


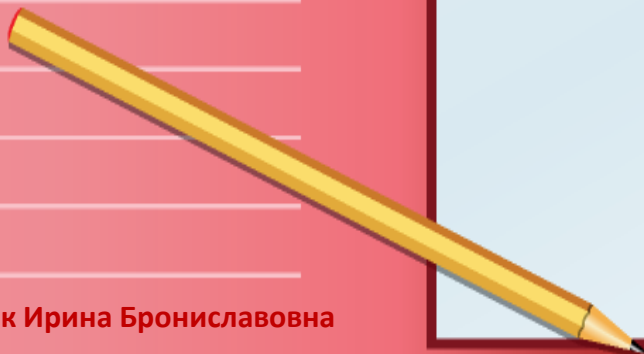


Эпиграф



**Мы нуждаемся в лучшем аппаратном
обеспечении для настольных прикладных
систем, лучшем программном
обеспечении и коммуникационной
инфраструктуре и, что наиболее важно,
лучшей отдаче от тех людей, кто
занимается тренировкой и
консультациями, кто строит из стандартных
блоков
путь к пониманию для самых разных
пользователей.**


Билл Гейтс (Bill Gates)





Глава 12

«Разработка интерфейсов информационного взаимодействия»



Тема 28. Программно- аппаратные интерфейсы.


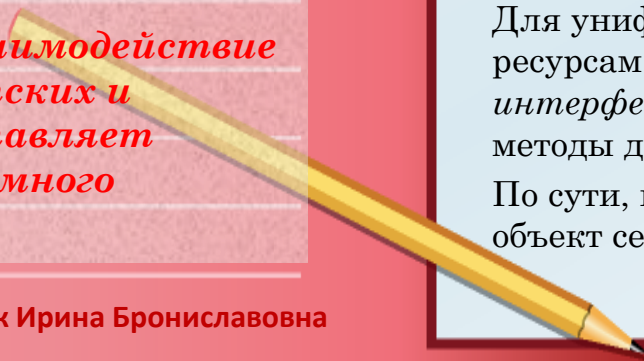
1. Интерфейсы взаимодействия уровня приложений.
2. Реализация интерфейса взаимодействия WEB - приложений.



Интерфейсы взаимодействия уровня приложений

Рассмотрим интерфейс взаимодействия двух приложений WEB-броузера и WEB-сервера (поскольку эти приложения обычно располагаются на разных машинах и, соответственно, на разных программно-аппаратных платформах используем термин *программно-аппаратный интерфейс*).

Согласованное взаимодействие объектов (клиентских и серверных) и составляет понятие программного интерфейса.



При реализации интерфейса взаимодействия WEB - приложений используется протокол **HTTP** (*HypertextTransferProtocol* - протокол передачи гипертекста), который представляет собой протокол прикладного уровня и обеспечивает возможность доступа к разнообразной информации, размещенной в сети WWW- WorldWideWeb.

Способность хранить и представлять данные разнообразных форматов (изображения, видео, аудио) делает сеть WWW с используемым HTTP уникальным средством размещения информации.


В настоящее время протокол HTTP используется системой WWW качестве одного из основных протоколов обладающего высокопроизводительными механизмами тиражирования информации, независимо от типа представления данных.

Протокол построен по объектно-ориентированной технологии и может использоваться для решения различных задач, (*например, управления распределенными информационными системами*).

Протокол HTTP позволяет получать доступ к информационным ресурсам и сервисам WWW-серверов.

Для унификации доступа к многофункциональным ресурсам сети WWW-серверы поддерживают комплекс интерфейсов, позволяющих структурировать уровни и методы доступа к сетевым ресурсам.

По сути, каждый из интерфейсов представляет собой объект сети со своими методами и своей структурой.



Интерфейсы взаимодействия уровня приложений

Рассмотрим составляющие программно-аппаратных интерфейсов на основе протоколов уровня приложений.

URI (*UniformResourceIdentifier*, Идентификатор ресурса),

URL (*UniformResourceLocator*, Местонахождение ресурса),


URN (*UniformResourceName*, Имя ресурса) – разные аспекты идентификации одного и того же сервиса, определяющие тип, метод доступа и расположение узла сети, на котором находятся ресурс, доступный через сеть Интернет.

Этот сервис состоит из трех частей.

- 1) Схема. Идентифицирует тип сервиса, через который можно получить доступ к сервису, например WWW-сервер.
- 2) Адрес. Идентифицирует адрес (хост) ресурса, например, www.ripn.net.
- 3) Имя или путь доступа. Идентифицирует полный путь к ресурсу на выбранном хосте, который мы хотим использовать для доступа к ресурсу, например, [/home/images/image1.gif](http://home/images/image1.gif).



Интерфейсы взаимодействия уровня приложений




Например, файл readme.txt, расположенный на сайте Microsoft (WWW-сервере), представляет собой ресурс с идентификатором: <http://www.microsoft.com/readme.txt> .

Это означает:


- ☐ для обращения к ресурсу должен использоваться протокол HTTP, (схема доступа отделена двоеточием ":" и указывает название использованного протокола),
- ☐ следующие два слэша отделяют адрес сервера www.microsoft.com, а также имя файла /readme.txt.

Как правило, когда имеют в виду компьютер, на котором расположен ресурс, используют значение URL или URN, а когда обозначают ресурс полностью (тип, хост, путь) используют URI. Нет ошибки, если используется одно обозначение вместо другого, но обязательно следует пояснить, что оно значит в контексте.





Интерфейсы взаимодействия уровня приложений



Идентификатор URI может содержать не только имя ресурса, но и параметры, необходимые для его представления.

Имя ресурса отделяется от строки параметров символом "?".

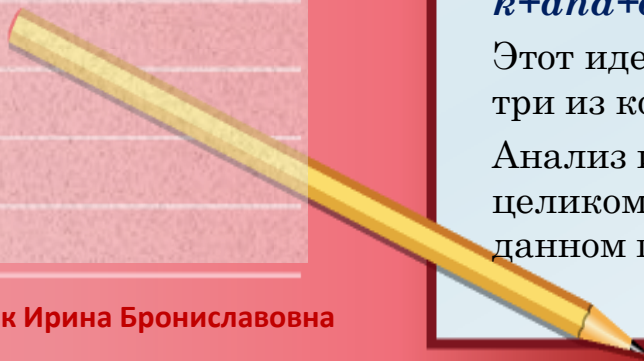
Строка параметров состоит из символьных групп с постоянной структурой (лексем), разделяемых символом "&", каждая такая лексема состоит из имени параметра и его значения, разделенных символом "=", символ пробела " " заменяется знаком "+".

Символы лексем, не входящие в набор символов ASCII, заменяются знаком "%" и шестнадцатеричным значением этого символа.

Для указанного ресурса вся строка параметров является одним строковым параметром, поэтому тип, порядок следования или уникальность имен отдельных параметров строки не существенны. Например:


<http://www.exe.com/bm/scrshell.run?in=10&go=ok+and+ok&event=1&event=2>

Этот идентификатор URI содержит 4 параметра, три из которых – числовые, а два имеют одно имя. Анализ и разбор значений отдельных параметров целиком возлагается на идентификатор URI, в данном примере на ресурс scrshell.run.





Реализация интерфейса взаимодействия WEB - приложений



Hyper Text Markup Language (HTML) - это язык описания информации, хранимой в сети WWW.

HTML-файл может содержать специальные коды, обозначающие присоединенную графическую, видео или аудио информацию или исполняемые коды среды отображения информации (Web-браузер - JavaScript, Java).

Для языков Java и JavaScript приложение Web-браузер представляет операционную систему или среду, в которой они выполняются, а Web-страница является ресурсом, выделенным для их работы.


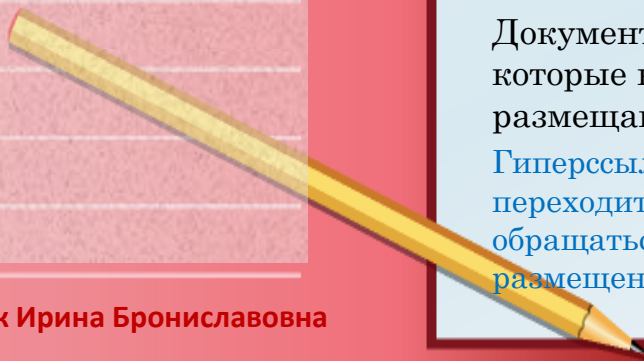
Эти языки не строят Web-страницу по данным пользователя, а используют ее как платформу для своих действий и действий пользователя.

Когда Web-браузер получает доступ к этому файлу, он сначала интерпретирует закодированную в HTML-файле информацию, а затем в соответствующей форме представляет эту информацию пользователю.

Буквы "HT" в названии протокола HTML обозначают "HyperText" – основную концепцию размещения информации в сети WWW.

Документы HyperText содержат специальные связи, которые называются *гиперссылками* (hyperlinks) и размещаются в тексте документа.

Гиперссылки позволяют пользователю не только переходить от одной части этого документа к другой, но и обращаться к другим связанным документам, размещенным в сети WWW.



Реализация интерфейса взаимодействия WEB - приложений

В настоящее время широко используется технология активных серверных страниц ASP (ActiveServerPages). Эта технология представляет применение того же самого стандарта CGI, только на уровне объектно-ориентированного подхода к построению Web-страниц.

Common Gateway Interface (CGI) – это стандарт расширения функциональности WWW, позволяющий WWW-серверам выполнять программы, аргументы которых может определять пользователь.

Интерфейс CGI расширяет возможности пользователя и позволяет ему выполнять программы, ассоциированные с данной Web-страницей, предоставляя таким образом возможность получения динамической информации из WWW-сервера.

Например, пользователь такого WWW-сервера может получить самую последнюю информацию о погоде, выполнив программу, которая запрашивает прогноз погоды на текущий момент из базы данных.

Интерфейс CGI в основном играет роль шлюза между WWW-сервером и внешними исполняемыми программами. Он получает запрос от пользователя, передает его внешней программе и затем возвращает результаты пользователю через построенную динамически Web-страницу.

При этом построенные Web-страницы могут коренным образом отличаться друг от друга, поскольку они формируются в прямой зависимости от параметров, определяемых пользователем.


Механизм интерфейса CGI также является универсальным и может передавать данные между любыми WWW-серверами. Так как CGI основан на исполняемых файлах, нет ограничений на тип программы, которая будет в нем исполняться. Программа может быть написана на любом из языков программирования, позволяющих создавать исполняемые модули.

CGI-программа также может быть написана с использованием командных языков операционных систем, таких как Perl или Shell.




Глава 12

«Разработка интерфейсов информационного взаимодействия»



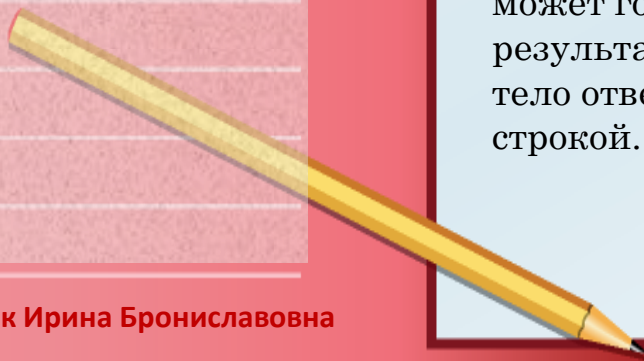
Тема 29. Интерфейс информационного взаимодействия программных приложений.

1. Интерфейс взаимодействия программных приложений на примере HTTP.
2. Передача запросов и ответов.



Интерфейс взаимодействия программных приложений на примере HTTP

Рассмотрим интерфейс
взаимодействия
программных
приложений на примере
HTTP.





Интерфейс реализуется последовательно.

Первый этап – это когда HTTP-клиент (браузер) соединяется с сервером. Для этого он использует протокол TCP/IP, и соединение происходит с известным клиенту TCP-портом. Принятый номер порта HTTP - 80; для других сервисов определены другие TCP-порты.


Вторым этапом является запрос клиента: клиент передает заголовок запроса (Requestheader) и, возможно (в зависимости от метода), тело сообщения запроса. В заголовке обязательно указываются метод, URL и версия HTTP. Там может быть еще несколько необязательных полей, которые тоже дают серверу информацию о том, как обрабатывать запрос.

Третий этап – ответ сервера, который состоит из заголовка (Responseheader), в котором сервер указывает версию HTTP и код статуса, который может говорить об успешном или неуспешном результате и его причинах. После заголовка идет тело ответа, отделенное от заголовка пустой строкой.





Интерфейс взаимодействия программных приложений на примере HTTP



Четвертым этапом является разрыв TCP/IP соединения.

Requestheader может выглядеть следующим образом:

GET /MyDoc.htm HTTP/1.1

Connection: Keep-Alive

User-Agent: Mozilla/3.0 (Win95; I)

Host: 212.54.196.226

*Accept: image/gif, image/x-bitmap,
image/jpeg, *.**

Здесь:

MyDoc.htm – имя запрашиваемого документа;

GET – тип запроса;

Host – IP-адрес;

Асепт – форматы данных "понимаемых" клиентом.

Интерфейс взаимодействия программных приложений на примере HTTP

Requestheader, приведенный ниже, получен от документа, содержащего форму:

POST /Scripts/ReadData.pl HTTP/1.1

Referer: http://212.54.196.226

Connection: Keep Alive

User-Agent: Mozilla/3.0 (Win95; I)

Host: 212.54.196.226

*Accept: image/gif, image/x-bitmap,
image/jpeg, *.**

Content-type: application/x-www-form-urlencoded

Content-length: 38

FirstName=Mary+Ann&LastName=Sylvester

Здесь:

POST – метод передачи данных из формы; Referer – адрес web-страницы, с которой пользователь перешел на документ, содержащий форму;

Content-type – способ кодировки передаваемых данных;

Content-length – количество передаваемых данных (байт); FirstName,

LastName – имена полей формы; Mary+Ann, Sylvester – передаваемые значения (пробел заменен знаком "+").

Интерфейс взаимодействия программных приложений на примере HTTP

Web-сервер отвечает на запрос браузера, посылая ему HTML-файл, которому предшествует Responseheader.

Типичный *Responseheader* содержит следующие данные:

HTTP/1.1 200 OK

Server: Microsoft-IIS/4.0

Date: Tue, 04 Apr 2005 00:26:34 GMT

Content-type: text/html

Set-Cookie:

*ASPSESSIONIDFFFYXKFR=ACMNFLJANKGBAMPB
EGNGLEAB*

<HTML>

{ HTML - код }

Этот заголовок сформирован сервером.

Строка "*200 OK*" – это статус запроса. Если бы сервер не смог обработать запрос, то он сформировал бы сообщение об ошибке, например, "*404 ObjectNotFound*".

Content-type – тип содержимого. Браузер отображает документ (интерпретирует его код именно как HTML-код, поскольку *Content-type* имеет значение *text/html*) и ждет, когда клиент запросит (щелкнув по гиперссылке) очередную страницу этого сайта или перейдет на другой сайт. Если страница содержит изображение (например, формата *jpeg*), оно будет направлено web-сервером клиенту вместе с другим *Responseheader*, где *Content-type* будет иметь значение *image/jpeg*.

Set-Cookie – устанавливает значение специальной информации записываемой на компьютере клиента. В этом поле хранится идентификатор текущей сессии.

Передача запросов и ответов

Рассмотрим пример и разберём подробнее HTTP запрос клиента. Он может выглядеть например так:

POST http: //localhost/ HTTP/1.1

*Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, */**

Accept-Language: ru

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: localhost

Proxy-Connection: Keep-Alive

param1=1¶m2=2

Из примера видно, что запрос начинается со слова "POST". Это слово означает метод передачи данных на сервер, в котором дополнительные данные запроса (*строка "param1=1¶m2=2"*) передаются после заголовка.

Передача запросов и ответов

В HTML документах метод передачи данных указывается в форме отправки сообщений. Например, для того, чтобы получить этот запрос, была использована следующая форма:

```
<form action="http://localhost/"  
method="post">
```

```
<input type="hidden" name="param1"  
value="1">
```

```
<input type="hidden" name="param2"  
value="2">
```

```
<input type="submit"></form>
```

Как видно из примера, параметры записываются в виде

*[имя параметра1]=[значения
параметра1]&[имя параметра2]=[значения
параметра2] & ...*

Передача запросов и ответов

Часто употребим метод запроса – "GET". Фактически все запросы, не требующие отправки данных – например запрос страницы, производятся этим способом. Изменим форму запроса:

```
<form action="http://localhost/" method="get">  
<input type="hidden" name="param1" value="1">  
<input type="hidden" name="param2" value="2">  
<input type="submit"></form>
```

Получим следующий HTTP запрос:

GET http://localhost/?param1=1¶m2=2 HTTP/1.1

*Accept: image/gif, image/x-bitmap, image/jpeg, image/pjpeg, */**

Accept-Language: ru

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

Host: localhost

Proxy-Connection: Keep-Alive

Как видно, строка "*param1=1¶m2=2*" переместилась выше и добавилась к строке "*http://localhost/*" после знака "?". Так же изменилось первое слово в HTTP заголовке, остальное осталось без изменения.


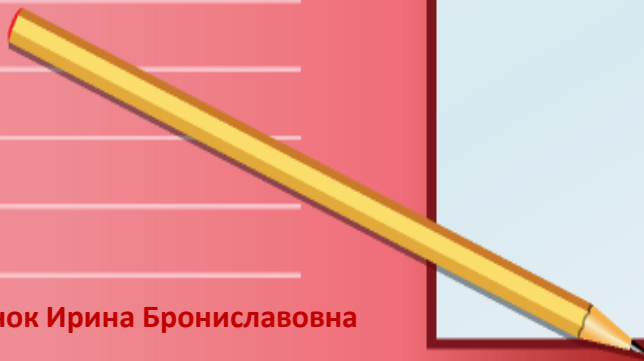
Достоинством метода GET является то, что в строке браузера видно, какие данные были отправлены.

Недостаток – длина отправляемых данных таким способом (в отличие от метода POST) ограничена – некоторые серверы, как и некоторые браузеры, имеют лимит на длину адреса запрашиваемого документа. Соответственно адрес с длинной строкой запроса может быть либо срезан, либо сервер возвратит ошибку "*414 Request-URI TooLong*".



Глава 12

«Разработка интерфейсов информационного взаимодействия»



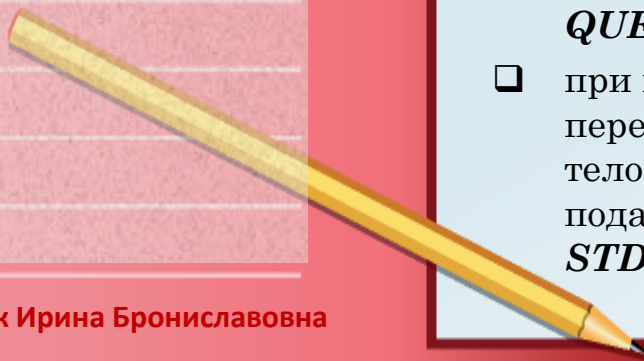
Тема 30. Реализация интерфейса
информационного взаимодействия
программных приложений.

1. Интерфейс взаимодействия сервера
с внешней программой.



Интерфейс взаимодействия сервера с внешней программой

Рассмотрим реализацию
интерфейса взаимодействия
сервера с внешней программой
(CommonGatewayInterface).



Переменные среды Common Gateway Interface (CGI).

Сервер при запуске CGI-скрипта (внешней программы) формирует среду окружения, в которой скрипт может найти всю доступную информацию о HTTP-соединении и о полученных в запросе параметрах.

Большинство переменных CGI стандартизованы.

Рассмотрим стандартные переменные CGI-окружения:

REQUEST_METHOD - это одно из самых главных полей, используемое для определения метода запроса HTTP.

Протокол HTTP использует для запроса к серверу методы GET и POST.

Отличие:

- ☐ в методе GET запрос является частью URL (например <http://www.localhost/myscript.cgi?a=request>), (для CGI: при GET запрос идет в переменную *QUERY_STRING*);
- ☐ при использовании метода POST данные передаются в теле HTTP-запроса (при GET тело запроса пусто), (для CGI: при POST подается на стандартный ввод скрипта – *STDIN*).



Интерфейс взаимодействия сервера с внешней программой



*Пример: **REQUEST_METHOD=GET***

QUERY_STRING– эта строка запроса при методе GET.


Запрос, отправляемый из формы, кодируется браузером, поскольку не все символы разрешены в URL (некоторые из них имеют специальное назначение). В методе `urlencode`: все пробелы заменяются в URL на знак “+”, а все специальные и непечатаемые символы на последовательность `%hh`, где `hh`– шестнадцатеричный код символа. Кроме того, разделителем полей формы является знак `&`, поэтому при обработке форм надо производить декодирование.

*Пример: **QUERY_STRING=***

name=user+chef&age=20&hobby=games

CONTENT_LENGTH – длина тела запроса в байтах. При методе запроса POST необходимо считать со стандартного входа (**STDIN**)

CONTENT_LENGTH – байт, а потом производить их обработку. Обычно методом POST пользуются для передачи форм, содержащих потенциально большие области ввода текста. При этом методе нет никаких ограничений, а при методе GET существуют ограничения на длину URL.





Интерфейс взаимодействия сервера с внешней программой



*Пример: **CONTENT_LENGTH=31***

CONTENT_TYPE – тип тела запроса (для форм, кодированных вышеуказанным образом, тип определяется как application/x-www-form-urlencoded).

GATEWAY_INTERFACE – версия протокола CGI.

*Пример: **GATEWAY_INTERFACE=CGI/1.1***

REMOTE_ADDR – IP-адрес удаленного хоста, делающего данный запрос.

*Пример: **REMOTE_ADDR=139.142.24.157***


REMOTE_HOST – это, если запрашивающий хост имеет доменное имя, то эта переменная содержит его, в противном случае, это тот же самый IP-адрес, что и **REMOTE_ADDR**.

*Пример: **REMOTE_HOST=idsoftware.com***

SCRIPT_NAME – имя скрипта (виртуальное), использованное в запросе. Для получения реального пути на сервере используется переменная **SCRIPT_FILENAME**.



Интерфейс взаимодействия сервера с внешней программой



Пример: ***SCRIPT_NAME=/cgi/guestbook.cgi***
SCRIPT_FILENAME – имя файла скрипта на сервере. Используя эту переменную, вы получаете именно физический путь к файлу.

Пример:
SCRIPT_FILENAME=/home/public/cgi/guestbook.cgi
SERVER_NAME – имя сервера, чаще всего доменное (такое как `www.microsoft.com`), но в редких случаях, за неимением такового оно может быть IP-адресом (типа `157.151.74.254`).

Пример: ***SERVER_NAME=www.tksite.edu***
SERVER_PORT – TCP-порт сервера, используемый для соединения.

По умолчанию HTTP-порт имеет номер 80, хотя в некоторых случаях он может быть и другим.

Пример: ***SERVER_PORT=80***
SERVER_PROTOCOL – версия протокола сервера.


Пример: ***SERVER_PROTOCOL=HTTP/1.1***
SERVER_SOFTWARE – программное обеспечение сервера.

Пример: ***Apache/1.0***





Интерфейс взаимодействия сервера с внешней программой



Основная информация о взаимодействии клиента и сервера может быть получена *из стандартных переменных окружения*.

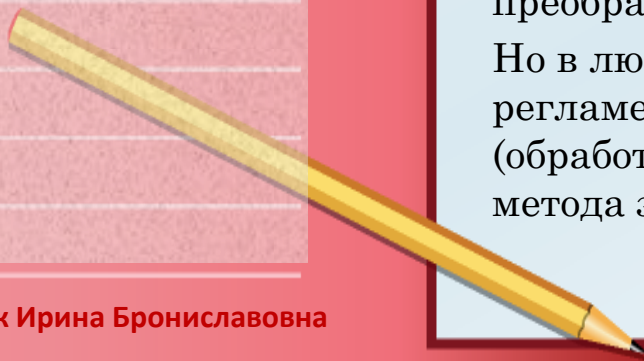
При каждом запуске CGI-сценарий работает так, как будто он прежде никогда не запускался. Программа сценария не переходит в состояние ожидания между отдельными вызовами. Каждый раз процедура инициализации запускается с самого начала.

До считывания входных данных необходимо прочесть значение переменной окружения ***REQUEST_METHOD*** и определить какую информацию она содержит get или post.

Дальнейшие действия зависят от этого значения.

Обработка входных данных зависит от целей написания конкретного сценария. В общем случае обработка данных означает их преобразование.

Но в любом случае есть строго регламентированная часть обработки (обработка входных данных), зависящая от метода запроса.



Интерфейс взаимодействия сервера с внешней программой

Обработка входных данных для метода *get*

Алгоритм обработки входных данных для метода *get* состоит из следующих шагов:

- ❑ Определение значения переменной *QUERY_STRING*.
- ❑ Декодирование имен и их значений. При этом учитывают, что все пробелы были заменены символом "+" и что все символы с десятичным кодом больше 128 преобразованы в символ "%" и следующим за ним шестнадцатеричным кодом символа.
- ❑ Формирование структуры "Имя-значение".

Метод *get* предполагает передачу данных программе через переменные среды.

В переменную *QUERY_STRING* попадают значения полей форм в формате:

*"Имя1=Значение1&Имя2=Значение2&Имя3=Значение3
"....*

Здесь ИмяN – значения атрибутов NAME, задающих имена управляющих элементов формы.

На месте ЗначениеN сервер записывает то значение атрибута VALUE, которое ввел пользователь в элементе ИмяN.

Строку *QUERY_STRING* обрабатывают обычными функциями языков программирования. Но ее нельзя модифицировать. Эта строка использует кодировку URL. После декодирования входных данных в массиве строк хранятся все пары имя=значение из входного набора. Этот массив разработчик использует в зависимости от поставленной задачи.

Интерфейс взаимодействия сервера с внешней программой

Обработка входных данных для метода post

Алгоритм обработки входных данных для метода post состоит из следующих шагов:

- ❑ Определение значения переменной ***CONTENT_LENGTH***.
- ❑ Декодирование имен и их значений. При этом учитывают, что все пробелы были заменены символом "+" и что все символы с десятичным кодом больше 128 преобразованы в символ "%" и следующим за ним шестнадцатеричным кодом символа.
- ❑ Формирование структуры "Имя - значение".

Необходимо считать данные из потока STDIN. Если переменная окружения ***CONTENT_TYPE*** содержит значение *application/x-www-form-urlencoded*, данные из потока STDIN также следует декодировать.


Метод post предполагает ввод данных через стандартный поток stdin.

Количество байт, которые следует считывать из STDIN, передается в переменной ***CONTENT_LENGTH***.



Интерфейс взаимодействия сервера с внешней программой

Рассмотрим интерфейс
взаимодействия внешней
программы с сервером.



Вывод данных для пользователя


Вне зависимости от метода ввода данных от пользователя, программа CGI направляет свой вывод в стандартный поток STDOUT.

Этот вывод может представлять собой HTML-документ или инструкции серверу, где получить необходимый документ. Преимущество последнего подхода в том, что cgi-модуль не должен формировать полный HTTP заголовок на каждый запрос.

Заголовок вывода сценария.

Как правило, вывод сценария интерпретируется сервером и посылается пользователю. Информация, позволяющая браузеру выяснить, какого типа файл идет к нему по сети, поступает до получения самого файла и называется заголовком. Поскольку сервер не может создать заголовок, сценарий сам должен обеспечивать отправку соответствующего заголовка. Таким образом, помимо собственно результатов обработки, сценарий должен поместить в выходной поток и корректный заголовок. Он состоит из строки состояния и затем полей ответа: общий заголовок (General-Header) и заголовок тела сообщения (Entity-Header), а также заголовок ответа (Response-Header).

Вывод начинается с маленького заголовка. Он содержит текстовые строки, в том же формате, как и в HTTP заголовке, и завершается пустой строкой (содержащей символ перевода строки). Любые строки заголовка, не являющиеся директивами сервера, посылаются непосредственно пользователю.



Интерфейс взаимодействия сервера с внешней программой

В настоящее время, CG1 спецификация определяет три директивы сервера представленные в таблице.

Имя	Пояснение
Content-type	MIME-тип возвращаемого документа.
Location	Это поле используется в случае, когда необходимо указать серверу, что возвращается не сам документ, а ссылка на него. Тогда сервер передает пользователю указание на перенаправление запроса.
Status	Задаёт серверу HTTP строку-статус, которая будет послана клиенту. Формат: <code>nnnxxxxx</code> , где <code>nnn</code> – 3-х цифровой статус-код, и <code>xxxxx</code> строка причины.

Наиболее интересная информация содержится в строке состояния.

Строка состояния имеет следующий формат:

HTTP/version – версия,

Status-Code – трехзначный код статуса идентифицирующий результат.


Status-Phrase – текстовая фраза, поясняющая код.

Код статуса используется браузером, а текстовая фраза предназначена для пользователя.

Первая цифра кода статуса предназначена для определения класса ответа.




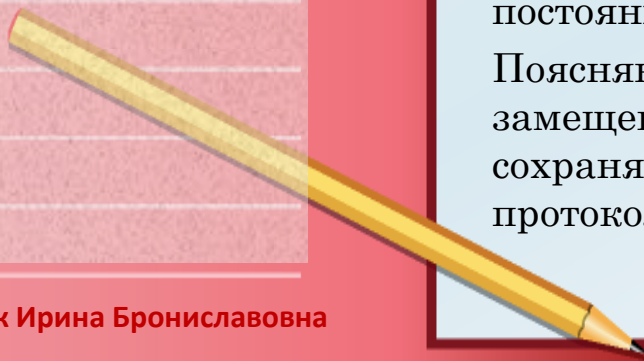
Интерфейс взаимодействия сервера с внешней программой



Существует пять категорий, определяемых первой цифрой кода:

- 1) **1xx**: пока не используется, зарезервирован для использования в будущем.
- 2) **2xx**: успех – запрос был полностью получен, понят и принят к обработке
- 3) **3xx**: перенаправление – пользователю следует предпринять дальнейшие действия для успешного выполнения запроса.
- 4) **4xx**: ошибка пользователя – запрос не может быть успешно обработан. Если пользователь еще не завершил запрос на момент получения ответа с кодом 4xx, то он должен немедленно прекратить передачу данных серверу.
- 5) **5xx**: ошибка Web-сервера – сервер не смог ответить на корректный запрос. Сервер посылает описание ошибочной ситуации и то, является ли это состояние временным или постоянным, в теле ответа.

Поясняющие текстовые фразы могут быть замещены любыми другими фразами, сохраняющими смысл и допускающимися протоколом.





Интерфейс взаимодействия сервера с внешней программой


В таблице перечислены
коды ответов HTTP.



Код статуса	Значение
200	ок
201	Успешная команда post
202	Запрос принят
203	Запрос get или HEAD выполнен
204	Запрос выполнен но нет содержимого
300	Ресурс обнаружен в нескольких местах
301	Ресурс удален навсегда
302	Ресурс отсутствует временно
304	Ресурс был изменен
400	Плохой запрос от клиента
401	Неавторизованный запрос
402	Необходима оплата за ресурс
403	Доступ запрещен
404	Ресурс не найден
405	Метод не применим для данного ресурса
406	Недопустимый тип ресурса
410	Ресурс недоступен
500	Внутренняя ошибка сервера
501	Метод не выполнен
502	Неисправный сценарий либо сервер перегружен
503	Сервер недоступен/тайм-аут сценария
504	Вторичный сценарий/тайм-аут сервера



Интерфейс взаимодействия сервера с внешней программой



Директива *Status* позволяет CGI-сценарию вернуть сообщение о состоянии обработки.

Если эта директива не задана, то сервер подразумевает 200 Ok.

Пример:

Status: 404 Notfound

На базе этой информации сервер и формирует окончательный заголовок, который и передается клиенту.

Например, набор директив:

HTTP/1.0 200 OK

Server: NCSA/1.0a6

Content-type: text/html

сообщает об успешном выполнении запроса.


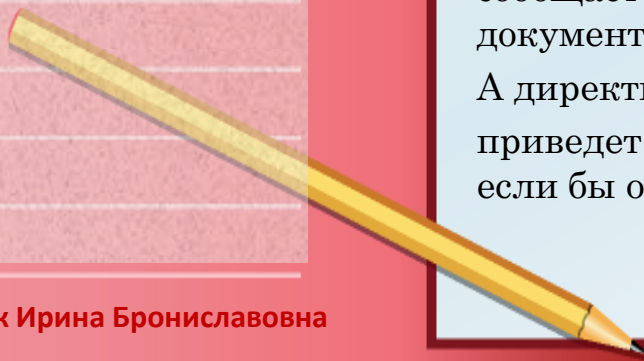
Здесь: *200* - код означающий успешную обработку запроса, что и поясняет "Ok".

Директива *Content-type: text/html*

обязательно присутствует, если есть тело ответа, и сообщает серверу, что далее последует HTML-документ.

А директива: *Location: http://host/file.txt*

приведет к тому, что Web-сервер выдаст file.txt, как если бы он был затребован клиентом.






Интерфейс взаимодействия сервера с внешней программой

Этапы работы сценария.


Как видно из вышеизложенного, работа сценария в общем случае состоит из трех частей:

- 1) Инициализация и обработка входных данных.
- 2) Основная часть работы и вывод пользователю.
- 3) Завершение своей работы.




При выполнении инициализации все параметры известны заранее (или их несложно узнать), а решаемые задачи более менее одинаковы для любого из сценариев. То, что выполняется на этапе обработки входящих данных, зависит от целей написания сценария. В общем случае обработка входящих данных означает их преобразование.

После инициализации работы, чтения входящих данных и их разделения на отдельные переменные, сценарий будет готов к выполнению основной части работы. В этой части работы сценарий готовит данные для вывода и шаги регламентируются не столь жестко, как на этапе инициализации. Все выполняемые действия определяются поставленными целями. Поскольку обработка данных в CGI довольно проста, а целью создания сценария почти всегда является подготовка некоторой исходящей информации, формализованные шаги обработки входящих и подготовки исходящих данных объединяются в одну фазу программы.





Интерфейс взаимодействия сервера с внешней программой

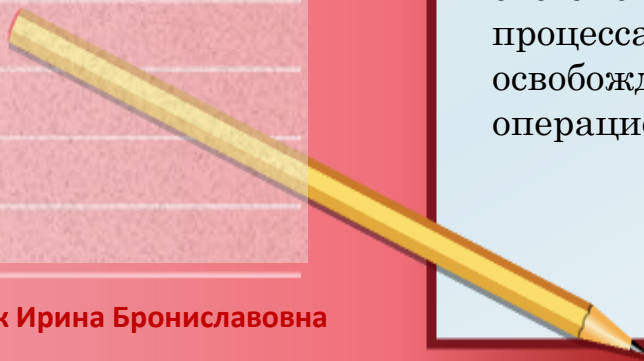


Вывод результатов. В простом CGI-сценарии исходящая информация представляет собой только заголовок и HTML-текст. Более сложные сценарии могут выводить только графику, графику с текстом или же вообще всю информацию, необходимую для повторного вызова сценария с некоторыми дополнительно введенными данными.

Завершение подразумевает всего лишь очистку и освобождение всех использованных ресурсов с последующим выходом из программы.

Если выполнялось распределение свободной памяти или других объектов, то они должны быть освобождены.


Если этого не сделать может возникнуть утечка или блокировка ресурсов всей системы и остановка работы любых других сценариев или вообще всего сервера. На большинстве платформ файлы и созданные в памяти объекты автоматически закрываются и освобождаются операционной системой после завершения использовавшего их процесса. Но и в данном случае неразумно при освобождении ресурсов целиком полагаться на операционную систему.





Глава 12

«Разработка интерфейсов информационного взаимодействия»



Тема 31. Комплексный подход к разработке пользовательского интерфейса.

1. Общие принципы создания интерфейсов.
2. Интерфейс человек-компьютер как отдельный компонент системы.

Общие принципы создания интерфейсов

При разработке интерфейса необходимо руководствоваться общими принципами.

1. Стратегия разработки интерфейса человек - компьютер.

2. Интерфейс человек-компьютер как отдельный компонент системы.

Так же как структуры данных в системе можно изолировать от алгоритмов обработки этих структур, мы можем до определенной степени отделить интерфейс человек-компьютер от прикладной.

Интерфейс необходимо проектировать отдельно, как и отдельно разрабатывать структуру файлов, обрабатываемых системой. Состав и форма представления входных и выходных данных должны стать предметом тщательного анализа разработчиков интерфейса.

3. Учет возможностей аппаратных и программных средств.

Разработчики систем, как, естественно, и другие специалисты, пользуются в работе своими старыми навыками. Этот внутренний консерватизм усиливается, а не ослабевает вследствие стремительного развития за последнее время аппаратных и программных средств. Однако невозможно разработать компоненты интерфейса, не понимая возможностей и ограничений основных элементов, из которых он может быть построен.

Общие принципы создания интерфейсов

4. Последовательность разработки.

В процессе разработки требуется новизна, причем необходимо следить за тем, чтобы эта новизна не растворилась в мелочах, а также за целесообразностью обилия подходов. В настоящее время многие пользователи имеют доступ к разным системам, и вряд ли они будут менять свои приемы работы при смене систем. Желательно развивать "семейство" программ, в рамках которых все они работают одинаково. Этому также может способствовать библиотека стандартных модулей, которые могут использоваться для разработки программных интерфейсов различных систем.

5. Использование принятых принципов разработки интерфейса.

Физическое взаимодействие пользователя с рабочей станцией имеет много общего с взаимодействием человека с машиной вообще. Поэтому существует большое число общепринятых в эргономике рекомендаций, которые можно легко перенести на разработку и организацию рабочей станции. В то же время форма представления информации на экране не одинакова для различных компьютерных систем: графический дизайн зависит от распределения информации на экране, словарного состава предложений, способа выделения ключевых элементов представления данных и т. д. Разработчики должны знать эти принципы и стараться также использовать знания из других областей при затруднениях в решении своих проблем.

Общие принципы создания интерфейсов

Хотя общие принципы определяют основу создания интерфейса, они не могут удовлетворять любого пользователя.

6. Понимание задач и пользователя.

Разработчик должен понимать не только вычислительный процесс, необходимый для решения задачи, но и оценивать действия пользователя, направленные на достижение цели задачи. Ему нужно знать особенности потенциальных пользователей системы. Важно отметить прогресс, достигнутый за последние годы в области разработки среды программирования – интерфейса между компьютером и программистом.

7. Привлечение пользователей.

Рекомендацию о том, что надо "понимать пользователя и задачу", легче дать, чем выполнить. Вряд ли можно ожидать, что системный аналитик или разработчик хорошо знаком со всеми областями применения своих разработок или глубоко чувствует психологические потребности потенциальных пользователей.

Однако, привлечение пользователей к разработке не может гарантировать ее абсолютной приемлемости. Даже если условия задачи остаются практически постоянными, потребности пользователей, как и они сами, меняются.

Общие принципы создания интерфейсов

Правильно спроектированный интерфейс, **как например, в случае с компьютерными играми**, должен быть настраиваемым на нужды разных пользователей, а также на одного пользователя в разные периоды его работы.

Проблема адаптации интерфейсов человек-компьютер - главное направление исследований в последнее время.

8. Оценка.

Как разработчик, придерживающийся рассмотренной выше стратегии, узнает, что он достиг требуемого результата?

Можно выделить несколько критериев, позволяющих оценить интерфейс и охватывающих три основных аспекта:

- ☐ простота освоения и запоминания операций системы;
- ☐ быстрота достижения целей задачи, решаемой с помощью системы;
- ☐ субъективная удовлетворенность при эксплуатации системы.

Интерфейс человек-компьютер как отдельный компонент системы

Хотя все три критерия можно отнести к любым областям применения, для конкретных применений важным будет какой-либо один из них.

Примеры:

Для систем, подобных системе управления авиатранспортом, важными являются факторы точности и скорости.

Для систем, рассчитанных на широкое применение, основным требованием является отсутствие предварительного обучения перед началом работы, поскольку часто отсутствует возможность организовать такие занятия.

При работе с системой, подобной электронной почте, пользователи должны чувствовать себя так же удобно, как и при работе с более простыми системами, иначе они просто откажутся от них.

Можно установить контрольное время, необходимое определенному пользователю для достижения заданного уровня знаний.

Критерий может также указать тип упражнений, помогающих добиться желаемого результата.

Подобный критерий можно сформулировать следующим образом: "После двух дней самостоятельных занятий с системой пользователь, работающий с ней впервые, освоит все команды, необходимые для работы с файлами, хранящимися на диске в иерархически связанных каталогах".


Сохранение полученных рабочих навыков по истечению некоторого времени – это другой критерий (связанный с первым), который определяет объем знаний, достаточный для возобновления деятельности после некоторого перерыва в работе.

Быстроту решения задачи можно оценить скоростью или точностью.

При оценке скорости учитывается не быстродействие системы, а время, необходимое для достижения поставленной цели. Поэтому для системы ввода данных важна не скорость работы с клавиатурой, а контрольная цифра, которую можно указать, например, так: "банковский служащий должен за час обработать не менее 20 счетов с ошибкой менее 1 %".

Критерий субъективной удовлетворенности отражает мнение пользователя о системе и удобстве работы с ней.


Этот критерий трудно оценить количественно, но его можно выразить, например, с помощью частоты, с которой пользователи обращаются к дополнительным устройствам.



Интерфейс человек-компьютер как отдельный компонент системы

Разработчик должен уметь измерять реальную производительность системы в соответствии с поставленными целями.

Комплексный учет требований позволяет создать стратегию проектирования интерфейса, которая может привести к созданию систем, удобных для использования людьми.



Для проведения этих измерений использовалось несколько методик.

Системы могут автоматически создавать и сохранять копию конкретного диалога, заносить в системный журнал время, затрачиваемое на выполнение различных этапов задания, или количество и тип ошибок.

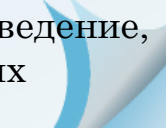
Пользователям предлагается ответить на вопросы или заполнить различные анкеты, чтобы можно было определить, удовлетворены ли они работой системы. За работой системы можно наблюдать визуально и даже записать на видеокассету для последующего анализа.


При использовании всех этих методов трудно быть уверенным в том, что получен действительно правильный результат и что любые замеченные вами отклонения присущи системе, а не определяются каким-либо внешним фактором.

Статистические методы, которые часто используются, требуют более строгого подхода к трактовке природы испытываемого объекта и способа проведения измерений.

Общеизвестны трудности выбора вопросов анкеты, на которые можно дать точные и четкие ответы.


Люди могут существенно изменить свое поведение, если узнают, что за ними наблюдают или их "испытывают".






Интерфейс человек-компьютер как отдельный компонент системы

Требования к пользовательским интерфейсам (ПИ)



Помимо идентификации основных возможностей ПИ требуются определенные ключевые характеристики поведения и внешнего вида ПИ. Ниже приведены эти характеристики, которые учитываются разработчиками ПИ.


- ☐ Выбор стиля ПИ.
- ☐ Платформа и другие стандарты ПИ для приложения.
- ☐ Совместимость с ведущим ПО, работающим на данной платформе (например приложение X или пакет Y).
- ☐ Содержание экрана (например, данные и функции, необходимые в ключевые моменты выполнения задач).
- ☐ Поведение экрана (например, входной фокус на первом элементе управления при отображении экрана).
- ☐ Характеристики внешнего вида экрана (например, использование графики; отображения данных, представления и эстетические свойства).



Интерфейс человек-компьютер как отдельный компонент системы

Требования к пользовательским интерфейсам (ПИ)

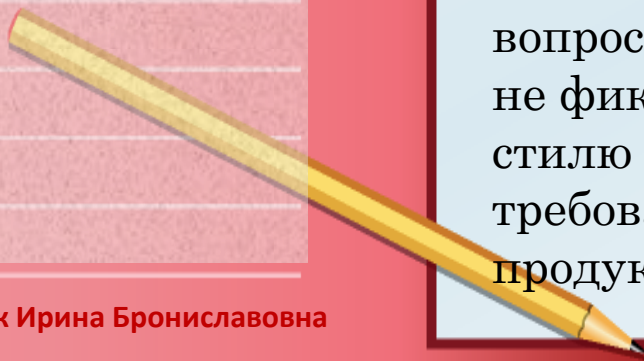
- ☐ Методы взаимодействия пользователей с системой (например, доступ к командам, способы образования комбинаций клавиш и т.д.).
- ☐ Возможности работы с клавиатурой, включая поведение средств табуляции и циклическую работу клавиши табуляции.
- ☐ Обратная связь пользователя в ответ на состояние системы и время отклика.
- ☐ Пользовательский контроль над различными функциями.
- ☐ Запоминание результатов операций расположения и изменения размеров окна, а также данных, состояния и контекста.
- ☐ Возможности навигации для приложения.
- ☐ Сохранение данных пользователя при навигации.



Интерфейс человек-компьютер как отдельный компонент системы

- ☐ Запоминание промежуточных данных пользователя при навигации.
- ☐ Интерактивное обучение, поддержка производительности и справочная система.
- ☐ Предотвращение ошибок и восстановление системы после ошибок.
- ☐ Стандартное использование цвета, индикаторов, графики и т.д.
- ☐ Средства обеспечения доступа для пользователей с физическими недостатками.

Многие из перечисленных выше вопросов ПИ зачастую явно и конкретно не фиксируются в руководствах по стилю ПИ, документах описания требований или спецификациях продукта.




Интерфейс человек-компьютер как отдельный компонент системы


Факторы, влияющие на GUI-ориентированное ПО, которые следует учитывать при разработке.

Для Web-ориентированного и HUI-ориентированного пользовательского интерфейса также характерны часть этих факторов.

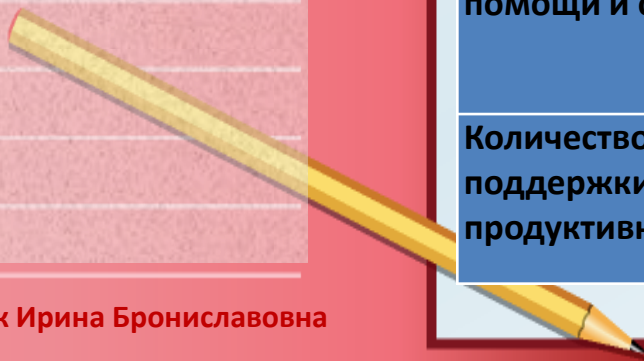
Основные факторы	Характеристики
Функциональные возможности	Прикладные характеристики объектов, команды и другие свойства
Возможности ПИ	Строка меню, всплывающее меню, пункты меню, панели инструментов, элементы панелей инструментов, операции "перетащить и поместить", операции с буфером обмена, клавиши быстрого выбора команд, клавиши доступа и т.д.
Количество объектов	Пиктограммы и двоичные отображения различного размера и разрешения, типы данных; поведение рабочего стола, системные функции; форматы печати
Количество объектных окон/страниц	Стандартные, память окна (размер, позиция, состояние)
Количество представлений на объект	Схемы представлений; поведение клавиатуры, поведение мыши
Количество настроек на объект	Количество настраиваемых свойств в расчете на окно и на пользовательский интерфейс в целом



Интерфейс человек-компьютер как отдельный компонент системы



Основные факторы	Характеристики
Количество командных окон на объект	Уникальные для приложения; общесистемные; поведение "затененных" команд
Количество элементов управления на окно объекта	Выпадающие списки, поля для ввода, кнопки со стрелками и т.д.
Количество элементов управления на командное окно	Аналогично предыдущему
Количество операций обратной связи с пользователем	"Песочные часы", индикаторы хода процесса и т.д.
Количество уникальных форматов печати	Схемы предварительного просмотра и печати
Количество экранов помощи и обучения	Включая организацию подобной помощи пользователю в последовательности смены окон ПИ
Количество панелей поддержки продуктивности	Включая различные применяемые методы



Интерфейс человек-компьютер как отдельный компонент системы

Полезное правило.

*Для каждого окна/страницы
ПИ следует предусматривать
подробную декомпозицию
работ по их созданию.*

Этот метод особенно полезен для разработчиков или бригад, которые впервые занимаются интерфейсом и не обладают достаточным опытом их структуризации.

Основные факторы	Характеристики
Количество сообщений на окно объекта	Обратная связь, ошибки, сообщения
Количество сообщений на командное Окно	Аналогично предыдущему
Количество уникальных типов внешнего вида/поведения	Специальные характеристики внешнего вида и поведения, требующие проектирования и разработки ПО
Количество пользовательских элементов управления	Нестандартные и специализированные элементы управления ПИ, требующие проектирования и разработки ПО
Возможности инсталляции/деинсталляции/обновления	Специальное ПО, требуемое для инсталляции, обновления и удаления приложения из системы пользователя
Уникальные операции с клавиатурой, мышью и другими устройствами	"Горячие клавиши", клавиши доступа, "жесты" (программируемые пользователем графические знаки, представляющие команду или ряд последовательно нажимаемых клавиш.