网络模型

OSI 参考模型

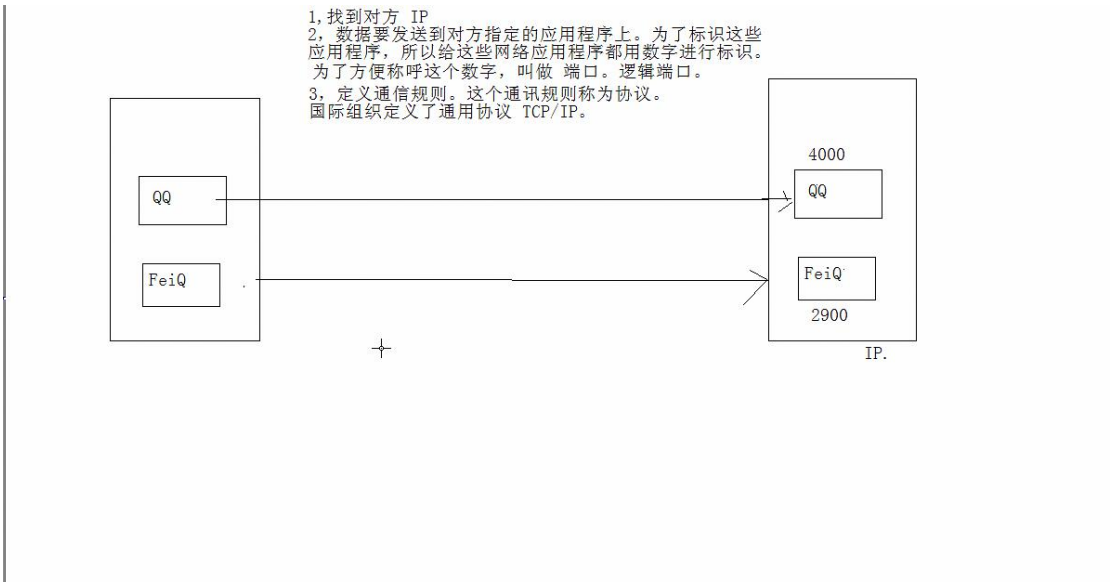
TCP/IP 参考模型

网络通讯要素

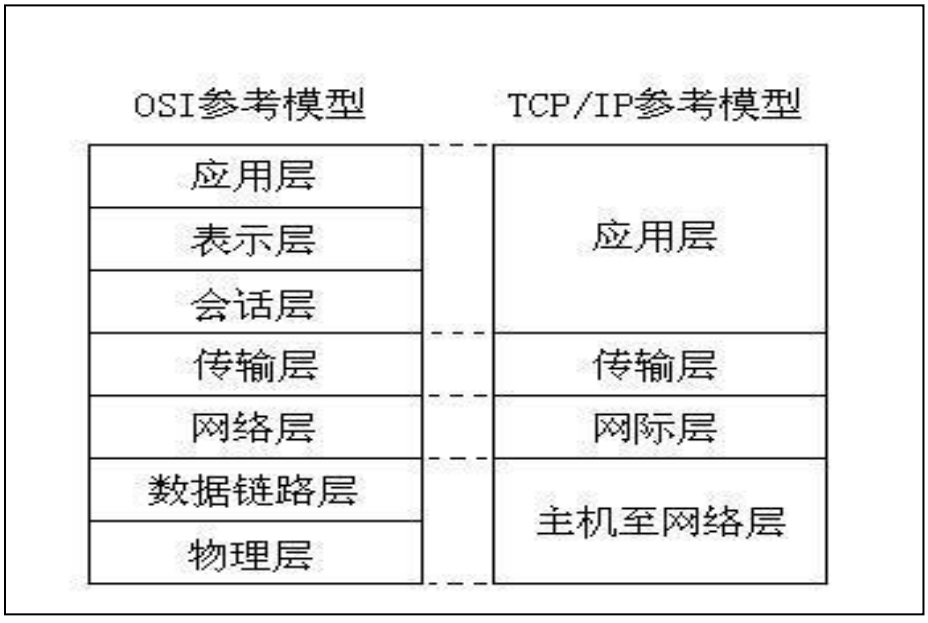
IP 地址

端口号

传输协议



127.0.0.1



这堂课 传输层 网络层
Java web 开发：应用层

传输层 TCP UDP
网际层 IP
应用层 http ftp

IP 地址:InetAddress
网络中设备的标识
不易记忆，可用主机名
本地回环地址：127.0.0.1 主机名：localhost

端口号
用于标识进程的逻辑地址，不同进程的标识
有效端口：0~65535，其中 0~1024 系统使用或保留端口。

传输协议
通讯的规则
常见协议：TCP，UDP

Java.net.*
InetAddress

UDP
将数据及源和目的封装成数据包中，不需要建立连接
每个数据报的大小在限制在 64k 内
因无连接，是不可靠协议
不需要建立连接，速度快

TCP
建立连接，形成传输数据的通道。
在连接中进行大数据量传输
通过三次握手完成连接，是可靠协议
必须建立连接，效率会稍低

Socket 就是为网络服务提供的一种机制。
通信的两端都有 Socket。
网络通信其实就是 Socket 间的通信。
数据在两个 Socket 间通过 IO 传输。

/*
需求：通过 udp 传输方式，将一段文字数据发送出去。，
定义一个 udp 发送端。
思路：
1，建立 updsocket 服务。
2，提供数据，并将数据封装到数据包中。
3，通过 socket 服务的发送功能，将数据包发出去。
4，关闭资源。

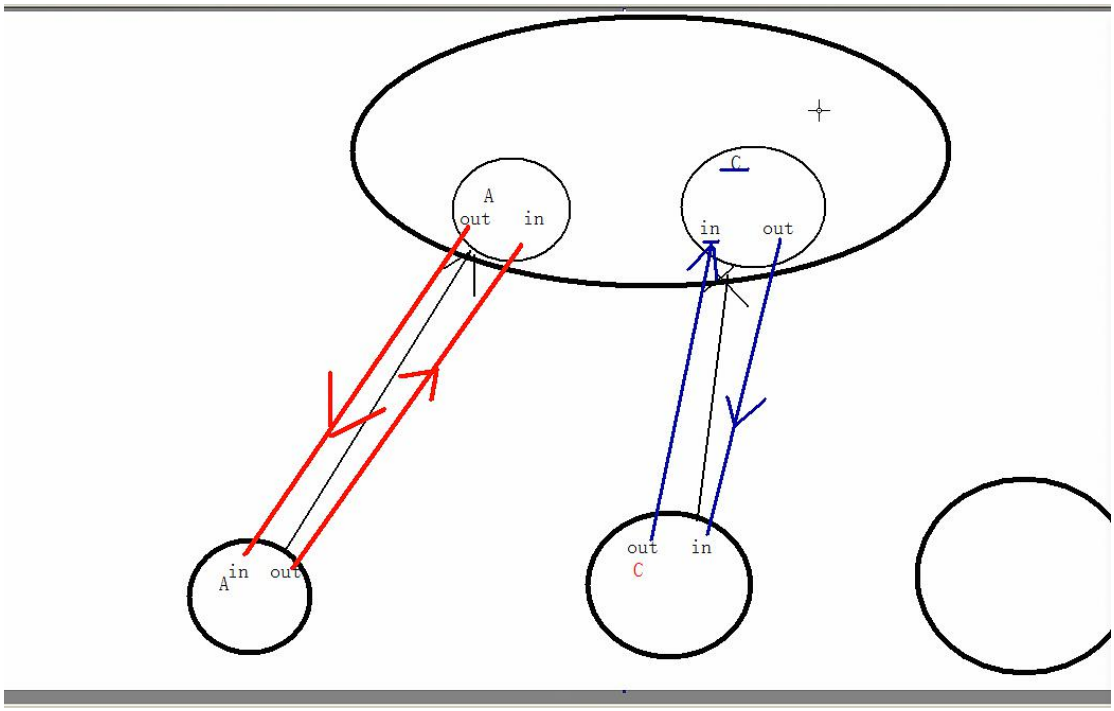
*/

/*
需求：
定义一个应用程序，用于接收 udp 协议传输的数据并处理的。

定义 udp 的接收端。
思路：
1，定义 udpsocket 服务。通常会监听一个端口。其实就是给这个接收网络应用程序定义数字标识。
　　方便于明确哪些数据过来该应用程序可以处理。

2，定义一个数据包，因为要存储接收到的字节数据。
因为数据包对象中有更多功能可以提取字节数据中的不同数据信息。
3，通过 socket 服务的 receive 方法将收到的数据存入已定义好的数据包中。
4，通过数据包对象的特有功能。将这些不同的数据取出。打印在控制台上。
5，关闭资源。

*/




```
/*
演示 tcp 传输。
1,tcp 分客户端和服务端。
2，客户端对应的对象是 Socket。
      服务端对应的对象是 ServerSocket。
*/


/*
客户端，
通过查阅 socket 对象，发现在该对象建立时，就可以去连接指定主机。
因为 tcp 是面向连接的。所以在建立 socket 服务时，
就要有服务端存在，并连接成功。形成通路后，在该通道进行数据的传输。

需求：给服务端发送给一个文本数据。
步骤：
1，创建 Socket 服务。并指定要连接的主机和端口。
*/
import java.io.*;
import java.net.*;
class    TcpClient
{
      public static void main(String[] args) throws Exception
      {
            //创建客户端的 socket 服务。指定目的主机和端口
```

```
            Socket s = new Socket("192.168.1.254",10003);
            //为了发送数据，应该获取 socket 流中的输出流。
            OutputStream out = s.getOutputStream();
            out.write("tcp ge men lai le ".getBytes());
            s.close();
        }
}


/*
需求：定义端点接收数据并打印在控制台上。

服务端:
1，建立服务端的 socket 服务。ServerSocket();
      并监听一个端口。
2，获取连接过来的客户端对象。
      通过 ServerSokcet 的  accept 方法。没有连接就会等，所以这个方法阻塞式的。
3，客户端如果发过来数据，那么服务端要使用对应的客户端对象，并获取到该客户端对象的读取流来读取发过来的数据。
      并打印在控制台。
4，关闭服务端。（可选）


*/
class    TcpServer
{
      public static void main(String[] args) throws Exception
      {
            //建立服务端 socket 服务。并监听一个端口。
            ServerSocket ss = new ServerSocket(10003);

            //通过 accept 方法获取连接过来的客户端对象。
            while(true)
            {
            Socket s = ss.accept();

            String ip = s.getInetAddress().getHostAddress();
            System.out.println(ip+".....connected");

            //获取客户端发送过来的数据，那么要使用客户端对象的读取流来读取数据。
            InputStream in = s.getInputStream();

            byte[] buf = new byte[1024];
            int len = in.read(buf);
```

```
            System.out.println(new String(buf,0,len));

            s.close();//关闭客户端.
            }
            //ss.close();
        }
}
```

s.shutdownOutput();//关闭客户端的输出流。相当于给流中加入一个结束标记-1.

/*

服务端

这个服务端有个局限性。当 A 客户端连接上以后。被服务端获取到。服务端执行具体流程。
这时 B 客户端连接，只有等待。
因为服务端还没有处理完 A 客户端的请求，还有循环回来执行下次 accept 方法。所以
暂时获取不到 B 客户端对象。

那么为了可以让多个客户端同时并发访问服务端。
那么服务端最好就是将每个客户端封装到一个单独的线程中，这样，就可以同时处理多个客户端请求。

如何定义线程呢？

只要明确了每一个客户端要在服务端执行的代码即可。将该代码存入 run 方法中。
*/

/*
客户端通过键盘录入用户名。
服务端对这个用户名进行校验。

如果该用户存在，在服务端显示 xxx，已登陆。
并在客户端显示 xxx，欢迎光临。

如果该用户存在，在服务端显示 xxx，尝试登陆。
并在客户端显示 xxx，该用户不存在。

最多就登录三次。
*/

/*
演示客户端和服务端。

1，
客户端：浏览器 (telnet)
服务端：自定义。

2，
客户端：浏览器。
服务端：Tomcat 服务器。

3，
客户端：自定义。(图形界面)
服务端：Tomcat 服务器。

*/

http 应用层　这个是请求数据头

```
GET / HTTP/1.1
Accept: text/html, application/xhtml+xml, image/jxr, */*
Accept-Language: en-US,en-GB;q=0.8,en;q=0.6,zh-Hans-CN;q=0.4,zh-Hans;q=0.2
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36 Edge/16.16299
Accept-Encoding: gzip, deflate
Host: 192.168.1.101:10004
Connection: Keep-Alive
```

URL

```
import java.net.*;
class URLDemo
{
	public static void main(String[] args) throws MalformedURLException
	{
		URL url = new URL("http://192.168.1.254/myweb/demo.html?name=haha&age=30");

		System.out.println("getProtocol() :"+url.getProtocol());
```

```
            System.out.println("getHost() :"+url.getHost());
            System.out.println("getPort() :"+url.getPort());
            System.out.println("getPath() :"+url.getPath());
            System.out.println("getFile() :"+url.getFile());
            System.out.println("getQuery() :"+url.getQuery());

            /*int port = getPort();
            if(port==-1)
                  port = 80;

            getPort()==-1
            */
      }
}
/*
 String getFile()
              获取此 URL 的文件名。
 String getHost()
              获取此 URL 的主机名（如果适用）。
 String getPath()
              获取此 URL 的路径部分。
 int getPort()
              获取此 URL 的端口号。
 String getProtocol()
              获取此 URL 的协议名称。
 String getQuery()
              获取此 URL 的查询部

*/
```

上升到应用层

```
            ta.setText("");
            String urlPath = tf.getText();//http://192.168.1.254:8080/myweb/demo.html

            URL url = new URL(urlPath);

            URLConnection conn = url.openConnection();

            InputStream in = conn.getInputStream();
```

ServerSocket(int port, int backlog)

connect(**SocketAddress** endpoint)