СОЗДАНИЕ ПРИЛОЖЕНИЕ «КЛИЕНТ-СЕРВЕР» И ДОСТУП К СЕРВЕРНОЙ БАЗЕ ДАННЫХ ИЗ КЛИЕНТА

Рассмотрим механизмы доступа к расположенной на сервере базе данных из клиента. Технология "клиент-сервер" применяется для централизованного доступа из удаленных ЭВМ к общей базе данных, расположенной на сервере. В языке Java механизм клиент-серверного взаимодействия основан на использовании потоков и сокетных соединений. Для клиента надлежит создать два потока: один поток для работы на ввод, второй — для работы на вывод. Аналогичная ситуация с сервером. В качестве языка запросов используем язык SQL.

Клиент должен обращаться в базу с сформированным SQL- запросом. Сервер должен выполнять запрос и возвращать ответ.

Это первое приложение в односторонней связи. В случае односторонней связи клиент отправляет на сервер, но сервер не отправляет обратно клиенту. При двусторонней связи клиент отправляет на сервер, а сервер отправляет обратно клиенту.
Всего в приложении TCP / IP 4 варианта.

APPLICATION NUMBER	FUNCTIONALITY
1st application	Client to server communication (one-way)
2nd application	Server to client communication (one-way)
3rd application	Server sends file contents to client (two-way, non-co
4th application	Chat program (two-way, continuous)

1-е Приложение клиент-сервер

Приложение состоит из двух программ. Клиентская программа, работающая на стороне клиента, и серверная программа, работающая на стороне сервера. Клиентская программа WishesClient.java отправляет серверу наилучшие пожелания, а серверная программа WishesServer.java получает сообщение и печатает на своем терминале (мониторе).

Клиентская программа – WishesClient.java

```
import java.net.Socket;
import java.io.OutputStream;
import java.io.DataOutputStream;
public class WishesClient
{
public static void main(String args[]) throws Exception
{
Socket sock = new Socket("127.0.0.1", 5000);
String message1 = "Accept Best Wishes, Serverji";
OutputStream ostream = sock.getOutputStream();
DataOutputStream dos = new DataOutputStream(ostream);
dos.writeBytes(message1);
dos.close();
ostream.close();
sock.close();
}
}
```

Socket sock = **new** Socket ("127.0.0.1", 5000);
Конструктор класса Socket принимает два параметра – строку, IP-адрес сервера и целое число, номер порта на сервере, к которому клиент хотел бы подключиться. 127.0.0.1 – это адрес по умолчанию локальной системы в компьютерных сетях.

OutputStream ostream = sock.getOutputStream ();

Метод getOutputStream() класса Socket возвращает объект OutputStream, здесь объект является ostream. Это отправная точка всего общения (программы). Здесь сокет связан с потоками. Потоки способствуют передаче данных.

DataOutputStream dos = **new** DataOutputStream (ostream);
dos.writeBytes (message1);
OutputStream является абстрактным классом; он не может быть использован напрямую. В приведенном выше коде он связан с конкретным классом DataOutputStream. Метод writeBytes() объекта DataOutputStream принимает строковое сообщение и передает его в Socket. Теперь клиентский сокет отправляется на другой сокет на сервере. Когда работа закончится, закройте потоки и сокет. Он освобождает дескрипторы (ссылки), связанные с системными ресурсами.

Ниже приведены исключения в вышеприведенной программе, создаваемые конструктором и различными методами.

- Socket(“127.0.0.1”, 5000) выдает UnknownHostException

- getOutputStream() генерирует IOException
- writeBytes (message1) выдает IOException
- Все методы close() выдают IOException
- Серверная программа – WishesServer.java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
import java.io.DataInputStream;
public class WishesServer
{
public static void main(String args[]) throws Exception
{
ServerSocket sersock = new ServerSocket(5000);
System.out.println("server is ready"); // message to know the server is running
Socket sock = sersock.accept();
InputStream istream = sock.getInputStream();
DataInputStream dstream = new DataInputStream(istream);
String message2 = dstream.readLine();
System.out.println(message2);
dstream .close(); istream.close(); sock.close(); sersock.close();
}
}
```

ServerSocket sersock = новый ServerSocket (5000);

У сервера есть два задания: одно, как и ожидалось, должно связываться, а другое связывает соединение с номером порта 5000. Для связи он использует Socket, а для привязки – ServerSocket.

Связывание – это не что иное, как выделение номера порта клиенту так долго, как ему хотелось бы; Между тем, если какой-либо другой клиент запрашивает номер порта 5000, он не должен выделяться сервером. Когда

клиент отключается, порт освобождается и может быть предоставлен другому клиенту сервером.
Socket sock = sersock.accept ();

accept() – это метод класса ServerSocket, используемый сервером для привязки соединения по номеру порта 5000, запрошенного клиентом.

InputStream istream = sock.getInputStream();

Метод getInputStream() объекта Socket возвращает объект InputStream, и это отправная точка серверной программы. Сервер использует входной поток при получении сообщения.

DataInputStream dstream = new DataInputStream (istream);

Поскольку InputStream является абстрактным классом, его нельзя использовать напрямую. Он связан с конкретным классом DataInputStream.

String message2 = dstream.readLine();

Метод readLine() объекта DataInputStream читает строку сообщения из сокета и возвращает ее. Это сообщение печатается на консоли.

Примечание. При компиляции этой программы вы получаете предупреждение из-за метода readLine() объекта DataInutStream; но программа выполняется. Чтобы избежать этого предупреждения, в следующей программе используется BufferedReader.
Выполнение клиентских и серверных программ

В одной системе, чтобы действовать как клиент и сервер, откройте два шDOS и обработайте одно как клиент, а другой – как сервер. Из одного приглашения DOS сначала запустите серверную программу, а из другого приглашения DOS запустите клиентскую программу. Вы получаете вывод при запросе сервера DOS.

Это приложение и следующее – только односторонняя связь, отправляющая или получающая. Но второй набор (после следующего) приложений является двусторонним, когда клиент и сервер могут отправлять и получать (оба).

Для лучшего понимания вопрос-ответ из пакета java.lang.

Сколько существует типов внутренних классов?
Ответ: 4 типа.

Что такое файлы JAR?
Ответ: JAR-файл – это заархивированный файл, сжатый JVM.

Как преобразовать строку в форму типа данных?
Ответ: Преобразование строки в тип данных – байтовое, короткое, целое, длинное, плавающее, двойное, символьное и логическое.

Как преобразовать объект в строку?
Ответ: Объект в строку – toString()

Как сравнить два объекта?
Ответ: Сравнение объектов – hashCode() & equals()