

====2020/08/17=====

1. 权限管理：

a. 多级树形结构；

b. 过程：

i. 设计数据库的表结构，id, name,

parent\_id (根节点为0) , parent\_code (所有上级节点-隔开) ；

ii. 首先查询数据库，把所有信息放到List中，遍历list找到根节点 (parent\_id=0) ，从根节点出发，查找子节点。

iii. 执行selectChild()方法，传入根节点，返回根节点，在中间对根节点操作。拿到根节点的id，通过模糊查询parent\_code找到所有子节点放到List1中，再通过一个方法，找到根节点直属的下一级子节点List2，将List2中的子节点加到根节点的children上，再对list2中的每个子节点selectChild()查找其子节点，递归调用，能将孙子节点加到子节点的children上，遍历完所有子节点，return回来拿到根节点。

====2020/08/14=====

1. 垃圾回收方式：

a. 标记清除：两次扫描，第一次进行标记，第二次进行回收，适用于垃圾少；

b. 复制收集：一次扫描，准备新空间，存活复制到新空间，适用于垃圾比例大。有局部性优点，在复制的过程中，会按照**对象被引用的顺序**将对象复制到新空间，于是，关系较近的对象被放在距离近的内存空间可能性高，内存缓存有更高的效率；

c. 引用计数：每个对象有一个引用计数，对象的引用增加，计数增加；引用计数为0，则回收对象；缺点：循环引用，不适合并行；

## 2. char和varchar:

a. 定长和变长；char是**固定长度**，插入长度小于定长时，会用**空格补齐**；varchar时按**实际长度存储**；因此char的速度快，**空间换时间**；

b. 存储容量；char最多存**255个字符**，varchar在**5.0**以前是255字符，在5.0以后是**65532字节**；

c. varchar2支持**null**；

d. 对于进程修改长度的数据，varchar会产生**行迁移**；一行数据存在一个block中，修改后block空间不足以存储，产生行迁移，数据库会将整行数据迁移到新的block中；行链接，初始插入一行数据，大小大于block，会链接多个block开存储这一行。

## 3. 工厂设计模式：

a. 建造一个工厂来创建对象；

b. 构造对象的实例，而不用关系构造对象实例的细节和过程。

## 4. hashMap、hasgTable、concurrentHashHap:

a. hashMap：数组+链表；

b. hashTable：通过synchronized来保证线程安全；

c. concurrentHashMap：线程安全，通过**锁分段技术**提高并发访问率，hashTable在并发环境下效率低，是因为所有的hashTable线程都必须竞争同一把锁；concurrentHashMap使用锁分段技术，将数据分成小段的存储，给每一段数据配一把锁，一个线程占用锁访问一个数据，其他分段的数据还能够被访问。

## 5. spring:

a. IOC（控制反转，Inversion of Controller）：把对象的控制权交给容器，通过容器来实现对象的装配和管理；

b. AOP（面向切面编程，Aspect-Oriented Programming）：把通用的功能提取出来，织入到应用程序中，比如事务、权限、日志；

6. 孤儿进程和僵尸进程：

a. 孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程所收养，并由init进程对它们完成状态收集工作。

b. 僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

7. 聚簇索引和非聚簇索引

a. 聚集索引包含索引和数据，索引的叶子节点就是对应的数据。索引顺序和表中记录的物理顺序是一致的。

b. 非聚集索引将索引和数据分开，索引的叶子节点指向数据的对应行，等于做了一个映射。索引顺序和物理顺序不一致。

c. 一个表只能有一个聚集索引，通常默认是主键。

d. 聚簇索引的顺序就是数据的物理存储顺序，非聚簇索引的索引顺序和物理顺序无关，因此只能由一个聚簇索引；在B+树中，聚簇索引的叶子节点就是数据节点，非聚簇索引的叶子节点仍然是索引节点，只不过由一个指针指向对应的数据块。

e. MyISAM索引文件和数据文件是分离的，索引文件仅保存数据记录的地址。而在InnoDB中，表数据文件本身就是按B+树组织的一个索引结构，这棵树的叶子节点的data域保存了完整的数据记录。

f. InnoDB的二级索引和主键索引有很大不同，InnoDB的二级索引的叶子节点包含主键值，而不是行指针（row pointers），减小移动数据维护二级索引的开销，不需要更新索引的行指针。（列值=索引值=主键值）

g. InnoDB的二级索引叶子节点存放key字段+主键值，myisam存放的是列值与行号的组合。

h. 主索引是系统创建，二级索引是我们自己创建；主索引是表的主键；

====2020/08/13====

1. 状态:

- a. 线程: 创建, 就绪, 运行, 阻塞, 死亡; New, Runnable, Running, Blocked, Dead;
- b. 进程: 创建, 就绪, 运行, 阻塞, 死亡;

2. blocked, waiting, waiting-timed:

- a. 阻塞状态等待事件发生, 转换为就绪状态;
- b. blocked: 等待monitor lock;
- c. waiting: 等待notify;
- d. waiting-timed: 等待notify或者定时器数到0;

3. 文件描述符: fd (file descriptor) ; linux中每打开一个文件, 都有一个小的整数与之对应, 就是文件描述符, 0标准输入, 1标准输出, 2标准保存输出, <输入重定向符, >输出重定向符;

4. IO多路复用: 通过一种机制, 监视多个文件描述符, 一旦某个文件描述符就绪, 就能通知程序进行相应的IO操作。数据从内核到用户空间。

5. IO多路复用模型:

- a. 多个io复用一個线程;
- b. select: 轮询;
- c. poll: 最大连接数无上限;
- d. epoll: 事件通知。

6. select:

- a. select函数, 返回就绪的文件描述符fd的个数, 找到就绪的文件描述符, 保存在fdset中。每次调用select, 需要轮询一遍fd, 查看就绪状态; 且支持的最大fd数量有限, 32位系统默认是1024; 需要把fdset拷贝从用户态拷贝到内核态, 开销大;
- b. poll: 不存在最大文件描述符限制;
- c. epoll: 事件通知的方式。包含epoll\_create, epoll\_ctl, epoll\_wait三个方法;
  - i. epoll\_create: 创建一个句柄 (类似于指针), 占用一个fd;

ii. `epoll_ctl`: 注册监听事件, 把所有fd拷贝进内核一次, 并为每一个fd指定一个回调函数, 不需要每次轮询遍历fd; 当fd就绪, 会调用回调函数, 把就绪的文件描述符和事件加入一个就绪链表, 并拷贝到用户空间内存, 应用程序不用亲自从内核拷贝。

iii. `epoll_wait`: 监听`epoll_ctl`中注册的fd, 在就绪链表中查看有没有就绪的fd, 不用遍历fd。两种工作方式;

1. 水平触发: 默认工作方式, `epoll_wait`检测到fd就绪, 通知程序, 不会立刻处理, 下次`epoll`还会通知;
2. 边缘触发: `epoll_wait`通知会被立刻处理, 下次不会通知;

#### 7. 同步异步、阻塞非阻塞:

- a. 同步异步: 描述的是用户线程和内核的交互方式, 同步是指线程发起io请求后需要等待或者轮询, 内核io操作完成后才能继续执行; 异步是指用户线程发起io请求后仍然继续执行, 当内核io完成后会通知用户线程, 或者调用用户线程注册的回调函数。
- b. 阻塞非阻塞: 描述用户线程调用内核io的方式: 阻塞是指io操作彻底完成后才返回到用户空间; 非阻塞指io操作被调用后, 立刻返回给用户一个状态值, 无需等待io操作彻底完成。

#### 8. io同步阻塞, 同步非阻塞, 多路复用:

- a. 同步阻塞io: 内核进行io操作时, 用户线程阻塞;
- b. 同步非阻塞: 用户线程在发起io请求后立即返回, 然后进行轮询, 不断发起io请求, 知道数据准备完成后, 才正在进行io操作;
- c. io多路复用: 建立在select函数基础上, 使用select函数避免同步非阻塞io的轮询等待过程; 用户将需要io操作的socket添加到select中, 线程阻塞等待select调用返回, 当数据准备完成时,

socket被激活，select函数返回，用户线程发起io请求，完成io操作。优势是用户可以在一个线程内同时处理多个socket的io请求。用户注册多个socket，不断调用select读取被激活的socket，同一线程同时处理多个io请求。

d. 异步io：内核读取数据，放在用户线程缓存区中，内核io完成后通知用户线程直接使用即可。

9. HashSet和TreeSet：

a. HashSet是哈希表+红黑树，TreeSet是红黑树；

b. HashSet允许空值，无序存储；自定义类实现TreeSet，需要实现Comparable接口；

c. HashSet的add、remove、contains时间复杂度 $O(1)$ ，TreeSet是 $O(\log n)$ ；

10. 集合线程安全：vector、stack、hashtable、枚举；

11. 适用于查找的数据结构：

12. mysql高并发：

a. 代码中sql语句；

b. 数据库字段、索引；

c. 加缓存，redis/memcache；

d. 读写分离，主从复制；

e. 分区表；

f. 垂直拆分，解耦模块；

g. 水平切分；

13. 查找数据结构：

基于线性表的查找：

数组的顺序查找： $O(1)$

根据下标随机访问的时间复杂度为 $O(1)$ ；

二分查找： $O(\log_2 n)$

分块查找：介于顺序查找和二分查找之间

跳跃链表： $O(\log_2 n)$

基于树的查找：

二叉排序树： $O(\log_2 n)$

平衡二叉树： $O(\log_2 n)$

B-树： $O(\log_2 n)$

B+树： $O(\log_{1.44} n)$

红黑树；

堆；

计算式查找：

哈希查找： $O(1)$

====2020/08/13====

1. 24点问题：

a. what：4个数，加减乘除得到24，返回true，否则false。

2. 文本传输：

a. 服务端：

- i. 创建服务器套接字并等待客户请求；
- ii. 收到请求并建立连接；
- iii. 按行读取客户端数据并写入到文件l；
- iv. 完成后向客户端发送响应。

b. 客户端：

- i. 创建套接字；
- ii. 按行读取文本文件并发送；

====2020/08/12====

### 1. 适配器模式：

a. 定义：将一个类的接口转换成客户端需要的另一个接口，主要目的是**兼容性**，让原本接口不匹配的两个类协同工作。

b. 角色：目标接口，被适配者，适配器。

c. 通过适配器类，继承源角色，实现目标角色的接口，在适配器类中进行具体实现，达到适配的目的。

d. 类、对象、接口 适配器：

i. 类适配器：通过继承来实现，继承源角色，实现目标目标角色接口；

ii. 对象适配器：适配器拥有源角色实例，通过组合来实现适配功能，持有源角色，实现目标接口；

iii. 接口适配器：接口中有多个方法，用抽象类实现这个接口和方法，在创建子类时，只需要重写其中几个方法就行。

### 2. 装饰者模式：

a. 定义：以透明动态的方式来动态扩展对象的功能，是继承关系的一种代替方案。

b. 角色：抽象类，抽象装饰者，装饰者具体实现。

c. 一个抽象类有A方法，定义一个类作为抽象装饰者继承该抽象类，再创建具体装饰者类继承抽象装饰者类，并对其进行方法扩展，不用改变原来层次结构。

### 3. 适配器模式，装饰者模式，外观模式，区别：



- a. 适配器模式将对象包装起来改变其接口;
- b. 装饰者模式包装对象扩展其功能;
- c. 外观模式保证对象简化其接口。

4. 代理模式:

a. what: **给某个对象提供一个代理对象，由代理对象控制该对象的引用。**

b. why:

**i. 中介隔离作用**，在客户类和委托对象之间，起到中介作用;

**ii. 开闭原则**，增加功能，对扩展开发，对修改封闭，给代理类增加新功能。

c. where: 需要隐藏某个类，使用代理模式。

d. how: 代理角色、目标角色、被代理角色，**静态代理，动态代理**;

i. 静态代理:

1. what: 代理类创建实例并调用方法，在程序运行前，代理类已经创建好了;

2. why:

a. 优点: 开闭原则，功能扩展;

b. 缺点: **接口发生改变，代理类也需要修改。**

3. where: 需要代理某个类。

4. how: 代理对象和被代理对象实现相同接口, 通过调用代理对象的方法来调用目标对象。

ii. 动态代理:

1. what: 程序运行时通过反射机制动态创建代理类;

2. why:

a. 优点: 不需要继承父类, 利用反射机制;

b. 缺点: 目标对象需要实现接口。

3. where: 代理某个类;

4. how: 实现InvocationHandler接口, 重写invoke方法, 返回值时被代理接口的一个实现类。

iii. Cglib代理:

1. what: 通过字节码创建子类, 在子类中采用方法拦截来拦截父类的方法调用, 织入横切逻辑, 完成动态代理。

2. why:

a. 优点：不需要接口；

b. 缺点：对final无效。

3. where：不需要接口，代理。

a. SpringAOP中，加入容器的目标对象有接口，用动态代理；

b. 目标对象没有接口，用CGLib代理。

4. how：字节码。

5. 接口和抽象类：

a. 相同点：

i. 不能直接实例化；

ii. 包含抽象方法，则必须实现。

b. 不同点：

i. 继承extends只能支持一个类抽象类，实现implements可以实现多个接口；

ii. 接口不能为普通方法提供方法体，接口中普通方法默认为抽象方法；

iii. 接口中成员变量是public static final，抽象类任意；

iv. 接口不能包含构造器、初始化块。