Урок 6

Работа с Сетью

Сокеты. Написание простого эхо-сервера и консольного клиента

Основы работы в Сети

В основу работы в сети, поддерживаемой в Java, положено понятие сокета, обозначающего конечную точку в Сети. Сокеты составляют основу современных способов работы в Сети, поскольку сокет позволяет отдельному компьютеру одновременно обслуживать много разных клиентов, предоставляя разные виды информации. Эта цель достигается благодаря применению порта — нумерованного сокета на отдельной машине.

Говорят, что серверный процесс «прослушивает» порт до тех пор, пока клиент не соединится с ним. Сервер в состоянии принять запросы от многих клиентов, подключаемых к порту с одним и тем же номером, хотя каждый сеанс связи индивидуален. Для управления соединениями со многими клиентами серверный процесс должен быть многопоточным или располагать какими-то другими средствами для мультиплексирования одновременного ввода-вывода.

Связь между сокетами устанавливается и поддерживается по определённому сетевому протоколу. Протокол Интернета (IP) является низкоуровневым маршрутизирующим сетевым протоколом, разбивающим данные на небольшие пакеты и посылающим их через Сеть по определенному адресу, что не гарантирует доставки всех этих пакетов по этому адресу. Протокол управления передачи (TCP) является сетевым протоколом более высокого уровня, обеспечивающим связывание, сортировку и повторную передачу пакетов, чтобы обеспечить надежную доставку данных. Ещё одним сетевым протоколом более низкого уровня, чем TCP, является протокол пользовательских дейтаграмм (UDP). Этот сетевой протокол может быть использован непосредственно для поддержки быстрой, не требующей постоянного соединения и ненадежной транспортировки пакетов.

Как только соединение будет установлено, в действие вступает высокоуровневый протокол, тип которого зависит от используемого порта. Протокол TCP/IP резервирует первые 1 024 порта для отдельных протоколов. Например, порт 21 выделен для протокола FTP, порт 23 — для протокола Telnet, порт 25 — для электронной почты, порт 80 — для протокола HTTP и т.д. Каждый сетевой протокол определяет порядок взаимодействия клиента с портом.

Например, протокол HTTP используется серверами и веб-браузерами для передачи гипертекста и графических изображений. Это довольно простой протокол для базового постраничного просмотра информации, предоставляемой веб-серверами. Рассмотрим принцип его действия. Когда клиент запрашивает файл у HTTP-сервера, это действие называется обращением. Оно состоит в том, чтобы отправить имя файла в специальном формате в предопределённый порт и затем прочитать содержимое этого файла. Сервер также сообщает код состояния, чтобы известить клиента, был ли запрос обслужен, а также причину, по которой он не может быть обслужен.

Главной составляющей Интернета является адрес, который есть у каждого компьютера в Сети. Изначально все адреса состояли из 32-разрядных значений, организованных по четыре 8-разрядных значения. Адрес такого типа определён в протоколе IPv4. Но в последнее время вступила в действие новая схема адресации, называемая IPv6 и предназначенная для поддержки намного большего адресного пространства. Правда, для сетевого программирования на Java обычно не приходится беспокоиться, какого типа адрес используется: IPv4 или IPv6, поскольку эта задача решается в Java автоматически.

Сокеты по протоколу TCP/IP служат для реализации надежных двунаправленных постоянных двухточечных потоковых соединений между хостами в Интернете. Сокет может служить для подключения системы ввода-вывода в Java к другим программам, которые могут находиться как на локальной машине, так и на любой другой в Интернете.

В Java поддерживаются две разновидности сокетов по протоколу TCP/IP: один — для серверов, другой — для клиентов. Класс ServerSocket предназначен для создания сервера, который будет обрабатывать клиентские подключения, тогда как класс Socket предназначен для обмена данными между сервером и клиентами по сетевому протоколу. При создании объекта типа Socket неявно

устанавливается соединение клиента с сервером.

Для доступа к потокам ввода-вывода, связанным с классом Socket, можно воспользоваться методами getInputStream() и getOuptutStream(). Каждый из этих методов может сгенерировать исключение типа IOException, если сокет оказался недействительным из-за потери соединения. Эти потоки ввода-вывода используются для передачи и приема данных.

Написание эхо-сервера

```
public class MainClass {
    public static void main(String[] args) {
        ServerSocket serv = null;
        Socket sock = null;
        try {
            serv = new ServerSocket(8189);
            System.out.println("Сервер запущен, ожидаем подключения...");
            sock = serv.accept();
            System.out.println("Клиент подключился");
            Scanner sc = new Scanner(sock.getInputStream());
            PrintWriter pw = new PrintWriter(sock.getOutputStream());
            while (true) {
                String str = sc.nextLine();
                if (str.equals("end")) break;
                pw.println("Эхо: " + str);
                pw.flush();
            }
        } catch (IOException e) {
            System.out.println("Ошибка инициализации сервера");
        } finally {
            try {
                serv.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Для начала создаётся объект класса ServerSocket, представляющий собой сервер, который прослушивает порт 8189. Метод server.accept() переводит основной поток в режим ожидания, поэтому, пока никто не подключится, следующая строка кода выполнена не будет. Как только клиент подключился, информация о соединении с ним запишется в объект типа Socket. Для обмена сообщениями с клиентом необходимо создать обработчики входящего и исходящего потока, в данном случае это — Scanner и PrintWriter, в дальнейшем будем использовать DataInputStream и DataOutputStream.

Поскольку мы создаём эхо-сервер, обработка данных производится следующим образом: сервер считывает сообщение, переданное клиентом, добавляет к нему фразу «Эхо: » и отсылает обратно. Если клиент прислал сообщение «end», общение с ним прекращается, и сокет закрывается.

Блок finally предназначен для гарантированного закрытия всех сетевых соединений и освобождения ресурсов.

Написание клиентской части

Ниже представлен код клиентской части чата.

```
public class MainClass {
    public static void main(String[] args) {
        new MyWindow();
    }
}
public class MyWindow extends JFrame {
    private JTextField jtf;
    private JTextArea jta;
    private final String SERVER_ADDR = "localhost";
    private final int SERVER_PORT = 8189;
    private Socket sock;
    private Scanner in;
    private PrintWriter out;
    public MyWindow() {
        try {
            sock = new Socket(SERVER_ADDR, SERVER_PORT);
            in = new Scanner(sock.getInputStream());
            out = new PrintWriter(sock.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
        setBounds(600, 300, 500, 500);
        setTitle("Client");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        jta = new JTextArea();
        jta.setEditable(false);
        jta.setLineWrap(true);
        JScrollPane jsp = new JScrollPane(jta);
        add(jsp, BorderLayout.CENTER);
        JPanel bottomPanel = new JPanel(new BorderLayout());
        add(bottomPanel, BorderLayout.SOUTH);
        JButton jbSend = new JButton("SEND");
        bottomPanel.add(jbSend, BorderLayout.EAST);
        jtf = new JTextField();
        bottomPanel.add(jtf, BorderLayout.CENTER);
        jbSend.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                if (!jtf.getText().trim().isEmpty()) {
                    sendMsg();
                    jtf.grabFocus();
                }
            }
        });
        jtf.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                sendMsg();
            }
        });
        new Thread(new Runnable() {
            @Override
            public void run() {
```

```
                try {
                    while (true) {
                        if (in.hasNext()) {
            String w = in.nextLine();
                            if (w.equalsIgnoreCase("end session")) break;
                            jta.append(w);
                            jta.append("\n");
                        }
                    }
                } catch (Exception e) {
                }
            }
        }).start();
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                super.windowClosing(e);
                try {
                    out.println("end");
                    out.flush();
                    sock.close();
                    out.close();
                    in.close();
                } catch (IOException exc) {
                }
            }
        });
        setVisible(true);
    }
    public void sendMsg() {
        out.println(jtf.getText());
        out.flush();
        jtf.setText("");
    }
}
```

Большая часть приведённого выше кода связана с созданием графического интерфейса. Поэтому разберём отдельные блоки, не касающиеся GUI.

```
public class MyWindow extends JFrame {
    // ...

    private final String SERVER_ADDR = "localhost";
    private final int SERVER_PORT = 8189;
    private Socket sock;
    private Scanner in;
    private PrintWriter out;
    public MyWindow() {
        try {
            sock = new Socket(SERVER_ADDR, SERVER_PORT);
            in = new Scanner(sock.getInputStream());
            out = new PrintWriter(sock.getOutputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
   // ...

   }
```

```
}
```

Константа SERVER_ADDR задаёт адрес сервера, к которому будет подключаться клиент, SERVER_PORT — номер порта. Для открытия соединения с сервером и обмена сообщениями используются объекты классов Socket, Scanner и PrintWriter, по аналогии с серверной частью.

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            while (true) {
                if (in.hasNext()) {
                    String w = in.nextLine();
                    if (w.equalsIgnoreCase("end session")) break;
                    jta.append(w);
                    jta.append("\n");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}).start();
```

Чтение данных с сервера организовано в отдельном потоке, как показано выше. Производится постоянный опрос входящего потока на наличие данных, если какие-то данные пришли, мы их записываем в многострочное текстовое поле на форме с помощью метода append(). Отдельный поток для чтения сделан для того, чтобы бесконечный цикл полностью не блокировал работу программы.

```
public void sendMsg() {
    out.println(jtf.getText());
    out.flush();
    jtf.setText("");
}
```

Отсылка сообщений на сервер организована с помощью метода sendMsg(), который извлекает сообщение из однострочного текстового поля и пишет его в исходящий поток.

Домашнее задание

1. Написать консольный вариант клиент\серверного приложения, в котором пользователь может писать сообщения как на клиентской стороне, так и на серверной. Т.е. если на клиентской стороне написать «Привет», нажать Enter, то сообщение должно передаться на сервер и там отпечататься в консоли. Если сделать то же самое на серверной стороне, сообщение, соответственно, передаётся клиенту и печатается у него в консоли. Есть одна особенность, которую нужно учитывать: клиент или сервер может написать несколько сообщений подряд. Такую ситуацию необходимо корректно обработать.

Разобраться с кодом с занятия — он является фундаментом проекта-чата.

ВАЖНО! *Сервер общается только с одним клиентом, т.е. не нужно запускать цикл, который будет ожидать второго/третьего/n-го клиента.*

Дополнительные материалы

1. Кей С. Хорстманн, Гари Корнелл. Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. — М.: Вильямс, 2014. — 864 с.
2. Брюс Эккель. Философия Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1 168 с.
3. Г. Шилдт. Java 8. Полное руководство // 9-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 1 376 с.
4. Г. Шилдт. Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. — М.: Вильямс, 2015. — 720 с.