Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Чувашский государственный университет имени И.Н. Ульянова»

СЕТЕВЫЕ СРЕДСТВА JAVA.

Назарова Ольга Васильевна

Сетевые классы имеют методы для установки сетевых соединений передачи запросов и сообщений. Многопоточность позволяет обрабатывать несколько соединений.

Сетевые приложения используют Internet-приложения, к которым относятся Web-браузер, e-mail, сетевые новости, передача файлов.

Для создания таких приложений используются сокеты, порты, протоколы TCP/IP, UDP.

Сетевые классы имеют методы для установки сетевых соединений передачи запросов и сообщений.

Многопоточность позволяет обрабатывать несколько соединений. Сетевые приложения используют Internet-приложения, к которым относятся Web-браузер,
e-mail, сетевые новости, передача файлов. Для создания таких приложений используются сокеты, порты, протоколы TCP/IP, UDP.

Приложения клиент/сервер используют компьютер, выполняющий специальную программу-**сервер**, которая обычно устанавливается на удаленном компьютере и предоставляет услуги другим программам-**клиентам**. **Клиент** - это программа, получающая услуги от сервера. Клиент устанавливает соединение с сервером и пересылает серверу запрос. Сервер осуществляет прослушивание клиентов, получает и выполняет запрос после установки соединения.

Клиент-серверные приложения основаны на использовании, в первую очередь, прикладных протоколов стека TCP/IP, таких как:

HTTP - Hypertext Transfer Protocol (WWW);

NNTP - Network News Transfer Protocol (группы новостей);

SMTP - Simple Mail Transfer Protocol (посылка почты);

POP3 - Post Office Protocol (чтение почты с сервера);

FTP - File Transfer Protocol (протокол передачи файлов);

TELNET – Удаление управление компьютерами.

Каждый компьютер, работающий по протоколам стека TCP/IP имеет уникальный сетевой адрес.

IP-адрес - это 32-битовое число, обычно записываемое как четыре числа, разделенные точками, каждое из которых изменяется от 0 до 255. IP-адрес может быть временным и выделяться динамически для каждого подключения или быть постоянным, как для сервера.

Обычно при подключении к компьютеру вместо числового IP адреса используются символьные имена (например — www.example.com), называемые доменными именами. Специальная программа **DNS** (Domain Name Sever) преобразует имя домена в числовой IP-адрес. Получить IP-адрес в программе можно с помощью объекта класса **InetAddress** из пакета **java.net**.

```
import java.net.*;
public class MyLocal {
public static void main(String[] args) {
InetAddress myIP = null;
try {
myIP = InetAddress.getLocalHost();
} catch (UnknownHostException e) {
System. out.println( " ошибка доступа ->" + e);
}
System. out.println( " Мой IP ->" + myIP);
}
```

Класс **InetAddress** не имеет public-конструкторов. Создать объект класса можно с помощью статических методов.

Метод **getLocalHost()** возвращает объект класса **InetAddress**, содержащий IP-адрес и имя компьютера, на котором выполняется программа.

Метод **getByName(Stringhost)** возвращает объект класса **InetAddress**, содержащий IP-адрес по имени компьютера, используя пространство имен DNS. IP-адрес может быть временным, различным для каждого соединения, однако он остается постоянным, если соединение установлено.

Метод **getByAddress(byte[] addr)** создает объект класса **InetAddress**, содержащий имя компьютера, по IP-адресу, представленному в виде массива байт.

Если компьютер имеет несколько IP, то получить их можно методом **getAllByName(Stringhost)**, возвращающим массив объектов класса **InetAddress**. Если IP для данной машины один, то массив будет содержать один элемент.

Метод **getByAddress(Stringhost, byte[] addr)** создает объект класса **InetAddress** с заданным именем и IP-адресом, не проверяя существование такого компьютера. Все эти методы являются потенциальными генераторами исключительной ситуации **UnknownHostException**, и поэтому их вызов должен быть обработан с помощью **throws** для метода или блока **try-catch**. Проверить доступ к компьютеру в данный момент можно с помощью метода **booleanisReachable(int timeout)**, который возвращает **true**, если компьютер доступен, где **timeout**–время ожидания ответа от компьютера в миллисекундах.

Следующая программа демонстрирует, как получить IP-адрес из имени домена с помощью сервера имен доменов (DNS), к которому обращается метод getByName().

```
import java.net.*;
public class IPfromDNS {

public static void main(String[] args) {
        InetAddress omgtu = null;
        try {
                omgtu = InetAddress.getByName("omgtu.ru");
                }
        catch (UnknownHostException e) {
                System.out.println( " ошибка доступа ->" + e);
                }
        System.out.println( "IP- адрес ->" + omgtu );
        }
}
```

Будет выведено: IP-адрес –>omgtu.ru/195.69.204.35

Сокетные соединения по протоколу TCP/IP

Сокеты (сетевые разъёмы) –это логическое понятие, соответствующее разъёмам, к которым подключены сетевые компьютеры и через которые осуществляется двунаправленная поточная передача данных между компьютерами.

Сокет определяется номером порта и IP-адресом. При этом *IP-адрес* используется для идентификации компьютера, *номер порта* –для идентификации процесса, работающего на компьютере. Когда одно приложение знает сокет другого, создается сокетное протоколо-ориентированное соединение по протоколу TCP/IP. Клиент пытается соединиться с сервером, инициализируя сокетное соединение. Сервер прослушивает сообщение и ждет, пока клиент не свяжется с ним. Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента. Сервер, в свою очередь, создает сокет, который будет использоваться для связи с клиентом, и посылает его клиенту с первым сообщением. После этого устанавливается коммуникационное соединение.

Сокетное соединение с сервером создается клиентом с помощью объекта класса **Socket**. При этом указывается IP-адрес сервера и номер порта.

При этом IP-адрес используется для идентификации компьютера, номер порта – для идентификации процесса, работающего на компьютере. Когда одно приложение знает сокет другого, создается сокетное соединение. Клиент пытается соединиться с сервером, инициализируя сокетное соединение. Сервер ждет, пока клиент не свяжется с ним. Первое сообщение, посылаемое клиентом на сервер, содержит сокет клиента. Сервер в свою очередь создает сокет, который будет использоваться для связи с клиентом, и посылает его клиенту с первым сообщением. После этого устанавливается коммуникационное соединение.

Если указано имя домена, то Java преобразует его с помощью DNS-сервера к IP-адресу:

```
try {
Socket socket = new Socket("localhost", 8030);
} catch (IOException e) {
System. out.println( " ошибка : " + e);
}.
```

Сервер ожидает сообщения клиента и должен быть запущен с указанием определенного порта. Объект класса **ServerSocket** создается с указанием конструктору номера порта и ожидает сообщения клиента с помощью метода **accept()**, который возвращает сокет клиента:

```
Socket socket = null ;
try {
ServerSocket server = new ServerSocket(8030);
socket = server.accept();
} catch (IOException e) {
System. out.println( " ошибка : " + e);
}.
```

Клиент и сервер после установления сокетного соединения могут получать данные из потока ввода и записывать данные в поток вывода с помощью методов **getInputStream()** и **getOutputStream()** или к **PrintStream** для того, чтобы программа могла трактовать поток как выходные файлы.

В следующем примере для посылки клиенту строки "привет!" сервер вызывает метод **getOutputStream()** класса **Socket**. Клиент получает данные от сервера с помощью метода **getInputStream()**. Для разъединения клиента и сервера после завершения работы сокет закрывается с помощью метода **close()** класса **Socket**. В данном примере сервер посылает клиенту строку "привет!", после чего разрывает связь.

```
// передача клиенту строки : MyServerSocket. java
import java.io.*;
import java.net.*;
public class MyServerSocket {
public static void main(String[] args) throws Exception {
Socket s = null;
try { // посылка строки клиенту
ServerSocket server = new ServerSocket(8030);
s = server.accept();
PrintStream ps = new PrintStream(s.getOutputStream());
ps.println( " привет !" );
ps.flush();
s.close(); // разрыв соединения
} catch (IOException e) {
System. out.println( " ошибка : " + e);
}
}
}
```

```
/* получение клиентом строки : MyClientSocket. java */
import java.io.*;
import java.net.*;
public class MyClientSocket {
public static void main(String[] args) {
Socket socket = null;
try {// получение строки клиентом
socket = new Socket( " имя _ компьютера " , 8030);
BufferedReader dis = new BufferedReader(new InputStreamReader(
socket.getInputStream()));
String msg = dis.readLine();
System.out.println(msg);
} catch (IOException e) {
System. out.println( " ошибка : " + e);
}
}
}
```

Аналогично клиент может послать данные серверу через поток вывода с помощью метода **getOutputStream()**, а сервер может получать данные с помощью метода **getInputStream()**.

Если необходимо протестировать подобный пример на одном компьютере, можно выступать одновременно в роли клиента и сервера, используя статические методы **getLocalHost()** класса **InetAddress** для получения динамического IP-адреса компьютера, который выделяется при входе в Internet.