Java知识体系总结

看过的就忘记了、总结了的才是自己的

2019-12-01 乙亥年 冬

微信：kk6521 **[备注：芝麻开门]**

目录

[第一章 计算机基础 5](#_Toc63426904)

[1. 计算机组成原理 5](#_Toc63426905)

[1.1 Cpu 5](#_Toc63426906)

[1.1.1 cpu如何造出来的 5](#_Toc63426907)

[1.1.2 cpu核和线程 5](#_Toc63426908)

[1.1.3 内存结构 7](#_Toc63426909)

[1.1.4 品牌系列 9](#_Toc63426910)

[1.1.5 指令集 10](#_Toc63426911)

[1.2 主板 11](#_Toc63426912)

[1.2.1 选择CPU对应芯片组 11](#_Toc63426913)

[1.2.2 选择版型 11](#_Toc63426914)

[1.2.3 选择品牌 12](#_Toc63426915)

[1.2.4 总线 12](#_Toc63426916)

[1.2.5 PCI总线 14](#_Toc63426917)

[1.2.6 PCIe总线 16](#_Toc63426918)

[1.2.7 外频 18](#_Toc63426919)

[1.3 显卡 19](#_Toc63426920)

[1.4 内存 19](#_Toc63426921)

[1.5 存储 20](#_Toc63426922)

[1.6 外设 22](#_Toc63426923)

[2. 计算机网络 23](#_Toc63426924)

[2.1网络层次划分 23](#_Toc63426925)

[2.2物理层 23](#_Toc63426926)

[2.2.1数据传输类型： 24](#_Toc63426927)

[2.2.2数据传输方式 24](#_Toc63426928)

[2.2.3传输介质 24](#_Toc63426929)

[2.3数据链路层 25](#_Toc63426930)

[2.3.1链路层特性 25](#_Toc63426931)

[2.3.2点对点协议 26](#_Toc63426932)

[2.3.3广播协议 27](#_Toc63426933)

[2.3.4 扩展的以太网 28](#_Toc63426934)

[2.3.5 链路层安全 29](#_Toc63426935)

[2.4网络层 30](#_Toc63426936)

[2.4.1 IP地址 30](#_Toc63426937)

[2.4.1.1. 分类IP【过时】 30](#_Toc63426938)

[2.4.1.2. 子网 31](#_Toc63426939)

[2.4.1.3. CIDR超网 31](#_Toc63426940)

[2.4.2 ARP协议 32](#_Toc63426941)

[2.4.3 协议报文 33](#_Toc63426942)

[2.4.3.1. IP数据报 33](#_Toc63426943)

[2.4.3.2.  ICMP协议 34](#_Toc63426944)

[2.4.3.3.  DHCP协议 34](#_Toc63426945)

[2.4.3.4.  VPN 35](#_Toc63426946)

[2.4.3.5.  NAT 35](#_Toc63426947)

[2.4.4 IP路由协议 35](#_Toc63426948)

[2.5传输层 36](#_Toc63426949)

[2.5.1UDP协议 37](#_Toc63426950)

[2.5.2 TCP协议 37](#_Toc63426951)

[2.5.2.1.  可靠传输 37](#_Toc63426952)

[2.5.2.2.  流量控制 38](#_Toc63426953)

[2.5.2.3.  拥塞控制 39](#_Toc63426954)

[2.5.2.4.  连接管理 40](#_Toc63426955)

[2.6应用层 41](#_Toc63426956)

[3. 机器虚拟化 42](#_Toc63426957)

[3.1 esxi 安装 43](#_Toc63426958)

[3.2 vsphere client的安装 44](#_Toc63426959)

[3.3 vcenter server的安装 45](#_Toc63426960)

[3.4使用web client访问vcenter 49](#_Toc63426961)

[3.5虚拟机网络设置 51](#_Toc63426962)

[3.5.1 Bridged（桥接模式） 51](#_Toc63426963)

[3.5.2 NAT（网络地址转换模式） 51](#_Toc63426964)

[3.5.3 Host-only(主机模式) 51](#_Toc63426965)

[3.6 类抖音服务部署架构 53](#_Toc63426966)

[4. 操作系统 55](#_Toc63426967)

[4.1 bios系统 55](#_Toc63426968)

[4.2 linux/unix分支 56](#_Toc63426969)

[4.2.1 Linux 发行版 56](#_Toc63426970)

[4.2.2 Linux 发行版（移动） 57](#_Toc63426971)

[4.2.3 Linux 发行版（国内 ）： 58](#_Toc63426972)

[4.2.4 Linux 桌面版 59](#_Toc63426973)

[4.2.5 Linux 服务器 59](#_Toc63426974)

[4.2.6 FreeBSD 60](#_Toc63426975)

[4.2.7 openwrt 61](#_Toc63426976)

[4.3 常用命令 61](#_Toc63426977)

[4.3.1 帮助 61](#_Toc63426978)

[4.3.2 文件命令 62](#_Toc63426979)

[4.3.3 网络命令 64](#_Toc63426980)

[4.3.4 用户命令 65](#_Toc63426981)

[4.3.5 系统其他命令 66](#_Toc63426982)

[4.4 进程和多线程 68](#_Toc63426983)

[4.4.1 进程的切换 69](#_Toc63426984)

[4.4.2 核心线程用户线程 69](#_Toc63426985)

[4.5 文件系统 69](#_Toc63426986)

[4.5.1 高层体系结构 70](#_Toc63426987)

[4.5.2 零拷贝 70](#_Toc63426988)

[4.5.3 主要结构 70](#_Toc63426989)

[4.6 网络 72](#_Toc63426990)

[4.6.1 负载均衡lvs ipvs 72](#_Toc63426991)

[4.6.2 代理上网 74](#_Toc63426992)

[4.6.3 防火墙 74](#_Toc63426993)

[4.7 shell脚本 74](#_Toc63426994)

[5. 容器虚拟化 75](#_Toc63426995)

[5.1 容器化历史 75](#_Toc63426996)

[5.2 docker安装部署 76](#_Toc63426997)

[5.2.1 单机安装 76](#_Toc63426998)

[5.2.2 Docker machine 77](#_Toc63426999)

[5.3 dockers 使用 79](#_Toc63427000)

[5.3.1镜像命令 79](#_Toc63427001)

[5.3.2容器命令 83](#_Toc63427002)

[5.3.3docker file 87](#_Toc63427003)

[5.3.4Docker Compose 编排 89](#_Toc63427004)

[5.3.5 harbor镜像仓库 89](#_Toc63427005)

[5.4 集群管理 92](#_Toc63427006)

[5.4.1 Docker 网络 92](#_Toc63427007)

[5.4.3 Docker Swarm 93](#_Toc63427008)

[5.4.4 图形化管理 99](#_Toc63427009)

[5.4.5 jnkins整合 100](#_Toc63427010)

[6. 容器管理 k8s 101](#_Toc63427011)

[6.1环境准备 101](#_Toc63427012)

[6.1.1主机配置 101](#_Toc63427013)

[6.1.2配置Docker 102](#_Toc63427014)

[6.2安装Kubernetes 103](#_Toc63427015)

[6.2.1安装主节点 103](#_Toc63427016)

[6.2.2初始化集群master 104](#_Toc63427017)

[6.2.3配置集群内部网络 105](#_Toc63427018)

[6.2.4添加Worker节点 105](#_Toc63427019)

[6.3部署容器tomcat 106](#_Toc63427020)

[6.4图形化管理 106](#_Toc63427021)

[6.4.1部署Dashboard 107](#_Toc63427022)

[6.4.2 Kuboard 109](#_Toc63427023)

[6.5其他进阶： 110](#_Toc63427024)

[6.5.1 Kubernetes 入门 110](#_Toc63427025)

[6.5.2 Kubernetes 进阶 110](#_Toc63427026)

[6.5.3 Kubernetes 高级 111](#_Toc63427027)

[6.5.4 Spring Cloud部署 111](#_Toc63427028)

[IASS硬件资源服务【暂略】 112](#_Toc63427029)

[7. 存储方案 113](#_Toc63427030)

[7.1 nas存储方案 113](#_Toc63427031)

[7.2 vsan 存储方案 114](#_Toc63427032)

[7.2.1vSan安装要求 114](#_Toc63427033)

[7.2.2配置vSan 116](#_Toc63427034)

[7.2.3查看vSan状态 120](#_Toc63427035)

[7.3 ceph 存储方案 122](#_Toc63427036)

[7.3.1 ceph 模块 122](#_Toc63427037)

[7.3.2ceph集群服务器规划 123](#_Toc63427038)

[7.3.3安装ceph 124](#_Toc63427039)

[7.3.4创建OSD 125](#_Toc63427040)

[7.3.5客户端挂载使用磁盘镜像 128](#_Toc63427041)

[7.3.6rdb在线扩容 129](#_Toc63427042)

[7.3.7 s3简单存储接口 130](#_Toc63427043)

[7.4 其他存储 130](#_Toc63427044)

[7.4.1GlusterFS 130](#_Toc63427045)

[7.4.1.1. 框架 130](#_Toc63427046)

[7.4.1.2 环境 132](#_Toc63427047)

[7.4.1.3 设置glusterfs 133](#_Toc63427048)

[7.4.2 fastdfs 135](#_Toc63427049)

[7.4.3 swift 135](#_Toc63427050)

[7.4.4 hdfs 135](#_Toc63427051)

[7.5 CDN方案 136](#_Toc63427052)

[7.5.1架构 136](#_Toc63427053)

[7.5.2使用场景 136](#_Toc63427054)

[7.5.3其他名词 136](#_Toc63427055)

[7.5.4回源host 137](#_Toc63427056)

[7.5.4 cdn流量控制 137](#_Toc63427057)

[7.7 星际存储 137](#_Toc63427058)

[7.7.1 swarm 137](#_Toc63427059)

[7.7.1 ipfs 137](#_Toc63427060)

前言：

业精于勤而荒于嬉，行成于思而毁于随。

2020-04-07

做自己喜欢的事叫坚持，做自己不喜欢的事叫突破。

2020-05-09

你要做的是把自己变得更强，而不仅仅是更努力

2020-05-14

**读不在三更五鼓，功只怕一暴十寒**

2020-05-16

不怕慢，就怕站

2020-05-19

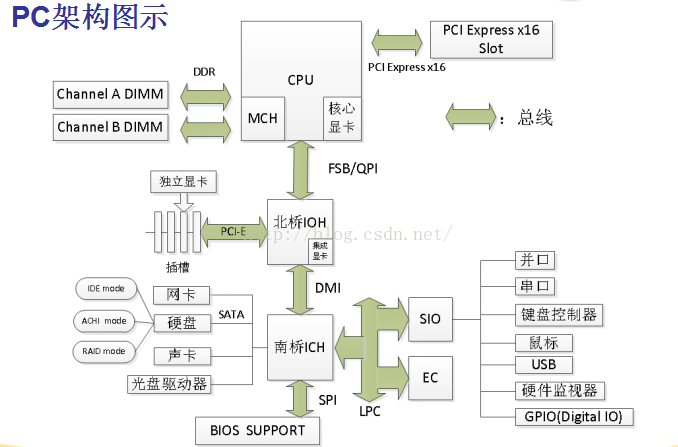
参考《Java 核心知识整理&面试.pdf》

# 计算机基础

# 计算机组成原理

|  |
| --- |
| **《服务器拆机图解》**[https://post.smzdm.com/p/alpozqwo/](https://post.m.smzdm.com/p/alpozqwo/)  《如何配组装电脑》<https://www.zhihu.com/question/19912421>  《小白如何自己组装一台电脑》<https://zhuanlan.zhihu.com/p/60381178>  《X86 架构》<https://blog.csdn.net/vito_bin/article/details/52748429>  《计算机科学速成课》https://github.com/1c7/Crash-Course-Computer-Science-Chinese  《完整学习计算机组成原理》<https://zhuanlan.zhihu.com/p/60958907> |

现在的计算机架构都是冯诺依曼架构，计算机的运算速度是由CPU决定的，购机第一要确定的是CPU的型号；主板芯片组是和CPU匹配的，CPU确认后就可以确认可适配的主板，主板根据不同的机箱会存在不同的尺寸，大尺寸也决定了插槽有更大的扩展性。内存的大小决定了加载存储的容量，会对CPU加载数据的速度造成影响；而存储（外存）是承载具体资料的，他的接口类型决定不同加载速度，空间大小也决定了断电后电脑存储的数据容量。



## 1.1 Cpu

### 1.1.1 cpu如何造出来的

|  |
| --- |
| 《cpu是怎么制造的?》<https://www.zhihu.com/question/23350000>  《开发一个芯片有多难》<https://www.zhihu.com/question/22962300> |

### 1.1.2 cpu核和线程

|  |
| --- |
| 《CPU的频率是什么》<https://zhuanlan.zhihu.com/p/30582175>  《多线程－概述及底层实现机制浅析》<https://www.jianshu.com/p/e1edc437d184>  《cpu工作原理》<https://www.zhihu.com/question/40571490> |

**插槽数**决定了一台电脑可以安装几个物理cpu数量

**单核cpu和多核cpu**

都是一个cpu，不同的是每个cpu上的核心数，多核cpu是多个单核cpu的替代方案，多核cpu减小了体积，同时也减少了功耗，但是一个核心只能同时执行一个线程

**物理核数量** = cpu数(机子上装的cpu的数量**【插槽数】**)\*每个cpu的核心数

**虚拟核：**所谓的4核8线程，4核指的是物理核心。通过**超线程**技术（仍然是当前线程的代码），用一个物理核模拟两个虚拟核，每个核两个线程，总数为8线程。在操作系统看来是8个核，但是实际上是4个物理核。通过超线程技术可以实现单个物理核实现线程级别的并行计算，但是比不上性能两个物理核，对于计算型的并发线程，超线程作用不大，但是对于io等阻塞型线程，超线程会并发使用cpu核心。

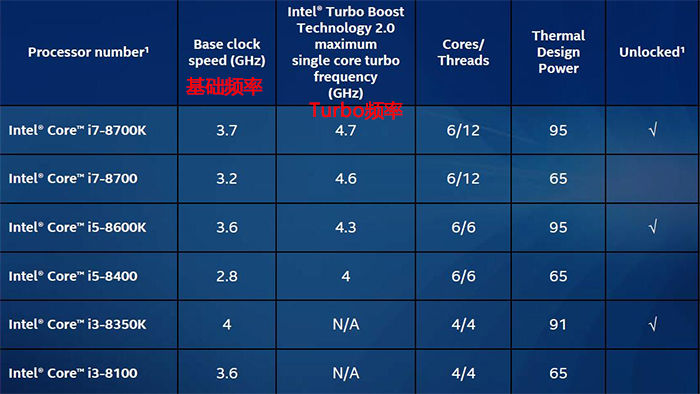


**频率：**

**主频**即CPU内核工作的时钟频率（CPU Clock Speed），比如一颗CPU是2.3G，无论是单核还是多核，所有的核心都是工作在2.3G。通常所说的某某CPU是多少兆赫的，而这个多少兆赫就是“CPU的主频”。CPU的主频表示在CPU内数字脉冲信号震荡的速度，与CPU实际的运算能力并没有直接关系。主频越高，CPU的运算速度就越快。但主频不等于处理器一秒钟执行的指令条数，一条指令的执行可能需要多个时钟周期。对于CPU，在有兼容性的前提下，主频越高，字节越长，CPU速度就越快。

**睿频**是Intel的一项加速技术，指当启动一个运行程序后，处理器会自动加速到合适的频率，而原来的运行速度会提升 10%~20% 以保证程序流畅运行的一种技术。**CPU在管理机制上允许一部分核心超频运行，以应对吃主频而不吃多线程的应用。**这样来说CPU的使用非常的灵活，比如I9 7980虽然主频才2.6，但是允许一个核心超频到4.4来使用，单核心性能几乎提升了一倍。因为大多数的应用程序是用不到多核的，只认频率不认核。但多核CPU都都不会主频太高，这样，Intel就会把多核CPU的不需要的核暂时关掉，只用单核运行，然后把单核的频率提高，高于标准的主频。

睿频的实质是多核切换成单核模式然后自动超频。既提高了日常使用的速度，又节能。



**位数**

32位更换到 64 位操作系统，最直接的一个好处就是可以使用 4GB 以上的内存，通常我们说的 32 位指的是 **IA32(x86-32) 指令集，而 64 位通常是指 AMD64(x86-64) 指令集**，后者是对前者的一个扩充，增加了一些(多媒体方面的)新指令，增强了安全机制，增加了几个寄存器，同时**以 64bit 为单位处理数据**，也就是说支持 4GB 以上的内存。但是，多数软件，尤其是 Windows 下的闭源软件，为了确保一个安装包在任何版本的操作系统，任何平台下都可以使用，往往会使用最老的，支持最广泛的指令集来进行编写(编译), 而不会使用 64 位新指令集中的功能。**当然，随着时间的推移，会有更多软件为 64 位系统提供优化。**

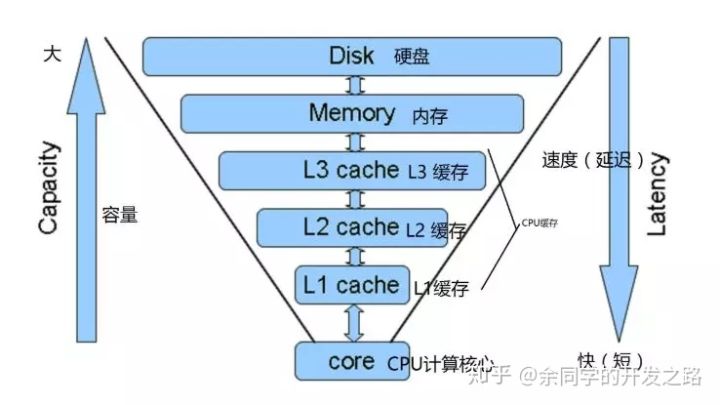
**CPU的评测工具：CPU-Z**



### 1.1.3 内存结构

|  |
| --- |
| 《彻底搞懂 CPU 中的内存结构》<https://zhuanlan.zhihu.com/p/55429568>  《多核cache一致性》<https://zhuanlan.zhihu.com/p/115114220>  《cache原理解析》<https://zhuanlan.zhihu.com/p/102293437> |

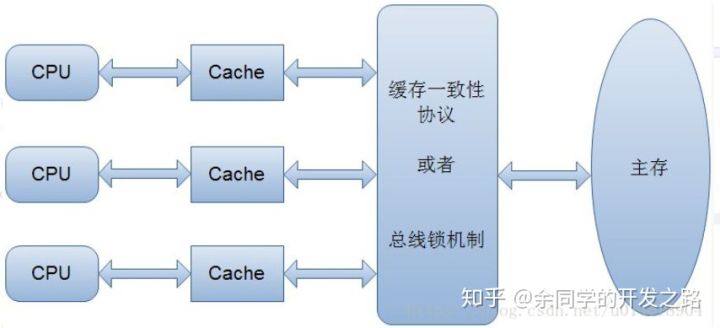
现在主要介绍一下 CPU 中的**多级缓存和乱序执行优化**，为后面学习多线程做铺垫。先来理解一下 CPU 的结构，后面再说 Java 虚拟机的内存模型。先放两张图看一下 CPU 和各级缓存、内存、硬盘之间的关系。

下面就来介绍一下为什么会出现多级缓存，以及会出现什么问题，CPU 又是如何解决的。

为什么会出现多级缓存呢？说的简单一点因为CPU 的频率太快了，而若是没有缓存，直接读取内存中的数据又太慢了，我们不想让 CPU 停下来等待，所以加入了一层读取速度大于内存但小于 CPU 的这么一层东西，这就是缓存。

加入缓存之后，CPU需要数据就问缓存要，缓存没有就从主存中读取，并保留一份在缓存中。下次读取就从缓存中读取，加快速度。



但是，我们经常听到 CPU 运行速度快，缓存次之，而内存慢一点，硬盘最慢。这又是为什么呢？

CPU 运行的快，那是因为一个时钟周期短，一个时钟周期是指机器码0和1变化，实质就是电信号一高一低之间所用的时间，这可都是电信号，速度很快！**也就 10 纳秒左右吧，1 秒等于 10 的 9 次方纳秒。Cpu的频率现在基本也到了瓶颈。**

缓存目前基本上都是采用 **SRAM 存储器**，SRAM 是英文 Static RAM 的缩写，它是一种具有静志存取功能的存储器，不需要刷新电路即能保存它内部存储的数据。不像 DRAM 内存那样需要刷新电路，每隔一段时间，固定要对 DRAM 刷新充电一次，否则内部的数据即会消失，因此 SRAM 具有较高的性能，但是 SRAM 也有它的缺点，即它的集成度较低，相同容量的 DRAM 内存可以设计为较小的体积，但是 SRAM 却需要很大的体积，这也是目前不能将缓存容量做得太大的重要原因。这中间也解释了为什么内存中的数据一断电就没有了。

而 RAM（随机读写存储器）的工作原理大致是当 CPU 读取主存时，将地址信号放到地址总线上传给主存，主存读到地址信号后，解析信号并定位到指定存储单元，然后将此存储单元数据放到数据总线上返回给 CPU。

磁盘慢就慢在它的读取是需要借助于磁头移动，这有寻址的过程，而这还是一个机械运动的过程，上面慢也是在和电信号打交道，而磁盘不仅需要电还需要摆臂，所以，最慢的就是它了。

我们为了提高 CPU 的利用率，添加了多级缓存，数据的读取和保存都要在主存上进行，若是单线程是没有问题的，一条路走下去，该读读该写写。但是在多线程的情况下就会出现问题，因为每个线程都有自己的缓存，假如线程 1 从主存中读取到 x，并对其加 1 ，此时还没有写回主存，线程 2 也从主存中读取 x ，并加 1 ，它们是不知道对方的，也不可以读取对方的缓存。这时都将 x 写回主存，那此时 x 的值就少了 1 。**这也就是多线程情况下带有缓存的问题，数据出现问题了，怎么办呢？**不能把缓存去掉吧，这时就需要一种协议，来保证不同的线程在读写主存数据时遵守某种规则以保证不会出现数据不一致的问题。这种协议有很多，其中用的比较多的是 **MESI 协议**，**主要用来保证缓存的一致性。**

MESI 为了保证多个缓存中共享数据的一致性，定义了 cache line 的四种状态，而线程对 cache line 的四种操作可能会产生不一致的状态，因此缓存控制器监听到本地操作和远程操作的时候，需要对地址一致的 cache line 状态进行一致性修改，从而保证数据在多个缓存之间保持一致性。(M: modified E: Exclusive S: shared I: invalid)

CPU 中每个缓存行（caceh line）使用 4 种状态进行标记（使用额外的两位(bit)表示）。

**M: 被修改（Modified)**

该缓存行只被缓存在该 CPU 的缓存中，并且是被修改过的（dirty)，即与主存中的数据不一致，该缓存行中的内存需要在未来的某个时间点（允许其它 CPU 读取请主存中相应内存之前）写回主存。当被写回主存之后，该缓存行的状态会变成独享（exclusive)状态。

**E: 独享的（Exclusive)**

该缓存行只被缓存在该 CPU 的缓存中，它是未被修改过的（clean)，与主存中数据一致。该状态可以在任何时刻当有其它 CPU 读取该内存时变成共享状态（shared)。同样地，当 CPU 修改该缓存行中内容时，该状态可以变成 Modified 状态。

**S: 共享的（Shared）**

该状态意味着该缓存行可能被多个 CPU 缓存，并且各个缓存中的数据与主存数据一致（clean)，当有一个 CPU 修改该缓存行中，其它 CPU 中该缓存行可以被作废（变成无效状态）。

**I: 无效的（Invalid）**

该缓存是无效的（可能有其它 CPU 修改了该缓存行）。

cache line 不同的状态之间可以相互转化，这也就是 MESI 协议的具体内容，比方说一个处于 M 状态的缓存行必须时刻监听所有试图读该缓存行相对应主存的操作，这种操作必须在缓存将该缓存行写回主存并将状态变成 S 状态之前被延迟执行。

在 MESI 出现之前的解决缓存一致性的方案是总线锁机制，这种解决方案效率很低，锁住总线期间，其他 CPU 无法访问内存。

说完了 CPU 中的多级缓存，再来看看 CPU 中的**乱序执行优化**，什么意思呢，处理器为提高运算速度而做出违背代码原有顺序的优化。虽然顺序变了，但是执行的结果是不会变的。

举个例子，我要去买杯饮料喝，正常逻辑是这样的，去店里->点单->付钱->喝饮料。那其实我还可以在去店里的路上同时就把【点单付钱】一起搞定，又或者我先【点单】再去店里付钱拿饮料。

类比到 CPU 中也是这样，求 x + y = ? 我计算 x 的值，再计算 y 的值，其实可以一起执行，或是先计算 y 的值，当然为了提高运算速度，它会同时计算 x 和 y 的值，一个 CPU 中又不是只有一个逻辑计算的单元。

乱序优化就是这样，为了提高效率 CPU 做出的优化，这在单核的时候是不会有问题的，但是在多核时代又会出现问题。

再举个例子，线程 1 需要借组一个 flag 变量执行一段逻辑，你能用 C线程 2 来乱序优化一下，先执行逻辑而不看 flag 变量的值？这是肯定不行的。

行还是不行肯定有一套规则，

这套规则也就是内存屏障，太高级了听不懂，换一种说法，就是说不同架构的处理器在其指令集中提供了不同的指令来发起内存屏障，对应在编程语言当中就是提供特殊的关键字来调用处理器相关的指令。结果就是有些指令能乱有些则不能！

好了，到这里就算把 CPU 级别的特性给说完了，这些知识对于我们理解 Java 内存模型（JMM） 以及多线程编程很有用。而 Java 内存模型就是借鉴了硬件的结构。

**总结一下，在 CPU 中为了提高运行效率，加了多级缓存和乱序执行优化（还有超线程）。加了多级缓存之后呢，会出现缓存不一致的情况，解决的办法就是定义了 MESI 等类似协议。对于乱序执行优化带来的问题，CPU 选择内存屏障来解决，即定义了一套指令集，什么样的指令不能执行乱序优化。**

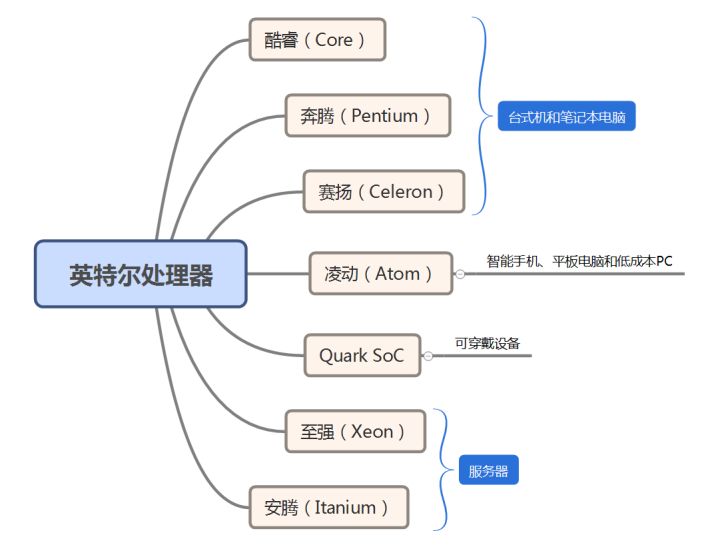
### 1.1.4 品牌系列

|  |
| --- |
| 《酷睿系列及分代》<https://zh.wikipedia.org/wiki/%E9%85%B7%E7%9D%BF>  《至强系列及分代》<https://zh.wikipedia.org/wiki/%E8%87%B3%E5%BC%BA> |

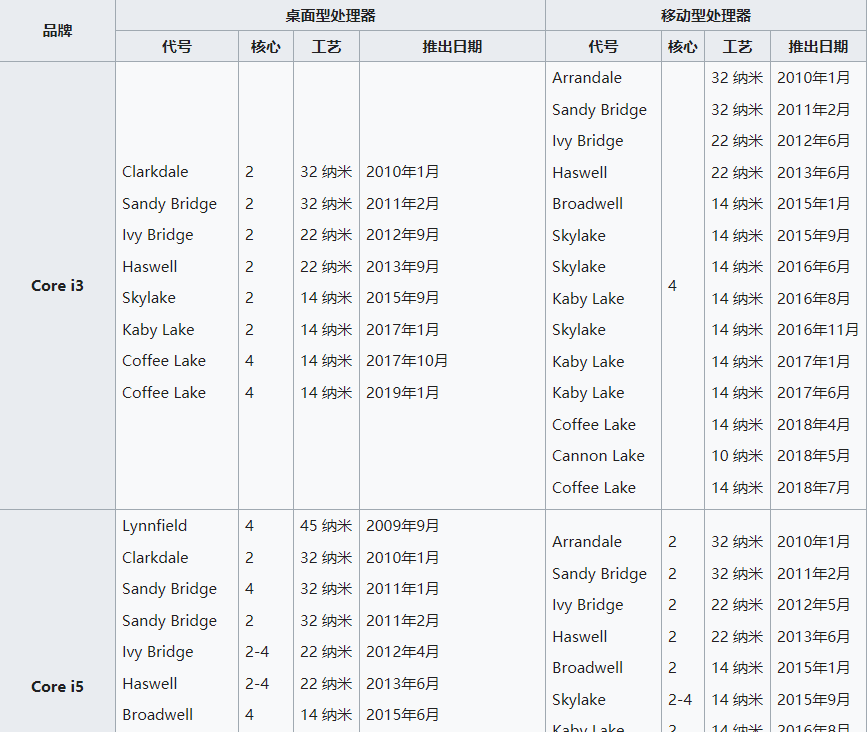
CPU有很多种，主要有两个品牌。**INTEL和AMD**是两家历史几乎同样悠久的，世界上仅存的两家（以前还有好几家，后来都没了，威胜还能生产，但市场上很难见到）民用级X86架构处理器生产商（不讨论arm系：高通）。INTEL目前来说要更强大一些（其实一直都是，AMD最强的时候也没超过50%占有率，不过要是把一些游戏机什么的也算上的话，那可是远超INTEL的）。

**英特尔处理器系列**

对于台式机和笔记本电脑，最常见的是酷睿、奔腾和赛扬系列，同代产品中他们的性能依次减弱，酷睿最强，奔腾次之，赛扬最弱。



酷睿下面还有四大系列，**从低到高分别是I3、I5、I7、I9，现在已经到了第9代**。



服务器CPU：以前的**Xeon分为E3、E5、E7三个系列**，现在的**Xeon分为Xeon E、Xeon W、Xeon D、Xeon Scalable**几个系列。另有加速卡Xeon Phi。只有部分Xeon E3、Xeon E处理器内置[GPU](https://zh.wikipedia.org/wiki/GPU" \o "GPU)。

早期Xeon的命名由定位、具体型号和代数三部分组成。

* 开头的E3、E5或E7代表等级定位。
* 后四位数字中，第一个数字代窗体个[主板](https://zh.wikipedia.org/wiki/%E4%B8%BB%E6%A9%9F%E6%9D%BF" \o "主板)可同时安装的最多的处理器数目，简称路，有单路、双路、四路和八路之分。第二个数字代表[CPU插座](https://zh.wikipedia.org/wiki/CPU%E6%8F%92%E5%BA%A7)类型，如第一代产品的1、2、4、6、8分别代表[BGA 1284](https://zh.wikipedia.org/w/index.php?title=BGA_1284&action=edit&redlink=1" \o "BGA 1284（页面不存在）)、[LGA 1155](https://zh.wikipedia.org/wiki/LGA_1155" \o "LGA 1155)、[LGA 1356](https://zh.wikipedia.org/wiki/LGA_1356" \o "LGA 1356)、[LGA 2011](https://zh.wikipedia.org/wiki/LGA_2011" \o "LGA 2011)和[LGA 1567](https://zh.wikipedia.org/wiki/LGA_1567)。第三位和第四位为性能区分用。
* 最后的v与数字代表微架构，如v2代表此处理器为[Ivy Bridge](https://zh.wikipedia.org/wiki/Ivy_Bridge%E5%BE%AE%E6%9E%B6%E6%A7%8B" \o "Ivy Bridge微架构)微架构，v3代表[Haswell](https://zh.wikipedia.org/wiki/Haswell%E5%BE%AE%E6%9E%B6%E6%A7%8B" \o "Haswell微架构)微架构，以此类推。代表Sandy Bridge微架构的v1通常省略，不标注。

以E5-2650 v2为例，E5代表处理器的定位为中高端工作站，2代表此处理器最高支持双路工作，6代表此处理器的插槽为LGA 2011，50为性能区分用，最后的v2代表此处理器采用Ivy Bridge微架构。

### 1.1.5 指令集

|  |
| --- |
| 《漫话CPU指令集架构》<https://zhuanlan.zhihu.com/p/46170108>  《[常见的CPU指令集介绍](https://www.cnblogs.com/yilang/p/10974327.html)》<https://www.cnblogs.com/yilang/p/10974327.html>  《分不清ARM和X86架构，别跟我说你懂CPU！》 <https://zhuanlan.zhihu.com/p/21266987>  《ARM架构和MIPS架构以及X86架构的区别》<https://www.zhihu.com/question/20155909>  《嵌入式X86、ARM、MIPS架构对比》<https://blog.csdn.net/dsh0828/article/details/79662794> |

处理器指令集架构（Instruction Set Architectures）目前市面上存在两种指令集架构类型:

Reduced Instruction Set Computing **(RISC) 精简指令集**，比如ARM,MIPS等

Complex Instruction Set Computing **(CISC) 复杂指令集**，比如Intel的X86等

简单说，一个就是只规定非常简单的2进制处理器指令，复杂的指令也只能是最基本的指令的叠加组合。二复杂指令集代表着本身就有一个很庞大的指令集库，一些复杂的指令已经在指令集库里，只需要调用即可。

当然，目前最火的当属基于RISC开发的ARM架构，在移动通信时代，凭借着低功耗打遍天下无敌手。手机，平板，自动驾驶等等等等，一招鲜吃遍天。而且RISC的构架也是ARM低功耗原因之一。RISC的CPU包含有较少的单元电路，因而面积小、功耗低；而CISC的CPU包含有丰富的电路单元，因而功能强、面积大、功耗大。

**X86，**依靠强有力的Intel，强势控制产业链，获取价值链上最丰厚的那部分利润.

**ARM，** 靠IP授权的商业模式**（高通）**，且技术上走与Intel差异化路线，加上一些些运气（踏对了手机这条路，谢谢TI-Nokia，Apple，Samsung for big.Little）走小而美的路线，但是凭借已经形成巨大的生态系统，占据优势.

**MIPS，**很学术很精美很帅，但是对指令集控制松散，导致生态系统分裂，没有形成合力，最终被市场抛弃。

**其他：SSE指令集，3DNow!+指令集，AVX指令集**

## 1.2 主板

|  |
| --- |
| 《如何选主板》<https://zhuanlan.zhihu.com/p/42709927> |

主板一定是根据CPU来选择的

### 1.2.1 选择CPU对应芯片组

芯片组是主板最重要的部分，选对芯片组的主板才能和CPU兼容。英特尔平台芯片组支持英特尔8代CPU的是300系芯片组的主板，档次从低到高分别为：H310、B360、H370、Z370。



H310：入门级，价格最低，适合家用办公。建议搭配奔腾G5400\G5500这类CPU使用。

B360：主流级，价格适中，适合后缀不带k的CPU。建议搭配酷睿i3-8100\i5-8400\i5-8500\i5-8600\i7-8700等。

H370：中高端，价格稍贵，建议搭配酷睿i7-8700使用。

Z370：高端级，价格贵，能超频，支持4000+高频内存，适合超频玩家。建议搭配i3-8350k/i5-8600k/i7-8700k等后缀带K的CPU。（注：英特尔后缀带k的CPU是可以超频的CPU）

### 1.2.2 选择版型

要根据机箱的大小选择合适的板型。最常见的是ATX大板和M-ATX小板，**大小板性能几乎没有差距**，仅扩展性有差别。



1. ATX型主板是高性能主板，芯片组多为X299、X399，适用于AMD的线程撕裂者、英特尔的i9-7920X等CPU，价格昂贵，很少见。
2. ATX型主板俗称“大板”，形状类似长方形，这类主板由于更大，所以有更多的插槽和接口，可扩展性较高。比如装双显卡、视频采集卡、PCI网卡等。
3. M-ATX型主板俗称“小板”，形状接近正方形，这类主板该有的接口都有，不过PCI-E插槽数量较少，可扩展性较小。但价格比大板便宜，预算不多的用户可以选择。
4. mini-ITX型主板体型更小，这类主板的接口仅属于刚刚够用，几乎没有扩展性，这种主板仅适用于桌面小机箱，较少见。

### 1.2.3 选择品牌

目前主流的主板品牌有：华硕、微星、技嘉、华擎、映泰、七彩虹。

记住以下口诀：高端主板买华硕；中端主板买技嘉、华擎；AMD主板买微星；入门主板买映泰、七彩虹。

**华硕：**三大板厂之首，主板bios界面做的简单易用。最出名的就是高端的ROG玩家国度系列主板，做工、用料都是顶级，超频玩家首选。华硕的中低端主板价格较贵，竞争力不强，所以买华硕建议买高端的主板。

**微星：**三大板厂之一，微星AMD芯片主板做的很好，比较出名的是迫击炮系列主板和战斧导弹系列主板，是AMD玩家首选。  
技嘉：三大板厂之一，技嘉主板用料扎实，比较出名的是AORUS“雕牌”系列主板，用料奢华，设计感强。

**华擎：**华擎被称为妖板，各种奇怪的主板比较多。华擎主板最大的特点是供电模块堆料比较夸张，比较出名的是z370 pro4系列和太极系列主板，性价比很高。

**映泰：**映泰主板价格便宜，面向中低端，做工用料多不错，稳定性很好，适合家用。

**七彩虹：**七彩虹主板价格便宜，没什么可圈可点的地方，稳定性很好，适合家用。

主板对电脑性能影响小，好主板和差主板主要体现在用料做工、外观设计、以及一些锦上添花的特色功能上。CPU供电模块越多、散热马甲越多，主板的超频能力越好。

### 1.2.4 总线

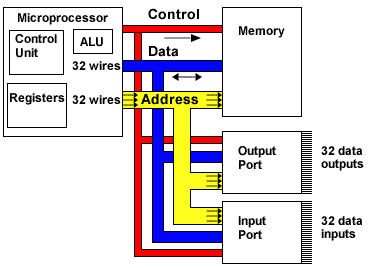
|  |
| --- |
| 《深入PCI与PCIe》<https://zhuanlan.zhihu.com/p/26172972>  **《深入PCI与PCIe-软件》**<https://zhuanlan.zhihu.com/p/26244141> |

任何**处理器系统**中通常都有三种类型的总线：

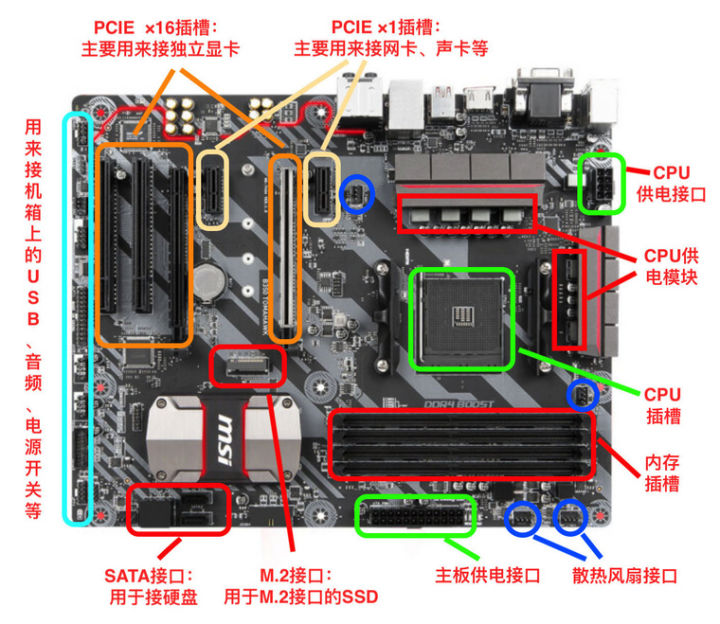
**地址总线：**这决定了处理器从中读取数据或将数据写入的内存中的位置。

**数据总线：**它包含已从内存位置读取或将被写入内存位置的内容。

**控制总线：**它管理组件之间的信息流，指示操作是读取还是写，并确保在合适的时间进行操作。



自PC在1981年被IBM发明以来，主板上都有扩展槽用于扩充计算机功能。现在最常见的扩展槽是PCIe插槽，实际上在你看不见的计算机主板芯片内部，各种硬件控制模块大部分也是以PCIe设备的形式挂载到了一颗或者几颗PCI/PCIe设备树上。固件和**操作系统正是通过枚举设备树**们才能发现绝大多数即插即用（PNP）设备的。



**PCI/PCIe的历史**

在我们看PCIe是什么之前，我们应该要了解一下PCIe的祖先们，这样我们才能对PCIe的一些设计有了更深刻的理解，并感叹计算机技术的飞速发展和工程师们的不懈努力。

**1. ISA (Industry Standard Architecture)**

**2. MCA (Micro Channel Architecture)**

**3. EISA (Extended Industry Standard Architecture)**

**4. VLB (VESA Local Bus)**

**5. PCI (Peripheral Component Interconnect)**

**6. PCI-X (Peripheral Component Interconnect eXtended)**

**7. AGP (Accelerated Graphics Port)**

**8. PCI Express (Peripheral Component Interconnect Express)**

科技的每一步前进都是为了解决前一代中出现的问题，这里的问题就是速度。作为扩展接口，它**主要用于外围设备的连接和扩展**，而外围设备吞吐速度的提高，往往会倒推接口速度的提升。**第一代ISA插槽**出现在第一代IBM PC XT机型上（1981），作为现代PC的盘古之作，8位的ISA提供了4.77MB/s的带宽（或传输率）。到了1984年，IBM就在PC AT上将带宽提高了几乎一倍，16位ISA第二代提供了8MB/s的传输率。但其对传输像图像这种数据来说还是杯水车薪。

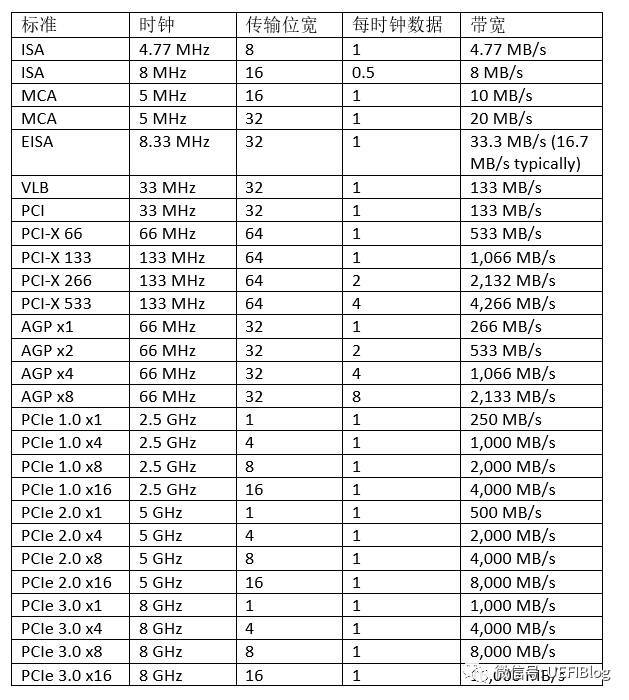
IBM自作聪明在PS/2产品线上引入了**MCA总线**，迫使其他几家PC兼容机厂商联合起来捣鼓出来**EISA**。因为两者都期待兼容ISA，导致速度没有多大提升。真正的高速总线始于**VLB**，它绑定自己的频率到了当时486 CPU内部总线频率：33MHz。而到了奔腾时代，内部总线提高到了66MHz，给VLB带来了严重的兼容问题，造成致命一击。

Intel在1992年提出**PCI（Peripheral Component Interconnect）总线协议**，并召集其它的小伙伴组成了名为 PCI-SIG (PCI Special Interest Group)（PCI 特殊兴趣组J）的企业联盟。从那以后这个组织就负责PCI和其继承者们（PCI-X和PCIe的标准制定和推广。

合作共赢的心态使得PCI标准得以广泛推广和使用。统一的标准撩拨起了外围设备制造商的创新，从那以后各种各样的PCI设备应运而生，丰富了PC的整个生态环境。PCI总线标准初试啼声就提供了133MB/s的带宽(33MHz时钟，每时钟传送32bit)。这对当时一般的台式机已经是超高速了，但对于服务器或者视频来说还是不够。于是AGP被发明出来专门连接北桥与显卡，而为服务器则提出PCI-X来连接高速设备。

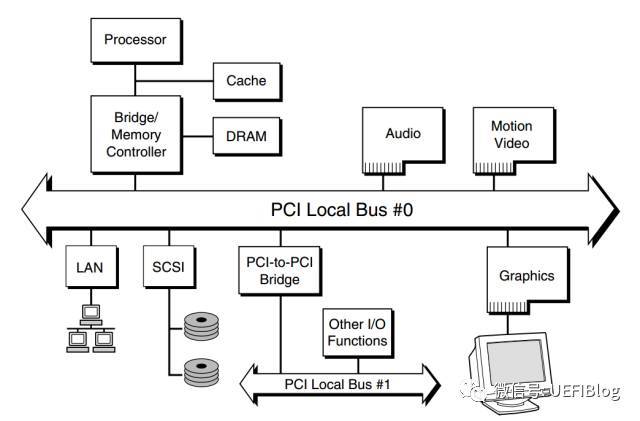
2004年，Intel再一次带领小伙伴革了PCI的命。PCI express（PCIe，注意官方写法是这样，而不是PCIE或者PCI-E）诞生了，其后又经历了两代，现在是第三代(gen3，3.0)，gen4有望在2017年公布，而gen5已经开始起草中。

下面这个大表列出所有的速度比较。其中一些x8,x16的概念后面细节部分有介绍。



### 1.2.5 PCI总线

桌面系统一般只有一个Host Bridge用于隔离处理器系统的存储器域与PCI总线域，并完成处理器与PCI设备间的数据交换。**每个Host Bridge单独管理独立的总线空间**，包括PCI Bus, PCI I/O, PCI Memory, and PCI Prefetchable Memory Space。桌面系统也一般只有一个Root Bridge，每个Root Bridge管理一个Local Bus空间，它下面挂载了一颗PCI总线树，在同一颗PCI总线树上的所有PCI设备属于同一个PCI总线域。一颗典型的PCI总线树如图：



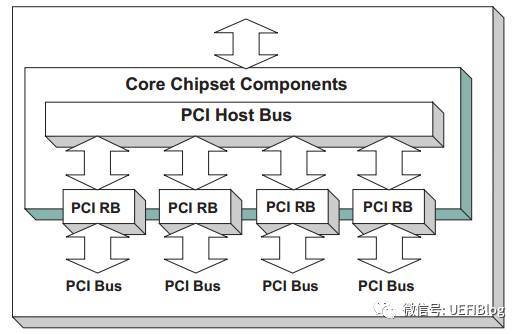
从图中我们可以看出 PCI 总线主要被分成三部分：

**1. PCI 设备。**符合 PCI 总线标准的设备就被称为 PCI 设备，PCI 总线架构中可以包含多个 PCI 设备。图中的 Audio、LAN 都是一个 PCI 设备。PCI 设备同时也分为主设备和目标设备两种，主设备是一次访问操作的发起者，而目标设备则是被访问者。

**2. PCI 总线。**PCI 总线在系统中可以有多条，类似于树状结构进行扩展，每条 PCI 总线都可以连接多个 PCI 设备/桥。上图中有两条 PCI 总线。

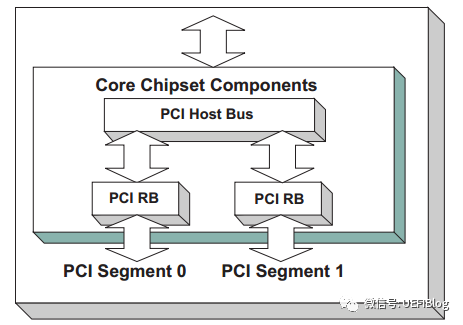
**3. PCI 桥。**当一条 PCI 总线的承载量不够时，可以用新的 PCI 总线进行扩展，而 PCI 桥则是连接 PCI 总线之间的纽带。

服务器的情况要复杂一点，举个例子，如Intel志强第三代四路服务器，共四颗CPU，每个CPU都被划分了共享但区隔的Bus, PCI I/O, PCI Memory范围，其构成可以表示成如下图：



可以看出，只有一个Host Bridge，但有四个Root Bridge，管理了四颗单独的PCI树，树之间**共享Bus等PCI空间**。

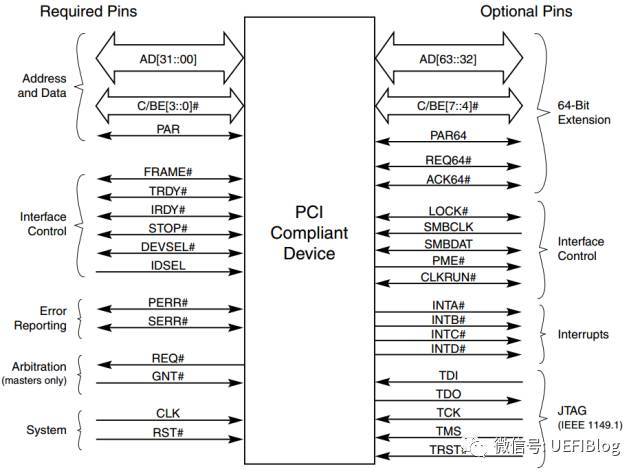
在某些时候，当服务器连接入大量的PCI bridge或者PCIe设备后，Bus数目很快就入不敷出了，这时就需要引入Segment的概念，扩展PCI Bus的数目。如下例：



如图，我们就有了两个Segment，每个Segment有自己的bus空间，这样我们就有了512个Bus数可以分配，但其他PCI空间因为只有一个Host Bridge所以是共享的。会不会有更复杂的情况呢? 在某些大型服务器上，会有多个Host bridge的情况出现，这里我们就不展开了。

PCI标准有什么特点吗？

**1. 它是个并行总线。**在一个时钟周期内32个bit（后扩展到64）同时被传输。引脚定义如下：



地址和数据在一个时钟周期内按照协议，分别一次被传输。

**2. PCI空间与处理器空间隔离。**PCI设备具有独立的地址空间，即PCI总线地址空间，该空间与存储器地址空间通过Host bridge隔离。处理器需要通过Host bridge才能访问PCI设备，而PCI设备需要通过Host bridge才能主存储器。在Host bridge中含有许多缓冲，这些缓冲使得**处理器总线与PCI总线工作在各自的时钟频率中，彼此互不干扰**。**Host bridge的存在也使得PCI设备和处理器可以方便地共享主存储器资源。处理器访问PCI设备时，必须通过Host bridge进行地址转换；而PCI设备访问主存储器时，也需要通过Host bridge进行地址转换**。

深入理解PCI空间与处理器空间的不同是理解和使用PCI的基础。

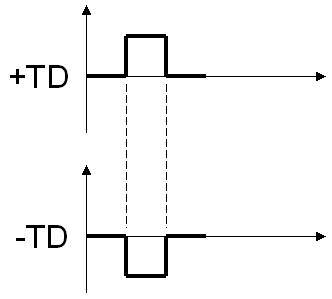
**3.扩展性强。**PCI总线具有很强的扩展性。在PCI总线中，Root Bridge可以直接连出一条PCI总线，这条总线也是该Root bridge所管理的第一条PCI总线，该总线还可以通过PCI桥扩展出一系列PCI总线，并以Root bridge为根节点，形成1颗PCI总线树。在同一条PCI总线上的设备间可以直接通信，并不会影响其他PCI总线上设备间的数据通信。隶属于同一颗PCI总线树上的PCI设备，也可以直接通信，但是需要通过PCI桥进行数据转发。

### 1.2.6 PCIe总线

PCI后期越来越不能适应高速发展的数据传输需求，PCI-X和AGP走了两条略有不同的路径，PCI-x不断提高时钟频率，而AGP通过在一个时钟周期内传输多次数据来提速。随着频率的提高，PCI并行传输遇到了干扰的问题：高速传输的时候，并行的连线直接干扰异常严重，而且随着频率的提高，干扰（EMI）越来越不可跨越。

|  |
| --- |
| 乱入一个话题，经常有朋友问我为什么现在越来越多的通讯协议改成串行了，SATA/SAS，PCIe，USB，QPI等等，经典理论不是并行快吗？一  次传输多个bit不是效率更高吗？从PCI到PCIe的历程我们可以一窥原因。 |

PCIe和PCI最大的改变是由并行改为串行，通过使用差分信号传输（differential transmission），如图



相同内容通过一正一反镜像传输，干扰可以很快被发现和纠正，从而可以将传输频率大幅提升。加上PCI原来基本是半双工的（地址/数据线太多，不得不复用线路），而串行可以全双工。综合下来，如果如果我们从频率提高下来得到的收益大于一次传输多个bit的收益，这个选择就是合理的。我们做个简单的计算:

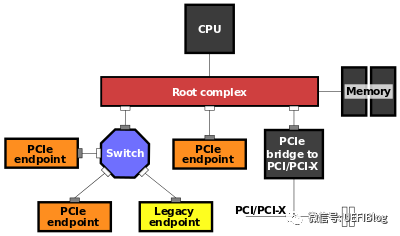
**PCI传输:** 33MHz x 4B = 133MB/s

**PCIe 1.0 x1:** 2.5GHz x 1b = 250MB/s (知道为什么不是2500M / 8=312.5MB吗？)

速度快了一倍!我们还得到了另外的好处，例如布线简单，线路可以加长（甚至变成线缆连出机箱！），多个lane还可以整合成为更高带宽的线路等等。

PCIe还在很多方面和PCI有很大不同：

**1. PCI是总线结构，而PCIe是点对点结构。**一个典型的PCIe系统框图如下：



一个典型的结构是一个root port和一个endpoint直接组成一个点对点连接对，而Switch可以同时连接几个endpoint。一个root port和一个endpoint对就需要一个单独的PCI bus。而PCI是在同一个总线上的设备共享同一个bus number。过去主板上的PCI插槽都公用一个PCI bus，而现在的PCIe插槽却连在芯片组不同的root port上。

**2.PCIe的连线是由不同的lane来连接的**，这些lane可以合在一起提供更高的带宽。譬如两个1lane可以合成2lane的连接，写作x2。两个x2可以变成x4，最大直到x16，往往给带宽需求最大的显卡使用。

**3.PCI配置空间从256B扩展为4k**，同时提供了PCIe memory map访问方式，我们在软件部分会详细介绍。

**4.PCIe提供了很多特殊功能**，如Complete Timeout(CTO)，MaxPayload等等几十个特性，而且还在随着PCIe版本的进化不断增加中，对电源管理也提出了单独的State（L0/L0s/L1等等）。这些请参见PCIe 3.0 spec，本文不再详述。

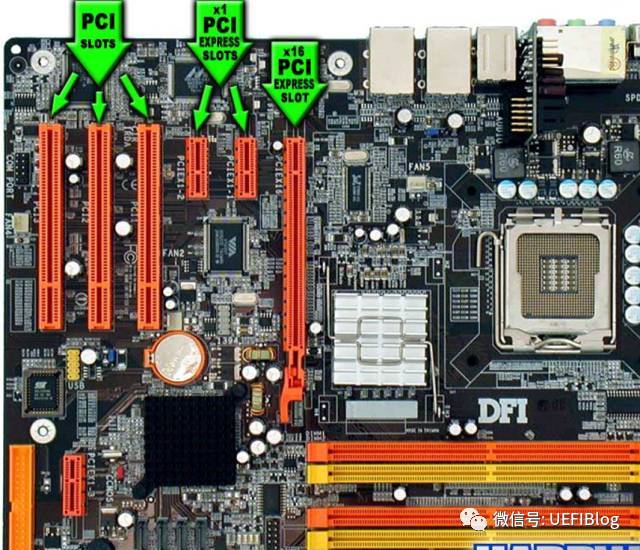
**5. 其他VC的内容**，和固件理解无关，本文不再提及。INT到MSI的部分会在将来介绍PC中断系统时详细讲解。

PCIe 1.0和2.0采用了8b/10b编码方式，这意味着每个字节（8b）都用10bit传输，这就是为什么2.5GHz和5GHz时钟，每时钟1b数据，结果不是312.5MB/s和625MB/s而是250MB/s和500MB/s。PCIe 3.0和4.0采用128b/130b编码，减小了浪费（overhead），所以才能在8GHz时钟下带宽达到1000MB/s（而不是800MB/s）。即将于今年发布的PCIe 4.0还会将频率提高一倍，达到16GHz，带宽达到2GB/s每Lane。



后记

对于一般用户来说，PCIe对用户可见的部分就是主板上大大小小的PCIe插槽了，有时还和PCI插槽混在一起，造成了一定的混乱，其实也很好区分：



如图，PCI插槽都是等长的，防呆口位置靠上，大部分都是纯白色。PCIe插槽大大小小，最小的x1，最大的x16，防呆口靠下。各种PCIe插槽大小如下：



常见问题：

Q:我主板上没有x1的插槽，我x1的串口卡能不能插在x4的插槽里。

A: 可以，完全没有问题。除了有点浪费外，串口卡也将已x1的方式工作。

Q:我主板上只有一个x16的插槽，被我的显卡占据了。我还有个x16的RAID卡可以插在x8的插槽内吗？

A: 你也许会惊讶，答案同样是：可以！你的RAID卡将以x8的方式工作。实际上来说，你可以将任何PCIe卡插入任何PCIe插槽中! PCIe在链接training的时候会动态调整出双方都可以接受的宽度。最后还有个小问题，你根本插不进去！呵呵，有些主板厂商会把PCIe插槽尾部开口，方便这种行为，不过很多情况下没有。这时怎么办？你懂的。。。。

Q: 我的显卡是PCIe 3.0的，主板是PCIe2.0的，能工作吗？

A: 可以，会以2.0工作。反之，亦然。

Q: 我把x16的显卡插在主板上最长的x16插槽中，可是benchmark下来却说跑在x8下，怎么回事?！

A: 主板插槽x16不见得就连在支持x16的root port上，最好详细看看主板说明书，有些主板实际上是x8。有个主板原理图就更方便了。

Q: 我新买的SSD是Mini PCIe的，Mini PCIe是什么鬼？

A: Mini PCIe接口常见于笔记本中，为54pin的插槽。多用于连接wifi网卡和SSD，注意不要和mSATA弄混了，两者完全可以互插，但大多数情况下不能混用（除了少数主板做了特殊处理），主板设计中的防呆设计到哪里去了！请仔细阅读主板说明书。另外也要小心不要和m.2(NGFF)搞混了，好在卡槽大小不一样。

### 1.2.7 外频

外频是由主板为CPU提供的基准时钟频率，一般常见的有100、133、166、200。我们说的FSB（Front System Bus）指的是系统前端总线，它是处理器与主板北桥芯片或内存控制集线器之间的数据通道，常见频率有400、333、533、800。

作为新手不必掌握那么多概念性的东西，只要记住以下几个公式：

**主频（核心频率）= 外频\*倍频（MHz）**

**Intel CPU前端总线（等效工作频率）= 外频\*4（MHz）**

**AMD CPU前端总线（等效工作频率）= 外频\*2（MHz）**

**CPU数据带宽 = 前端总线（等效工作频率）\*位宽/8（MB/s）（通常位宽为64位）**

**内存带宽 = 内存等效工作频率\*位宽/8（MB/s）（通常位宽为64位）（如，ddr3 1600带宽为12800MB/S）**

前端总线频率

总线是将信息以一个或多个源部件传送到一个或多个目的部件的一组传输线。通俗的说，就是多个部件间的公共连线，用于在各个部件之间传输信息。人们常常以MHz表示的速度来描述总线频率。总线的种类很多，前端总线的英文名字是Front Side Bus，通常用FSB表示，是将CPU连接到北桥芯片的总线。计算机的前端总线频率是由CPU和北桥芯片共同决定的。

北桥芯片负责联系内存、显卡等数据吞吐量最大的部件，并和南桥芯片连接。CPU就是通过前端总线（FSB）连接到北桥芯片，进而通过北桥芯片和内存、显卡交换数据。前端总线是CPU和外界交换数据的最主要通道，因此前端总线的数据传输能力对计算机整体性能作用很大，如果没足够快的前端总线，再强的CPU也不能明显提高计算机整体速度。数据传输最大带宽取决于所有同时传输的数据的宽度和传输频率，即数据带宽＝（总线频率×数据位宽）÷8。目前PC机上所能达到的前端总线频率有266MHz、333MHz、400MHz、533MHz、800MHz几种，最高到1066MHz。前端总线频率越大，代表着CPU与北桥芯片之间的数据传输能力越大，更能充分发挥出CPU的功能。现在的CPU技术发展很快，运算速度提高很快，而足够大的前端总线可以保障有足够的数据供给给CPU，较低的前端总线将无法供给足够的数据给CPU，这样就限制了CPU性能得发挥，成为系统瓶颈。

外频与前端总线频率的区别

前端总线的速度指的是CPU和北桥芯片间总线的速度，更实质性的表示了CPU和外界数据传输的速度。而外频的概念是建立在数字脉冲信号震荡速度基础之上的，也就是说，100MHz外频特指数字脉冲信号在每秒钟震荡一万万次，它更多的影响了PCI及其他总线的频率。之所以前端总线与外频这两个概念容易混淆，主要的原因是在以前的很长一段时间里（主要是在Pentium 4出现之前和刚出现Pentium 4时），前端总线频率与外频是相同的，因此往往直接称前端总线为外频，最终造成这样的误会。随着计算机技术的发展，人们发现前端总线频率需要高于外频，因此采用了QDR（Quad Date Rate）技术，或者其他类似的技术实现这个目的。这些技术的原理类似于AGP的2X或者4X，它们使得前端总线的频率成为外频的2倍、4倍甚至更高，从此之后前端总线和外频的区别才开始被人们重视起来，目前的主流产品均采用这些技术。

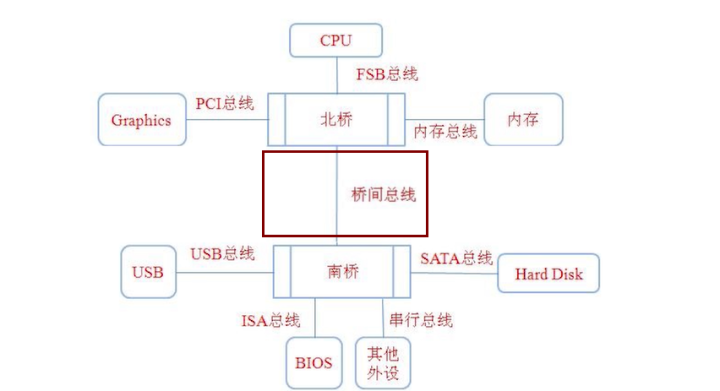
## 1.3 显卡

NVIDIA Corporation GK104GL [GRID K520] (rev a1)

## 1.4 内存

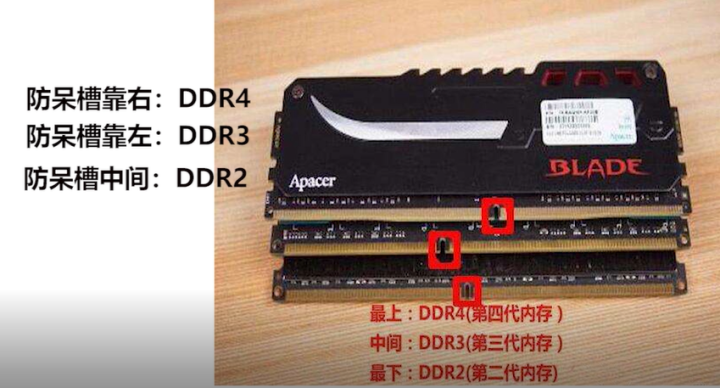
|  |
| --- |
| 《内存条怎么选》[**https://zhuanlan.zhihu.com/p/63604430**](https://zhuanlan.zhihu.com/p/63604430)  **《每个程序员都应该了解的内存知识》**[**https://lrita.github.io/2018/06/30/programmer-should-know-about-memory-1/**](https://lrita.github.io/2018/06/30/programmer-should-know-about-memory-1/) |

内存条是CPU可通过总线寻址，并进行读写操作的电脑部件。或者可以理解内存条是CPU连接其他设备的重要通道。虽然现在已经**没有了北桥**。

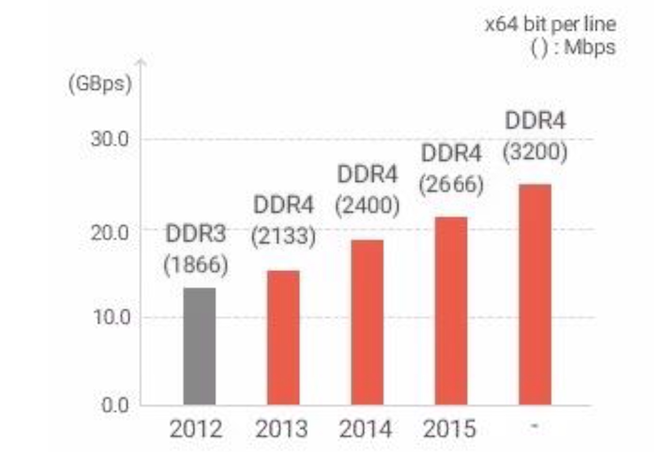


简单来说，CPU要运行程序需要从硬盘调取，但硬盘又有距离，所以需要用内存条来做一个连接。所谓几代只得内存条的类型，名字叫做Double Data Rate双倍速率同步动态随机存储器。简称为DDR几 目前最高是DDR4 也就是四代内存。

当前最受欢迎的是DDR3和DDR4的内存，那么两代之间又有什么差别呢！



从性能上来讲，代数越高 一般频率也越高，性能自然也越高，同时由于电压比DDR3低20%，DDR4可以降低功耗并降低TCO



从容量上来讲，现在工作中配置用的最多的是4G 8G 16G；那么就顺便讲讲内存条上的数字字符都代表什么意思。  
其中KVR代表品牌名 这边表示金士顿，L便是低压 X表示旗舰级 在同等性能和负载下相比标准版DDR3功耗可降低15%或者更多；16（频率1600）三代条子 表示的是DDR3 有时会看到4D之类的表示DDR4；数字4代表的就是4GB的内存。



**其他：颗粒，是否原厂，频率，时序见参考资料。**

## 1.5 存储

|  |
| --- |
| 《阵列 RAID详解》<https://blog.51cto.com/molinux/516008> |

**磁盘接口：ide、sata、 scsi、 sas 、 ssd 存储方案的选择**

**磁盘文件格式windows : nfs linux：ext3 ext4 mac：xnfs**

**磁盘挂载命令：mnt nfs**

硬件：

阵列卡的全称叫[磁盘阵列](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98%E9%98%B5%E5%88%97)卡 是用来做 RAID（[廉价冗余磁盘阵列](https://baike.baidu.com/item/%E5%BB%89%E4%BB%B7%E5%86%97%E4%BD%99%E7%A3%81%E7%9B%98%E9%98%B5%E5%88%97/10168623)）的。磁盘阵列是一种把若干硬[磁盘驱动器](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98%E9%A9%B1%E5%8A%A8%E5%99%A8)按照一定要求组成一个整体，整个磁盘阵列由阵列控制器管理的系统。[冗余](https://baike.baidu.com/item/%E5%86%97%E4%BD%99/67649)磁盘阵列RAID(Redundant Array of Independent Disks)技术1987年由[加州大学伯克利分校](https://baike.baidu.com/item/%E5%8A%A0%E5%B7%9E%E5%A4%A7%E5%AD%A6%E4%BC%AF%E5%85%8B%E5%88%A9%E5%88%86%E6%A0%A1/3755024" \t "_blank)提出，最初的研制目的是为了组合小的廉价磁盘来代替大的昂贵磁盘，以降低大批量数据存储的费用（当时RAID称为dundant Array of Inexpensive Disks 廉价的磁盘阵列），同时也希望采用冗余信息的方式，使得磁盘失效时不会使对数据的访问受损失，从而开发出一定水平的数据保护技术。

第一种是 **IDE阵列卡** ，以前主要用在一些数据重要或要接很多个硬盘的[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)与工作站电脑中，可以支持 RAID 0、1 、0+1、3、5。 现基本上已经淘汰了。

第二种是 **SATA阵列卡**，主要作用于大容量数据存储、网吧、[数据安全](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%AE%89%E5%85%A8)等服务器领域，同时一些低端卡也满足了一些家用客户的需求，能够支持 RAID 0、1、0+1、5 、6。

第三种是 **SCSI阵列卡** 使用在高端[工作站](https://baike.baidu.com/item/%E5%B7%A5%E4%BD%9C%E7%AB%99)或者是服务器中，可以支持很多块[SCSI接口](https://baike.baidu.com/item/SCSI%E6%8E%A5%E5%8F%A3)的硬盘。能够支持RAID 0、1、0+1、3、5 。这种阵列卡性能很好速度很快 当然价格也比较高。不过，现基本上已经淘汰了。

第四种是 **SAS阵列卡** 主要使用在一些高端工作站与[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)中，已经取代了昔日的SCSI接口，并且可以兼容SATA接口硬盘，能够支持 **RAID 0、1、0+1、5 、50、6、60**。



**RAID阵列：（Redundant Array of indenpensive Disk） 独立磁盘冗余阵列**: 磁盘阵列是把多个磁盘组成一个阵列,当作单一磁盘使用,它将数据以分段(striping)的方式储存在不同的磁盘中,存取数据时,**阵列中的相关磁盘一起动作,大幅减低数据的存取时间,同时有更佳的空间利用率**。磁盘阵列利用的不同的技术,称为RAID level,不同的level针对不同的系统及应用,以解决数据安全的问题。简单来说，RAID把多个硬盘组合成为一个逻辑扇区，因此，操作系统只会把它当作一个硬盘。

一般高性能的磁盘阵列都是以硬件的形式来达成,进一步的把磁盘存取控制及磁盘阵列结合在一个控制器(RAID controler)或控制卡上,针对不同的用户解决人们对磁盘输输入输出系统的四大要求:  
  (1)增加存取速度,

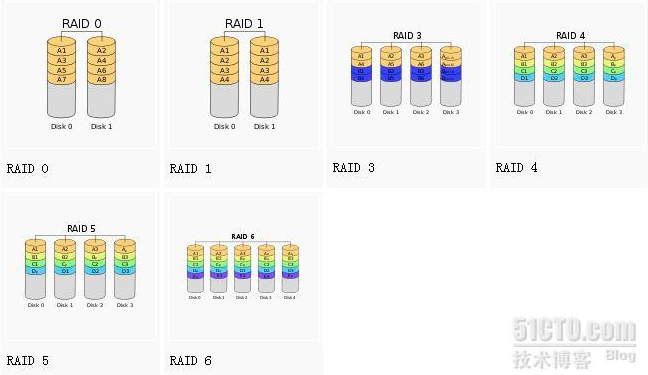
  (2)容错(fault tolerance),即安全性  
  (3)有效的利用磁盘利用率;  
  (4)尽量的平衡CPU,内存及磁盘的性能差异,提高主机的整体工作性能。  
**可提供的功能：**

   1.冗余(容错)

   2.性能提升

**RAID分类:** 1. 硬件RAID：用RAId接口卡来实现；需要内核支持其驱动，并且该类设备设备显示为SCSI设备，代号为/dev/sd\*  
 2. 软件RAID：用内核中的MD(multiple devices)模块实现，该类设备在/etc/下表示为：md\*；在现在的RH 5版本中使用mdadm工具管理软RAID；（虽然来说可以用软件模拟实现raid，但是相对对磁盘控制的功能及性能不如硬件实现的好，生产环境中最好使用硬件raid

**几种常见RAID类型描述：**



**1. RAID-0** ：striping（条带模式），至少需要两块磁盘，做RAID分区的大小最好是相同的（可以充分发挥并优势）;而数据分散存储于不同的磁盘上，在读写的时候可以实现并发，所以相对其读写性能最好；但是没有容错功能，任何一个磁盘的损坏将损坏全部数据；

**2. RAID-1**：mirroring（镜像卷），至少需要两块硬盘，raid大小等于两个raid分区中最小的容量（最好将分区大小分为一样），可增加热备盘提供一定的备份能力；数据有冗余，在存储时同时写入两块硬盘，实现了数据备份；但相对降低了写入性能，但是读取数据时可以并发，几乎类似于raid-0的读取效率；

**3. RAID-5**：需要三块或以上硬盘，可以提供热备盘实现故障的恢复；采用奇偶效验，可靠性强，且只有同时损坏两块硬盘时数据才会完全损坏，只损坏一块硬盘时，系统会根据存储的奇偶校验位重建数据，临时提供服务；此时如果有热备盘，系统还会自动在热备盘上重建故障磁盘上的数据；

存储方式：简单来说就是，磁盘阵列的第一个磁盘分段是校验值,第二个磁盘至后一个磁盘再折回第一个磁盘的分段是数据,然后第二个磁盘的分段是校验值,从第三个磁盘再折回第二个磁盘的分段是数据,以此类推,直到放完数据为止。这样数据与校验值的循环分离存储就可以达到一定的故障重建功能；但是raid-5的控制较为复杂，且计算大量的校验码，可能给系统造成额外计算的负担(软raid来说，硬件有自己的数据处理能力)

## 1.6 外设

暂略