zookeeper

zookeeper是一个分布式协调服务的开源框架，主要用来解决分布式集群中应用系统的一致性问题，例如怎样避免同时操作同一数据造成脏读的问题。

zookeeper本质上是一个分布式的小文件存储系统，提供基于类似于文件系统的目录树方式的数据存储，并且可以对书中的节点进行有效管理，从而用来维护和监控你存储的数据的状态变化。通过监控这些数据状态的变化，从而可以达到基于数据的集群管理，诸如：统一命名服务，分布式配置管理，分布式消息队列，分布式锁，分布式协调等服务。

zookeeper特性：

全局数据一致性

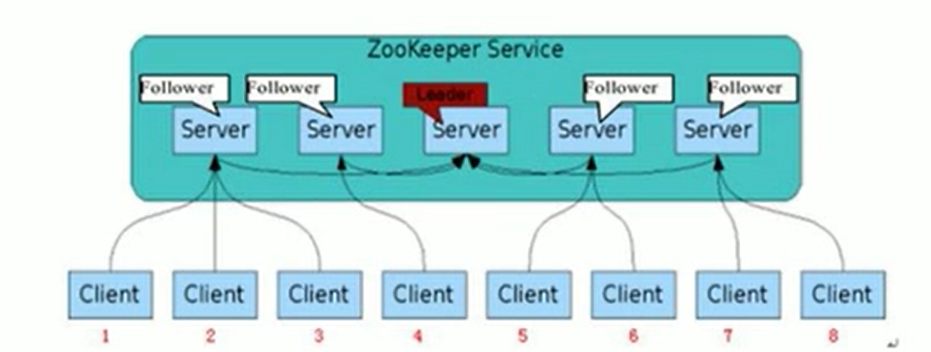
可靠性

顺序性

数据更新原子性

实时性

zookeeper集群角色：



1. Leader：

zookeeper集群工作的核心，事务请求（写操作）的唯一调度和处理者，保证集群事务处理的顺序性，集群内部各个服务器的调度者。对于create，setData，delete等有写操作的请求，则需要统一转发给leader处理，leader需要决定编号，执行操作，这个过程称为一个事务。

1. Follower:

处理客户端非事务（读操作）请求，转发事务请求给leader，参与集群Leader选举投票。

1. Observer:

针对访问量比较大的zookeeper集群，增加观察者角色。观察者角色，观察zookeeper集群的最新状态变化，并将这些状态同步过来，其对于非事务请求可以进行独立处理，对于事务请求，则会转发给leader服务器进行处理。不会参与任何形式的投票，只提供非事务服务，通常用于不影响集群事务处理能力的前提下提升集群的非事务处理能力。

zookeeper集群的搭建：

过程：

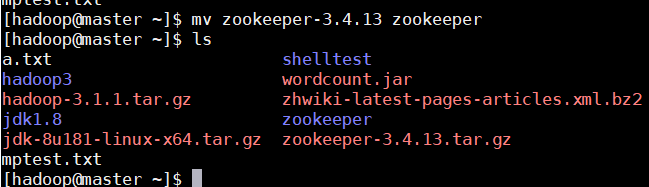
1. 安装jdk（卸载open jdk rpm -qa | grep java; rpm -e --nodeps javapath）
2. 配置主机名称到IP地址的映射配置，配置环境变量
3. 设置myid（1，2，3……）
4. 远程复制分发安装文件
5. 启动zookeeper

注：如果要想使用observer模式，可在对应节点的配置文件中添加如下配置：peerType=observer；其次，必须在配置文件指定哪些节点被指定为Observer。

过程截图：

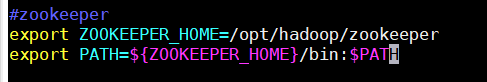
jdk的安装在之前就弄了，这里省略，可以查看hadoop3.1搭建的时候的相关资料；

解压zookeeper：



创建环境变量：

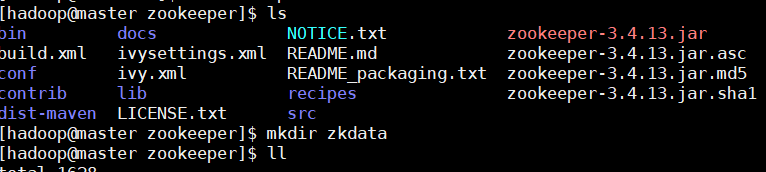
在hadoop根目录下，vim .bashrc，添加如下内容：



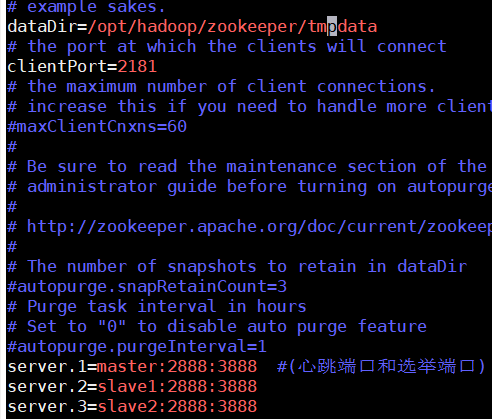
创建zoo.cfg配置文件：



创建数据存放文件夹：mkdir tmpdata



修改zoo.cfg文件：

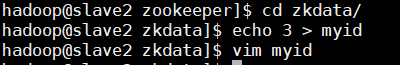


创建myid:

mkdir zkdata

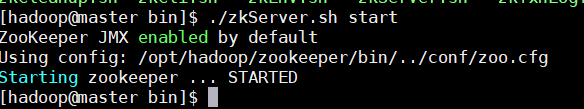
vim myid: 1

远程将zookeeper文件夹复制到其他主机，然后修改myid



启动zookeeper:

在三个节点都启动



查看启动状态：（有错误）

数据模型：树形结构

Znode

节点类型：临时节点和永久节点

节点属性：通过get获取，dataVersion，cversion

cZxid:Znode创建的事务id

mZxid:被修改的事务的id

ctime

mtime

ephemeraOwner如果是临时节点，其值表示与该节点绑定的id，如果为0，表示不是临时节点

numChildren子节点个数

Shell客户端：

连接：

1. 本地连接zkCli.sh
2. 远程连接zkCli.sh -server ip

创建节点：

create [-s] [-e] path data acl

-s 序列化

-e 临时节点，会话断开后，会消失

acl权限控制

节点查看：

ls /xxx 列举

get /xxx 获取信息

更新节点：

set path data [version]

删除节点：

delete path [version]:需要先删除子节点

quota命令(限制/约束)：

listquota path

setquota -n|-b val path

-n节点个数，连同自己本身，如果超出数目，在日志文件中会有warn

-b数据长度

val对应的值

delquota [-n|-b] path 删除quota

history命令：列出历史命令

redo命令：重新执行指定的历史命令，常结合history命令使用

Watcher：

分布式数据发布/订阅功能

分布式的通知功能

三个过程：客户端向服务端注册Wathcer，服务端事件发生触发Wathcer，客户端回调Wathcer得到触发事件情况

Wathcer机制特点：

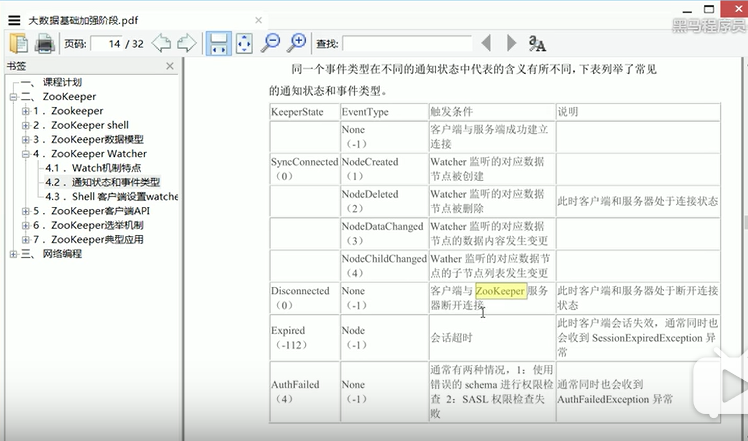
一次性触发：不会再次触发

事件封装：WathedEvent对象来封装，通知状态，事件类型，节点路径

event异步发送：

先注册再触发：

事件类型和通知状态：



Shell设置Wathcer：

help命令查看什么命令操作可以设置 监听

客户端：get path watch

Java API:

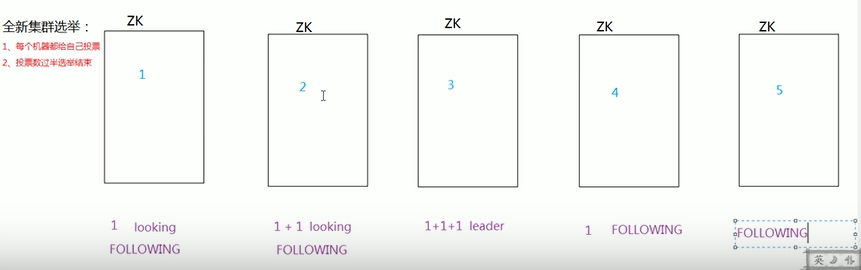
类：org.apache.zookeeper.Zookeeper

接口：org.apache.zookeeper.Watcher，回调方法process

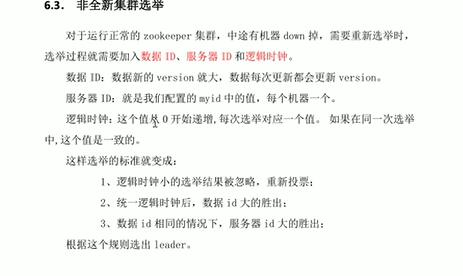
watcher操作

选举机制：

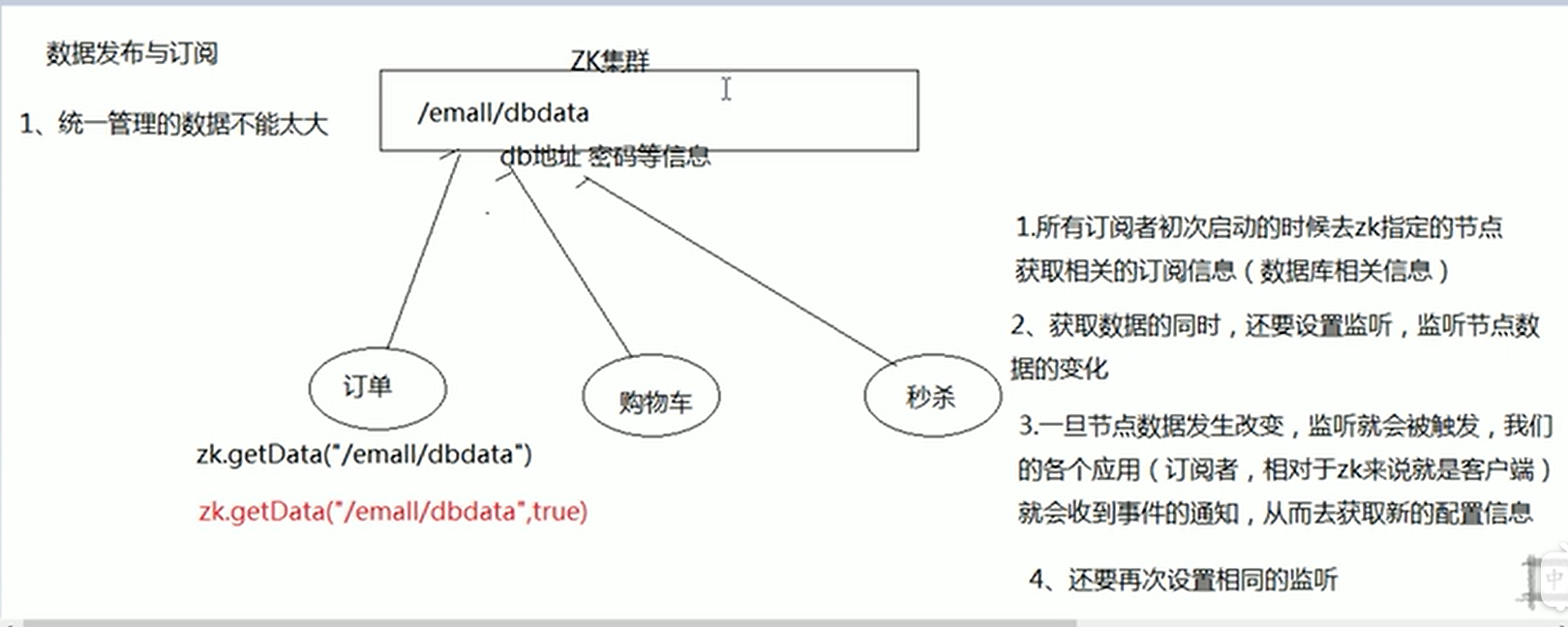
全新集群选举：主要参考服务器的myid的值



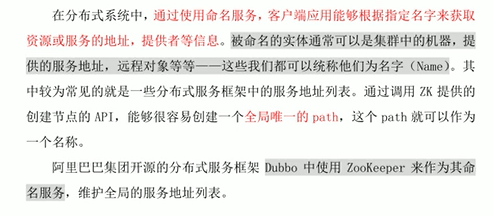
非全新集群选举：

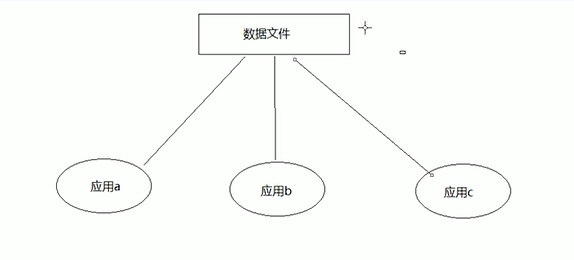


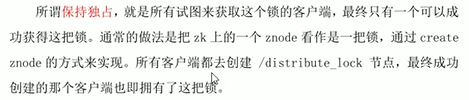
数据发布与订阅：

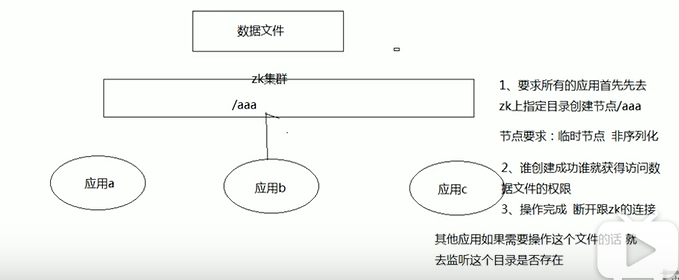


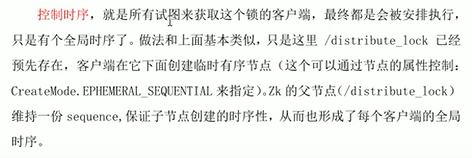
命名服务和分布式锁：

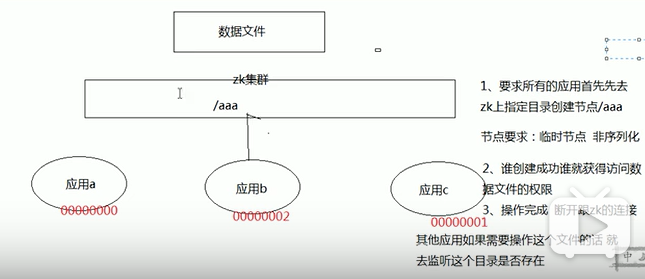
  
分布式锁分为保持独占和控制时序两类。“脏数据”





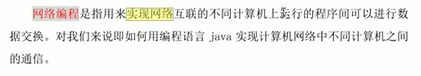


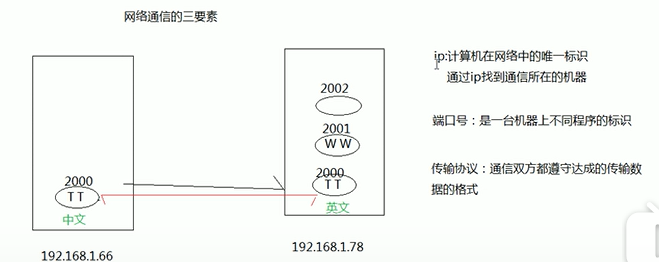


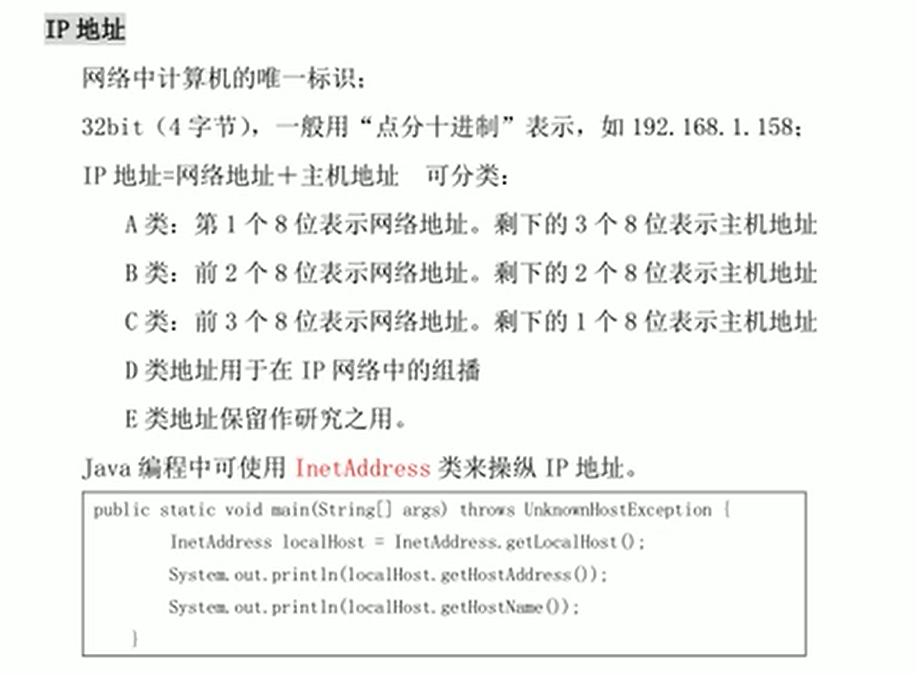


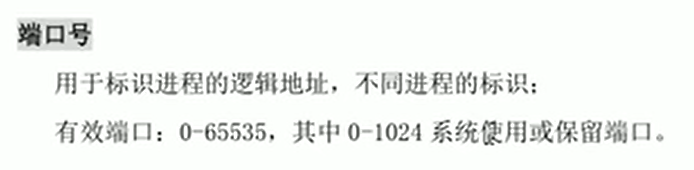
网络通信：

网络编程：



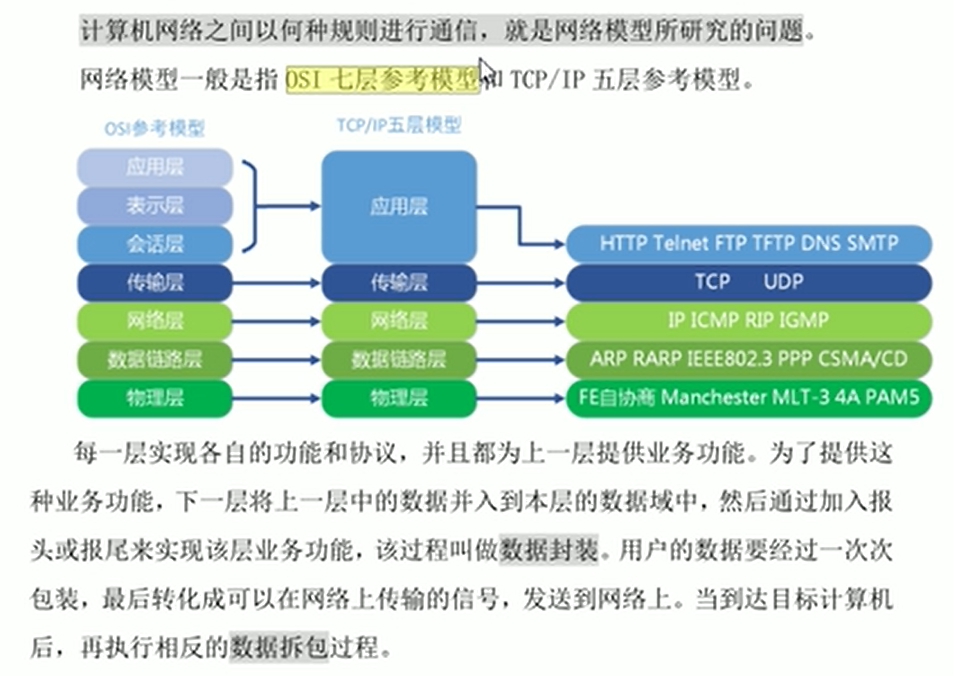


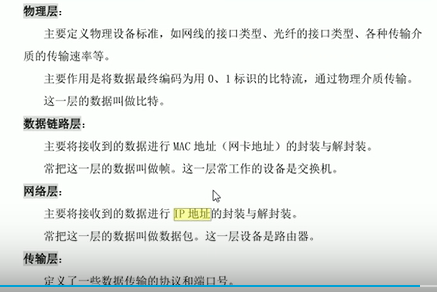






网络模型：





Socket：



socket + 传输协议（TCP/UDP）

基于UDP协议的Socket通信：

核心类：DatagramSocket

过程：

客户端：

//创建发送端的socket对象， DatagramSocket

//把要发送的数据打包

//发送打包好的数据

//资源释放

服务器端：

//创建服务端的socket对象， DatagramSocket

//接收数据

//数据解析

//释放资源

基于TCP协议的Socket通信：

服务端：

核心API：ServerSocket



监听连接：是阻塞的方法，server.accept()获取一个 socket对象

客户端：

过程：

//创建客户端的socket对象，Socket

//指定目的地址和端口号

//通过socket建立的连接，输出数据

//释放资源

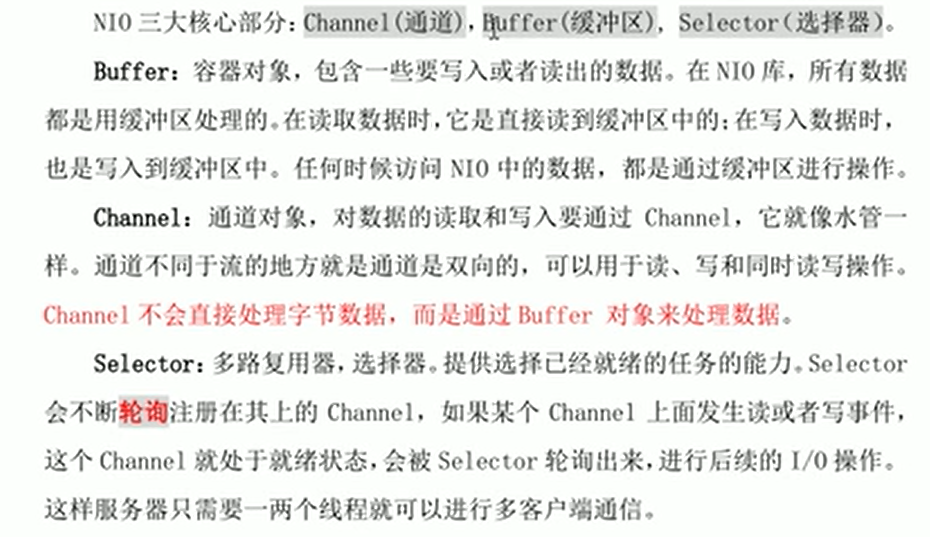
IO通信模型：

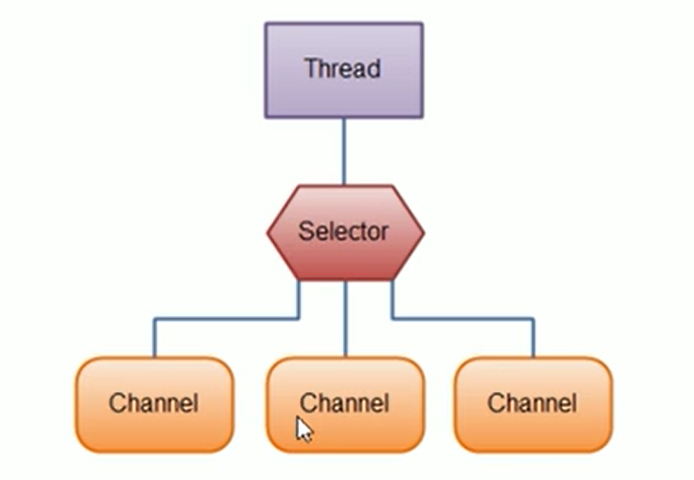
BIO（阻塞模式）blocking IO

多线程通信：

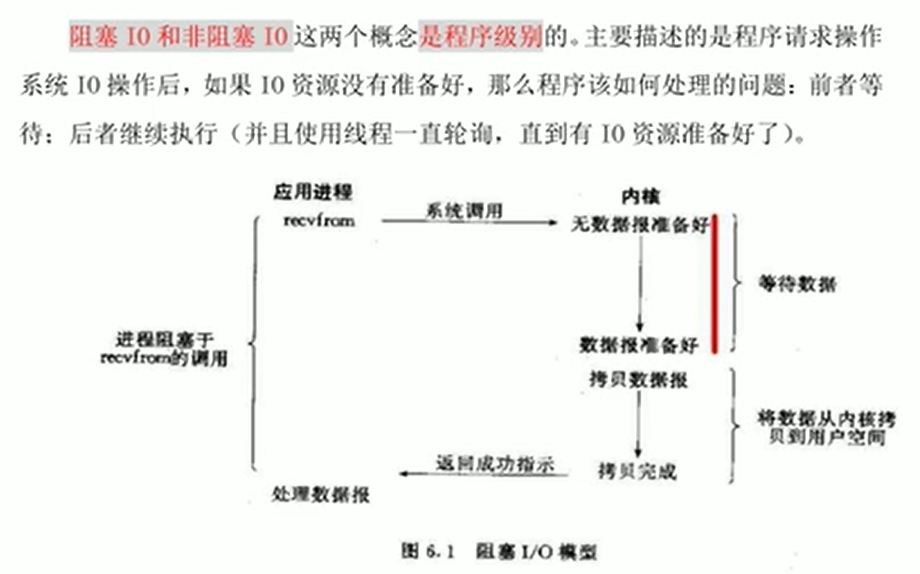


NIO（非阻塞模式）no-blocking IO:



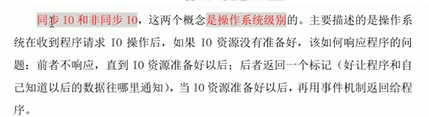


阻塞/非阻塞：





同步/非同步：



RPC：远程过程调用协议

