Конспект лекции
Введение в сетевые протоколы

Цель и задачи лекции

Цель – изучить основные сетевые протоколы и поддержку их Java.

Задачи:

1. Описать основные сетевые протоколы.
2. Изучить основные классы Java для работы с сетью

План занятия

1. Сетевая модель OSI
2. Пакет java.net

Сетевая модель OSI

OSI расшифровывается как Open System Interconnection. На русском языке это звучит следующим образом: Сетевая модель взаимодействия открытых систем (эталонная модель). Эту модель можно смело назвать стандартом. Именно этой модели придерживаются производители сетевых устройств, когда разрабатывают новые продукты.

Сетевая модель OSI состоит из 7 уровней, причем принято начинать отсчёт с нижнего. Перечислим их:

7. Прикладной уровень (application layer)

6. Представительский уровень или уровень представления (presentation layer)

5. Сеансовый уровень (session layer)

4. Транспортный уровень (transport layer)

3. Сетевой уровень (network layer)

2. Канальный уровень (data link layer)

1. Физический уровень (physical layer)

Прикладной уровень

Прикладной уровень или уровень приложений(application layer) – это самый верхний уровень модели. Он осуществляет связь пользовательских приложений с сетью. Эти приложения нам всем знакомы: просмотр веб-страниц (HTTP), передача и приём почты (SMTP, POP3), приём и получение файлов (FTP, TFTP), удаленный доступ (Telnet) и т.д.

Представительский уровень

Представительский уровень или уровень представления данных (presentation layer) – он преобразует данные в соответствующий формат. На примере понять проще: те картинки (все изображения) которые вы видите на экране, передаются при пересылке файла в виде маленьких порций единиц и ноликов (битов). Так вот, когда Вы отправляете своему другу фотографию по электронной почте, протокол Прикладного уровня SMTP отправляет фотографию на нижний уровень, т.е. на уровень Представления. Где Ваша фотка преобразуется в удобный вид данных для более низких уровней, например в биты (единицы и нолики).

Именно таким же образом, когда Ваш друг начнет получать Ваше фото, ему оно будет поступать в виде все тех же единиц и нулей, и именно уровень Представления преобразует биты в полноценное фото, например JPEG.

Вот так и работает этот уровень с протоколами (стандартами) изображений (JPEG, GIF, PNG, TIFF), кодировок (ASCII, EBDIC), музыки и видео (MPEG) и т.д.

Сеансовый уровень

Сеансовый уровень или уровень сессий(session layer) – как видно из названия, он организует сеанс связи между компьютерами. Хорошим примером будут служить аудио и видеоконференции, на этом уровне устанавливается, каким кодеком будет кодироваться сигнал, причем этот кодек должен присутствовать на обеих машинах. Еще примером может служить протокол SMPP (Short message peer-to-peer protocol), с помощью него отправляются хорошо известные нам СМСки и USSD запросы. И последний пример: PAP (Password Authentication Protocol) – это старенький протокол для отправки имени пользователя и пароля на сервер без шифрования.

Транспортный уровень

Транспортный уровень (transport layer) – этот уровень обеспечивает надёжность передачи данных от отправителя к получателю. На самом деле всё очень просто, например вы общаетесь с помощью веб-камеры со своим другом или преподавателем. Нужна ли здесь надежная доставка каждого бита переданного изображения? Конечно нет, если потеряется несколько битов из потокового видео Вы даже этого не заметите, даже картинка не изменится (м.б. изменится цвет одного пикселя из 900000 пикселей, который промелькнет со скоростью 24 кадра в секунду).

Для примера. Вам друг пересылает (например, через почту) в архиве важную информацию или программу. Вы скачиваете себе на компьютер этот архив. Вот здесь надёжность нужна 100%, т.к. если пару бит при закачке архива потеряются – Вы не сможете затем его разархивировать, т.е. извлечь необходимые данные. Или представьте себе отправку пароля на сервер, и в пути один бит потерялся – пароль уже потеряет свой вид и значение изменится.

Таким образом, когда мы смотрим видеоролики в интернете, иногда мы видим некоторые артефакты, задержки, шумы и т.п. А когда мы читаем текст с веб-страницы – потеря (или скжение) букв не допустима, и когда скачиваем программы – тоже все проходит без ошибок.

На этом уровне я выделю два протокола: UDP и TCP. UDP протокол (User Datagram Protocol) передает данные без установления соединения, не подтверждает доставку данных и не делает повторы. TCP протокол (Transmission Control Protocol), который перед передачей устанавливает

соединение, подтверждает доставку данных, при необходимости делает повтор, гарантирует целостность и правильную последовательность загружаемых данных.

Следовательно, для музыки, видео, видеоконференций и звонков используем UDP (передаем данные без проверки и без задержек), а для текста, программ, паролей, архивов и т.п. – TCP (передача данных с подтверждением о получении, затрачивается больше времени).

Сетевой уровень

Сетевой уровень (network layer) – этот уровень определяет путь, по которому данные будут переданы. И, между прочим, это третий уровень Сетевой модели OSI, а ведь существуют такие устройства, которые как раз и называют устройствами третьего уровня – маршрутизаторы.

Все мы слышали об IP-адресе, вот это и осуществляет протокол IP (Internet Protocol). IP-адрес – это логический адрес в сети.

На этом уровне достаточно много протоколов и разбор всех этих протоколов – задача отдельного сссетевого курса, а мы изучаем Java.

Как об IP-адресе все слышали и о команде ping – это работает протокол ICMP.

Те самые маршрутизаторы (с которыми мы и будет работать в дальнейшем) используют протоколы этого уровня для маршрутизации пакетов (RIP, EIGRP, OSPF).

Канальный уровень

Канальный уровень (data link layer) – нужен для взаимодействия сетей на физическом уровне. Наверное, все слышали о MAC-адресе, вот он является физическим адресом. Устройства канального уровня – коммутаторы, концентраторы и т.п.

IEEE (Institute of Electrical and Electronics Engineers - Институт инженеров по электротехнике и электронике) определяет канальный уровень двумя подуровнями: LLC и MAC.

LLC – управление логическим каналом (Logical Link Control), создан для взаимодействия с верхним уровнем.

MAC – управление доступом к передающей среде (Media Access Control), создан для взаимодействия с нижним уровнем.

Приведем пример. В Вашем компьютере (ноутбуке, коммуникаторе) имеется сетевая карта (или какой-то другой адаптер), так вот для взаимодействия с ней (с картой) существует драйвер. Драйвер – это некоторая программа - верхний подуровень канального уровня, через которую как раз и можно связаться с нижними уровнями, а точнее с микропроцессором (железо) – нижний подуровень канального уровня.

Типичных представителей на этом уровне много. PPP (Point-to-Point) – это протокол для связи двух компьютеров напрямую. FDDI (Fiber Distributed Data Interface) – стандарт передаёт данные на расстояние до 200 километров. CDP (Cisco Discovery Protocol) – это проприетарный (собственный) протокол принадлежащий компании Cisco Systems, с помощью него можно обнаружить соседние устройства и получить информацию об этих устройствах.

Физический уровень

Физический уровень (physical layer) – самый нижний уровень, непосредственно осуществляющий передачу потока данных. Протоколы нам всем хорошо известны: Bluetooth, IRDA (Инфракрасная связь), медные провода (витая пара, телефонная линия), Wi-Fi, и т.д.

Пакет java.net

Сокеты — это логическое понятие, соответствующее разъемам, 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.

Сокет - это одна конечная точка двустороннего канала связи между двумя программами, работающими на разных компьютерах в сети. Сокет привязан к номеру порта, так что транспортный уровень может идентифицировать приложение, которому предназначены данные для отправки.

Класс Socket

В java существует специальный пакет "java.net", содержащий класс java.net.Socket. Socket в переводе означает "гнездо", название это было дано по аналогии с гнёздами на аппаратуре, теми самыми, куда подключают штепсели. Соответственно этой аналогии, можно связать два "гнезда", и передавать между ними данные. Каждое гнездо принадлежит определённому хосту (Host - хозяин, держатель). Каждый хост имеет уникальный IP (Internet Packet) адрес.

Гнёзда монтируются на порт хоста (port). Порт обозначается числом от 0 до 65535 и логически обозначает место, куда можно пристыковать (bind) сокет. Если порт на этом хосте уже занят каким-то сокетом, то ещё один сокет туда пристыковать уже не получится. Таким образом, после того, как сокет установлен, он имеет вполне определённый адрес, символически записывающийся так [host]:[port], к примеру - 127.0.0.1:8888 (означает, что сокет занимает порт 8888 на хосте 127.0.0.1)

Для того, чтобы облегчить жизнь, чтобы не использовать неудобозапоминаемый IP адрес, была придумана система DNS (DNS - Domain Name Service). Цель этой системы - сопоставлять IP адресам символьные имена. К примеру, адресу "127.0.0.1" в большинстве компьютеров сопоставленно имя "localhost" (в просторечье - "локалхост").

Локалхост, фактически, означает сам компьютер, на котором выполняется программа, он же - локальный компьютер. Вся работа с локалхостом не требует выхода в сеть и связи с какими-либо другими хостами.

Клиентский сокет

Итак, вернёмся к классу java.net.Socket. Наиболее удобно инициализировать его следующим образом:

```
public Socket(String host, int port) throws UnknownHostException, IOException
```

Сокетное соединение с сервером создается клиентом с помощью объекта класса Socket. При этом указывается IP-адрес сервера и номер порта. Если указано символьное имя домена, то Java преобразует его с помощью DNS-сервера к IPадресу. Например, если сервер установлен на этом же компьютере, соединение с сервером можно установить из приложения клиента с помощью инструкции:

```
Socket socketClient = new Socket("ИМЯ_СЕРВЕРА", 8030);
```

В строковой константе host можно указать как IP адрес сервера, так и его DNS имя. При этом программа автоматически выберет свободный порт на локальном компьютере и поставит в соответствие туда сокет, после чего будет предпринята попытка связаться с другим сокетом, адрес которого указан в параметрах инициализации. При этом могут возникнуть два вида исключений: неизвестный адрес хоста - когда в сети нет компьютера с таким именем или ошибка отсутствия связи с этим сокетом.

Так же полезно знать функцию

```
public void setSoTimeout(int timeout) throws SocketException
```

Эта функция устанавливает время ожидания (timeout) для работы с сокетом. Если в течение этого времени никаких действий с сокетом не произведено (имеется ввиду получение и отправка данных), то он самоликвидируется. Время задаётся в секундах, при установке timeout равным 0 сокет становится "вечным".

Для некоторых сетей изменение timeout невозможно или установлено в определённых интервалах (к примеру от 20 до 100 секунд). При попытке установить недопустимый timeout, будет выдано соответственное исключение.

Программа, которая открывает сокет этого типа, будет считаться клиентом, а программа-владелец сокета, к которому вы пытаетесь подключиться, далее будет называться сервером. Фактически, по аналогии гнездо-штепсель, программа-сервер - это и будет гнездо, а клиент как раз является тем самым штепселем.

Сокет сервера

Для создания и открытия сокета сервера в Java существует следующий класс: java.net.ServerSocket

Наиболее удобным инициализатором для него является следующий:

```
public ServerSocket(int port, int backlog, InetAddress bindAddr) throws IOException
```

Как видно, в качестве третьего параметра используется объект ещё одного класса - java.net.InetAddress Этот класс обеспечивает работу с DNS и IP именами, по этому вышеприведённый инициализатор в программах можно использовать так:

```
ServerSocket(port, 0, InetAddress.getByName(host)) throws IOException
```

Для этого типа сокета порт установки указывается прямо, поэтому, при инициализации, может возникнуть исключение, говорящее о том, что данный порт уже используется либо запрещён к использованию политикой безопасности компьютера.

После установки сокета, вызывается метод

```
public Socket accept() throws IOException
```

Эта функция погружает программу в ожидание того момента, когда клиент будет присоединяться к сокету сервера. Как только соединение установлено, функция возвратит объект класса Socket для общения с клиентом.

Датаграммные сокеты

Датаграммные сокеты работают по протоколу UDP, в Java за это отвечает класс java.net.DatagrammSocket

Датаграммные сокеты не гарантируют доставку пакетов данных, но работают быстрее потоковых и обеспечивают возможность широковещательной рассылки пакетов данных одновременно всем узлам сети. Для работы с такими сокетами приложение должно создать объект класса DatagramSocket, а также подготовить объект класса DatagramPacket. В него будет записан блок данных, принятый от партнера по сети.

В классе DatagramSocket имеются два конструктора, прототипы которых представлены ниже:

```
public DatagramSocket(int port);

public DatagramSocket();
```

Первый из этих конструкторов позволяет определить порт для сокета, второй предполагает использование любого свободного порта.

Канал, а также входные и выходные потоки создавать не нужно. Данные передаются и принимаются методами send и receive, определенными в классе DatagramSocket:

```
public void send(DatagramPacket p);

public void receive(DatagramPacket p);
```

В качестве параметра этим методам передается ссылка на пакет данных (соответственно, передаваемый и принимаемый), определенный как объект класса DatagramPacket.

После использования сокет нужно закрыть методом close:

```
public void close();
```

Перед тем как принимать или передавать данные с использованием методов receive и send вы должны подготовить объекты класса DatagramPacket. Метод receive запишет в такой объект принятые данные, а метод send - перешлет данные из объекта класса DatagramPacket узлу, адрес которого указан в пакете.

Подготовка объекта класса DatagramPacket для приема пакетов выполняется с помощью следующего конструктора:

```
public DatagramPacket(byte buf[], int length);
```

Этому конструктору передается ссылка на массив buf, в который нужно будет записать данные, и размер массива length.

Если вам нужно подготовить пакет для передачи, воспользуйтесь конструктором, который дополнительно позволяет задать адрес IP iaddr и номер порта port узла назначения:

```
public DatagramPacket(byte buf[], int length, InetAddress iaddr, int port);
```

Информация о том, в какой узел и на какой порт необходимо доставить пакет данных, хранится не в сокете, а в пакете, то есть в объекте класса DatagramPacket.

Литература и ссылки

1. Спецификация языка Java https://docs.oracle.com/javase/specs/jls/se8/html/index.html
2. Язык программирования Java SE 8. Подробное описание, 5-е издание, Джеймс Гослинг, Билл Джой, Гай Стил, Гилад Брача, Алекс Бакли; 672 стр., с ил.; 2015, 2 кв.; Вильямс.
3. Шилдт Г. Java 8. Полное руководство. 9-е издание. — М.: Вильямс, 2012. — 1377 с.
4. Эккель Б. Философия Java. 4-е полное изд. — СПб.: Питер, 2015. — 1168 с.

Вопросы для самоконтроля

1. Опишите основные сетевые протоколы.
2. Какие классы в Java отвечают за сетевое взаимодействие?