# Socket编程原理

## 简介

1. 五层模型为:物理层，数据链路层，网络层，传输层，应用层。

Socket位于传输层之上，应用层之下的接口层，基于TCP/IP协议。

1. 对于一个 Socket，以 客户端的 IP 及端口号 <=> 服务器端的 IP 以及端口号 作为唯一标识。
2. TCP 的数据是流，大家可以类比文件流，而 UDP 则是基于数据包。CP 没有数据边界，每次接收数据以字节为单位，如果想区分两次发送的数据，除非在数据中加入分割字符 (如 http 的 \ r\n\r\n)，否则，TCP 无法区分数据边界，而 UDP 每次发送的数据都被打为一个独立的数据包，因此几次发送的数据边界很清晰，我们每次接收也是按照数据包为单位进行接收。

## UDP编程

* 1. 客户端编程

1. 创建客户端发送端口（不带IP地址和端口）
2. 创建要发送的数据报（加上服务端端口和地址信息）
3. 发送数据报
4. 创建要接收数据的数据报（不带端口和地址）
5. 接收数据
   1. 服务端编程
6. 创建服务端监听端口（加上服务端的端口和地址信息）
7. 创建要接收的数据报（不带IP地址和端口）
8. 接收数据报
9. 利用接收的数据报获取客户端的端口和IP地址信息
10. 创建要发送的数据报（加上客户端的端口和IP地址信息）
11. 发送数据报

具体文件在同一文件夹下。

# 服务器模型

## 阻塞式服务器

1. 简介

阻塞式服务器是最好实现的服务器，也是问题最多的服务器。**客户端发送到服务器的请求，服务器会进行排队，依次处理请求。前一个请求没有处理完成，服务器不会处理后面的请求。**这种服务器很容易进行攻击，只需要向服务器发送一个处理时间很长的请求，就会将其他的请求堵在门外，导致其他请求无法得到处理，所以，这种服务器更多的是作为理论模型，实际应用并不多。

## 并发服务器

1. 简介

**并发式服务器，这种服务器处理请求时，每接收到一个请求，就启动一个线程处理该请求。**这种模式的服务器，好处是不会出现阻塞式服务器请求被拥堵的情况，但是也是存在问题的，**服务器启动线程是有一定的开销的，请求数量不多的时候，服务器启动线程没有问题，但是请求过多时，将会导致服务器的资源耗尽**。所以，会存在一种方式——**建立线程池来处理请求，每当请求到来时，向线程池申请线程进行处理，这样，线程池开放多少线程是固定的，不会导致系统资源耗尽**，但是依然会有一些问题，当线程池被占用满时，还是有可能出现请求被阻塞的情况，所以这种方式是一种折中的方式。但是，对于并发请求不是很多的场景来说， 使用这种方式是完全可以的。

## 异步服务器

1. 简介

异步服务器，使用该种方式，一般要借助于系统的异步 IO 机制，如 select 或 poll，这种方式，**当一个请求到达时，我们可以先将请求注册，当有数据可以读取时，会得到通知，这时候我们处理请求，这样，服务器进程没有必要阻塞处理，也不会存在很大的系统开销，因此，目前对于并发量要求比较高的服务器，一般都是采用这种方式**。

## TCP的阻塞式服务器

* 1. 服务端

1. 创建服务器对象
2. 监听客户端发起的连接，建立TCP连接
3. 建立输入流,读取该连接中的数据流
4. 建立输出流，向客户端写出处理后的字符串
5. 关闭TCP连接
   1. 客户端
6. 建立TCP连接，创建连接服务器的Socket
7. 先建立输出流，写出请求字符串，发送给服务端
8. 建立输入流，读取服务器响应
9. 关闭TCP连接

具体见同文件夹下文件