网络编程

一、 什么是网络编程

> 网络编程的本质是两个设备之间的数据交换，当然，在计算机网络中，设备主要指计算机。数据传递本身没有多大的难度，不就是把一个设备中的数据发送给两外一个设备，然后接受另外一个设备反馈的数据。
>
> 现在的网络编程基本上都是基于请求/响应方式的，也就是一个设备发送请求数据给另外一个，然后接收另一个设备的反馈。
>
> 在网络编程中，发起连接程序，也就是发送第一次请求的程序，被称作客户端(Client)，等待其他程序连接的程序被称作服务器(Server)。客户端程序可以在需要的时候启动，而服务器为了能够时刻响应连接，则需要一直启动。例如以打电话为例，首先拨号的人类似于客户端，接听电话的人必须保持电话畅通类似于服务器。

> 总结： 通过编码的方式让不同计算机之间通过网络相互通信(传递数据)

二、 网络编程要解决的核心的问题

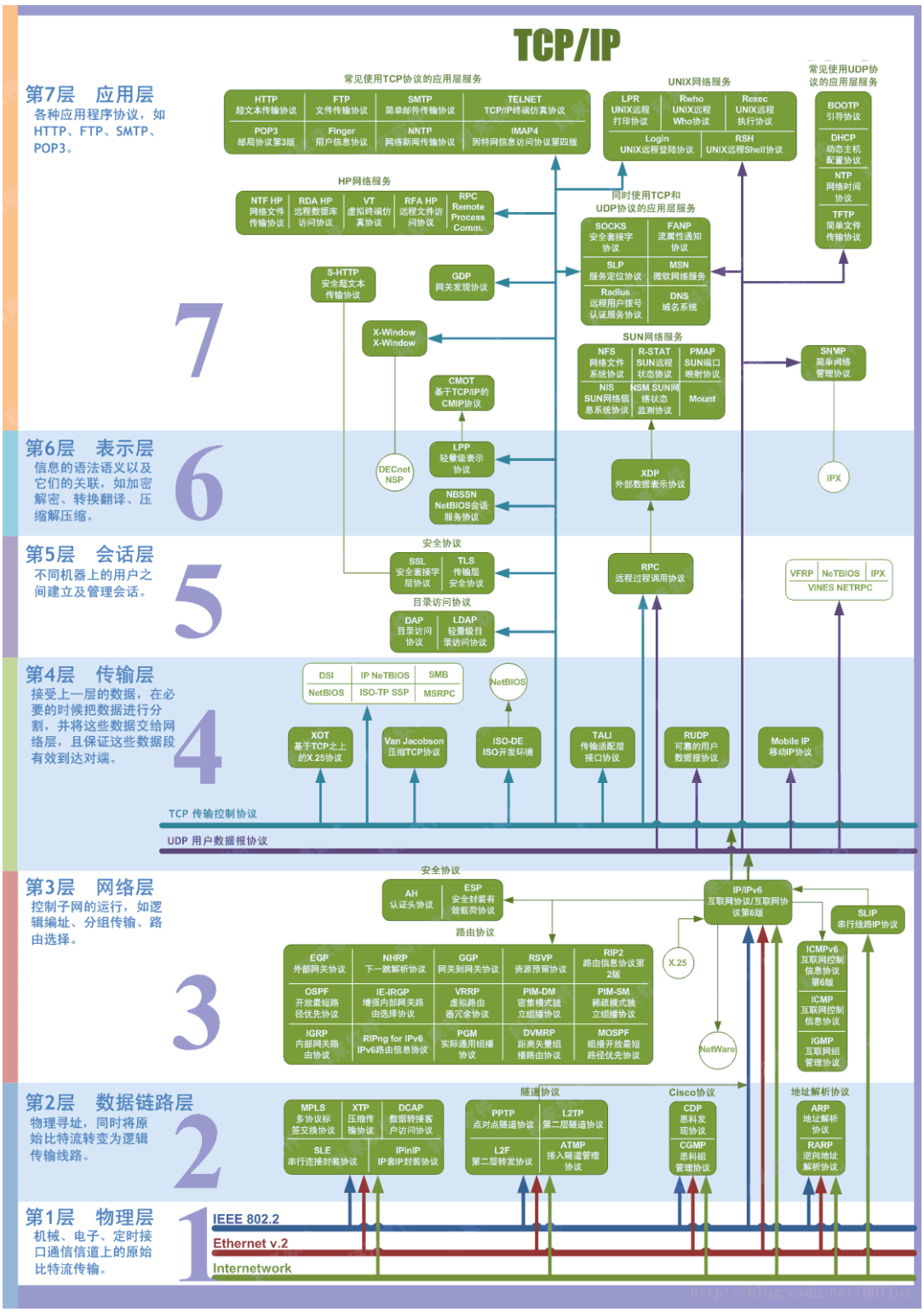
> 1. 寻址: 使用ip+端口
> 2. 协议: 数据的传输规则|方式

三、 常见的协议有哪些

> 1). UDP 协议
>
> 面向非连接的协议,传递数据时不关心连接状态直接发送,他的传输效率高,传递的数据不安全
>
> 2). TCP/IP 协议
>
> 面向连接的协议,传递数据时关系连接的状态,连接成功才会发送数据,他的传输效率相对于UDP协议要低,会通过三次握手机制保证数据的完整性
>
> 3). Http 协议
>
> 属于应用层协议,层面比较高,http协议底层还是通过tcp协议传输

四、 网络中的OSI模型

*OSI*是Open System Interconnection的缩写，意为开放式系统互联. OSI 七层模型通过七个层次化的结构模型使不同的系统不同的网络之间实现可靠的通讯，因此其最主要的功能就是帮助不同类型的主机实现数据传输



注意: 所处的层越低传输效率越高

五、 常见的网络编程

浏览器 ----------> 服务器

chorme tomcat

c/s:访问某个服务器端的时候,需要特定的客户端

b/s:浏览器,是一个公共的客户端,通过浏览器可以给不同的服务器发送数据

客户端:

1 c/s模式中对应的客户端

2 b/s中的浏览器,通用的客户端

3 通过代码实现: http-client(模拟浏览器)

4 自己书写一个客户端

服务器:

自己编写

六、 java中网络编程的实现方式

java中实现网络编程,也叫(Socket)编程,通过java实现网络编程主要有以下几种方式

1 java BIO:基于java阻塞io实现网络编程(Block IO 同步阻塞IO)

2 java NIO:基于java的同步非阻塞IO实现网络编程(New IO 同步非阻塞IO)

七、 java中网络编程的开发思路

使用java语言开发网络编程,主要用到其核心类:Socket,翻译为:套接字,socket类中封装了系列网络编程的api

1 开发服务端

ServerSocket:专门用来书写服务器端

1 服务器端一直活着

2 服务器端的accept()能够重复调用

2 开发客户端

Socket:专门用来书写客户端

八、 网络编程的实现

1.使用BIO中网络编程开发服务器端

```
public static void main(String[] args) throws IOException {
    //创建serversocket对象
```

```
    ServerSocket serverSocket = new ServerSocket(8989);
    System.out.println("服务已经启动");
    //等待连接
    Socket socket = serverSocket.accept();
    InputStream is = socket.getInputStream();
    //获取客户端的输入流
    DataInputStream dataInputStream = new DataInputStream(is);
    System.out.println(dataInputStream.readUTF());
    //获取客户端的输出流
    OutputStream outputStream = socket.getOutputStream();
    DataOutputStream dataOutputStream = new
DataOutputStream(outputStream);
    dataOutputStream.writeUTF("服务端响应结果.....");
}
```

2.使用BIO中网络编程开发客户端

```
public static void main(String[] args) throws IOException {
    //创建客户端对象
    Socket socket = new Socket();
    socket.connect(new InetSocketAddress("127.0.0.1",8989));
    //获取输出流
    OutputStream outputStream = socket.getOutputStream();
    DataOutputStream dataOutputStream = new
DataOutputStream(outputStream);
    dataOutputStream.writeUTF("你好,服务端servers");
    //获取输入流
    InputStream inputStream = socket.getInputStream();
    DataInputStream dataInputStream = new DataInputStream(inputStream);
    System.out.println(dataInputStream.readUTF());
}
```

九、 网络编程服务端之支持多客户端访问

```
public static void main(String[] args) throws IOException {
    //创建serversocket对象
    ServerSocket serverSocket = new ServerSocket(8989);
    System.out.println("服务已经启动");
    while(true){
        //等待连接
        Socket socket = serverSocket.accept();
        InputStream is = socket.getInputStream();
        //获取客户端的输入流
        DataInputStream dataInputStream = new DataInputStream(is);
        System.out.println(dataInputStream.readUTF());
        //获取客户端的输出流
        OutputStream outputStream = socket.getOutputStream();
```

```
        DataOutputStream dataOutputStream = new
DataOutputStream(outputStream);
        dataOutputStream.writeUTF("服务端响应结果.....");
    }
}
```

十、 服务端的多线程操作

```
public static void main(String[] args) throws IOException {
    //创建serversocket对象
    final ServerSocket serverSocket = new ServerSocket(8989);
    System.out.println("服务已经启动");
    while(true){
        Thread thread = new Thread(new Runnable() {
            @Override
            public void run() {
                //等待连接
                try {
                    Socket socket = serverSocket.accept();
                    System.out.println(socket.getLocalAddress());
                    InputStream is = socket.getInputStream();
                    //获取客户端的输入流
                    DataInputStream dataInputStream = new
DataInputStream(is);
                    System.out.println(dataInputStream.readUTF() +
Thread.currentThread().getId());
                    //获取客户端的输出流
                    OutputStream outputStream = socket.getOutputStream();
                    DataOutputStream dataOutputStream = new
DataOutputStream(outputStream);
                    dataOutputStream.writeUTF("服务端响应结
果.....:"+Thread.currentThread().getId());
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();
    }
}
```

十一、 线程池(Executors)的服务端

```
public static void main(String[] args) throws IOException {
```

```
    //创建线程池
    ExecutorService pool = Executors.newFixedThreadPool(10);
    //创建serversocket对象
    final ServerSocket serverSocket = new ServerSocket(8989);
    System.out.println("服务已经启动");
    while(true){
       pool.submit(new Runnable() {
           @Override
           public void run() {
               //等待连接
               try {
                   Socket socket = serverSocket.accept();
                   System.out.println(socket.getLocalAddress());
                   InputStream is = socket.getInputStream();
                   //获取客户端的输入流
                   DataInputStream dataInputStream = new
DataInputStream(is);
                   System.out.println(dataInputStream.readUTF() +
Thread.currentThread().getId());
                   //获取客户端的输出流
                   OutputStream outputStream = socket.getOutputStream();
                   DataOutputStream dataOutputStream = new
DataOutputStream(outputStream);
                   dataOutputStream.writeUTF("服务端响应结
果.....:"+Thread.currentThread().getId());
               } catch (IOException e) {
                   e.printStackTrace();
               }
           }
       });
    }
}
```