第十二章 网络编程

1 基本概念

1.1 计算机网络

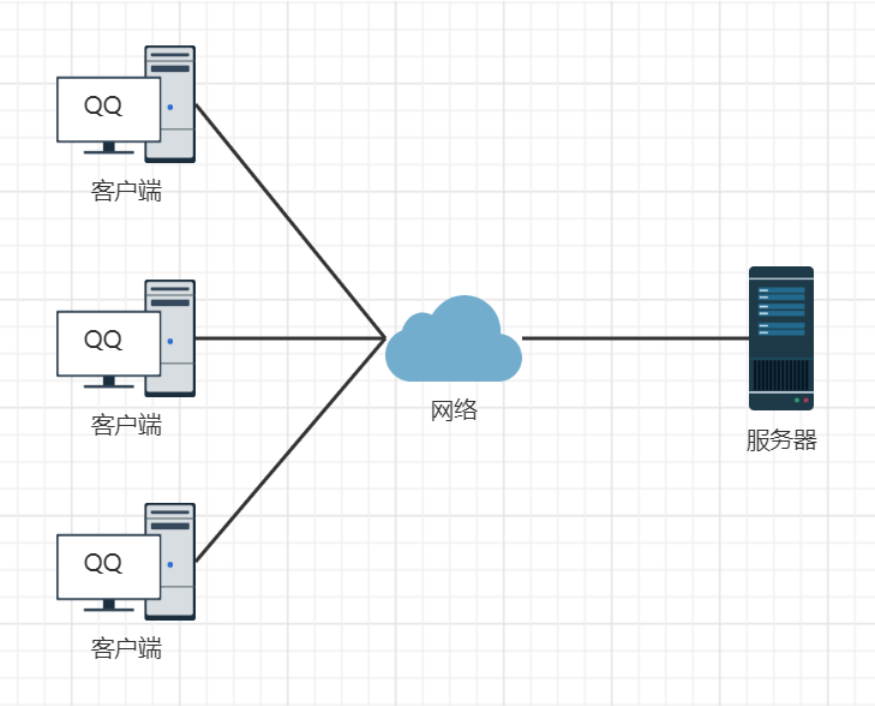
　也称计算机通信网，其是利用通信线路将地理上分散的、具有独立功能的计算机系统和通信设备按不同的形式连接起来，以功能完善的网络软件及协议实现资源共享和信息传递的系统。最简单的计算机网络就只有两台计算机和连接它们的一条链路，即两个节点和一条链路。

　通过编程的方式，使得计算机网络中不同计算机上的应用程序间能够进行数据的传输，这就是网络编程要做的事情。

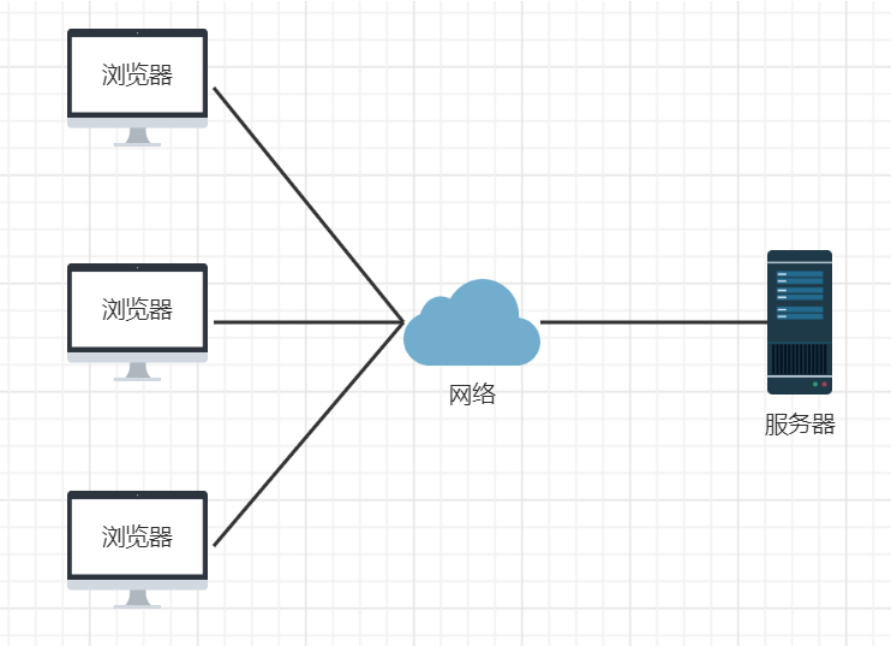
1.2 软件结构

常见的软件结构有C/S和B/S

- Client/Server（C/S结构），表示 客户端/服务器 的软件结构，例如QQ、微信、百度网盘客户端等，只要是需要我们下载安装，并且和服务器通信的这一类软件，都属于C/S的软件结构。



- Browser/Server（B/S结构），表示 浏览器/服务器 的软件结构，例如淘宝网、新浪新闻、凤凰网等，只要是需要使用浏览器，并且和服务器通信的这一类软件，都属于B/S的软件结构



C/S和B/S对比：

- C/S在图形的表现能力上以及运行的速度上肯定是强于B/S的

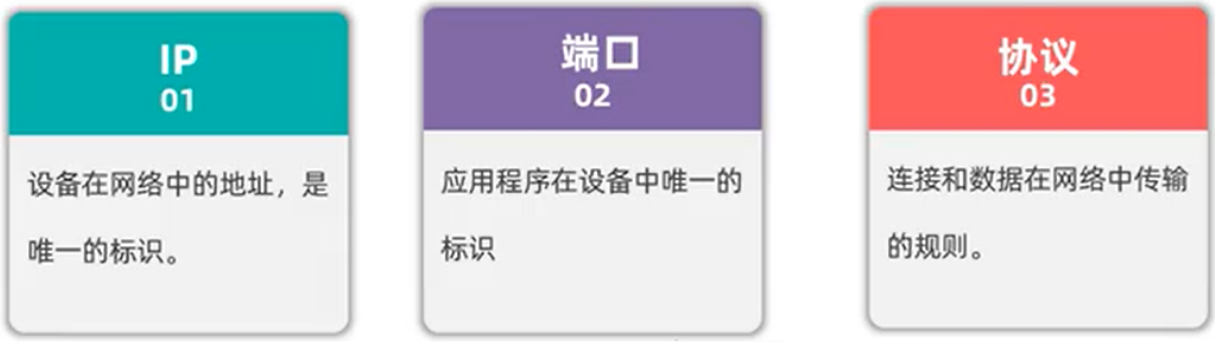
- C/S/需要运行专门的客户端，并且它不能跨平台，用c++在windows下写的程序肯定是不能在linux下运行
- B/S不需要专门的客户端，只要系统中安装了浏览器即可访问，方便用户的使用。
- B/S是基于网页语言的、与操作系统无关，所以跨平台也是它的优势

随着网页技术以及浏览器的进步，B/S在表现能力上的处理以及运行的速度上会越来越快，所以现在越来越多的C/S结构的软件，也推出了对应B/S的版本，例如webQQ，在线文档工具、在线画图工具等。同时也包括很多网页版的游戏，也是随着前端技术的发展，慢慢出现的。

> 无论是C/S结构的软件，还是B/S结构的软件，都必须依赖网络编程。

1.3 通信三要素

如果两台计算机上的应用程序能够实现通信，那么必须解决3个问题，找到对方计算机，找到对方应用程序，采用相同的通信协议。



- IP地址

可认为IP地址是个标识号，通过这个标识号能够找到网络世界中唯一的那个计算机。

- 端口

  端口号可以用来标识计算机中唯一的那个应用程序。网络的通信，本质上是两个应用程序的通信。每台计算机都有很多的应用程序，在网络通信时，就采用端口号进行区分这些应用程序。

- 协议

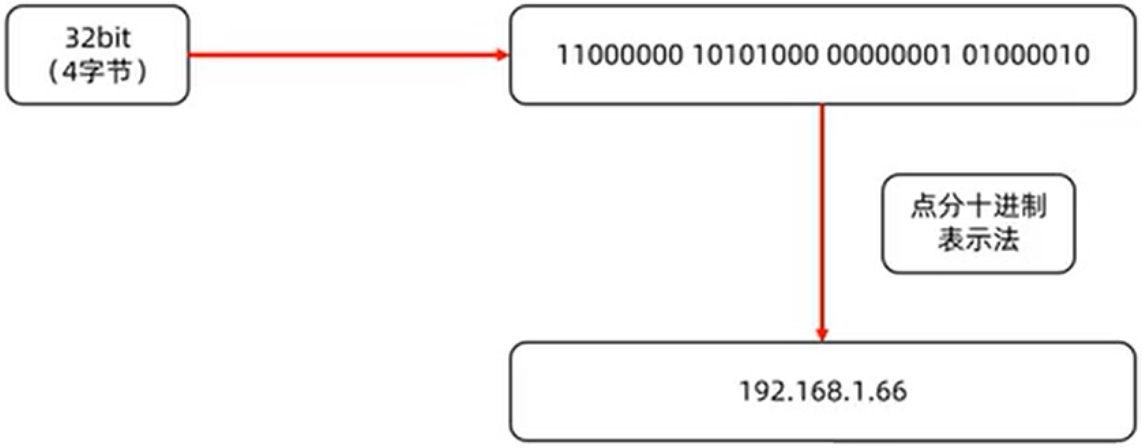
  当我们和其他人沟通交流的时候都要使用互相能听懂的语言。计算机也一样，计算机与计算机通过网络进行数据和信息交换的时候，也要使用同样的“语言”，这个语言被称为网络通讯协议。

  网络通信协议对数据的传输格式、传输速率、传输步骤等做了统一规定，通信双方必须同时遵守才能完成数据交换。常见的协议有UDP协议和TCP协议。

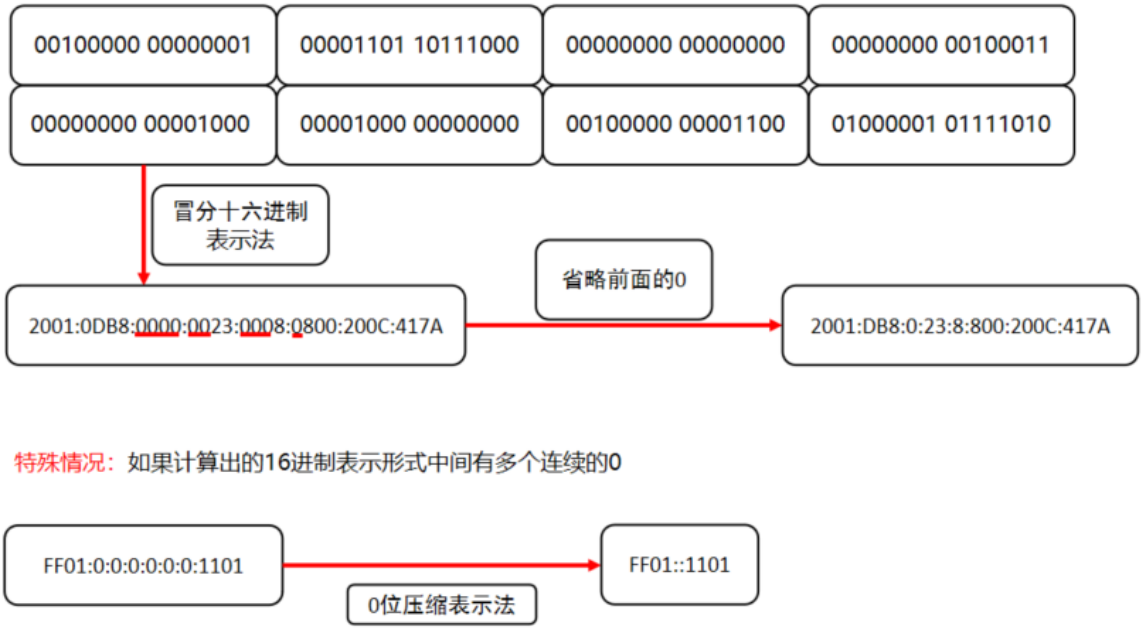
2 IP地址

全称”互联网协议地址”，也称IP地址。是分配给上网设备的数字标签。常见的IP分类为：ipv4和ipv6

IPv4：共32位，表示范围43亿左右，一般使用点分4段表示法



IPv6：共128位，表示范围更大，号称可以为地球上每一粒沙子分配一个IP



IP域名：

本质上也是一个IP地址，可读性更好，更容易记忆，需要使用dns域名解析服务器解析。

公网、内网IP：

- 公网IP，是可以连接互联网的IP地址
- 内网IP，也叫局域网IP，只能组织机构内部使用

  192.168.开头的就是常见的局域网地址，范围即为192.168.0.0--192.168.255.255，专门为组织机构内部使用

DOS常用命令：

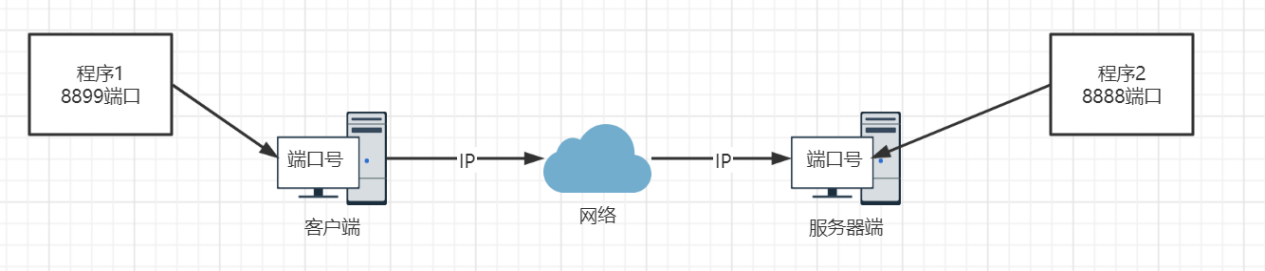
- ipconfig：查看本机IP地址
- ping IP地址：检查网络是否连通

特殊IP地址：

- 127.0.0.1：是回送地址，可以代表本机地址，一般用来测试使用

3 端口号

端口号用来标记正在计算机设备上运行的应用程序，其范围是0~65535。其中，0~1023之间的端口号用于系统内部的使用，我们自己普通的应用程序要使用1024以上的端口号即可，同时也要避免和一些知名应用程序默认的端口冲突，例如：oracle启动后默认占用端口号1521，mysql启动后默认占用端口号3306，redis启动后默认占用端口号6379，tomcat启动后默认占用端口号8080。



> 程序1，在客户端电脑中的内存中运行着，并且占用了端口号8899
>
> 程序2，在服务器端电脑中的内存中运行着，并且占用了端口号8888
>
> 俩个程序，通过IP+端口号的方式，找到对方进行通信，传输信息

端口分类：

- 周知端口

  0~1023，被预先定义的知名应用占用，如: HTTP占用80，FTP占用21，MySQL占用3306

- 注册端口

  1024~49151，分配给用户进程或某些应用程序

- 动态端口

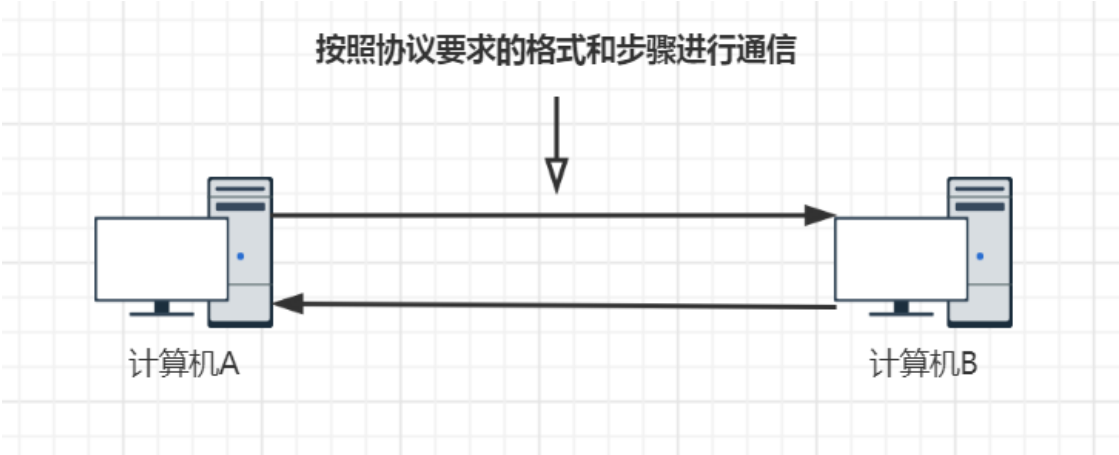
  49152到65535，之所以称为动态端口，是因为它 一般不固定分配某种进程，而是动态分配

注意事项：

- **自己开发的程序一般选择使用注册端口**
- **同一时刻一个设备中两个程序的端口号不能重复，否则出错**

4 通信协议

计算机网络中，不同设备进行连接和通信的规则被称为**网络通信协议**。通信协议，对俩台计算机之间所传输数据的传输格式、传输步骤等做了统一规定要求，通信双方必须同时遵守才能完成数据交换。



OSI（Open System Interconnect），即开放式网络互连标准。 一般叫OSI参考模型，是ISO（国际标准化组织）在1985年研究的网络互连模型，它共包含七层，具体可参考下图。

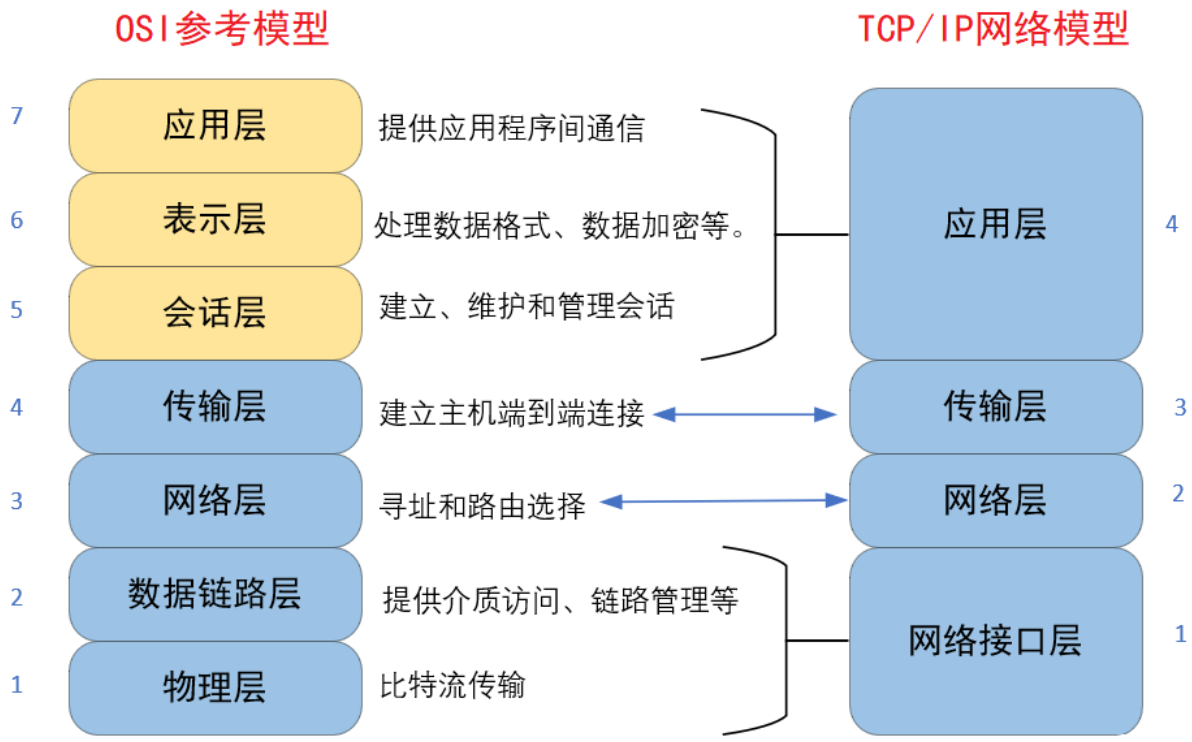
疑问：为什么要分层?

要解决计算机网络中数据的传输，涉及的问题很多很复杂，分层可以将大问题拆分成小问题，更利于问题的解决。

TCP/IP网络模型，是事实上的国际标准，它被简化为了四个层，从下到上分别依次是应用层、传输层、网络层、网络接口层。

- 应用层：主要负责应用程序的协议，例如HTTP协议、FTP协议等。

- 传输层：主要使网络程序进行通信，在进行网络通信时，可以采用TCP协议，也可以采用UDP协议。
- 网络层：网络层是整个TCP/IP协议的核心，它主要用于将传输的数据进行分组，将分组数据发送到目标计算机或者网络。
- 链路层：链路层是用于定义物理传输通道，通常是对某些网络连接设备的驱动协议，例如针对光纤、网线提供的驱动。



OSI网络参考模型	TCP/IP网络模型	各层对应	面向操作
应用层	应用层	HTTP、FTP、SMTP...	应用程序需要关注的：浏览器，邮箱。 程序员一般在这一层开发
表示层			
会话层			
传输层	传输层	UDP、TCP...	选择使用的TCP，UDP协议
网络层	网络层	IP...	封装源和目标IP
数据链路层	数据链路层+物理	比特流...	物理设备中传输
物理层			

5 TCP和UDP

虽然完整的通信过程比较复杂，但是JavaAPI中把这些通信实现的细节进行了封装，使得我们可以直接使用相应的类和接口，来进行网络程序开发，而不用考虑通信的细节。

`java.net`包中对常见的俩种通信协议进行了封装和支持：UDP和TCP

- UDP，用户数据报协议(User Datagram Protocol) （**了解**）

  UDP是**无连接通信协议**，在数据传输时，数据的发送端和接收端不建立连接，也不能保证对方能接收成功。

  例如，当一台计算机向另外一台计算机发送数据时（UDP），发送端不会确认接收端是否存在，就会直接发出数据，同样接收端在收到数据时，也不会向发送端反馈是否收到数据。

  由于使用UDP协议消耗资源小，**通信效率高**，所以通常都会用于音频、视频和普通数据的传输，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。

  但是在传输重要数据时，不建议使用UDP协议，因为它**不能保证数据传输的完整性**。

- TCP，传输控制协议 (Transmission Control Protocol) （**重要**）

  TCP协议是**面向连接**的通信协议，即传输数据之前，在发送端和接收端建立连接，然后再传输数据，它提供了两台计算机之间**可靠的、无差错的**数据传输。

  在TCP连接中，将计算机明确划分为客户端与服务器端，并且由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”的过程，四次挥手断开连接。

TCP的三次握手：

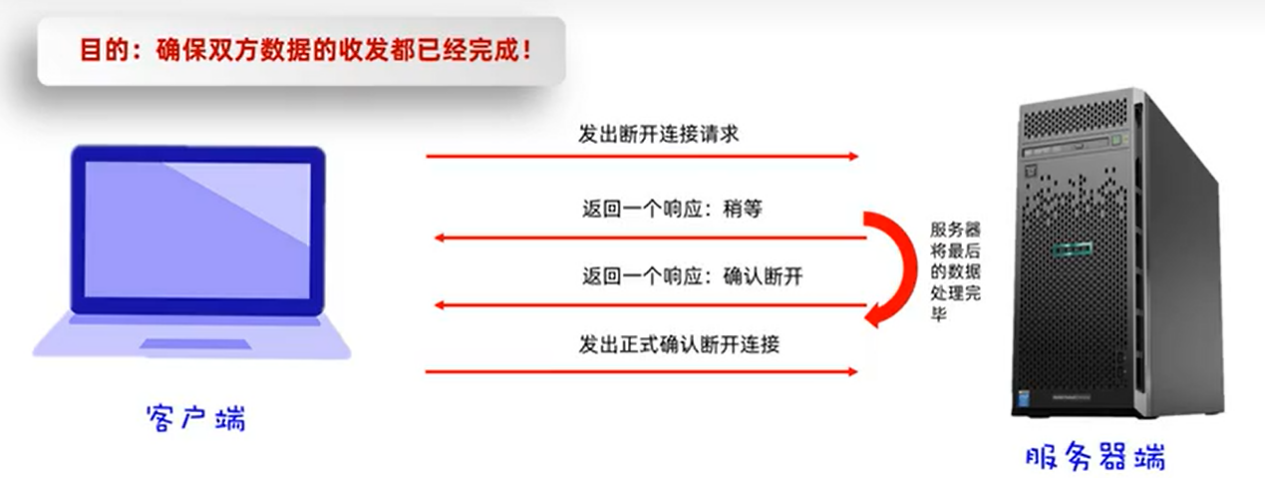
TCP协议中，在发送数据的准备阶段，客户端与服务器之间的三次交互，以保证连接的可靠

- 第一次握手，客户端向服务器端发出连接请求，等待服务器确认
- 第二次握手，服务器端向客户端回送一个响应，通知客户端收到了连接请求
- 第三次握手，客户端再次向服务器端发送确认信息，确认连接

> 完成上述的三次握手后，客户端和服务器端的连接就已经建立了，在这个安全的、可靠的连接基础之上，就可以开始进行数据传输了。
>
> TCP协议应用的十分广泛，例如下载文件、浏览网页、远程登录等。

TCP的四次挥手：

用于断开连接



6 TCP网络编程

6.1 概述

在TCP通信协议下，计算机网络中不同设备上的应用程序之间可以通信，通信时需严格区分客户端（Client）与服务器端（Server）。

在Java中，对于这样基于TCP协议下连接通信的客户端和服务端，分别进行了抽象：

- `java.net.ServerSocket`类表示服务端
- `java.net.Socket`类表示客户端

使用`Socket`和`ServerSocket`进行的编程，也称为套接字编程。

6.2 通信流程

TCP客户端和服务器进行通信，其通信流程是固定的，具体如下：

服务器端：

- 创建ServerSocket（需绑定端口，方便客户端连接）

- 调用ServerSocket对象的accept()方法接收一个客户端请求，得到一个Socket
- 调用Socket的getInputStream()和getOutputStream()方法获取和客户端相连的IO流

  输入流可以读取客户端发送过来的数据

  输出流可以发送数据到客户端
- 操作完成，关闭资源

构造方法

方法名	说明
ServletSocket(int port)	创建绑定到指定端口的服务器套接字

相关方法

方法名	说明
Socket accept()	监听要连接到此的套接字并接受它

注意：accept方法是阻塞的，作用是等待客户端连接，如果有客户端连接则立马返回，如果没有客户端连接则一致阻塞等待。

客户端：

- 创建Socket连接服务端（需指定服务器ip地址、端口），找对应的服务器进行连接
- 调用Socket的getInputStream()和getOutputStream()方法获取和服务端相连的IO流

输入流可以读取服务端输出流写出的数据

输出流可以写出数据到服务端的输入流

- 操作完成，关闭资源

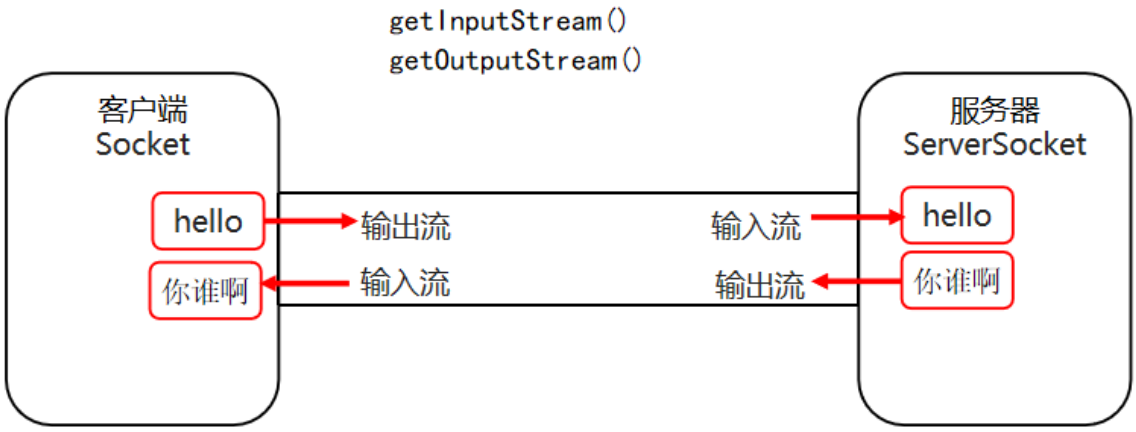
> 注意，在整个过程中，服务端不能主动连接客户端，必须由客户端先行发起连接才行

构造方法

方法名	说明
Socket(InetAddress address,int port)	创建流套接字并将其连接到指定IP指定端口号
Socket(String host, int port)	创建流套接字并将其连接到指定主机上的指定端口号

相关方法

方法名	说明
InputStream getInputStream()	返回此套接字的输入流
OutputStream getOutputStream()	返回此套接字的输出流



6.3 基础案例

搭建TCP客户端，发送信息到服务器

```
1  package com.briup.chap12;
2
3  public class Test063_TcpClient {
4      public static void main(String[] args) throws Exception {
5          //1.创建Socket对象(指定服务器ip port) 连接到服务器
6          Socket socket = new Socket("127.0.0.1",8989);
7          System.out.println("成功连接到 8989服务器，socket: " +
   socket);
8
9          //2.获取数据传输的IO流
10         InputStream is = socket.getInputStream();
11         OutputStream os = socket.getOutputStream();
12
13         //3.数据传输【跟服务器 业务配合 起来】
14         // 先发送数据给服务器
15         os.write("hello server, 我是Tcp客户端".getBytes());
16         System.out.println("数据发送完成！");
17
18         // 再接收从服务器返回的数据
```

```
        byte[] buff = new byte[1024];
        int len = is.read(buff);
        System.out.println("read: " + new String(buff,0,len));

        //4.关闭资源
        os.close();
        is.close();
        socket.close();
    }
}
```

搭建TCP服务器端，接收客户端发送的信息，并返回数据给客户端

```
package com.briup.chap12;

public class Test063_TcpServer {
    public static void main(String[] args) throws IOException
{
        // 1.创建ServerSocket(需绑定端口，方便客户端连接)
        ServerSocket server = new ServerSocket(8989);
        System.out.println("服务器成功启动，端口 8989 ...");

        //2.调用ServerSocket的accept()方法接收客户端请求，得到一个
Socket
        //  如果客户端成功连接到服务器，直接返回 套接字对象(通信)
        //  如果没有客户端连接  服务器，则阻塞
        Socket socket = server.accept();
        System.out.println("客户端成功连接: " + socket);

        //3.获取网络通信的IO流对象
        InputStream is = socket.getInputStream();
        OutputStream os = socket.getOutputStream();

        //4.数据传输
```

```
20          //  先读取 客户端发送过来的数据 并输出
21          byte[] buff = new byte[1024];
22          int len = is.read(buff);
23          System.out.println("成功读取: ");
24          System.out.println(new String(buff,0,len));
25
26          //  再返回一条数据给 客户端
27          System.out.println("即将发送数据 给 客户端...");
28          os.write("hello world client".getBytes());
29
30          System.out.println("数据成功发送!");
31
32          //5.关闭资源
33          os.close();
34          is.close();
35          socket.close();
36          server.close();
37      }
38  }
```

6.4 反转案例

搭建一个TCP客户端，从键盘录入整行数据（遇到quit结束录入）然后发送给服务器，再接收服务器返回的数据并输出。

```
1   package com.briup.chap12;
2
3   public class Test064_ReversalClient {
4       public static void main(String[] args) throws Exception {
5           //1.创建Socket对象(指定服务器ip port) 连接到服务器
6           Socket socket = new Socket("127.0.0.1",8989);
7           System.out.println("成功连接到 8989服务器，socket: " +
    socket);
8
```

```
9           //2.获取数据传输的IO流
10          InputStream is = socket.getInputStream();
11          OutputStream os = socket.getOutputStream();
12
13          //定义增强流 更好的实现功能
14          // 打印流 写
15          PrintStream ps = new PrintStream(os);
16          // 缓冲流 读取
17          Reader r = new InputStreamReader(is);
18          BufferedReader br = new BufferedReader(r);
19
20          //3.核心业务：数据传输
21          // 从键盘录入整行数据 发送给服务器，遇到quit结束录入
22          Scanner sc = new Scanner(System.in);
23          System.out.println("请录入发送数据： ");
24          String line = null;
25          while(true) {
26              //录入数据  并发送
27              line = sc.nextLine();
28              ps.println(line);
29              System.out.println("发送数据成功");
30
31              if("quit".equals(line))
32                  break;
33
34              //从服务器接收 返回的信息(反转字符串)
35              String msg = br.readLine();
36              System.out.println("接收信息： " + msg);
37          }
38
39          //4.关闭资源
40          System.out.println("客户端即将关闭");
41          br.close();
42          ps.close();
43          socket.close();
44      }
```

```
45  }
46
```

搭建一个TCP服务器，逐行接收从客户端发送过来的字符串（读取到quit字符串则结束读取），然后对字符串进行反转，最后把反转的字符串返回给客户端。

```
1   public class Test064_ReversalServer {
2       //StringBuffer类测试
3       public static void main00(String[] args) {
4           StringBuffer sb = new StringBuffer();
5           //追加内容
6           sb.append("hello world");
7           System.out.println(sb);
8
9           //反转
10          sb.reverse();
11          System.out.println("反转后: " + sb);
12
13          //获取长度
14          int len = sb.length();
15
16          //清空
17          sb.delete(0, len);
18          System.out.println("清空后: " + sb);
19      }
20
21      public static void main(String[] args) throws Exception {
22          //1.搭建服务器
23          ServerSocket server = new ServerSocket(8989);
24          System.out.println("服务器成功启动，端口: 8989");
25
26          //2.接收客户端连接
27          Socket socket = server.accept();
28          System.out.println("客户端成功连接: " + socket);
```

```
29
30          //3.获取IO流并增强
31          PrintStream ps =
32                  new PrintStream(socket.getOutputStream());
33          // 缓冲流 读取
34          BufferedReader br =
35                  new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
36
37          //4.业务操作
38          //  4.1 逐行收取数据
39          String line = null;
40          // 可变字符序列类，类似String
41          StringBuffer sb = new StringBuffer();
42          while(true) {
43              //清空序列
44              sb.delete(0, sb.length());
45
46              line = br.readLine();
47              System.out.println("read: " + line);
48
49              //读取到quit,则结束
50              if("quit".equals(line))
51                  break;
52
53              // 4.2 对收取的数据 进行 反转
54              sb.append(line);
55              //反转
56              sb.reverse();
57              //获取字符序列 包含的 字符串
58              String msg = sb.toString();
59
60              //4.3 将反转的字符串 返回给 客户端
61              ps.println(msg);
62              System.out.println("服务器 成功发送 反转信息...");
63          }
```

```
64
65          //5.关闭资源
66          System.out.println("服务器即将关闭!");
67          ps.close();
68          br.close();
69          socket.close();
70          server.close();
71      }
72  }
```

6.5 传输对象

准备一个stud.txt文件，放到src/dir目录下，内容如下：

```
1  010.tom.19
2  001.zs.21
3  003.lucy.19
4  002.jack.20
```

搭建TCP客户端，逐行读取stud.txt中数据，然后转化为Student对象，最后将所有对象发送到服务器端。

Student基础类：

```java
package com.briup.chap12;

//注意，必须实现序列化接口
public class Student implements Serializable {
    private static final long serialVersionUID = 1L;

    private String id;
    private String name;
    private int age;

    //省略...
}
```

客户端代码：

```java
package com.briup.chap12;

/*
 * 逐行读取stud.txt中 数据 --> Student对象
 * 再将所有对象 发送到 服务器
 */
public class Test065_ObjectClient {
    public static void main(String[] args) throws Exception {
        //1.搭建客户端
        Socket socket = new Socket("127.0.0.1",8989);
        System.out.println("客户端成功连接，socket: " + socket);

        //2.获取流对象 并包装增强
        OutputStream os = socket.getOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(os);

        //3.解析文件 得到对象 并发送
        BufferedReader br =
                new BufferedReader(new
FileReader("src/dir/stud.txt"));
        String line = null;
```

```
21          List<Student> list = new ArrayList<>();
22          while((line = br.readLine()) != null) {
23              //拆分数据 001.zs.20
24              String[] arr = line.split("[.]");
25              String id = arr[0];
26              String name = arr[1];
27              int age = Integer.parseInt(arr[2]);
28
29              //封装学生对象 并添加到 list集合
30              Student s = new Student(id,name,age);
31              list.add(s);
32          }
33
34          // 发送集合(含所有学生)
35          oos.writeObject(list);
36          System.out.println("学生发送成功，数量: " + list.size());
37
38          //4.关闭资源
39          System.out.println("客户端即将关闭");
40          oos.close();
41          socket.close();
42      }
43  }
```

搭建TCP服务器，接收从客户端发送过来的所有学生，然后遍历输出。

```
1   package com.briup.chap12;
2
3   /*
4    * 搭建服务器，接收从客户端发送过来的所有学生
5    * 遍历输出
6    */
7   public class Test065_ObjectServer {
8       public static void main(String[] args) throws Exception {
```

```
        //1.搭建服务器，指定端口
        ServerSocket server = new ServerSocket(8989);
        System.out.println("服务器启动成功  端口 8989...");

        //2.接收客户端的连接
        Socket socket = server.accept();
        System.out.println("客户端成功连接: " + socket);

        //3.获取输入流 并 包装增强
        InputStream is = socket.getInputStream();
        ObjectInputStream ois = new ObjectInputStream(is);

        //4.接收 遍历数据
        List<Student> list = (List<Student>)ois.readObject();
        System.out.println("成功接收学生数量: " + list.size());
        for (Student s : list) {
            System.out.println(s);
        }

        //5.资源关闭
        System.out.println("服务器即将终止");
        ois.close();
        socket.close();
        server.close();
    }
}

```

6.6 多线程案例

搭建一个TCP客户端，读取键盘信息，然后发送给服务器，遇到quit退出。

```
package com.briup.chap12;

```

```java
/*
 * 接盘录入信息  然后发给服务器，遇到quit结束
 */
public class Test066_ChatClient {
    public static void main(String[] args) throws Exception {
        //1.搭建客户端
        Socket socket = new Socket("127.0.0.1",8989);
        System.out.println("客户端成功连接，socket: " + socket);

        //2.封装得到IO流
        PrintStream ps = new
PrintStream(socket.getOutputStream());

        //3.读取键盘信息 然后发送给服务器
        Scanner sc = new Scanner(System.in);
        String line = null;
        while(true) {
            line = sc.nextLine();
            if("quit".equals(line))
                break;

            ps.println(line);
        }

        //4.关闭资源
        System.out.println("客户端即将关闭");
        ps.close();
        socket.close();
    }
}
```

搭建TCP服务器，分离多线程接收客户端发送过来的数据，然后进行输出。

```java
package com.briup.chap12;
```

```
2
3  public class Test066_ChatServer {
4      //多线程 服务器
5      public static void main(String[] args) throws Exception {
6          //1.搭建服务器，指定端口
7          ServerSocket server = new ServerSocket(8989);
8          System.out.println("服务器启动成功  端口 8989...");
9
10         while(true) {
11             //2.接收客户端的连接
12             Socket socket = server.accept();
13             System.out.println("客户端成功连接: " + socket);
14
15             // 单独分离子线程 为 当前客户端提供服务
16             Thread th = new Thread() {
17                 @Override
18                 public void run() {
19                     //1.获取IO流
20                     BufferedReader br = null;
21                     try {
22                         br = new BufferedReader(new
   InputStreamReader(socket.getInputStream()));
23                         //2.数据操作
24                         String line = null;
25                         while((line = br.readLine()) != null)
   {
26                             System.out.println("read: " +
   line);
27                         }
28                     }catch(Exception e) {
29                         e.printStackTrace();
30                     }finally {
31                         //3.关闭资源
32                         System.out.println("客户端对应资源即将关
   闭!");
33                         try {
```

```
                            if(br != null)
                                br.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                        try {
                            if(socket != null)
                                socket.close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }
            };
            th.start();
        }
    }

    //实现 单线程服务器，接收多个客户端 聊天信息
    public static void main01(String[] args) throws Exception
{
        //1.搭建服务器，指定端口
        ServerSocket server = new ServerSocket(8989);
        System.out.println("服务器启动成功  端口 8989...");

        while(true) {
            //2.接收客户端的连接
            Socket socket = server.accept();
            System.out.println("客户端成功连接：" + socket);

            //3.封装IO流对象，逐行读取聊天信息并输出
            BufferedReader br =
                    new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            String line = null;
            while((line = br.readLine()) != null) {
```

```
68                  System.out.println("read: " + line);
69              }
70
71              //4.关闭资源
72              System.out.println("客户端即将关闭: " + socket);
73              br.close();
74              socket.close();
75              //能否关闭 server
76          }
77      }
78  }
```

7 UDP网络编程

7.1 概述

在UDP通信协议下，两台计算机之间进行数据交互，并不需要先建立连接，客户端直接往指定的IP和端口号上发送数据即可，但是它并不能保证数据一定能让对方收到。

`java.net.DatagramSocket` 和 `java.net.DatagramPacket` 是UDP编程中使用到的俩个类，客户端和服务器端都使用这个俩类

`java.net.DatagramSocket` 负责接收和发送数据

`java.net.DatagramPacket` 负责封装要发送的数据和接收到的数据

注意，UDP协议下的编程，作为扩展和了解即可

7.2 案例

UDP接收端搭建，接收数据：

```
1  package com.briup.chap12;
2
3  public class Test072_UDPServer {
4
5      public static void main(String[] args) {
6
7          byte[] buf = new byte[1024];;
8          int port = 9999;
9
10         DatagramSocket socket = null;
11         DatagramPacket packet = null;
12
13         try {
14             //创建socket，并指定监听的端口号
15             socket = new DatagramSocket(port);
16             //创建packet，用于接收数据
17             //指定buf从下标0开始，最多接收length长度的数据
18             packet = new DatagramPacket(buf, 0,buf.length);
19
20             System.out.println("服务器启动，等待客户端发送数据过
   来");
21             //receive方法会阻塞，等待客户端发送数据过来
22             //使用packet接收数据，数据存放在packet中的buf数组中
23             socket.receive(packet);
24             //把收的数据转为字符串输出，数据从buf的下标0开始，长度为
   packet.getLength()
25             System.out.println("服务器接收的数据为:"+new
   String(buf,0,packet.getLength()));
26         } catch (IOException e) {
27             e.printStackTrace();
28         }finally {
29             if(socket!=null){
```

```
                socket.close();
            }
        }
    }
}
```

UDP发送端搭建，发送数据：

```
package com.briup.chap12;

public class Test072_UDPClient {
    public static void main(String[] args) {
        String ip = "127.0.0.1";
        int port = 9999;

        DatagramSocket socket = null;
        DatagramPacket packet = null;

        try {
            byte[] buf = "hello world 中国".getBytes();

            //socket负责发送数据
            socket = new DatagramSocket();
            //打好数据报,并指定要发生到的ip和端口号
            //buf中从0开始，全部数据都发送
            packet = new DatagramPacket(buf,
0,buf.length,InetAddress.getByName(ip), port);
            //发送数据
            socket.send(packet);
            System.out.println("客户端发送数据完毕");
        } catch (IOException e) {
            e.printStackTrace();
        }finally {
            if(socket!=null){
                socket.close();
            }
```

```
28            }
29        }
30    }
```

> 注意，即使不启动服务器，客户端也可以发送数据，因为UDP不提前建立连接，也不保证数据会让对方收到