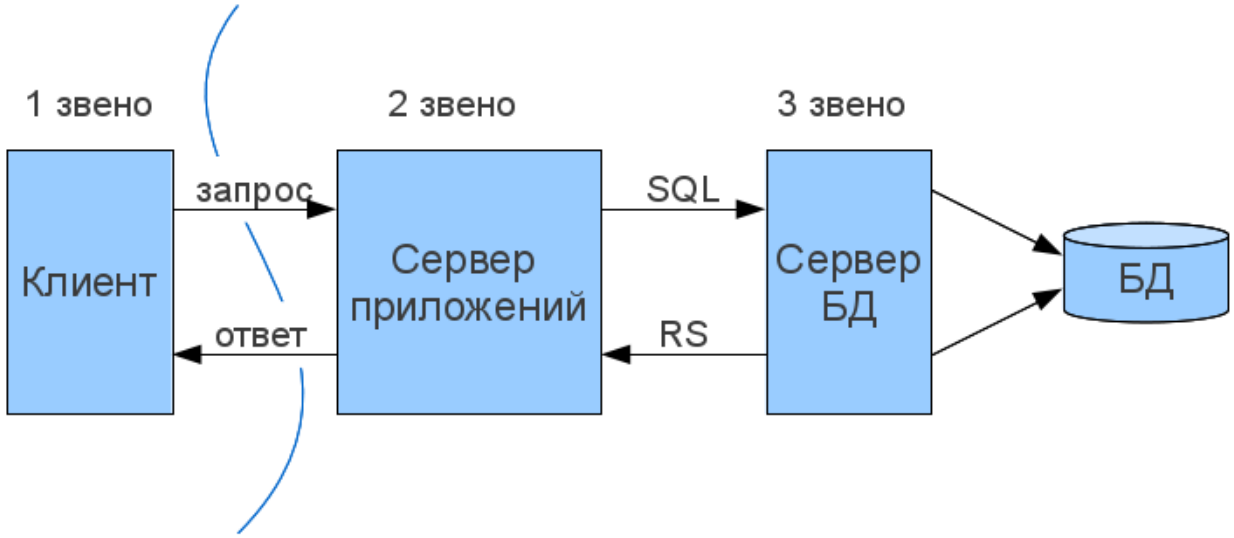

Рис. 6 Трехзвенная архитектура

Трехзвенная архитектура может быть расширена до многозвенной (N-tier, Multi-tier) путем выделения дополнительных серверов, каждый из которых будет представлять собственные сервисы и пользоваться услугами прочих серверов разного уровня.

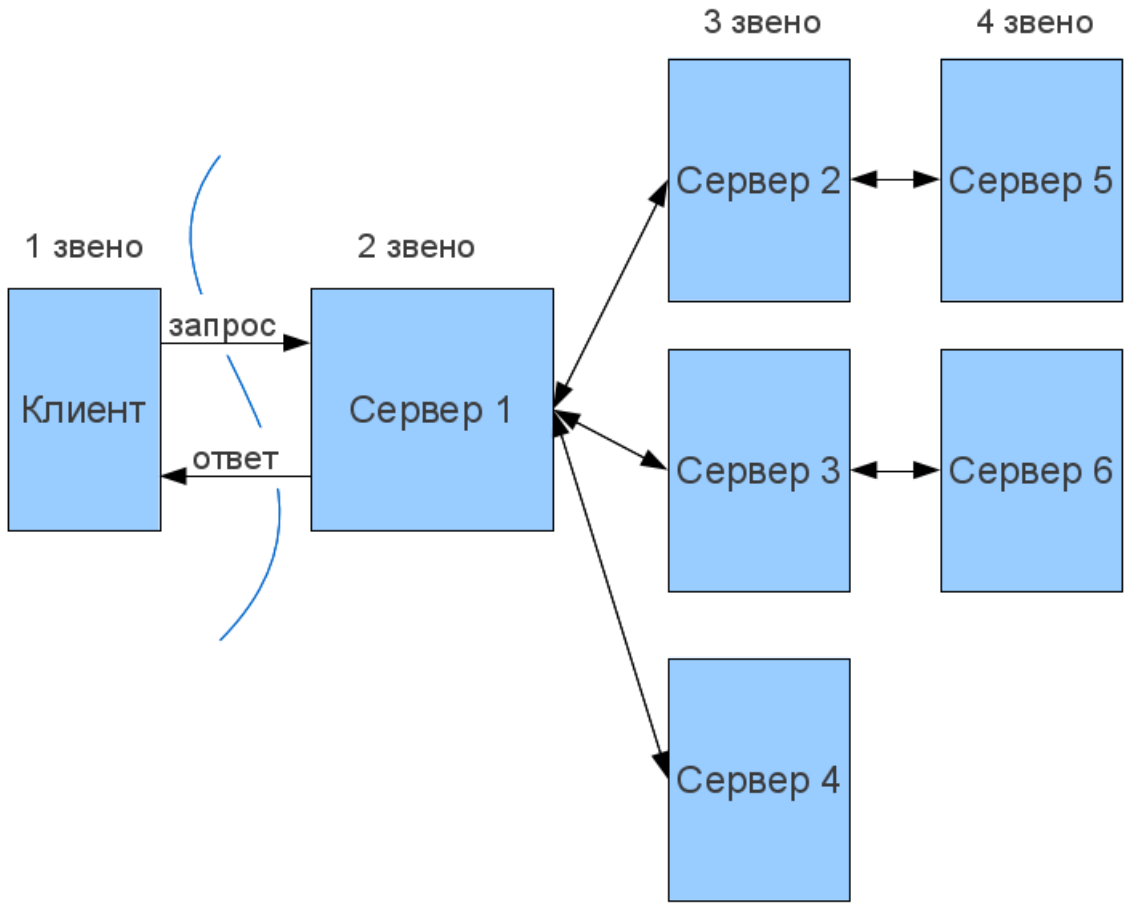

Рис. 7 Многозвенная архитектура

Двухзвенная архитектура проще, так как все запросы обслуживаются одним сервером, но именно из-за этого она менее надежна и предъявляет повышенные требования к производительности сервера.

Трехзвенная архитектура сложнее, но, благодаря тому, что функции распределены между серверами второго и третьего уровня, эта архитектура предоставляет:

- Высокую степень гибкости и масштабируемости.
- Высокую безопасность (т.к. защиту можно определить для каждого сервиса или уровня).
- Высокую производительность (т.к. задачи распределены между серверами).

Раздел №4. Протоколы взаимодействия и их виды

TCP/IP – совокупность протоколов передачи информации. TCP/IP – это особое обозначение всей сети, которая функционирует на основе протоколов TCP, а также IP.

TCP – вид протокола, который является связующим звеном для установки качественного соединения между 2 устройствами, передачи данных и верификации их получения.

IP – протокол, в функции которого входит корректность доставки сообщений по выбранному адресу. При этом информация делится на пакеты, которые могут поставляться по-разному.

MAC – вид протокола, на основании которого происходит процесс верификации сетевых устройств. Все устройства, которые подключены к сети Интернет, содержат свой оригинальный MAC-адрес.

ICMP – протокол, который ответственен за обмен данными, но не используется для процесса передачи информации.

UDP – протокол, управляющий передачей данных, но данные не проходят верификацию при получении. Этот протокол функционирует быстрее, чем протокол TCP.

HTTP – протокол для передачи информации (гипертекста), на базе которого функционируют все сегодняшние сайты. В его возможности входит процесс запрашивания необходимых данных у виртуально удаленной системы (файлы, веб-страницы и прочее).

FTP – протокол передачи информации из особого файлового сервера на ПК конечного пользователя.

POP3 – классический протокол простого почтового соединения, который ответственен за передачу почты.

SMTP – вид протокола, который может устанавливать правила для передачи виртуальной почты. Он ответственен за передачу и верификацию доставки, а также оповещения о возможных ошибках.

Раздел №5. Клиент-серверные технологии

Архитектура клиент-сервер применяется в большом числе сетевых технологий, используемых для доступа к различным сетевым сервисам.

Типы сервисов:

- Web-серверы

Изначально предоставляли доступ к гипертекстовым документам по протоколу HTTP (Hyper Text Transfer Protocol). Сейчас поддерживают расширенные возможности, в частности, работу с бинарными файлами (изображения, мультимедиа и т.п.).

- Серверы приложений

Предназначены для централизованного решения прикладных задач в некоторой предметной области. Для этого пользователи имеют право запускать серверные программы на исполнение. Использование серверов приложений позволяет снизить требования к конфигурации клиентов и упрощает общее управление сетью.

- Серверы баз данных

Серверы баз данных используются для обработки пользовательских запросов на языке SQL. При этом, СУБД находится на сервере, к которому и подключаются клиентские приложения.

- Файл-серверы

Файл-сервер хранит информацию в виде файлов и предоставляет пользователям доступ к ней. Как правило, файл-сервер обеспечивает и определенный уровень защиты от несанкционированного доступа

- Прокси-сервер

Во-первых, действует как посредник, помогая пользователям получить информацию из Интернета и, при этом, обеспечивая защиту сети.

Во-вторых, сохраняет часто запрашиваемую информацию в кэш-памяти на локальном диске, быстро доставляя ее пользователям, без повторного обращения к Интернету.

- Файрволы (брандмауэры)

Межсетевые экраны, анализирующие и фильтрующие проходящий сетевой трафик, с целью обеспечения безопасности сети.

- Почтовые серверы

Предоставляют услуги по отправке и получению электронных почтовых сообщений.

- Серверы удаленного доступа (RAS)

Эти системы обеспечивают связь с сетью по коммутируемым линиям. Удаленный сотрудник может использовать ресурсы корпоративной ЛВС, подключившись к ней с помощью обычного модема.

Для доступа к тем или иным сетевым сервисам используются клиенты, возможности которых характеризуются понятием «толщины». Оно определяет конфигурацию оборудования и программное обеспечение, имеющиеся у клиента. Рассмотрим возможные граничные значения:

«Тонкий» клиент»

Этот термин определяет клиента, вычислительных ресурсов которого достаточно лишь для запуска необходимого сетевого приложения через web-интерфейс. Пользовательский интерфейс такого приложения формируется средствами статического HTML (выполнение JavaScript не предусматривается), вся прикладная логика выполняется на сервере. Для работы тонкого клиента достаточно лишь обеспечить возможность запуска web-браузера, в окне которого и осуществляются все действия. По этой причине web-браузер часто называют "универсальным клиентом".

«Толстый» клиент»

Таковым является рабочая станция или персональный компьютер, работающие под управлением собственной дисковой операционной системы и имеющие необходимый набор программного обеспечения. К сетевым серверам «толстые» клиенты обращаются, в основном, за дополнительными услугами (например, доступ к web-серверу или корпоративной базе данных).

Так же под «толстым» клиентом подразумевается и клиентское сетевое приложение, запущенное под управлением локальной ОС. Такое приложение совмещает компонент представления данных (графический пользовательский интерфейс ОС) и прикладной компонент (вычислительные мощности клиентского компьютера).

В последнее время все чаще используется еще один термин:«rich»-client. «Rich» -клиент, своего рода, компромисс между «толстым» и «тонким» клиентом. Как и «тонкий» клиент, «rich»-клиент также представляет графический интерфейс, описываемый уже средствами XML и включающий некоторую функциональность толстых клиентов (например, интерфейс drag-and-drop, вкладки, множественные окна, выпадающие меню и т.п.)

Прикладная логика «rich»-клиента также реализована на сервере. Данные отправляются в стандартном формате обмена, на основе того же XML (протоколы SOAP, XML-RPC) и интерпретируются клиентом.

Некоторые основные протоколы «rich»-клиентов на базе XML приведены ниже:

- XAML (eXtensible Application Markup Language) — разработан Microsoft и используется в приложениях на платформе .NET.
- XUL (XML User Interface Language) — стандарт, разработанный в рамках проекта Mozilla, используется, например, в почтовом клиенте Mozilla Thunderbird или браузере Mozilla Firefox.
- Flex — мультимедийная технология на основе XML, разработанная Macromedia/Adobe.

Протокол передачи данных — набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами. Эти соглашения задают единообразный способ передачи сообщений и обработки ошибок при взаимодействии ПО.

Сетевой протокол – набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.

Раздел №6. Практические аспекты реализации. Конфигурирование сети

Настройка IP-адреса, шлюза по умолчанию, маски подсети

Отредактируйте файл конфигурации /etc/network/interfaces, например так:

$ sudo nano /etc/network/interfaces

И допишите в него:

Для статического IP:

iface eth0 inet static

address 192.168.0.1

netmask 255.255.255.0

gateway 192.168.0.254

dns-nameservers 192.168.0.254 8.8.8.8

auto eth0

Где:

- iface eth0 inet static - указывает, что интерфейс (iface eth0) находится в диапазоне адресов IPv4 (inet) со статическим ip (static);
- address 192.168.0.1 - указывает что IP адрес (address) нашей сетевой карты 192.168.0.1;
- netmask 255.255.255.0 - указывает что наша маска подсети (netmask) имеет значение 255.255.255.0;
- gateway 192.168.0.254 - адрес шлюза (gateway) по умолчанию 192.168.0.254 (не является обязательным);
- dns-nameservers 192.168.0.254 8.8.8.8 - адреса DNS серверов (о ниж мы расскажем позже)
- auto eth0 - указывет системе что интерфейс eth0 необходимо включать автоматически при загрузке системы с вышеуказанными параметрами.

eth0 – имя подключаемого своего интерфейса. Список интерфейсов можно посмотреть набрав:

$ ip addr

Раздел №7. Практические аспекты реализации. Конфигурирование web-сервера

Apache2 – web-сервер общего назначения гипертекстовой обработки данных.

Директивы

Как подчеркивалось выше, значения конфигурационных переменных хранятся в конфигурационных файлах. Эти переменные и есть директивы. Основное время администрирования «сервера Apache (90 %) затрачивается на определение того, значения каких директив будут изменяться, и значения, которые эти директивы должны задавать. Следует обратить внимание на то, что далеко не все директивы автоматически распознаются сервером Apache. Существует достаточно большое подмножество директив, которые называются основными. Эти директивы устанавливаются по умолчанию. Другие директивы распознаются в зависимости от того, какой набор модулей скомпилирован при построении данного конкретного сервера Apache. Поэтому указание директивы в конфигурационном файле совсем не означает, что она возымеет какое-либо действие: работающий вариант сервера вполне может не иметь модуля, который взаимодействует с данной директивой.

Модули

Сервер Apache имеет ядро, гарантирующее выполнение основных функций. Ядро обеспечивает работу директив, возможность чтения конфигурационных файлов, усеченную возможность управления доступом, возможность дополнения функциональных возможностей и полдесятка других основных функциональных возможностей. В частности, директивы, которые перечислены в приложении А, "Основные директивы", всегда присутствуют в стандартном дистрибутиве сервера Apache. Кроме того, сервер Apache разработан таким образом, что всегда существует возможность варьирования основных функциональных возможностей. Функциональные части могут быть и не подключены к первоначальной исполняемой программе. Эти субсекции называются модулями, причем достаточно значительная их часть поставляется по умолчанию в стандартном дистрибутиве.

Дескрипторы

Иногда модули представляют в распоряжение специальные дескрипторы, которые являются методами обработки файлов или запросов каким-то специальным способом. Иногда дескрипторы именуются таким образом, что к ним можно обращаться непосредственно с помощью конфигурационных директив.

Дескриптор	*Модуль*	*Действие*
send-as-is	mod_asis	Обслуживать файлы и заголовки в их текущем состоянии
cgi-script	mod_cgi	Выполнение CGI-сценариев
imap-file	mod imap	Файл правил обработки изображений
server-info	mod info	Отображение конфигурационной информации сервера
server-parsed	mod include	Найти и заместить все вставленные на стороне сервера модули
server-status	mod_status	Отображение информации о статусе сервера
type-map	mod negotiation	Анализировать как файл карты типа

Как показано в столбце "Модуль", для получения доступа к определенному дескриптору необходимо, чтобы в текущем httpd был включен соответствующий модуль. Дескрипторы активизируются с помощью директивы AddHandler.

Другим способом активизации определенного дескриптора может служить директива SetHandler. Директива SetHandler предназначена для использования внутри скобок < D i r e c t o r y > и л и < L o c a t i o n > . Например, для того, чтобы все файлы, содержащиеся в каталоге / i m a g e s обрабатывались как файлы правил обработки изображений, можно воспользоваться следующей директивой:

```
< L o c a t i o n / i m a g e s >
        SetHandler imap-file
< / L o c a t i o n >
```

```
  <Directory /var/www/testcgi>
     AllowOverride None
     Options FollowSymLinks
     Options +ExecCGI
     AddHandler cgi-script .cgi .pl .py
  </Directory>

  <Directory /var/www/test_front_end>
     AllowOverride None
     Options FollowSymLinks
     Options +ExecCGI
     AddHandler cgi-script .cgi .pl .py
  </Directory>
```

Основы установки конфигурирования

Установить веб-сервер Apache можно выполнив команду:

```
sudo apt-get update && apt-get install apache2
```

Веб-сервер Apache может запускать несколько веб-сайтов на одном компьютере. Каждый запущенный сайт («виртуальный хост» в терминологии Apache) должен иметь свою собственную конфигурацию. Виртуальный хост может быть IP или именем.

В этом руководстве мы сосредоточимся на втором типе, так как он проще в настройке и не требует нескольких IP-адресов (виртуальные хосты на основе имён позволяют нескольким веб-сайтам использовать один и тот же IP-адрес).

Виртуальный хост по умолчанию

Виртуальный хост по умолчанию определён в каталоге */etc/apache2/sites-available* внутри файла 000-default.conf. Рассмотрим его:

```
<VirtualHost *:443>
        [...]
        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        [...]

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        [...]
</VirtualHost>
```

Директива `<VirtualHost>` на первой строке применяется для группы параметров, используемых Apache для конкретного виртуального хоста. Первое, что вы увидите в ней, – инструкцию `*:80`. Она указывает IP-адрес и порт, используемый виртуальным хостом.

Несколько сайтов могут быть определены в одном и том же файле или по отдельности. В обоих случаях первое определение считается значением по умолчанию, если ни один другой виртуальный хост не соответствует запросу клиента.

Директива на строке 3 не является обязательной, она используется, чтобы указать контактный адрес. Обычно в качестве аргумента директивы предоставляют действительный адрес электронной почты, чтобы было проще связаться с администратором.

`DocumentRoot` в строке 4 является обязательным, это важно для конфигурации виртуального хоста. Аргумент этой инструкции должен иметь доступ к файловой системе. Указанный каталог будет считаться корневым каталогом виртуального хоста и не должен содержать завершающий символ «/». В этом случае корневая директория документа — `/var/www/html`. Если мы посмотрим на её содержимое, то увидим, что она содержит страницу `index.html`, которую вы до этого видели в качестве страницы приветствия сервера.

Последние две команды на строках 8–9, представленные в этом `VirtualHost`, – `ErrorLog` и `CustomLog`. Используя первый, вы указываете файл, в

который сервер будет записывать возникающие ошибки. Второй используется для регистрации запросов, отправленных на сервер в указанном формате.

Новый виртуальный хост

Вы видели, как определяется виртуальный хост по умолчанию. Теперь предположим, что вы хотите разместить другой веб-сайт с помощью вашего веб-сервера. Для этого вам нужно определить новый виртуальный хост.

Как сказано выше, файлы виртуальных хостов должны быть определены внутри каталога `/etc/apache2/sites-available` (по крайней мере в дистрибутивах на основе Debian). Поэтому создадим этот файл там. Прежде чем сделать это, следует создать каталог, который будет использоваться как document root, а также создать базовую страницу, которая будет отображаться при открытии сайта:

```
$ sudo mkdir /var/www/example && echo "Welcome to example!" > /var/www/example/index.html
```

Теперь можно приступить к настройке виртуального хоста:

```
<VirtualHost *:80>
DocumentRoot /var/www/example
ServerName www.example.local
</VirtualHost>
```

Это минимальная конфигурация, необходимая для его запуска. Здесь вы можете увидеть новую директиву `ServerName`. Это то, что определяет ваш виртуальный хост. Сохраним этот файл как `example.conf`. Чтобы активировать ваш виртуальный хост, используйте команду `a2ensite`. Эта команда создаёт символическую ссылку файла в каталоге `/etc/apache2/sites-enabled`:

```
sudo a2ensite example.conf
```

После этого следует перезагрузить конфигурацию сервера:

```
sudo systemctl reload apache2.service
```

Раздел №8. Практические аспекты реализации. Общий шлюзовой CGI-интерфейс

CGI-интерфейс (Common Gateway Interface – общий шлюзовой интерфейс) – одно из первых решений, созданных для доставки динамической web-информации.

Последовательность работы CGI состоит из следующих этапов.

- Получение Web-сервером информации от клиента-браузера. Для передачи данных Web-серверу используются формы. Когда пользователь заполнит всю форму, он нажимает эту кнопку, и данные из полей формы передаются программе CGI.
- Анализ и обработка полученной информации. Данные, извлеченные из HTML-формы, передаются для обработки CGI-программе. Они не всегда могут быть обработаны CGI-программой самостоятельно. Например, они могут содержать запрос к базе данных. В этом случае CGI-программа на основании полученной информации формирует запрос к ядру СУБД, выполняющейся на том же или удаленном компьютере.
- Создание нового HTML-документа и пересылка его браузеру. После обработки полученной информации CGI-программа создает динамический (виртуальный) HTML-документ, или формирует ссылку на уже существующий документ и передает результат через сервер браузеру клиента

CGI – Common Gateway Interface является стандартом интерфейса, который служит для связи внешней программы с веб-сервером. Программу, которая работает по такому интерфейсу совместно с веб-сервером, принято называть шлюзом, многие больше любят названия скрипт или CGI-программа.

Сам протокол разработан таким образом, чтобы можно было использовать любой язык программирования, который может работать со стандартными устройствами ввода/вывода. А это умеет даже сама операционная система, поэтому часто если вам не требуется сложный скрипт, его можно просто сделать в виде командного файла.

Все скрипты, как правило, помещают в директорию cgi-bin сервера, но это совсем даже не обязательно, в принципе скрипт может располагаться где угодно только при этом большинство Web-серверов требуют специальной настройки.

Чаще всего, хотя наверно почти всегда, скрипты используются для создания динамических страниц. Связано это с тем, что само содержимое веб-сервера является статическим и не будет меняться просто так, для этого должен приложить руку веб-мастер. Технология CGI позволяет просто поменять содержимое веб-сервера. Простым примером может служить скрипт, который при каждом новом обновлении страницы вставляет в нее новую ссылку(банер) или анекдот. Более сложными скриптами являются гостевые книги, чаты, форумы и естественно поисковые сервера или базы данных построенные на технологиях интернета.

Передача данных шлюзу осуществляется в следующем формате:

имя=значение&имя1=значение1&...

Здесь "имя" это название параметра, а "значение" его содержимое. Методов передачи данных в таком формате существует два - GET и POST. При использовании метода GET данные передаются серверу вместе с URL:

http://.../cgi-bin/test.cgi?имя=значение&имя1=значение1&...

При использовании метода POST данные посылаются внутри самого HTTP запроса.

Так как длина URL ограничена, то методом GET нельзя передать большой объем данных, а метод POST обеспечивает передачу данных не ограниченных по длине.

Получение данных самим скриптом также различается. При использовании метода GET данные следующие за "?" помещаются в переменную среды QUERY_STRING. При использовании POST содержимое запроса перенаправляется в стандартный поток ввода, т.е. в stdin.

Чтобы шлюз мог узнать какой метод используется для передачи данных, сервер создает переменную среды REQUEST_METHOD, в которую записывает GET или POST.

В имена и значения параметров при передаче кодируются браузером URL методом, т.е. все символы не принадлежащие к латинскому алфавиту и числам кодируются в виде %HH, где HH - шестнадцатеричное значение кода символа. Также кодируются все символы, которые нельзя использовать, т.е. !#%^&()=+ и пробел. Символ "&" используется, как мы уже видели для разделения пар "имя=значение", "=" используется в парах "имя=значение", "%" для кодирования символов, "пробел" кодируется символом "+"(плюс), сам же плюс кодируется через "%", и т.д. Поэтому при анализе полученных данных требуется их переводить в нормальных вид.

Пример кодировки символов и букв при передаче:

Передается строка: !@#$%^&()-=_+ абвгд*

Скрипт получает : %21@%23%24%25%5E%26%28%29-%3D_%2B+%E0%E1%E2%E3%E4*

Размер передаваемых данных методом POST содержится в переменной окружения CONTENT_LENGTH:

Передается : a=4&b=1

```
CONTENT_LENGTH = 7
```

Так и любая программа, программа CGI шлюза должна получить данные, обработать их и вывести результат работы в наглядной форме.

Получение данных. Пример CGI-программы

Весь процесс получения данных от веб-сервера можно представить следуюшим образом:

1. Получить все необходимые переменные окружения. Наиболее важной на данном этапе является REQUEST_METHOD.
2. Получить данные от сервера в зависимости от метода передачи:

```
Если (REQUEST_METHOD="GET") то
  Взять данные из переменной окружения QUERY_STRING
Иначе
  Если (REQUEST_METHOD="POST") то
    Проанализировать переменную QUERY_STRING
    Получить длинну данных из CONTENT_LENGTH
    Если (CONTENT_LENGTH>0) то
      Считать CONTENT_LENGTH байт из sdtin как данные.
  Иначе
    Выдать сообщение об ошибке и выйти.
```

3. Декодировать все полученные данные и, если надо, разбить их на пары "имя=значение" в удобную для программы форму.

С анализом переменной REQUEST_METHOD думаю все ясно, а вот, что значит в методе POST проанализировать QUERY_STRING, наверно, не все ясно. При передаче методом POST сервер посылает данные через стандартный поток ввода, но это не значит, что пользователь не пользовался URL'ом для передачи данных. Примером может служить многопользовательский шлюз, в котором для идентификации пользователя используется URL, а для передачи данных stdin:

```
http://.../cgi-bin/guestbook.cgi?user=bob&rec=0
```

В этом случае шлюзу гостевой книги (guestbook.cgi) сообщается два параметра user и rec, с помощью которых она может узнать "куда записывать" или "как обрабатывать" данные поступающие через поток ввода.

Считывание данных через поток sdtin должно осуществляться в динамическую память, или же во временный файл, если размер памяти ограничен или данные слишком велики для полного размещения в ОЗУ. Это связано с тем, что при использовании статических буферов может произойти его переполнение.

Пример:

```
char cgi_data[1000];
...
long content_length=atol(getenv("CONTENT_LENGTH"));
fread(cgi_data,content_length,1,stdin);
...
```

Надеюсь все сразу видно :-)). Если content_length>1000, то произойдет переполнение cgi_data. Переполнение буферов это излюбленный метод атаки хакеров. Вместо этого лучше выделять память динамически:

```
char *cgi_data;
...
long content_length=atol(getenv("CONTENT_LENGTH"));
cgi_data=(char *)malloc(content_length);
if (cgi_data!=NULL)
   fread(cgi_data,content_length,1,stdin);
...
```

После получения данных от сервера их надо еще декодировать. Можно это сделать сразу, а можно по мере надобности. Если Вы будете это делать сразу, то Вам также придется их разбить на куски, так как при декодировании могут появиться лишние знаки "&" и "=", которые больше не позволят вам отделять пары "имя=значение" друг от друга.

Вот пример процедуры, которая декодирует данные из буфера:

```
/* Возвращает верхний регистр символа*/
char upperchar(char ch)
{
 if ((ch>='a') && (ch<='z'))
 {
    ch='A'+(ch - 'a');
```

```
    return ch;
  }
 else return ch;
};

/* Переводит из Hex в Dec*/
char gethex(char ch)
{
 ch=upperchar(ch);
 if ((ch>='0')&&(ch<= '9')) return (ch-'0');
 if ((ch>='A')&&( ch<='F')) return (ch-'A'+10);
};

/*
 Ищет и возвращает параметр с именем name, в buffer.
 Если параметр name не найден, возвращает NULL.

Пример : message = getparam(post_buffer,"message=");

Замечание : символ "=" после имени параметра не удаляется
 и входит в возвращаемый результат, поэтому рекомендуется
 искать параметр вместе с символом "=".
*/

char *getparam(char *buffer,char *name)
{
 if (buffer==NULL) return NULL;

 char *pos;
 long leng=512,i=0,j=0;
 char h1,h2,Hex;

 char *p=(char *)malloc(leng);
 pos=strstr(buffer,name);
 if (pos == NULL) return NULL;

 if ((pos!=buffer) && (*(pos-1)!='&')) return NULL;
```

```
 pos+=strlen(name);


 while ( (*(pos+i)!='&')&&( *(pos+i)!='\0' ))
 {
  if ( *(pos+i)=='%' )
  {
    i++;
    h1=gethex(*(pos+i));
    i++;
    h2=gethex(*(pos+i));
    h1=h1<<4;
    *(p+j)=h1+h2;
  }
  else
  {
    if (*(pos+i)!='+') *(p+j)=*(pos+i);
     else *(p+j)=' ';
  };
  i++;
  j++;
  if (j >= leng) p=(char*)realloc(p,leng+20);
  leng+=20;
 };
 if (j < leng) p=(char*)realloc(p,j+1);


 *(p+j)='\0';
 return p;
};
```

Теперь используя функцию getparam Вы сможете в любое время получить какой-нибудь параметр. Конечно эта процедура еще далека от совершенства, так как использует стандартный realloc и не обрабатывает ошибку нехватки памяти, но все же ее можно использовать для данных не особо больших размеров.

Кстати надо еще сказать, что нецелесообразно размещать поступающие данные размером > 64 Кб. в памяти, если, например, скрипт предназначен для загрузки файлов, то можно (а вернее нужно) напрямую организовать запись в файл, иначе файл размером в пару мегабайт ваш скрипт не обработает.

Вывод информации

Для того, чтобы начать вывод данных шлюз должен сообщить браузеру тип выводимых данных и как с ними работать. Это действие производит заголовок.

Существуют два типа заголовков, вернее даже три, но один из них используется реже (о нем мы поговорим отдельно). Так вот, первый заголовок осуществляет переадресацию браузера на другой ресурс в сети и записывается вот как:

Location: URL-адрес ресурса

Под URL-адресом может быть что угодно, от заранее заготовленной страницы ответа, до страницы или файла на чужом сервере. Очень полезная штука для любителей преадресовки :-)

Пример:

Location: http://www.mjk.msk.ru/~dron/88.gif

Второй тип заголовка описывает данные которые будет выводить непосредственно скрипт. Задает их тип, длину и другие вспомогательные параметры. Самый простой заголовок записывается в виде:

Content-type: тип данных

Тип данных создается из двух составляющих: его типа и формата. То есть, сначала указывается один из следующих типов:

- **application** - данные для какого-либо приложения
- **audio** - аудио данные, например **RealAudio**
- **video** - видео данные, например **MPEG** или **AVI**
- **image** - изображение
- **text** - текст

Далее через /(слэш) указывается формат данных.

Примеры заголовков:

Content-type: text/html
Content-type: image/gif
Content-type: video/mpeg

Еще заголовок может включать в себя вспомогательные параметры, **Content-length** - длина содержимого, **Expires** - время существования в кэше и

т.д. Каждый параметр указывается в отдельной строке и заканчивается переносом строки **\n.**

Пример:

```
Content-type: image/gif
Content-length: 1052
```

После того как заголовок сформирован и отправлен надо отправить пустую строку, для того, чтобы отделить его от основных данных.

```
 ...
printf("Content-type: text/html\n");
printf("\n");
printf("<html><h1>Simple CGI for example !!!</h1></html>");
...
```

Если отделяющая пустая строка не будет отправлена, то сервер выдаст ошибку **500 Internal Server Error : Bad header.**

После отправки заголовка скрипт может приступать к выводу данных, но естественно только в указанном им формате.

Переменные среды о сервере

Эти переменные сервер устанавливает для того, чтобы шлюз мог узнать с каким сервером он работает. Сюда входят данные о портах сервера, его версии, типе интерфеса CGI и т.д. В каждой версии сервера часто прибавляются новые переменные, но следующие переменные должен устанавливать любой сервер любой версии.

GATEWAY_INTERFACE

Указывает версию интерфейса CGI, который поддерживает сервер. Например:

```
CGI/1.1
```

SERVER_NAME

Содержит IP адрес сервера или его доменное имя. Например:

```
www.mjk.msk.ru
```

SERVER_PORT

Номер порта, по которому сервер получает http запросы. Стандартный порт для этого 80.

SERVER_PROTOCOL

Версия протокола Http, который использует сервер для обработки запросов. Например:

```
HTTP/1.1
```

SERVER_SOFTWARE

Название и версия программы сервера. Например:

Apache/1.3.3 (Unix) (Red Hat/Linux)

Эти переменные обеспечивают все необходимые данные о сервере, на котором запускается скрипт. Если Ваш сервер сконфигурирован для работы с одним хостом, то скорее информация эта вам не понадобится. Сейчас же большинство серверов позволяют создавать так называемые "виртуальные" хосты. Т.е. это один компьютер, который поддерживает много **IP** адресов и различает запросы от клиентов по требуемому хосту, на которые он соответственно выдает странички с сайтов. Тут уже могут понадобиться данные о портах сервера (т.к. многие хосты просто "сидят" на других портах, например 8080, 8081 и т.д.) и его **IP** адрес с именем.

Переменные среды о запросе

При помощи переменных данного типа шлюз узнает полную информацию о запросе к нему. Т.е. каким методом будут передаваться данные, их тип, длину и т.д.

AUTH_TYPE

Тип авторизации используемой сервером. Например:

Basic

CONTENT_FILE

Путь к файлу с полученными данными. Используется только в серверах под Windows. Например:

```
c:\website\cgi-temp\103421.dat
```

CONTENT_LENGTH

Длинна переданной информации в байтах. То бишь сколько надо считать байтов из **stdin**. Например:

```
10353
```

CONTENT_TYPE

Тип содержимого посланного серверу клиентом. Например:

```
text/html
```

OUTPUT_FILE

Файл для вывода данных, используется только серверами под Windows. Аналогично CONTENT_FILE.

PATH_INFO и **PATH_TRANSLATED**

В современных веб-серверах появилась возможность после имени скрипта указывать еще какой-то определенный путь. Эти переменные работают следующим образом. Предположим существует скрипт с именем **1.cgi** в каталоге сервера **/cgi-bin**, тогда при вызове скрипта в таком виде:

http://.../cgi-bin/1.cgi/dir1/dir2
данные переменные установятся следующим образом:

PATH_INFO=/dir1/dir2
PATH_TRANSLATED=/home/httpd/html/dir1/dir2

Переменные будут указывать на папку относительно корневой директории сервера. При этом **PATH_TRANSLATED** будет содержать абсолютный путь до этого каталога на диске сервера. В данном случае корневым каталогом сервера считается **/home/httpd/html/**, и еще замечу, что это путь в **Unix** системах.

Под **dos/win** системами переменная **PATH_INFO** не изменится, а **PATH_TRANSLATED** будет содержать **d:\apache\htdocs\dir1\dir2** (в данном случае корнем сервера является директория **d:\apache\htdocs**).

QUERY_STRING

Содержит данные переданные через **URL**. Такие данные указываются после имени шлюза и знака **?**. Пример:

```
http://.../cgi-bin/1.cgi?d=123&name=kostia
```

тогда переменная **QUERY_STRING** будет содержать

```
d=123&name=kostia
```

и еще незабывайте, что данные передаваемые таким образом кодируются методом **URL**.

REMOTE_ADDR

Содержит **IP** адрес пользователя пославшего запрос шлюзу. Если Вы обращаетесь к любому шлюзу в интернете, то данная переменная будет содержать ваш **IP** адрес. Пример:

```
192.168.1.36
```

REMOTE_HOST

Содержит ваше доменное имя, при условии, что вы прописаны на каком-либо **DNS** сервере. Например, если ваш Dial-UP провайдер регистрирует все свои динамические **IP** адреса на **DNS** сервере, то при обращении к шлюзу, эта переменная может содержать примерно следующее:

```
d6032.dialup.cornell.edu
или
dial57127.mtu-net.ru
или
ppp-130-66.dialup.metrocom.ru
```

(брал прямо из логов сервера :-)

REQUEST_METHOD

Мы раньше говорили об этой переменной. Она содержит метод передачи данных шлюзу: **GET** или **POST**.

REQUEST_LINE

Содержит строку из запроса протокола HTTP. Например:

```
GET /cgi-bin/1.cgi HTTP/1.0
```

SCRIPT_NAME

Содержит имя вызванного скрипта. Например: **1.cgi.**

Все эти переменные, обеспечат все самые необходимые данные о запросе к шлюзу.

Переменные среды о клиенте

Далее рассмотрим еще несколько переменных, которые несут в себе информацию о клиенте пославшем запрос.

HTTP_ACCEPT

Эта переменная пречисляет все типы данных, которые может получать и обрабатывать клиент. Часто содержит просто ***/***, т.е. клиент может получать все подряд. Пример:

```
*/*,image/gif,image/x-xbitmap
```

HTTP_REFERER

Содержит **URL** страницы, с которой был произведен запрос, т.е. которая содержит ссылку на шлюз. Пример:

```
http://www.firststeps.ru/index.html
```

HTTP_USER_AGENT

Содержит в себе название и версию браузера, которым пользуется клиент для запроса. Полезно для того, чтобы игнорировать браузеры ненавистных вам фирм :-))). Пример:

```
MyBrowser/1.0 (MSIE 4.0 Compatible, Win98)
```

В большинстве случаев эти переменные не несут в себе полезной для скрипта информации, если только Большинство счетчиков пользуется этой переменной для того, чтобы не учитывать хиты с других хостов, на которые этот счетчик не зарегистрирован.

Еще существует несколько переменных, которые тоже относятся к переменным о клиенте:

HTTP_ACCEPT_ENCODING

Указывает набор кодировок, которые может получать клиент. Например:

```
koi8-r, gzip, deflate
```

HTTP_ACCEPT_LANGUAGE

Содержит в себе список языков в кодах **ISO**, которые может принимать клиент. Например:

```
ru, en, fr
```

HTTP_IF_MODIFIED_SINCE

Содержит в себе дату, новее которой должны быть получаемые данные.

HTTP_FROM

Содержит список почтовых адресов клиента.

Обработка простой формы

```
<form action="http://localhost/cgi-bin/primer.cgi" method=GET>
Введите свое имя пользователя:
<input type=text maxlength=150 name=user>
<p><input type=submit value=Send>
</form>
```

Формы небольшого размера лучше всего посредством метода **GET.** Во-первых получить данные из переменной окружения намного легче, чем считать их из потока. Чтобы считать данные из потока надо точно знать их размер, позаботиться о выделении памяти и многом другом. Тут же обо всем позаботится сервер и встроенные средства программирования вашего языка. Во-вторых пользователь сможет обратиться к вашему скрипту непосредственно из адресной строки браузера. Например, многие программы для поиска информации в интернете используют различные поисковые сервера. Для того чтобы сделать запрос к одному из них требуется всего лишь вызвать браузер и сообщить ему **URL**. В **Windows** это делается просто в командной строке (а значит и просто сделать программе):

```
start http://www.abc.com/cgi-bin/search.cgi?word=hello&language=ru
```

Таким образом программа может сразу вызвать браузер с уже подготовленной страничкой и пользователю не придется даже знать адрес этого поисковика и как он работает, все знает программа.

С методом **POST** в этом отношении сложнее, для этого программе нужно уметь работать по протоколу **HTTP** и связываться с серверами в интернете. Размер и сложность такой программы будет на порядок выше. Поэтому, если Ваш ресурс может быть полезен и при этом передаваемые ему данные не будут превышать 32 Кб, то лучше метода **GET** не найти.

```
#include <stdio.h>
#include <stdlib.h>

//Здесь надо вставить процедуру получения
//параметра по его имени...


char *getparam(...)
{
};

int main()
```

```
{
  char *user=NULL;
  char *content=NULL;
  char *request_method=getenv("REQUEST_METHOD");
  if (strcmp(request_method,"GET")!=0)
            {
                        printf("Content-type: text/html\n\n");
                        printf("Unknown REQUEST_METHOD. Use only GET !\n");
                        return -1;
            };

  content=getenv("QUERY_STRING");
  user=getparam(content,"user=");
  printf("Content-type: text/html\n\n");
  printf("User name=\"%s\"\n",user);
};
```

После того как вся программа будет собрана и скомпилированна, расположите ее в директории **cgi-bin** вашего веб-сервера. Теперь можно смело пробовать форму написанную выше.

```
http://localhost/cgi-bin/primer.cgi?user=hello
```

Таким образом, можно видеть тот же результат, что и при использовании формы:

```
User name="hello"
```

Отлавливать возможные ошибки уже на этапе создания шлюза, иначе без этого он может стать «дырой» в системе безопасности вашего сервера.

Раздел №9. Практические аспекты реализации. Библиотека CGICC

Установка

Для установки библиотеки CGICC в консоли достаточно ввести команду:

```
apt install libcgicc5 libcgicc5-dev
```

Однако полноценная конфигурация данным набором действий не ограничивается, поскольку необходимо осуществлять сборку разработанного приложения, использующего данную библиотеку. Процесс сборки будет представлен с задействованием утилиты make.

Make – основные сведения

make — утилита предназначенная для автоматизации преобразования файлов из одной формы в другую. Правила преобразования задаются в скрипте с именем Makefile, который должен находиться в корне рабочей директории проекта. Сам скрипт состоит из набора правил, которые в свою очеред описываются:

- целями (то, что данное правило делает);
- реквизитами (то, что необходимо для выполнения правила и получения целей);
- командами (выполняющими данные преобразования).В общем виде синтаксис makefile можно представить так:

```
# Индентация осуществляется исключительно при помощи символов табуляции,
# каждой команде должен предшествовать отступ
<цели>: <реквизиты>
        <команда #1>
        ...
        <команда #n>
```

То есть, правило make это ответы на три вопроса:

{Из чего делаем? (реквизиты)} ---> [Как делаем? (команды)] ---> {Что делаем? (цели)}

Несложно заметить, что процессы трансляции и компиляции очень красиво ложатся на эту схему:

{исходные файлы} ---> [трансляция] ---> {объектные файлы}

{объектные файлы} ---> [линковка] ---> {исполнимые файлы}

Для установки и настройки необходимой инфраструктуры необходимо выполнить следующий набор действий.

В консоли выполнить:

```
sudo apt install build-essential
sudo apt install gcc
```

Пример make-файла:

```
all:
    g++ -O3 -s main.cpp -o postr.cgi -lcgicc
```

```
clean:
	rm -f postr.cgi
```

Где:

- main.cpp – файл исходного кода на C++;
- postr.cgi – целевой исполняемый cgi-файл.

Для сборки сервиса необходимо выполнить команду make в директории с файлом исходного кода.

Раздел№10. Практические аспекты реализации. Организация web-страницы. HTML

Введение в HTML

Для быстрого старта сразу же ввести в курс дела создадим веб-страницу без последовательного изучения правил HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>Моя первая веб-страница</title>
 </head>
 <body>

  <h1>Заголовок страницы</h1>
  <p>Основной текст.</p>

 </body>
</html>
```

Обратите внимание, что расширение у файла должно быть именно html.

Инструментарий

Для эффективной работы не обойтись без необходимых и привычных инструментов, в том числе и при написании кода HTML. Поэтому для начальной разработки веб-страниц или даже небольшого сайта – так называется набор страниц, связанных между собой ссылками и единым оформлением, нам понадобятся следующие программы.

- Текстовый редактор.
- Браузер для просмотра результатов.
- Валидатор — программа для проверки синтаксиса HTML и выявления ошибок в коде.
- Графический редактор.
- Справочник по тегам HTML.

Текстовый редактор HTML-документ можно создавать в любом текстовом редакторе, хоть Блокноте, тем не менее, для этой цели подойдет не всякая программа. Нужна такая, чтобы поддерживала следующие возможности:

- подсветка синтаксиса — выделение тегов, текста, ключевых слов и параметров разными цветами. Это облегчает поиск
- нужного элемента, ускоряет работу разработчика и снижает возникновение ошибок;
- работа с вкладками. Сайт представляет собой набор файлов, которые приходится править по отдельности, для чего
- нужен редактор, умеющий одновременно работать сразу с несколькими документами. При этом файлы удобно
- открывать в отдельных вкладках, чтобы быстро переходить к нужному документу;
- проверка текущего документа на ошибки.

Ссылки на некоторые подобные редакторы приведены ниже.

PSPad

http://www.pspad.com/ru/download.php

HtmlReader

http://manticora.ru/download.htm

Notepad++

http://notepad-plus.sourceforge.net/ru/site.htm

EditPlus

http://www.editplus.com

Браузер это программа, предназначенная для просмотра веб-страниц. На первых порах подойдет любой браузер, но с повышением опыта и знаний потребуется завести целый «зверинец», чтобы проверять правильность отображения сайта в разных браузерах. Дело в том, что каждый браузер имеет свои уникальные особенности, поэтому для проверки универсальности кода требуется просматривать и корректировать код с их учетом. На сегодняшний день наибольшей популярностью пользуются три браузера: Firefox, Internet Explorer и Opera.

Mozilla Firefox

Перспективный и развивающийся браузер, получивший признание во всем мире. Его особенность – простота и расширяемость, которая получается

за счет специальных расширений, как они называются. Изначально Firefox имеет набор только самых необходимых функций, но, устанавливая желаемые расширения, в итоге можно нарастить браузер до системы, выполняющей все необходимые для вашей работы действия. Браузер Firefox является открытой системой, разрабатываемый группой Mozilla.

Где скачать

http://www.mozilla.ru/products/firefox/

Microsoft Internet Explorer (IE)

Один из старейших браузеров, который бесплатно поставляется вместе с операционной системой Windows. Это и определило его популярность. Версия IE 7 по удобству приблизилась к своим давним конкурентам, в частности, появились вкладки. К сожалению, этот браузер хуже всех поддерживает спецификацию HTML, поэтому для корректного отображения в IE приходится порой отдельно отлаживать код специально под него.

Где скачать

http://www.microsoft.com/rus/windows/ie/default.mspx

Opera

Быстрый и удобный браузер, поддерживающий множество дополнительных возможностей, повышающих комфортность работы с сайтами.

Где скачать

http://ru.opera.com/download/

Safari

Разработанный компанией Apple этот браузер встроен в iPhone и операционную систему MacOS на компьютерах Apple. Также имеется версия под Windows.

Где скачать

http://www.apple.com/ru/safari/

Google Chrome

Самый свежий браузер, появившийся на рынке в конце 2008 году. Разработан компанией Google.

Где скачать

http://www.google.com/chrome?hl=ru

Валиданиция HTML-документа

Валидация HTML-документа предназначена для выявления ошибок в синтаксисе веб-страницы и расхождений со спецификацией HTML. Соответственно, программа или система для такой проверки называется валидатором. Как проверить HTML-файл на валидность? Если есть доступ в Интернет, то следует зайти по адресу http://validator.w3.org и ввести путь к проверяемому документу или сайту в специальной форме. После проверки будут показаны возможные ошибки или появится надпись, что документ прошел валидацию успешно.

Tidy

Для проверки локального HTML-файла или при отсутствии подключения к Интернету, предназначена программа Tidy. Некоторые редакторы, например, PSPad, уже содержат встроенный Tidy и валидацию документа можно провести без дополнительных средств.

Где скачать

http://tidy.sourceforge.net

Графический редактор необходим для обработки изображений и их подготовки для публикации на веб-странице. Самой популярной программой такого рода является Photoshop, ставший стандартом для обработки фотографий и создания графических изображений для сайтов. Но в большинстве случаев мощь Photoshop-а избыточна, и лучше воспользоваться чем-нибудь более простым и проворным. В частности, программа Paint.Net позволяет сделать все необходимые манипуляции с изображениями, вдобавок бесплатна для использования.

Скачать Paint.Net

http://www.getpaint.net/download.html

Справочник по тегам HTML. Запоминать все теги и их параметры наизусть на первых порах сложно, поэтому требуется периодически заглядывать в руководство, чтобы уточнить тот или иной вопрос. Вообще, хороший справочник нужен всем, независимо от уровня подготовки.

Справочники в Интернете

Описание тегов HTML (на английском языке)

http://www.w3.org/TR/html4/index/elements.html

Раздел№11. Практические аспекты реализации. Организация web-страницы. JavaScript

Введение

Сценарии JavaScript размещаются внутри Web-страницы и не могут существовать отдельно от нее. Для выполнения JS-сценариев не нужен компилятор, они выполняются браузером. JS-сценарий - это обычный текст, и вы можете просмотреть код сценария невооруженным взглядом - без какого-нибудь специального программного обеспечения.

Что же такоеjavaScript? Это язык программирования с синтаксисом, сходным с языком Java, использующийся в составе HTML-страниц для увеличения их функциональности. Первоначально языкjavaScript был разработан компаниями Netscape и Sun Microsystems, за его основу был взят язык Sunjava.

JavaScript позволяет реализовать те функции страницы, которые невозможно реализовать стандартными тегами HTML. Сценарии запускаются в результате наступления какого-нибудь события, например пользователь нажал кнопку или изменился размер окна. JavaScript имеет доступ к свойствам документа и свойствам браузера. Например, najavaScript вы можете легко изменять заголовок окна браузера или текст в строке состояния.

Для написания сценариев JavaScript не нужно никакое специальное программное обеспечение - достаточно простого текстового редактора. Так что вы можете использовать свой любимый HTML-редактор для написания JavaScript-кода.

Объектная модель JavaScript

В JavaScript используется объектная модель документа, в рамках которой каждый HTML-контейнер можно рассматривать как совокупность свойств, методов и событий, происходящих в браузере. По сути, это связь между HTML-страницей и браузером. В этой главе мы не будем вникать в ее подробности, а желающие "бежать впереди паровоза" могут узнать больше по следующей ссылке: https://developer.mozilla.org/en-US/docs/Web/ JavaScript/Guide/Details_of_the_ Object_Model

Пока вам нужно знать, что есть старший класс Window, позволяющий обращаться к методам и свойствам HTML-страницы и браузера. Например, метод close() позволяет закрыть окно браузера, а свойство location – обратиться к адресной строке браузера.

Если вы не знакомы с объектно-ориентированным программированием, тогда, наверное, не особо понимаете, о чем идет речь, когда мы говорим об объектах и методах. В JavaScript все основано на классах и объектах (поскольку это объектно-ориентированный язык), и без них вы не сможете написать свои программы. Именно поэтому мы начали разговор об объектной модели JavaScript в этой вводной главе.

Объект можно воспринимать как совокупность данных и методов (функций) для их обработки. В JavaScript с некоторыми объектами также связываются определенные события. Давайте разберемся, что такое объект, как говорится, "на пальцах". Представим, что объект - это человек. Путь наш объект называется Human. У такого объекта может быть масса характеристик - имя, пол, дата рождения и т.д. Все это называется свойствами объекта. Обратиться к свойству можно так: «объект.свойство»

Метод - это функция обработки данных. Например, метод, возвращаемый год рождения человека может называться GetYear. Обращение к методу производится так:

объект.метод(параметры);

Например:

Human.GetYear();

Связать метод с функцией можно так:

объект.метод = имя_функции;

С объектом может быть связано какое-нибудь событие. Например, при рождении человека может генерироваться событие OnBirth. Для каждого события можно определить его обработчик - функцию, которая будет на него реагировать. Что будет делать эта функция, зависит от события. Например, обработчик OnBirth может заносить в некую общую таблицу базы данных информацию об объекте - имя, пол и дату рождения. Такая таблица может использоваться для ускорения поиска нужного объекта. Теперь рассмотрим еще одно, более абстрактное, понятие – класс. Класс – это образец объекта, если хотите - тип переменной объекта. Пусть мы разработали класс Human, тогда объект, то есть экземпляр класса Human, может называться Human1. Вы можете создавать несколько объектов класса Human - имена у них будут разными:

John new Human;

Mary = new Human;

создаем объект John класса Human

создаем объект Mary класса Human

Пример программы на JavaScript

Чтобы ваша JavaScript-программа (или сценарий) запустилась, ее нужно внедрить в HTML-код документа (или "связать" программу с документом). Обратите внимание - я использовал два разных термина – внедрение и связка. Сделано это умышленно, потому что есть разные способы взаимного существования JavaScript и HTML. Самый простой из них - использование тега <SCRIPT>, который обычно расположен до тега <BODY> (принято размещать тег <SCRIPT> в теге <HEAD>), например:

```
<SCRIPT>
document.write("<hl>Привет,  мир!</hl>");
</SCRIPT>
```

Данный сценарий будет выполнен при открытии страницы и выведет строку:

```
<hl>Привет,  мир!</hl>
```

Однако не всегда нужно, чтобы ваша программа начинала работать сразу при открытии страницы. Чаще всего требуется, чтобы она запускалась при определенном событии, например при нажатии какой-то кнопки. При нажатии кнопки генерируется событие onClick. Обработчик для этого события можно указать прямо в HTML-коде, например:

```
<FORM>
<INPUT TYPE=button  VALUE="Назад"  onClick="history.back()">
</FORM>
```

Разумеется, не всегда обработчик события компактный. Иногда он просто не помещается в одну-две строчки. Конечно, максимальная длина значения атрибута HTML-тега составляет 1024 символа, что вполне хватило бы для несложных обработчиков. Однако такие длинные обработчики значительно ухудшают читабельность HTML-кода, поэтому если обработчик события достаточно длинный, его принято оформлять в виде функции, например:

```
<form>
<input  type="button"  value="Нажми меня"
onClick="openWindow()">
</form>
```

Функцию open Window() нужно определить в теге <SCRIPT>. Полный код (HTML и JavaScript) нашего первого "проекта" приведен в листинге:

```
<html>
<head>
<script  language="JavaScript">
function  openWindow() {
msgWindow=  open("index.html")
</script>
</head>
<body>
<form>
<input  type="button"  value="Открыть главную страницу "
onClick="openWindow()">
</form>
</body>
</html>
```

Данная страница отобразит кнопку «Открыть главную страницу» (рис. 2.1), при нажатии которой будет открыто новое окно с загруженной страницей index.html.

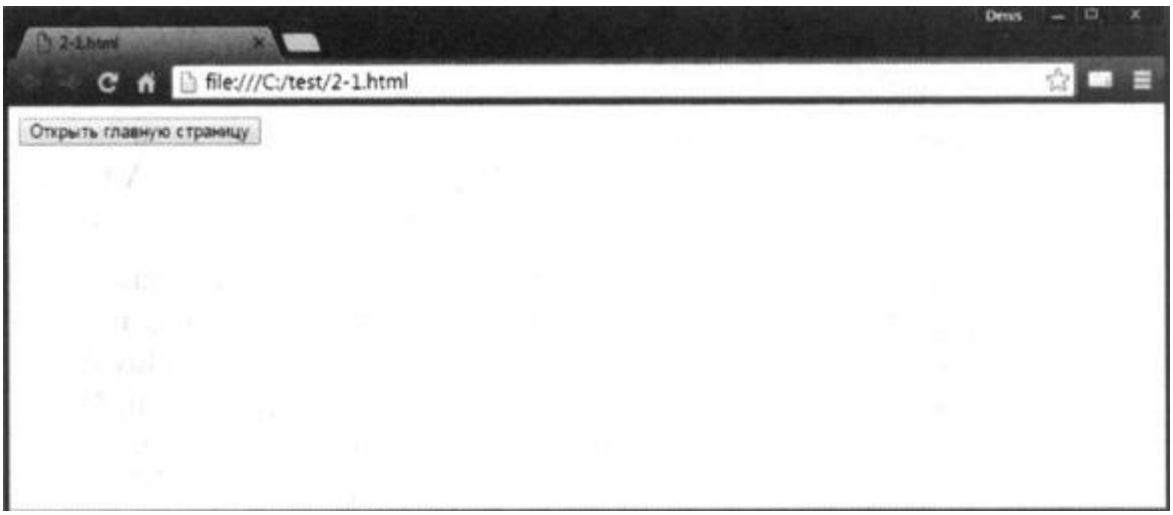

Рис. 9 Пример результата работы JavaScript-сценария

Обратите внимание: мы указали атрибут language тега <SCRIPT>, указы-вающий язык программирования, на котором написан сценарий (кроме JavaScript сценарии можно писать еще на VBS - Visual Basic Script). Кроме атрибута language у тега <SCRIPT> есть еще один очень полезный атрибут – src, указывающий файл, в котором содержится JS-код. Если код JavaScript очень большой, и вы не хотите загромождать им HTML-код страницы , можете вынести его в отдельный файл, а затем подключить его следующим способом:

```
<s c ript  src="javascript.js"  type= "text/javasc ript " >< / scrip t >
```

Четвертый (и последний) атрибут тега <SCRIPT> называется defer. Он откладывает выполнение сценария до полной загрузки страницы (по умолчанию сценарий начинает выполняться сразу при открытии страницы):

```
<script  def er>
11 code
</script.>
```

Аргумент language, как уже было отмечено выше, задает язык программирования, на котором написан сценарий, и может принимать следующие значения:

- JavaScript – классический вариант языка программирования, который рассматривается в этой книге (разработка Netscape и Sun).
- JScript – разновидность языка программирования JavaScript, разработанная компанией Microsoft. Компании Microsoft обязательно

нужно построить дом заново и заново изобрести велосипед, поэтому разница между JavaScript иJScipt заключается не только в названии, но и в под-ходах к программированию. Учтите, что поддержка JScript в некоторых браузерах может быть ограниченной, поэтому язык J avaScript более предпочтителен.

- VBS, VBScript – сценарий на языке программирования VBScript, в ос-нове которого лежит Visual Basic. Поддерживается браузером Internet Explorer. Остальные браузеры или вовсе не поддерживают VBS или поддержка весьма ограничена.

Комментарии в JavaScript

В этой главе вы уже успели познакомиться с комментариями, во всяком случае неявно. Теперь настало время познакомиться с ними явно. Комментарии в JS могут быть однострочными и многострочными. Однострочный комментарий начинается с двух знаков / /, с этим типом комментариев вы уже знакомы:

```
//комментарий
i++; //увеличиваем i
```

Комментарием является все, что находится после / / и до конца строки. Многострочный комментарий начинается символами /* и заканчивается

символами * /, например:

```
/* это пример
многострочного
комментария */
```

Какие комментарии использовать – зависит от вас. Однострочные комментарии удобно использовать для комментирования отдельных строчек кода. Многострочные подойдут для объяснения того, что делает целый блок, например, описать, что делает функция и какие параметры ей нужно передать.

Диалоговые окна в JavaScript

Для взаимодействия с пользователем, то есть для ввода данных и вывода результатов работы программы, как правило, используются HTML-формы и возможность вывода HTML-кода прямо в документ (метод document. write). Этот способ удобен тем, что вы, используя HTML и CSS, можете оформить форму ввода данных так, как вам заблагорассудится. То же самое можно

сказать и о выводе данных. Из J S-сценария вы можете выводить любой HTML-код, позволяющий как угодно оформить вывод.

Но в некоторых ситуациях этих возможностей оказывается очень много. Иногда нужно просто вывести диалоговое сообщение, например сообщить пользователю, о том, что введенный им пароль слишком простой или слишком короткий. В этом разделе мы разберемся, как выводить диалоговые окна, позволяющие выводить короткие сообщения (например, сообщения об ошибках ввода) и обеспечивающие ввод данных.

Метод alert() - простое окно с сообщением и кнопкой ОК

Метод alert() объекта window используется для отображения простого окна с сообщением и одной кнопкой - ОК. Такое окно может использоваться, например, для отображения сообщений об ошибках (короткий/простой/неправильный пароль). Окно, кроме донесения до пользователя сообщения, больше не предусматривает никакого взаимодействия с ним. Методу alert() передается только одна строка - отображаемая. Чтобы отобразить многострочное сообщение, разделяйте строки символом \n:

```
window.alert("Привет,  мир!");
window.alert("Привет,\nмир!");
```

Первая программа была не "Hello, world", но давайте не будем изменять традиции и все-таки напишем эту программу, пусть она будет и непервой – хоть тут мы будем отличаться от всех остальных программистов.

Сценарий, демонстрирующий использование метода alert(),

```
<html>
<head>
  <title>Alert</title>
</head>
<body>
  <script>
    window.alert("Привет, мир!");
  </script>
</body>
</html>
```

Метод confirm() - окно с кнопками ОК и Cancel

Другой часто используемый метод - confirm(). Он выводит окно с сообщением и двумя кнопками - ОК и Cancel, позволяя пользователю выбрать одну из них. Проанализировав возвращаемое методом значение (true, если нажата кнопка ОК и false - в противном случае), вы можете выполнить то или иное действие. Для нашего примера мы будем просто выводить с помощью alert() название нажатой кнопки.

```
<head>
  <title>Confirm</title>
</head>
<body>
  <script>
   if (window.confirm("Нажмите ОК или Отмена")) {
      window.alert("ОК");
   }
   else {
      window.alert("Отмена");
   }
</script>
</body>
</html>
```

Метод prompt() ·диалоговое окно для ввода данных

Метод prompt() отображает диалоговое окно с полем ввода, сообщением и кнопками ОК и Cancel. Введенное пользователем значение можно потом будет присвоить какой-то переменной. Диалог возвращает введенную пользователем строку. Если пользователь ничего не ввел, диалог возвращает значение null.

Методу prompt() нужно передать два параметра - строку, которая будет отображена в качестве приглашения ввода (над полем для ввода данных), и значение по умолчанию, которое будет передано в сценарий, если пользователь поленится ввести строку и просто нажмет ОК.

Раздел№12. Практические аспекты реализации. Организация web-страницы. CSS

Введение

Каскадные таблицы стилей, или Cascading Style Sheets (CSS), обеспечивают творческую свободу в разметке и дизайне веб-страниц. Пользуясь CSS, вы сможете украсить текст страниц привлекательными заголовками, буквицами и рамками, как в красочных глянцевых журналах. Можно точно разместить и позиционировать изображения, сделать колонки и создать баннеры, выделить ссылки динамическими эффектами. Можно также добиться постепенного появления и исчезновения элементов, перемещения объектов по странице или медленного изменения цвета кнопки при прохождении над ней указателя мыши.

Каскадные таблицы стилей как раз и предназначены для упрощения процесса оформления веб-страниц. Следующие подразделы будут посвящены изучению основ CSS. Что такое CSS? CSS — это язык стилей. Он используется для того, чтобы придать страницам на HTML — фундаментальном языке Всемирной паутины — совершенный вид.

Код на языке HTML образует структуру веб-страниц с их содержимым (контентом). Хотя страница, созданная с помощью только языка HTML, не очень красива, без него Всемирной паутины не существовало бы. Поэтому для получения от CSS наибольшей отдачи ваш HTML-код должен предоставить однородную, хорошо скроенную основу. В данном разделе вы познакомитесь с основами каскадных таблиц стилей, узнаете, как создавать HTML-код, более приспособленный под нужды CSS.

Важно отметить, что повсеместное использование на сайте CSS существенно облегчает создание HTML-кода. Вам больше не нужно будет применять средства HTML для создания и улучшения дизайна страниц (собственно, HTML для этого никогда и не предназначался). Все, что связано с графическим дизайном страниц, обеспечивается с помощью CSS. Соответственно, работа над основным кодом веб-страниц на языке HTML упрощается, поскольку HTML-страницы, написанные для совместной работы с CSS, имеют менее громоздкий, более понятный и прозрачный код. При этом, естественно, страницы загружаются быстрее, что немаловажно для рядовых пользователей.

Верстка HTML и CSS

HTML придает особый вид тексту путем деления его на логические блоки и их определения на веб-странице: например, главное значение элемента h1 — создать заголовок, предшествующий основному содержимому страницы. Заголовки второго, третьего уровней и т. д. — подзаголовки — позволяют делить содержимое страниц на менее важные, но связанные разделы. У веб-страницы, как и у книги, которую вы держите в руках, должна быть логическая структура. У каждой главы этой книги есть заголовок (отформатированный, например, элементом h1), а также несколько разделов и, соответственно, подзаголовков (например, с элементом h2), которые, в свою очередь, содержат подразделы с заголовками более низкого уровня. Представьте, насколько сложнее было бы читать эту книгу, если бы весь текст состоял из одного длинного абзаца, без деления на разделы, подразделы, пункты, без выделения примечаний, гиперссылок и т. д.

Помимо заголовков, в HTML есть множество других элементов для разметки содержимого веб-страницы, а также для определения назначения ее каждого логического фрагмента. Наиболее часто применяют следующие элементы: p — для создания абзацев текста, ul — для создания маркированных (ненумерованных) списков.

Далее по степени применения идут элементы, отображающие специфичное содержимое, например abbr — сокращения, аббревиатуры, code — программный код. При написании HTML-кода для CSS используйте элементы, размещая их рядом друг с другом, насколько это возможно. Ориентируйтесь на роль, которую играет фрагмент текста на веб-странице, а не на внешний вид, который текст приобретает благодаря этому элементу. Например, перечень ссылок на панели навигации — это и не заголовок, и не абзац текста. Больше всего это похоже на маркированный список. Таким образом, выбираем элемент ul. Элементы маркированного списка расположены вертикально один за другим, а нам требуется горизонтальная панель навигации, в которой все ссылки располагаются горизонтально. Об этом можно не беспокоиться. До появления CSS для достижения определенных визуальных эффектов дизайнеры пользовались элементом font и другими средствами HTML:

```
<p>
 <strong>
  <font color="#0066FF" size="5" face="Verdana,
   Arial, Helvetica, sans-serif">Urban Agrarian
   Lifestyle</font></strong>
  <br />
```

```
   <font color="#FF3300" size="4" face="Georgia,
     Times New Roman, Times, serif">
   <em>
  <strong>A Revolution in Indoor Agriculture
   <br /></strong></em></font>
     Lorem ipsum dolor sit amet...
</p>
```

Используя CSS, можно добиться тех же результатов (и даже лучше) с гораздо меньшим объемом HTML-кода:

```
<h1>The Urban Agrarian Lifestyle</h1>
<h2>A Revolution in Indoor Agriculture</h2>
<p>Lorem ipsum dolor sit amet...</p>
```

Кроме того, применяя CSS для дизайна веб-страницы, вы используете HTML по его прямому назначению, то есть именно для разметки кода веб-страницы на логические фрагменты, не заботясь о форматировании и внешнем виде страницы.

Элементы div и span

Элементы div и span применяются на протяжении всего существования Всемирной паутины. Обычно они используются для организации и группирования контента, с чем не всегда справляются другие HTML-элементы. Они похожи на пустые сосуды, которые вы сами и заполняете. Элемент div (предназначен для деления на фрагменты) определяет любой отдельный блок содержимого, как, например, абзац или заголовок, поэтому называется блочным. Поскольку у элементов div и span нет никаких свойств для визуализации, вы можете применять к ним CSS-стили, чтобы фрагменты внутри этих тегов выглядели так, как вам хочется. Однако вы также можете логически объединить любой набор таких элементов, как заголовок, несколько абзацев, маркированный список и т. д., в единственном блоке div.

Элемент div — замечательное средство разбивки веб-страницы на такие логические фрагменты, как баннер, колонтитул, боковая панель и т. д. Элемент span применим к строчным элементам страницы, то есть к словам, фразам, которые находятся в пределах абзаца текста или заголовка. Его можно использовать точно так же, как и другие строчные HTML-теги, к примеру, как a (элемент

привязки, позволяющий добавить ссылку к фрагменту текста) или strong (позволяющий отформатировать фрагмент текста в абзаце полужирным начертанием).

Можно применить элемент span, например, к названию компании и затем использовать CSS, чтобы выделить это название другим шрифтом, цветом и т. д.

Рассмотрим пример использования этих элементов в работе. К ним добавлены атрибуты id и class, предназначенные в том числе для применения стилей к фрагментам страницы.

```
<div id="footer">
  <p>Собственность 2015, <span class="bizName">SuperCo.com</span></p>
  <p>Звоните в службу поддержки потребителей по номеру 555-555-5501
    для подробной информации.</p>
</div>
```

Разговор об этих элементах не ограничится кратким введением. Они часто встречаются на тех веб-страницах, где широко применяется CSS, и эта книга поможет вам научиться использовать их в комбинации с CSS для получения творческого контроля над веб-страницами

Создание стилей и таблиц стилей

Определение стиля в CSS, устанавливающего внешний вид какого-либо элемента (фрагмента) веб-страницы, — это всего лишь правило, которое сообщает браузеру, что и каким образом форматировать: изменить цвет шрифта заголовка на синий, выделить фото красной рамкой, создать меню шириной 150 пикселов для списка гиперссылок. Если бы стиль мог говорить, он сказал бы: «Браузер, сделай, чтобы вот это выглядело так-то».

Фактически определение стиля состоит из двух основных элементов: самого элемента веб-страницы, который непосредственно подлежит форматированию браузером, — селектора, а также команд форматирования — блока объявления.

Селекторами могут быть заголовок, абзац текста, изображение и т. д. Блоки объявления могут, например, окрасить текст в синий цвет, добавить красную рамку (границу) вокруг абзаца, установить фотографию в центре страницы — возможности форматирования бесконечны.

Например, тело сайта включает простой стиль:

```
body {
  font-family: "Franklin-Book", Helvetica, Arial, sans-serif;
  color: #222;
}
```

Разумеется, CSS-стили не могут быть написаны на обычном языке, как, например, предыдущий абзац. У них есть собственный язык. В частности, чтобы установить красный цвет и размер шрифта 1,5 em для всех абзацев на веб-странице, нужно написать следующее:

```
p { color: red; font-size: 1.5em; }
```

Этот стиль как бы говорит браузеру: «Раскрась текст всех абзацев веб-страницы, заключенных в элемент p, красным цветом и установи размер шрифта равным 1,5 em

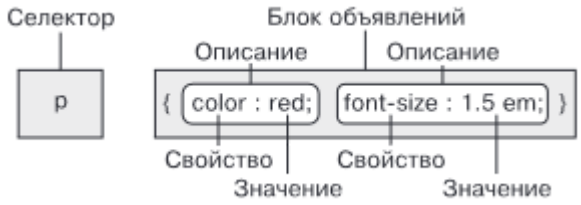

Рис. 10 Структура стилей CSS

- **Селектор**. Как уже было отмечено, селектор сообщает браузеру, к какому элементу или элементам веб-страницы применяется стиль: к заголовку, абзацу, изображению или гиперссылке. Селектор p обращается к элементу p, передавая браузеру, что все элементы p нужно форматировать, используя объявления, указанные в данном стиле. Благодаря большому разнообразию селекторов, предлагаемых языком CSS, и небольшому творческому потенциалу вы сможете мастерски форматировать веб-страницы (в следующей главе селекторы описаны более подробно).
- **Блок объявления**. Код, расположенный сразу за селектором, содержит все команды форматирования, которые можно применить к этому селектору. Блок начинается с открывающей фигурной скобки { и заканчивается закрывающей}.
- **Объявление**. Между открывающей и закрывающей фигурными скобками блока объявления можно добавить одно или несколько объявлений — команд форматирования. Каждое объявление имеет две части — свойство и значение. Двоеточие отделяет имя свойства от его значения, и все объявление заканчивается точкой с запятой.

- **Свойство**. Каскадные таблицы стилей предлагают большой выбор команд форматирования, называемых свойствами. Свойство представляет собой слово или несколько написанных через дефис слов, определяющих конкретный стиль. У большинства свойств есть соответствующие простые для понимания имена, такие как font-size, margin-top, background-color и т. д. (в переводе с английского: размер шрифта, верхний отступ, цвет фона). В следующих главах будет описано множество полезных свойств CSS. После имени свойства нужно добавить двоеточие, чтобы отделить его от значения.
- **Значение**. Наконец настал тот момент, когда вы можете задействовать свой творческий потенциал, присваивая значения CSS-свойствам: к примеру, устанавливая фоновый цвет синим, красным, фиолетовым, салатовым и т. д. Как будет описано в следующих главах, различные свойства каскадных таблиц стилей требуют указания определенных типов значений — цвета (red или #FF0000), размера (18px, 200% или 5em), URL-адреса (images/background.gif), а также опре-деленных ключевых слов (top, center или bottom).

У стилей может быть множество свойств форматирования, и есть возможность упорядочить код таблицы стилей, разбивая объявления на строки. Например, поместите селектор и открывающую скобку на одной строке, каждое объявление – далее на отдельных строках, а закрывающую фигурную скобку – отдельно на последней строке стиля. Это будет выглядеть следующим образом:

```
p {
  color: red;
  font-size: 1.5em;
}
```

Любой браузер игнорирует символы пробела и табуляции, так что вы можете спокойно добавлять их, создавая хорошо читаемые стили CSS. Кроме того, при перечислении свойств полезно сделать отступ табуляцией или несколькими пробелами для явного отделения селектора от блока объявления. К тому же один пробел между двоеточием и значением свойства, конечно, необязателен, но он обеспечивает дополнительную удобочитаемость стилей. Фактически можно добавить любое количество пробелов там, где вам захочется. Например, color:red, color: red и color : red — все варианты будут правильно работать.

Раздел №13. Практические аспекты реализации. Установка и конфигурирование PostgreSQL

Установка PostgreSQL

Чтобы установить PostgreSQL на ваш сервер Debian, выполните следующие действия от имени пользователя root или пользователя с привилегиями sudo. Установка начинается обновления индекса пакета APT:

```
sudo apt update
```

Установите сервер PostgreSQL и пакет contrib, который предоставляет дополнительные функции для базы данных PostgreSQL:

```
sudo apt install postgresql postgresql-contrib
```

После завершения установки запустится служба PostgreSQL. Чтобы проверить установку, используйте инструмент psql для печати версии сервера:

```
sudo -u postgres psql -c "SELECT version();"
```

Роли и методы аутентификации PostgreSQL

PostgreSQL обрабатывает разрешения на доступ к базе данных, используя концепцию ролей. В зависимости от того, как вы настроили роль, она может представлять пользователя базы данных или группу пользователей базы данных.

PostgreSQL поддерживает несколько методов аутентификации. Наиболее часто используемые методы:

1) Доверие — роль может подключаться без пароля, если соблюдаются критерии, определенные в pg_hba.conf .
2) Пароль — роль может подключиться, указав пароль. Пароли могут быть сохранены как scram-sha-256 md5 и password (открытый текст)

3) Идентификатор — поддерживается только для подключений TCP / IP. Он работает путем получения имени пользователя операционной системы клиента с дополнительным отображением имени пользователя.
4) Peer — То же, что и Ident, но поддерживается только для локальных подключений.

Аутентификация клиента PostgreSQL определяется в файле конфигурации с именем pg_hba.conf . Для локальных подключений PostgreSQL настроен на использование метода одноранговой аутентификации. Пользователь «postgres» создается автоматически при установке PostgreSQL. Этот пользователь является суперпользователем для экземпляра PostgreSQL и эквивалентен пользователю root MySQL.

Чтобы войти на сервер PostgreSQL как «postgres», переключитесь на пользовательский postgres и получите доступ к приглашению PostgreSQL с помощью утилиты psql :

```
sudo su - postgres
psql
```

Отсюда вы можете взаимодействовать с сервером PostgreSQL. Чтобы выйти из оболочки PostgreSQL, введите:

```
q
```

Вы можете использовать команду sudo для доступа к приглашению PostgreSQL без переключения пользователей:

```
sudo -u postgres psql
```

Пользователь postgres обычно используется только с локального хоста.

Создание роли и базы данных PostgreSQL

Команда createuser позволяет создавать новые роли из командной строки. Только суперпользователи и роли с привилегией CREATEROLE могут создавать новые роли. В следующем примере мы создадим новую роль с

именем kylo , базу данных с именем kylodb и предоставим этой роли права kylodb к базе данных.

Сначала создайте роль, введя следующую команду:

```
sudo su - postgres -c "createuser kylo"
```

Затем создайте базу данных с помощью команды createdb :

```
sudo su - postgres -c "createdb kylodb"
```

Чтобы предоставить пользователю права доступа к базе данных, подключитесь к оболочке PostgreSQL:

```
sudo -u postgres psql
```

Выполните следующий запрос:

```
grant all privileges on database kylodb to kylo ;
```

Удаленный доступ к серверу PostgreSQL

По умолчанию сервер PostgreSQL прослушивает только локальный интерфейс 127.0.0.1. Если вы хотите подключиться к серверу PostgreSQL из удаленных мест, вам необходимо настроить сервер на прослушивание общедоступного интерфейса и отредактировать конфигурацию, чтобы принимать удаленные подключения.

Откройте файл конфигурации postgresql.conf и добавьте listen_addresses = '*' в раздел CONNECTIONS AND AUTHENTICATION . Это дает серверу команду прослушивать все сетевые интерфейсы.

```
sudo nano /etc/postgresql/11/main/postgresql.conf
```

/etc/postgresql/11/main/postgresql.conf

#--

 # CONNECTIONS AND AUTHENTICATION

#--

 # - Connection Settings -

listen_addresses = ''   # what IP address(es) to listen on;*

Сохраните файл и перезапустите службу PostgreSQL, чтобы изменения вступили в силу:

sudo service postgresql restart

Выходные данные должны показать, что сервер PostgreSQL прослушивает все интерфейсы (0.0.0.0). Последний шаг — настроить сервер для приема удаленных входов в систему путем редактирования файла pg_hba.conf .

Ниже приведены несколько примеров, показывающих различные варианты использования:

/etc/postgresql/11/main/pg_hba.conf

 # TYPE DATABASE   USER   ADDRESS   METHOD

 # The user jane will be able to access all databases from all locations using an md5 password

host   all   jane   0.0.0.0/0   md5

 # The user jane will be able to access only the janedb from all locations using an md5 password

host   janedb   jane   0.0.0.0/0   md5

 # The user jane will be able to access all databases from a trusted location (192.168.1.134) without a password

host   all   jane   192.168.1.134   trust

Коллективное выполнение лабораторной работы

За каждым из участников команды из трех человек должна быть зафиксирована роль, в соответствии с которой должны быть решены задачи, определенные в графе «Зона ответственности».

Роль	«Специалист по дизайну графических пользовательских интерфейсов»
Зона ответственности	Ход выполнения. В соответствии с вариантом задания реализовать web-страницу (web-интерфейс) с использованием технологий HTML, JavaScript и CSS со следующим(ми) функционалом/характеристиками: – web-страница должна содержать стилевые правила для каждого текстового блока; – web-страница должна содержать элементы интерактивного взаимодействия с использованием JavaScript; – web-страница должна вызывать cgi-сервис и получать от него данные методом GET/POST; – реализовать системную интеграцию с остальными программными модулями, разработанными другими участниками команды.
Приоритет при защите	первый

Роль	«Разработчик Web и мультимедийных приложений»
Зона ответственности	Ход выполнения. В соответствии с вариантом задания реализовать cgi-сервис на языке C/C++ со следующим(ми) функционалом/ характеристиками: – cgi-сервис должен производить проверку корректности введенных данных на web-странице; – cgi-сервис должен выводить данные сервера на результирующую web-страницу; – в cgi-сервисе должно быть реализовано подключение к СУБД (PostgreSQL) и чтение/запись данных из/в таблиц БД; – реализовать системную интеграцию с остальными программными модулями, разработанными другими участниками команды.
Приоритет при защите	второй

Роль	«Администратор баз данных»
Зона ответственности	Ход выполнения. В соответствии с вариантом задания установить и сконфигурировать СУБД PostgreSQL со следующим(ми) функционалом/характеристиками: – СУБД должна разрешать подключение только под конкретным пользователем или набором пользователей; – предусмотреть подключение к БД минимум под двумя ролями (администратор БД, пользователь), которые должны иметь отличия в правах; – создать БД в соответствии с вариантом задания и выдать соответствующие права для подключения к ней; – реализовать системную интеграцию с остальными программными модулями, разработанными другими участниками команды.
Приоритет при защите	третий

Подготовка к защите лабораторной работы

Каждый участник команды из трех человек должен по результатам проведенной работы подготовить устный доклад на 3-5 минут, в рамках которого он должен пояснить, что было сделано, какие возникли сложности, проблемы в реализации этапов и какие были сделаны выводы в процессе их реализации.

Специалист по дизайну графических пользовательских интерфейсов должен быть готов ответить на следующие вопросы.

1. Какие элементы и атрибуты являются основными для HTML?
2. Какова структура CSS и как CSS применяется при верстке web-страниц?
3. Как следует использовать JavaScript для управления DOM и обработкой событий?

Разработчик Web и мультимедийных приложений должен быть готов ответить на следующие вопросы.

1. Каково назначение и принцип работы web-сервера Apache?
2. Каково назначение CGI-протокола?
3. Каковы основные отличия клиентских запросов GET и POST?

Администратор баз данных должен быть готов ответить на следующие вопросы.

1. Какие основные типы данных реализованы в СУБД PosgtreSQL?
2. Что такое хранимая процедура и каково ее применение?

3. Опишите структуру конфигурационного файла pg_hba.conf.

Список литературы

1. Макфарланд Д. Новая большая книга CSS. — СПб.: Питер, 2016. — 720 с.: ил. — (Серия «Бестселлеры O’Reilly»). ISBN 978-5-496-02080-0.
2. Никольский А. П. Javascript на примерах. Практика, практика и только практика - СПб.: Наука и Техника, 2018. - 272 с., ил. ISBN 978-5-94387-762-9.
3. Новиков Б. А. Основы технологий баз данных: учебное пособие / Б. А. Новиков, Е. А. Горшкова, Н. Г. Графеева; под ред. Е. В. Рогова. — 2-е изд. — М.: ДМК Пресс, 2020. — 582 с. ISBN 978-5-97060-841-8.
4. Хокинс, Скотт. Администрирование Web-сервера Apache и руководство по электронной коммерции.: Пер. с англ. М. : Издательский дом "Вильяме", 2001. — 336 с. : ил. — Парал. тит. англ. ISBN 5-8459-0212-6 (рус.).

Варианты заданий для выполнения лабораторных работ

Номер варианта задания	Задание на лабораторную работу
1	Разработка программного обеспечения для автоматизированной/информационной системы железной дороги
2	Разработка программного обеспечения для автоматизированной/информационной системы авиакомпании
3	Разработка программного обеспечения для автоматизированной/информационной системы аэропорта
4	Разработка программного обеспечения для автоматизированной/информационной системы морского порта
5	Разработка программного обеспечения для автоматизированной/информационной системы автобусного вокзала
6	Разработка программного обеспечения для автоматизированной/информационной системы школы
7	Разработка программного обеспечения для автоматизированной/информационной системы библиотеки
8	Разработка программного обеспечения для автоматизированной/информационной системы университета
9	Разработка программного обеспечения для автоматизированной/информационной системы службы занятости
10	Разработка программного обеспечения для автоматизированной/информационной системы службы социальной защиты
11	Разработка программного обеспечения для автоматизированной/информационной системы поликлиники
12	Разработка программного обеспечения для автоматизированной/информационной системы обязательного медицинского страхования
13	Разработка программного обеспечения для автоматизированной/информационной системы пенсионного фонда
14	Разработка программного обеспечения для автоматизированной/информационной системы выставочного комплекса
15	Разработка программного обеспечения для автоматизированной/информационной системы для организации НИОКР

16	Разработка программного обеспечения для автоматизированной/информационной системы издательства
17	Разработка программного обеспечения для автоматизированной/информационной системы редакции газеты
18	Разработка программного обеспечения для автоматизированной/информационной системы типографии
19	Разработка программного обеспечения для автоматизированной/информационной системы гостиницы
20	Разработка программного обеспечения для автоматизированной/информационной системы киноцентра
21	Разработка программного обеспечения для автоматизированной/информационной системы фирмы по прокату автомобилей
22	Разработка программного обеспечения для автоматизированной/информационной системы букмекерской фирмы
23	Разработка программного обеспечения для автоматизированной/информационной системы фондовой биржи
24	Разработка программного обеспечения для автоматизированной/информационной системы банка
25	Разработка программного обеспечения для автоматизированной/информационной системы лизинговой компании
26	Разработка программного обеспечения для автоматизированной/информационной системы туристического агентства
27	Разработка программного обеспечения для автоматизированной/информационной системы фильмотеки
28	Разработка программного обеспечения для автоматизированной/информационной системы агентства недвижимости
29	Разработка программного обеспечения для автоматизированной/информационной системы страховой организации
30	Разработка программного обеспечения для автоматизированной/информационной системы автошколы
31	Разработка программного обеспечения для автоматизированной/информационной системы оператора связи
32	Разработка программного обеспечения для автоматизированной/информационной системы автосервиса
33	Разработка программного обеспечения для автоматизированной/информационной системы для оказания госуслуг

34	Разработка программного обеспечения для автоматизированной/информационной системы фирмы по сборке и продаже компьютеров и комплектующих
35	Разработка программного обеспечения для автоматизированной/информационной системы транспортной фирмы
36	Разработка программного обеспечения для автоматизированной/информационной системы супермаркета
37	Разработка программного обеспечения для автоматизированной/информационной системы книжного магазина
38	Разработка программного обеспечения для автоматизированной/информационной системы ломбарда
39	Разработка программного обеспечения для автоматизированной/информационной системы ГИБДД
40	Разработка программного обеспечения для автоматизированной/информационной системы спортивного клуба
41	Разработка программного обеспечения для автоматизированной/информационной системы интернет-провайдера
42	Разработка программного обеспечения для автоматизированной/информационной системы интернет-магазина
43	Разработка программного обеспечения для автоматизированной/информационной системы интернет-аукциона
44	Разработка программного обеспечения для автоматизированной/информационной системы почтовой службы
45	Разработка программного обеспечения для автоматизированной/информационной системы предприятия ЖКХ
46	Разработка программного обеспечения для автоматизированной/информационной системы рекламного агентства
47	Разработка программного обеспечения для автоматизированной/информационной системы курьерской фирмы
48	Разработка программного обеспечения для автоматизированной/информационной системы ресторанного комплекса
49	Разработка программного обеспечения для автоматизированной/информационной системы службы такси
50	Разработка программного обеспечения для автоматизированной/информационной системы службы технической поддержки
51	Разработка программного обеспечения для автоматизированной/информационной системы <свой вариант> (необходимо сформулировать тему)