УВАЖАЕМЫЕ СТУДЕНТЫ! Законспектируйте в своей рабочей тетради по дисциплине приведенную лекцию (объемом 4-5 страницы).

Результаты работы, фотоотчет, предоставить преподавателю на e-mail: igor-gricenko-95@mail.ru в течении ТРЕХ дней.

При возникновении вопросов по приведенному материалу обращаться по следующему номеру телефона: (072)132-63-42

ВНИМАНИЕ!!! При отправке работы, не забывайте указывать ФИО студента, наименование дисциплины, дата проведения занятия (по расписанию).

Лекция 12

Тема: Формы в PHP

Цель: Изучить методы создания и работы с формами PHP

Основы клиент-серверных технологий

В самом начале курса мы уже говорили о том, что PHP – это скриптовый язык, обрабатываемый *сервером*. Сейчас мы хотим уточнить, что же такое *сервер*, какие функции он выполняет и какие вообще бывают *серверы*. Если речь идет о *сервере*, невольно всплывает в памяти понятие *клиента*. Все потому, что эти два понятия неразрывно связаны. Объединяет их компьютерная *архитектура клиент-сервер*. Обычно, когда говорят «*сервер*», имеют в виду *сервер* в *архитектуре клиент-сервер*, а когда говорят «*клиент*» – имеют в виду *клиент* в этой же *архитектуре*. Так что же это за архитектура? Суть ее в том, чтобы разделить функции между двумя подсистемами: *клиентом*, который отправляет запрос на выполнение каких-либо действий, и *сервером*, который выполняет этот запрос. Взаимодействие между *клиентом* и *сервером* происходит посредством стандартных специальных протоколов, таких как TCP/IP и z39.50. На самом деле протоколов очень много, они различаются по уровням. Мы рассмотрим только протокол прикладного уровня *HTTP* (чуть позднее), поскольку для решения наших программистских задач нужен только он. А пока вернемся к *клиент-серверной архитектуре* и разберемся, что же такое *клиент* и что такое *сервер*.

Сервер представляет собой набор программ, которые контролируют выполнение различных процессов. Соответственно, этот набор программ установлен на каком-то компьютере. Часто компьютер, на котором установлен *сервер*, и называют

сервером. Основная функция компьютера-сервера – по запросу *клиента* запустить какой-либо определенный процесс и отправить *клиенту* результаты его работы.

Клиентом называют любой процесс, который пользуется услугами *сервера*. *Клиентом* может быть как пользователь, так и программа. Основная задача *клиента* – выполнение приложения и осуществление связи с *сервером*, когда этого требует приложение. То есть *клиент* должен предоставлять пользователю интерфейс для работы с приложением, реализовывать логику его работы и при необходимости отправлять задания *серверу*.

Взаимодействие между *клиентом* и *сервером* начинается по инициативе *клиента*. *Клиент* запрашивает вид обслуживания, устанавливает сеанс, получает нужные ему результаты и сообщает об окончании работы.

Услугами одного *сервера* чаще всего пользуется несколько *клиентов* одновременно. Поэтому каждый *сервер* должен иметь достаточно большую производительность и обеспечивать безопасность данных.

Логичнее всего устанавливать *сервер* на компьютере, входящем в какую-либо сеть, локальную или глобальную. Однако можно устанавливать *сервер* и на отдельно стоящий компьютер (тогда он будет являться одновременно и *клиентом* и *сервером*).

Существует множество типов *серверов*. Вот лишь некоторые из них.

- **Видеосервер**

  Такой *сервер* специально приспособлен к обработке изображений, хранению видеоматериалов, видеоигр и т.п. В связи с этим компьютер, на котором установлен видеосервер, должен иметь высокую производительность и большую память.
- **Поисковый** *сервер* предназначен для поиска информации в Internet.
- **Почтовый** *сервер* предоставляет услуги в ответ на запросы, присланные по электронной почте.
- **Сервер WWW** предназначен для работы в Internet.
- **Сервер баз данных** выполняет обработку запросов к базам данных.
- *Сервер* защиты данных предназначен для обеспечения безопасности данных (содержит, например, средства для идентификации паролей).

- **Сервер приложений** предназначен для выполнения прикладных процессов. С одной стороны взаимодействует с *клиентами*, получая задания, а с другой – работает с базами данных, подбирая необходимые для обработки данные.
- **Сервер удаленного доступа** обеспечивает коллективный удаленный доступ к данным.
- **Файловый сервер** обеспечивает функционирование распределенных ресурсов, предоставляет услуги поиска, хранения, архивирования данных и возможность одновременного доступа к ним нескольких пользователей.

Обычно на компьютере-*сервере* работает сразу несколько программ-*серверов*. Одна занимается электронной почтой, другая распределением файлов, третья предоставляет web-страницы.

Из всех типов *серверов* нас в основном интересует *сервер WWW*. Часто его называют *web-сервером*, *http-сервером* или даже просто *сервером*. Что представляет собой *web-сервер*? Во-первых, это хранилище информационных ресурсов. Во-вторых, эти ресурсы хранятся и предоставляются пользователям в соответствии со стандартами Internet (такими, как протокол передачи данных *HTTP*). Как предоставляются данные в соответствии с этим протоколом, мы рассмотрим чуть позже. Работа с документами *web-сервера* осуществляется при помощи браузера (например, IE, Opera или Mozilla), который отсылает *серверу* запросы, созданные в соответствии с протоколом *HTTP*. В процессе выполнения задания *сервер* может связываться с другими *серверами*.

Далее в ходе лекции, говоря «*сервер*», мы будем подразумевать *web-сервер*.

В качестве примеров *web-серверов* можно привести *сервер Apache* группы *Apache*, Internet Information Server (IIS) компании Microsoft, SunOne фирмы Sun Microsystems,WebLogic фирмы BEA Systems, IAS (Inprise Application Server) фирмы Borland, WebSphere фирмы IBM, OAS (Oracle Application Server).

На рис. 12.1 и в таблице 12.1 приведена статистика использования различных *серверов* среди всех доменов Internet от NetCraft http://news.netcraft.com/.

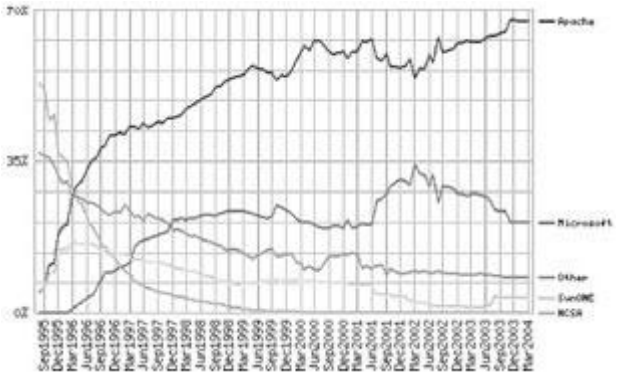

Рис. 12.1. Статистика использования ведущих web-серверов

Таблица 12.1. Ведущие разработчики web-серверов					
Разработчик	**Февраль 2004**	**Проценты**	**Март 2004**	**Проценты**	**Изменение**
Apache	31703884	67.21	32280582	67.20	-0.01
Microsoft	9849971	20.88	10099760	21.02	0.14
SunONE	1657295	3.51	1651575	3.44	-0.07
Zeus	755227	1.60	762716	1.59	-0.01

Как видно из приведенной таблицы, *сервер Apache* занимает лидирующие позиции. Все, что мы когда-либо будем говорить о *web-серверах*, ориентировано на *Apache*, если не указано иное. О том, как установить его на свой компьютер, мы уже рассказывали в самой первой лекции. А теперь, как было обещано, обратимся к протоколу *HTTP*.

Протокол HTTP и способы передачи данных на сервер

Internet построен по многоуровневому принципу, от физического уровня, связанного с физическими аспектами передачи двоичной информации, и до прикладного уровня, обеспечивающего интерфейс между пользователем и сетью.

HTTP (HyperText Transfer Protocol, *протокол передачи гипертекста*) – это протокол прикладного уровня, разработанный для обмена гипертекстовой информацией в Internet.

HTTP предоставляет набор методов для указания целей запроса, отправляемого *серверу*. Эти методы основаны на дисциплине ссылок, где для указания ресурса, к

которому должен быть применен данный метод, используется универсальный идентификатор ресурсов (Universal Resource Identifier) в виде местонахождения ресурса (Universal Resource Locator, ***URL***) или в виде его универсального имени (Universal Resource Name, ***URN***).

Сообщения по сети при использовании протокола *HTTP* передаются в формате, схожем с форматом почтового сообщения Internet (RFC-822) или с форматом сообщений MIME (Multiperposal Internet Mail Exchange).

HTTP используется для коммуникаций между различными пользовательскими программами и программами-шлюзами, предоставляющими доступ к существующим Internet-протоколам, таким как SMTP (протокол электронной почты), NNTP (протокол передачи новостей), FTP (протокол передачи файлов), Gopher и WAIS. *HTTP* разработан для того, чтобы позволять таким шлюзам через промежуточные программы-*серверы* (proxy) передавать данные без потерь.

Протокол реализует *принцип запрос/ответ*. Запрашивающая программа – *клиент* инициирует взаимодействие с отвечающей программой – *сервером* и посылает запрос, содержащий:

- метод доступа;
- адрес URI;
- версию протокола;
- сообщение (похожее по форме на MIME) с информацией о типе передаваемых данных, информацией о *клиенте*, пославшем запрос, и, возможно, с содержательной частью (телом) сообщения.

Ответ *сервера* содержит:

- строку состояния, в которую входит версия протокола и код возврата (успех или ошибка);
- сообщение (в форме, похожей на MIME), в которое входит информация *сервера*, метаинформация (т.е. информация о содержании сообщения) и тело сообщения.

В протоколе не указывается, кто должен открывать и закрывать соединение между *клиентом* и *сервером*. На практике соединение, как правило, открывает *клиент*, а *сервер* после отправки ответа инициирует его разрыв.

Давайте рассмотрим более подробно, в какой форме отправляются запросы на *сервер*.

Форма запроса клиента

Клиент отсылает *серверу* запрос в одной из двух форм: в полной или сокращенной. Запрос в первой форме называется соответственно *полным запросом*, а во второй форме – простым запросом.

Простой запрос содержит метод доступа и адрес ресурса. Формально это можно записать так:

<Простой-Запрос> := <Метод> <символ пробел>
<Запрашиваемый-URI> <символ новой строки>

В качестве метода могут быть указаны *GET*, *POST*, *HEAD*, PUT, DELETE и другие. О наиболее распространенных из них мы поговорим немного позже. В качестве запрашиваемого URI чаще всего используется *URL*-адрес ресурса.

Пример *простого запроса*:

GET http://phpbook.info/

Здесь *GET* – это метод доступа, т.е. метод, который должен быть применен к запрашиваемому ресурсу, а http://phpbook.info/ – это *URL*-адрес запрашиваемого ресурса.

Полный запрос содержит строку состояния, несколько заголовков (заголовок запроса, общий заголовок или заголовок содержания) и, возможно, тело запроса. Формально общий вид *полного запроса* можно записать так:

<Полный запрос> := <Строка Состояния>
(<Общий заголовок>|<Заголовок запроса>|
<Заголовок содержания>)
<символ новой строки>
[<содержание запроса>]

Квадратные скобки здесь обозначают необязательные элементы заголовка, через вертикальную черту перечислены альтернативные варианты. Элемент <Строка состояния> содержит *метод запроса* и URI ресурса (как и *простой запрос*) и, кроме того, используемую версию протокола *HTTP*. Например, для вызова внешней программы можно задействовать следующую строку состояния:

POST http://phpbook.info/cgi-bin/test HTTP/1.0

В данном случае используется метод *POST* и протокол *HTTP* версии 1.0.

В обеих формах запроса важное место занимает URI запрашиваемого ресурса. Чаще всего URI используется в виде *URL*-адреса ресурса. При обращении к *серверу* можно применять как полную форму *URL*, так и упрощенную.

Полная форма содержит тип протокола доступа, адрес *сервера* ресурса и адрес ресурса на *сервере* (рисунок 12.2).

В сокращенной форме опускают протокол и адрес *сервера*, указывая только местоположение ресурса от корня *сервера*. Полную форму используют, если возможна пересылка запроса другому *серверу*. Если же работа происходит только с одним *сервером*, то чаще применяют сокращенную форму.

Рис.12.2. Полная форма URL

Далее мы рассмотрим наиболее распространенные *методы отправки запросов*.

Методы

Как уже говорилось, любой запрос *клиента* к *серверу* должен начинаться с указания метода. Метод сообщает о цели запроса *клиента*. Протокол *HTTP* поддерживает достаточно много методов, но реально используются только три: *POST*, *GET* и *HEAD*. Метод *GET* позволяет получить любые данные, идентифицированные с помощью URI в запросе ресурса. Если URI указывает на программу, то возвращается результат работы программы, а не ее текст (если, конечно, текст не есть результат ее работы). Дополнительная информация, необходимая для обработки запроса, встраивается в сам запрос (в строку статуса). При использовании метода *GET* в поле тела ресурса возвращается собственно затребованная информация (текст HTML-документа, например).

Существует разновидность метода *GET* – условный *GET*. Этот метод сообщает *серверу* о том, что на запрос нужно ответить, только если выполнено условие, содержащееся в поле if-Modified-Since заголовка запроса. Если говорить более

точно, то тело ресурса передается в ответ на запрос, если этот ресурс изменялся после даты, указанной в if-Modified-Since.

Метод *HEAD* аналогичен методу *GET*, только не возвращает тело ресурса и не имеет условного аналога. Метод *HEAD* используют для получения информации о ресурсе. Это может пригодиться, например, при решении задачи тестирования гипертекстовых ссылок.

Метод *POST* разработан для передачи на *сервер* такой информации, как аннотации ресурсов, новостные и почтовые сообщения, данные для добавления в базу данных, т.е. для передачи информации большого объема и достаточно важной. В отличие от методов *GET* и *HEAD*, в *POST* передается тело ресурса, которое и является информацией, получаемой из полей форм или других источников ввода.

До сих пор мы только теоретизировали, знакомились с основными понятиями. Теперь пора научиться использовать все это на практике. Далее в лекции мы рассмотрим, как посылать запросы *серверу* и как обрабатывать его ответы.

Использование HTML-форм для передачи данных на сервер

Как передавать данные *серверу*? Для этого в языке HTML есть специальная конструкция – формы. Формы предназначены для того чтобы получать от пользователя информацию. Например, вам нужно знать логин и пароль пользователя для того, чтобы определить, на какие страницы сайта его можно допускать. Или вам необходимы личные данные пользователя, чтобы была возможность с ним связаться. Формы как раз и применяются для ввода такой информации. В них можно вводить текст или выбирать подходящие варианты из списка. Данные, записанные в форму, отправляются для обработки специальной программе (например, скрипту на PHP) на *сервере*. В зависимости от введенных пользователем данных эта программа может формировать различные web-страницы, отправлять запросы к базе данных, запускать различные приложения и т.п.

Разберемся с синтаксисом *HTML-форм*. Возможно, многие с ним знакомы, но мы все же повторим основные моменты, поскольку это важно.

Итак, для создания формы в языке HTML используется тег *FORM*. Внутри него находится одна или несколько команд *INPUT*. С помощью атрибутов *action* и

method тега *FORM* задаются имя программы, которая будет обрабатывать данные формы, и метод запроса, соответственно. Команда *INPUT* определяет тип и различные характеристики запрашиваемой информации. Отправка данных формы происходит после нажатия кнопки *input* типа *submit*. Создадим форму для регистрации участников заочной школы программирования.

Листинг 4.0. form.html (html, txt)

После обработки браузером этот файл будет выглядеть примерно так:

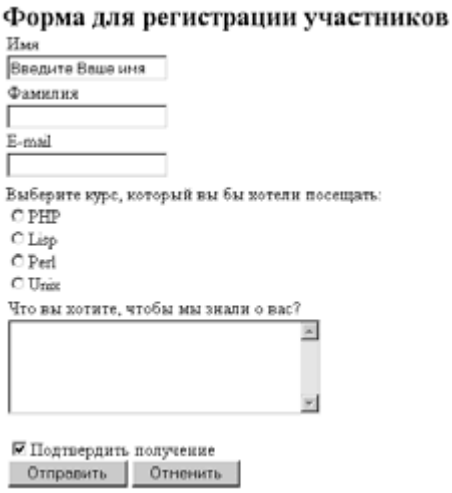

Рис. 4.3. Пример html-формы

Вот так создаются и выглядят *HTML-формы*. Будем считать, что мы научились или вспомнили, как их создавать. Как мы видим, в форме можно указывать метод передачи данных. Посмотрим, что будет происходить, если указать метод *GET* или *POST*, и в чем будет разница.

Для метода GET

При отправке данных формы с помощью метода *GET* содержимое формы добавляется к *URL* после знака вопроса в виде пар имя=значения, объединенных с помощью амперсанта &:

action?name1=value1&name2=value2&name3=value3

Здесь *action* – это *URL*-адрес программы, которая должна обрабатывать форму (это либо программа, заданная в атрибуте *action* тега *form*, либо сама текущая программа, если этот атрибут опущен). Имена name1, name2, name3 соответствуют именам элементов формы, а value1, value2, value3 – значениям этих элементов. Все специальные символы, включая = и &, в именах или значениях этих параметров

будут опущены. Поэтому не стоит использовать в названиях или значениях элементов формы эти символы и символы кириллицы в идентификаторах.

Если в поле для ввода ввести какой-нибудь служебный символ, то он будет передан в его шестнадцатеричном коде, например, символ $ заменится на %24. Также передаются и русские буквы.

Для полей ввода текста и пароля (это элементы *input* с атрибутом type=text и type=password), значением будет то, что введет пользователь. Если пользователь ничего не вводит в такое поле, то в строке запроса будет присутствовать элемент name=, где name соответствует имени этого элемента формы.

Для кнопок типа checkbox и radio button значение value определяется атрибутом VALUE в том случае, когда кнопка отмечена. Не отмеченные кнопки при составлении строки запроса игнорируются целиком. Несколько кнопок типа checkbox могут иметь один атрибут NAME (и различные VALUE), если это необходимо. Кнопки типа radio button предназначены для одного из всех предложенных вариантов и поэтому должны иметь одинаковый атрибут NAME и различные атрибуты VALUE.

В принципе создавать *HTML-форму* для передачи данных методом *GET* не обязательно. Можно просто добавить в строку *URL* нужные переменные и их значения.

http://phpbook.info/test.php?id=10&user=pit

В связи с этим у передачи данных методом *GET* есть один существенный недостаток – любой может подделать значения параметров. Поэтому не советуем использовать этот метод для доступа к защищенным паролем страницам, для передачи информации, влияющей на безопасность работы программы или *сервера*. Кроме того, не стоит применять метод *GET* для передачи информации, которую не разрешено изменять пользователю.

Несмотря на все эти недостатки, использовать метод *GET* достаточно удобно при отладке скриптов (тогда можно видеть значения и имена передаваемых переменных) и для передачи параметров, не влияющих на безопасность.

Для метода POST

Содержимое формы кодируется точно так же, как для метода *GET* (см. выше), но вместо добавления строки к *URL* содержимое запроса посылается блоком данных как часть операции *POST*. Если присутствует атрибут ACTION, то значение *URL*, которое там находится, определяет, куда посылать этот блок данных. Этот метод, как уже отмечалось, рекомендуется для передачи больших по объему блоков данных.

Информация, введенная пользователем и отправленная *серверу* с помощью метода *POST*, подается на стандартный ввод программе, указанной в атрибуте *action*, или текущему скрипту, если этот атрибут опущен. Длина посылаемого файла передается в *переменной окружения* CONTENT_LENGTH, а тип данных – в переменной CONTENT_TYPE.

Передать данные методом *POST* можно только с помощью *HTML-формы*, поскольку данные передаются в теле запроса, а не в заголовке, как в *GET*. Соответственно и изменить значение параметров можно, только изменив значение, введенное в форму. При использовании *POST* пользователь не видит передаваемые *серверу* данные.

Основное преимущество *POST* запросов – это их большая безопасность и функциональность по сравнению с GET-запросами. Поэтому метод *POST* чаще используют для передачи важной информации, а также информации большого объема. Тем не менее не стоит целиком полагаться на безопасность этого механизма, поскольку данные *POST* запроса также можно подделать, например создав html-файл на своей машине и заполнив его нужными данными. Кроме того, не все *клиенты* могут применять метод *POST*, что ограничивает варианты его использования.

При отправке данных на *сервер* любым методом передаются не только сами данные, введенные пользователем, но и ряд переменных, называемых *переменными окружения*, характеризующих *клиента*, историю его работы, пути к файлам и т.п. Вот некоторые из *переменных окружения*:

- ***REMOTE_ADDR*** – IP-адрес хоста (компьютера), отправляющего запрос;
- ***REMOTE_HOST*** – имя хоста, с которого отправлен запрос;

- ***HTTP_REFERER*** – адрес страницы, ссылающейся на текущий скрипт;
- ***REQUEST_METHOD*** – метод, который был использован при отправке запроса;
- ***QUERY_STRING*** – информация, находящаяся в *URL* после знака вопроса;
- ***SCRIPT_NAME*** – виртуальный путь к программе, которая должна выполняться;
- ***HTTP_USER_AGENT*** – информация о браузере, который использует *клиент*

Обработка запросов с помощью PHP

До сих пор мы упоминали только, что запросы *клиента* обрабатываются на *сервере* с помощью специальной программы. На самом деле эту программу мы можем написать сами, в том числе и на языке PHP, и она будет делать с полученными данными все, что мы захотим. Для того чтобы написать эту программу, необходимо познакомиться с некоторыми правилами и инструментами, предлагаемыми для этих целей PHP.

Внутри PHP-скрипта существует несколько способов получения доступа к данным, переданным *клиентом* по протоколу *HTTP*. До версии PHP 4.1.0 доступ к таким данным осуществлялся по именам переданных переменных (напомним, что данные передаются в виде пар «имя переменной, символ «=», значение переменной»). Таким образом, если, например, было передано first_name=Nina, то внутри скрипта появлялась переменная $first_name со значением Nina. Если требовалось различать, каким методом были переданы данные, то использовались ассоциативные массивы *$HTTP_POST_VARS* и *$HTTP_GET_VARS*, ключами которых являлись имена переданных переменных, а значениями – соответственно значения этих переменных. Таким образом, если пара first_name=Nina передана методом *GET*, то $HTTP_GET_VARS["first_name"]="Nina".

Использовать в программе имена переданных переменных напрямую небезопасно. Поэтому было решено начиная с PHP 4.1.0 задействовать для обращения к переменным, переданным с помощью HTTP-запросов, специальный массив – *$_REQUEST*. Этот массив содержит данные, переданные методами *POST* и *GET*, а также с помощью *HTTP* cookies. Это суперглобальный ассоциативный массив, т.е.

его значения можно получить в любом месте программы, используя в качестве ключа имя соответствующей переменной (элемента формы).

Пример 4.2. Допустим, мы создали форму для регистрации участников заочной школы программирования, как в приведенном выше примере. Тогда в файле 1.php, обрабатывающем эту форму, можно написать следующее:

```
<?php
$str = "Здравствуйте,
   ".$_REQUEST["first_name"]. "
   ".$_REQUEST["last_name"]."! <br>";
$str .="Вы выбрали для изучения курс по
   ".$_REQUEST["kurs"];
echo $str;
?>
```

Тогда, если в форму мы ввели имя «Вася», фамилию «Петров» и выбрали среди всех курсов курс по PHP, на экране браузера получим такое сообщение:

```
Здравствуйте, Вася Петров!
Вы выбрали для изучения курс по PHP
```

После введения массива *$_REQUEST* массивы *$HTTP_POST_VARS* и *$HTTP_GET_VARS* для однородности были переименованы в *$_POST* и *$_GET* соответственно, но сами они из обихода не исчезли из соображений совместимости с предыдущими версиями PHP. В отличие от своих предшественников, массивы *$_POST* и *$_GET* стали суперглобальными, т.е. доступными напрямую и внутри функций и методов.

Приведем пример использования этих массивов. Допустим, нам нужно обработать форму, содержащую элементы ввода с именами first_name, last_name, kurs (например, форму form.html, приведенную выше). Данные были переданы методом *POST*, и данные, переданные другими методами, мы обрабатывать не хотим. Это можно сделать следующим образом:

```
<?php
$str = "Здравствуйте,
   ".$_POST ["first_name"]."
```

```
    ".$_POST ["last_name"] ."! <br>";
$str .= "Вы выбрали для изучения курс по ".
        $_POST["kurs"];
echo $str;
?>
```

Тогда на экране браузера, если мы ввели имя «Вася», фамилию «Петров» и выбрали среди всех курсов курс по PHP, увидим сообщение, как в предыдущем примере:

Здравствуйте, Вася Петров!

Вы выбрали для изучения курс по PHP

Для того чтобы сохранить возможность обработки скриптов более ранних версий, чем PHP 4.1.0, была введена директива *register_globals*, разрешающая или запрещающая доступ к переменным непосредственно по их именам. Если в файле настроек PHP параметр register_globals=On, то к переменным, переданным *серверу* методами *GET* и *POST*, можно обращаться просто по их именам (т.е. можно писать $first_name). Если же register_globals=Off, то нужно писать $_REQUEST["first_name"] или $_POST["first_name"], $_GET["first_name"], $HTTP_POST_VARS["first_name"], $HTTP_GET_VARS["first_name"]. С точки зрения безопасности эту директиву лучше отключать (т.е. register_globals=Off). При включенной директиве *register_globals* перечисленные выше массивы также будут содержать данные, переданные *клиентом*.

Иногда возникает необходимость узнать значение какой-либо *переменной окружения*, например метод, использовавшийся при передаче запроса или IP-адрес компьютера, отправившего запрос. Получить такую информацию можно с помощью функции *getenv()*. Она возвращает значение *переменной окружения*, имя которой передано ей в качестве параметра.

```
<?
getenv("REQUEST_METHOD");
    // возвратит использованный метод
echo getenv ("REMOTE_ADDR");
    // выведет IP-адрес пользователя,
```

```
    // пославшего запрос
?>
```

Как мы уже говорили, если используется метод *GET*, то данные передаются добавлением строки запроса в виде пар «имя_переменной=значение к *URL*-адресу ресурса». Все, что записано в *URL* после знака вопроса, можно получить с помощью команды

```
getenv("QUERY_STRING");
```

Благодаря этому можно по методу *GET* передавать данные в каком-нибудь другом виде. Например, указывать только значения нескольких параметров через знак плюс, а в скрипте разбирать строку запроса на части или можно передавать значение всего одного параметра. В этом случае в массиве *$_GET* появится пустой элемент с ключом, равным этому значению (всей строке запроса), причем символ «+», встретившийся в строке запроса, будет заменен на подчеркивание «_».

Методом *POST* данные передаются только с помощью форм, и пользователь (*клиент*) не видит, какие именно данные отправляются *серверу*. Чтобы их увидеть, хакер должен подменить нашу форму своей. Тогда *сервер* отправит результаты обработки неправильной формы не туда, куда нужно. Чтобы этого избежать, можно проверять адрес страницы, с которой были посланы данные. Это можно сделать опять же с помощью функции *getenv()*:

```
getenv("HTTP_REFERER");
```

Теперь самое время решить задачу, сформулированную в начале лекции.

Пример обработки запроса с помощью PHP

Напомним, в чем состояла задача, и уточним ее формулировку. Нужно написать форму для регистрации участников заочной школы программирования и после регистрации отправить участнику сообщение. Мы назвали это сообщение универсальным письмом, но оно будет немного отличаться от того письма, которое мы составили на предыдущей лекции. Здесь мы также не будем отправлять что-либо по электронной почте, дабы не уподобляться спамерам, а просто сгенерируем это сообщение и выведем его на экран браузера. Начальный вариант формы регистрации мы уже приводили выше. Изменим его таким образом, чтобы каждый

регистрирующийся мог выбрать сколько угодно курсов для посещения, и не будем подтверждать получение регистрационной формы.

Здесь все достаточно просто и понятно. Единственное, что можно отметить, – это способ передачи значений элемента checkbox. Когда мы пишем в имени элемента kurs[], это значит, что первый отмеченный элемент checkbox будет записан в первый элемент массива kurs, второй отмеченный checkbox – во второй элемент массива и т.д. Можно, конечно, просто дать разные имена элементам checkbox, но это усложнит обработку данных, если курсов будет много.

Скрипт, который все это будет разбирать и обрабатывать, называется 1.php (форма ссылается именно на этот файл, что записано в ее атрибуте *action*). По умолчанию используется для передачи метод *GET*, но мы указали *POST*. По полученным сведениям от зарегистрировавшегося человека, скрипт генерирует соответствующее сообщение. Если человек выбрал какие-то курсы, то ему выводится сообщение о времени их проведения и о лекторах, которые их читают. Если человек ничего не выбрал, то выводится сообщение о следующем собрании заочной школы программистов (ЗШП).

Заключение

Подведем итоги. Мы научились отличать *клиента* от *сервера* и компьютер-*сервер* от программы-*сервера*, познакомились с основными методами, используемыми для передачи данных на *сервер*, изучили средства, предлагаемые языком PHP для обработки клиентских запросов, и испробовали их на практике. В принципе этого достаточно для того, чтобы создавать клиент-серверные приложения.