

====2020/08/25====

一、redis的数据结构

1. String字符串;
2. List列表;
3. Hash;
4. Set集合;
5. Zet有序集合。

二、Mysql高并发

1. 代码中sql语句优化, 加索引;
2. 数据库的字段优化, 用合适的数据类型;
3. 加缓存, 用redis缓存;
4. 主从复制, 读写分离;
5. 分区表;
6. 垂直拆分, 水平拆分;

三、适用于查找的数据结构

1. 基于线性表的查找: 数组 (1) , 折半查找 ($\log n$) , 分块查找 (介于折半查找和二分查找之间) , 跳跃表 ($\log n$) ;
2. 基于树的查找: 二叉排序树, 平衡二叉树, B树, B+树;
3. 计算式查找: hash。

四、跳跃表:

1. 跳跃表是基于有序链表的扩展, 在链表的基础上增加了跳跃功能, 对于n个元素的链表, 采用 $\log n + 1$ 层索引指针, 查找、插入、删除的时间复杂度是 $\log n$ 。插入新元素时, 利用抛硬币的形式, 决定该节点是否在上一层建立索引。

五、流量控制和拥塞控制

1. 流量控制: 是指发送方发送数据太快, 接收方来不及接收, 造成数据丢失; 流量控制就是让发送方不要发送太快。利用滑动窗口实现流量控制。滑动窗口receiver window 的单位是字节。
2. 拥塞控制: 拥塞控制的对象是整个网络, 包括主机和路由器, 是全局性的考虑。通过拥塞控制避免短时间内大量流量的注入, 而引起的网络拥塞。主要方式有慢开始、拥塞避免、快重传、快恢复。

- a. 慢开始：发送方维护一个拥塞窗口cwnd (congestion window)，表示网络拥塞状况，发送窗口总是不超过拥塞窗口，当网络通畅时，增大拥塞窗口。拥塞窗口初始值为1，每次增加一倍大小，当拥塞窗口大小增大到阈值时，采用拥塞避免算法；
- b. 拥塞避免：当拥塞窗口达到阈值后，每次对拥塞窗口增加1，而不是翻倍；当发生拥塞时，执行快恢复；
- c. 快重传：接收方在收到失序的数据包后立即发送重复确认，当发送方连续3次收到重复确认序号时，立即重传可能丢失的报文段；同时触发快恢复；
- d. 快恢复：当发生快重传事件后，发送方会认为网络可能拥堵，将窗口大小和慢开始阈值减半，然后执行拥塞避免；

3. 流量控制是端到端的流量发送速率问题，拥塞控制是全局性问题，使整个网络平衡均匀。

六、红黑树和avl树的区别

- 1. avl树是高度平衡的，插入和删除操作会引起rebalance，效率低；红黑树不是高度平衡的，插入最多旋转2次，删除最多旋转3次；

七、数据库日志：重做日志（redo）、回滚日志（undo）、二进制日志（binlog）、错误日志、慢查询日志、通用查询日志、中继日志。

- a. 重做日志：持久性，记录事务执行后的状态。
- b. 回滚日志：原子性，保证事务发生前的版本
- c. 二进制日志：实现备份，是增量备份，只记录改变的数据。（备份）
- d. 错误日志：启动停止以及运行过程中的错误信息
- e. 慢查询日志：查询时间长或无索引的查询语句
- f. 通用查询日志：记录所有查询
- g. 中继日志：主从复制，读取主服务器的二进制日志，本地回放，实现同步。（复制）

八、binlog：

- 1. 在主服务器master中把数据更新记录到二进制日志；主服务器在每次进行数据更新提交事务成功后，会把本次事件记录到二进制日志中；
- 2. 从服务器slave会启动一个IO线程，把主服务器的二进制日志读取，写到自己的中继日志中；

3. 从服务器启动一个线程执行中继日志中的语句。

九、反射：动态获取信息，动态调用对象。

1. 什么是反射：反射机制是指程序在运行时能够获得自身的信息。在java中，只要给定类的名字，就可以通过反射来获得类的所有信息。

a. Java中的反射机制，对于任意一个类，都能够知道这个类的所有属性和方法，并且能够调用它的任意一个方法，这种动态获取信息以及动态调用对象的功能称为Java语言的反射机制。

2. 反射作用：在运行时判断任意一个一个类的对象，对象所属的类，类具有的变量和方法，对象的方法，在运行时调用对象的方法，创建新的类对象；

a. 功能篇一

i. 在运行时判断任意一个对象所属的类；

ii. 在运行时构造任意一个类的对象；

iii. 在运行时判断任意一个类锁具有的方法和属性；

iv. 在运行时调用任意一个对象的方法；

v. 生成动态代理。

3. 如何使用：

a. 使用Class类的forName()静态方法，传入类的全路径；Class c1 = Class.forName("类的全路径");

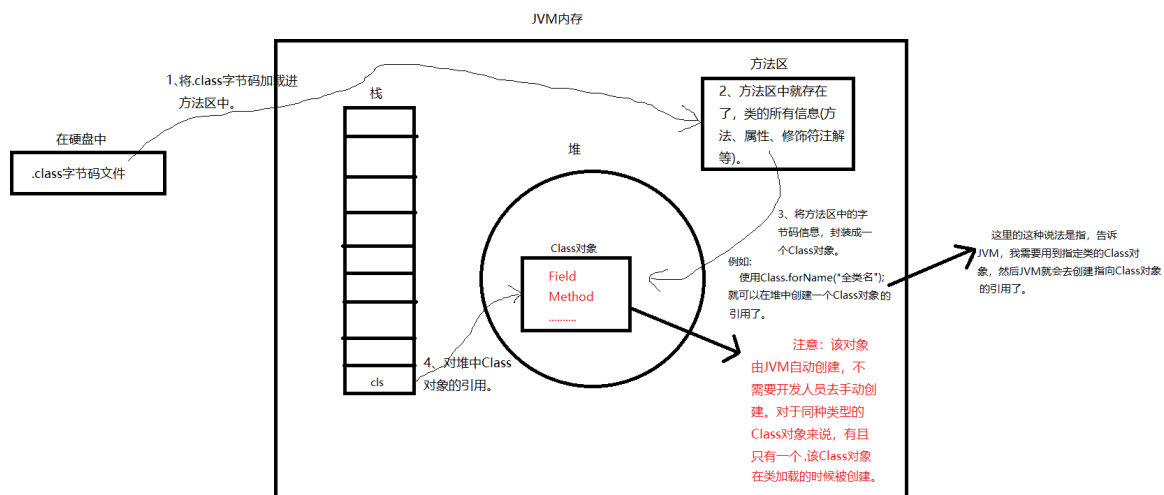
b. 调用某个类的class属性来获取该类对于的Class对象，这个属性保存的时Class对象的内存地址，Class c2 = Employee.class;

c. 调用某个对象的getClass()方法；Class c3 = e.getClass();

4. 反射的原理：

a. 因为Java中的Class也是一个对象，在硬盘中时一个文件，载入内存中就可以看成一个对象。Class对象的成员变量有Field、Constructor、Method、Modifier，对应的值就是属性、构造器、方法名、修饰符。

b.



https://blog.csdn.net/hyly_zhang

c. 反射原理:

- i. 开发人员保存的.java文件, 经过编译生成.class字节码文件; .class字节码文件保存在硬盘中。
- ii. 创建对象的过程, class字节码文件会加载到JVM内存的方法区, 方法区中就存在这个类的所有信息, 包括属性、方法、构造器、修饰符、注解等信息;
- iii. 在类加载的时候, JVM把方法区中字节码信息封装成一个class对象, 对同一类型的class对象, 有且只有一个, 保存在堆中, 在类加载的时候创建该class对象;
- iv. 使用 `Class.forName()` 传入全类名, 就可以创建这个class对象的引用, 引用保存在栈中, 就是告诉jvm需要用到的class对象, jvm会创建指向这个class对象的引用。
- v. 然后对这个class对象进行操作。

====2020/08/24=====

一、csrf:

1. cross-site request forgery, 跨站点请求伪造;
2. 网站A, 恶意网站B, 用户C:
 - a. 用户C打开浏览器访问网站A, 验证信息返回cookie, 用户A登录成功;
 - b. 同时, 用户A再同一浏览器访问恶意网站B, 网站B返回一些攻击性代码, 发送请求访问网站A;
 - c. 浏览器收到攻击性代码, 在用户不知情的情况下携带Cookie向网站A发送请求, 网站A会根据cookie信息, 以用户的权限处理该请求, 导致网站B的恶意代码被执行。
3. 防御: 验证HTTP Referer字段, 在请求地址中添加token, 在HTTP头中自定义属性并验证.
 - a. 验证http referer字段, 它记录了HTTP请求的来源地址;
 - b. 在请求中添加token, 因为用户信息存在cookie中, 黑客拿到cookie就能进行验证, 所以我们要使用token, 不存在cookie中, token利用js代码, 通过遍历把token写到所有的form标签中。
 - c. 在HTTP头中自定义属性并验证。

二、cookie、session、token:

1. cookie: 是一个数据, 由服务端生成, 发送给浏览器, 在浏览器存储, 以kv的形式存储, 下次访问同一网站时, 把cookie发送给服务器。
2. session: 服务端保存用户信息, 保存在内存中, 当访问数量多时, 会占用服务器性能, 用户离开网站的时候, session会被销毁; 当用户第一次通过浏览器使用用户名和密码访问服务器时, 服务器会验证用户数据, 验证成功后在服务器端写入session数据, 向客户端浏览器返回sessionid, 浏览器将sessionid保存在cookie中, 当用户再次访问服务器时, 会携带sessionid, 服务器会拿着sessionid从数据库获取session数据, 然后进行用户信息查询, 查询到, 就会将查询到的用户信息返回, 从而实现状态保持。缺点: 服务器压力大, CSRF跨站点请求伪造, 扩展性差。
3. token:

a. 服务器认证成功后，会对用户信息进行加密，生成加密字符串token，返回给客户端；浏览器会将接收到的token存储在Local Storage中；再次访问服务端，服务端对浏览器的token进行解密，对解密后的数据进行验证。

b. 是服务端生成的字符串，作为身份认证的令牌，发送给客户端，客户端使用token使用数据，不需要再输入账号密码验证数据库，减轻服务器压力，减少数据库频繁查询。

4. cookie和token的区别：

a. cookie：用户登录成功后，会在服务端生成session，返回cookie携带sessionId，客户端和服务端同时保存，用户再次操作时，需要带上cookie，在服务端进行验证，cookie有状态。

b. token：只保存在客户端，服务器收到数据后，进行解密验证，token是无状态的，适合跨平台。

三、TCP长连接和短连接

1. 短连接：tcp连接后，完成一次读写之后，立刻断开连接；优点是保证所有连接都是有效的连接，缺点是创建TCP连接和销毁连接的消耗较大；

2. 长连接：tcp建立连接后，完成一次读写操作，连接并不会主动关闭，后续的读写操作会继续使用这个连接。

四、死锁的必要条件：

1. 互斥条件，资源不允许其他进程访问；
2. 请求和保持请求，进程请求其他资源，被占用，占用自己的资源不放；
3. 不可剥夺，资源未使用完之前，不可被剥夺；
4. 循环等待，死锁后，存在进程-资源环形链。

====2020/08/23====

一、事务隔离性问题：

1. 脏读：一个事务读取到了另一个未提交的数据；
2. 不可重复读：一个事务内，多次读取同一个数据，在这个过程中，另一个事务对数据进行了修改，第一个事务前后读到的数据不一样；
3. 幻读：事务不是独立执行的，一个事务对表中的数据进行修改，涉及到表中的全部数据行，事务开启，这时，第二个事务向表中插入了一行新数据。那么，第一个事务提交后，用户会发现表中还有没有修改的数据。

二、事务隔离级别：

1. 未提交读：允许脏读，一个事务可以读到另一个事务未提交的数据；
2. 提交读：避免脏读，一个事务等待另一个事务提交后才能读取数据，oralce默认；
3. 可重复读：避免不可重复读，开始读数据时，事务开启时，不再允许修改操作，innodb默认；
4. 序列化：事务串行执行，可避免脏读不可重复读幻读。

====2020/08/22=====

一、数据库的三范式

1. 第一范式，字段不可分，原子性，数据库表的每一列都是不可分割的原子数据，而不能是集合，数组等非原子数据项；
2. 第二范式，非主键字段完全依赖于主键，通过主键能确定所有其他字段，部分依赖，由部分主键确定，多对多；
3. 第三范式，非主键字段不能相互依赖，不能传递依赖，a推出b，b推出c，一对多，分2张表。
4. 反范式，减少冗余会产生笛卡尔积，所以用空间换时间。

二、sql查询语句：select id from student order by score desc group by class limit 5;

====2020/08/22=====

一、事务隔离级别：

1. 隔离级别：

- a. 未提交读：允许脏读，可能读到其他会话中未提交事务修改的数据；
- b. 提交读：只能读到已经提交的数据，Oracle；
- c. 可重复度：在同一事务内的查询都是事务开始时刻一致的，Innodb。消除了不可重复度，存在幻读；
- d. 序列化：串行化读，每次读都需要获得表级共享锁，读写相互都会阻塞。

====2020/08/21=====

一、进程和线程：

1. 区别：

- a. 进程是程序的一次执行，是**资源分配**的基本单位；线程是**CPU调度**的基本单位，是进程中的一条**执行流程**，线程间**共享地址空间和文件资源**；
- b. 一个进程可以包含**多个线程**；
- c. 线程的**切换、创建、上下文切换**的开销小；
- d. 当进程只有一个线程时，可以认为进程就**等于**线程；有**多个线程**时，线程之间**共享虚拟内存和文件资源**。

2. 上下文切换：

- a. 进程的**上下文切换**：进程间共享CPU资源，CPU从一个进程切换到另一个进程；
- b. 两个线程属于同一进程：因为虚拟内存是共享的，所以资源不变，只需要切换线程的**寄存器和私有数据**。

3. 进程是程序的一次执行，是资源分配的基本单元；线程是进程当中的一条执行流程，线程之间可以并发运行且共享相同的**地址空间和文件资源**，每个线程都有独立的寄存器和栈，确保线程的控制流是相对独立的，线程是cpu调度和分派的基本单元，线程缺点是一个**线程挂掉**，进程中的其他线程也会挂掉。

4. 进程的数据结构是，PCB进程控制块，是进程的唯一标识，包含进程描述信息、进程控制信息、资源分配信息、CUP状态信息（断点处继续执行）。PCB如何组织，每个PCB通过链表的方式，把相同状态的进程链接在一起，组成队列，就绪队列、阻塞队列；另一种方式是索引表。

5. 进程的创建：**分配进程标识符，申请PCB，分配资源，加入就绪队列**

- a. 分配一个唯一的**进程标识符**，并申请一个空白的**PCB进程控制块**，PCB是有限的，申请失败则创建失败；
- b. 为进程**分配资源**，如果资源不足，则会进入等待状态；
- c. 初始化PCB；
- d. 如果进程的**就绪队列**能够接纳新进程，就插入到就绪队列，等待被调度。

6. 进程终止：**回收资源、撤销PCB，如果有子进程，会终止所有子进程**

- a. 包括**正常结束、异常结束、外界干预 (kill信号)**
 - b. **查找**终止进程的**PCB**进程控制块;
 - c. 如果处于执行状态, 则**立即结束执行**, 将**CPU资源**分配给其他进程;
 - d. 如果有**子进程**, 则终止所有子进程;
 - e. 将进程**资源**归还给父进程或操作系统;
 - f. 将PCB从队列中删除。
7. 阻塞, 当进程需要等待某一时间完成时, 可以调用阻塞语句把自己阻塞等待, 必须由其他进程唤醒。
8. 唤醒: 从阻塞队列, 插入到就绪队列。
9. 阻塞和挂起:
- a. 相同点: 都会释放CPU;
 - b. 阻塞的进程在内存中, 进程等待资源时发生,
 - c. 挂起的进程在外存中;
10. 孤儿进程是指, 父进程**退出**, 子进程仍在运行, 子进程将将为孤儿进程, 孤儿进程被init进程收养, 由init进程对他们完成回收工作; 僵尸进程是指, 子进程退出, 父进程没有调用wait或者waitpid去获取子进程的状态信息, 子进程的进程描述符仍然保存在系统中, 他就是一个僵尸进程。
11. 调度算法分类: (**时钟中断**)
- a. **非抢占式调度算法**, 一个进程运行, 直到**阻塞或退出**, 才会调用另一个进程;
 - b. **抢占式调度算法**, 时间片机制, 在时间间隔的末端发生时钟中断, 把CPU控制返回给调度程序, 一个进程只允许运行某段时间, 如果时间片结束, 仍在运行, 则**挂起**, 调度程序从就绪队列中挑选另一个进程。
12. 调度原则:
- a. CPU利用率;
 - b. 系统吞吐量, 单位时间CPU完成进程的数量;
 - c. 周转时间, 进程运行时间+等待时间;
 - d. 等待时间, 进程处于就绪队列的等待时间;
 - e. 响应时间, 交互式强的应用 (鼠标键盘), 响应时间要短。
13. 调度算法:

- a. **先来先服务**，非抢占式，先进入就绪队列先运行；
- b. **最短作业优先调度算法**，优先选择运行时间最短的进程；
- c. **高响应比优先调度算法**，响应比 = (等待时间 + 执行时间) / 执行时间；
- d. **时间片轮转调度算法**，进程分配一个时间片，一个时间片内没有运行完的进程回到就绪队列尾部，20~50ms；
- e. **最高优先级调度算法**，静态优先级，动态优先级是随着时间推移，增加等待进程的优先级；
- f. **多级反馈队列调度算法**：
 - i. 设置多个队列，优先级从高到低，优先级高时间片短；
 - ii. 新进程放在第一级队列末尾，先来先服务原则排队等待调度，如果第一级队列的进程在规定时间内没有运行完，会转到第二级队列末尾；
 - iii. 当较高优先级的队列为空，才调度较低优先级的队列；
 - iv. 进程运行时，有新进程进入较高优先级队列，则停止执行当前进程，并移入到原队列末尾，让较高优先级进程运行。
 - v. 兼顾长短作业，且有较好的响应时间。

二、参考资料：

1. <https://mp.weixin.qq.com/s/52Ud2ebDbsoLztVjrd-QNA>

====2020/08/19====

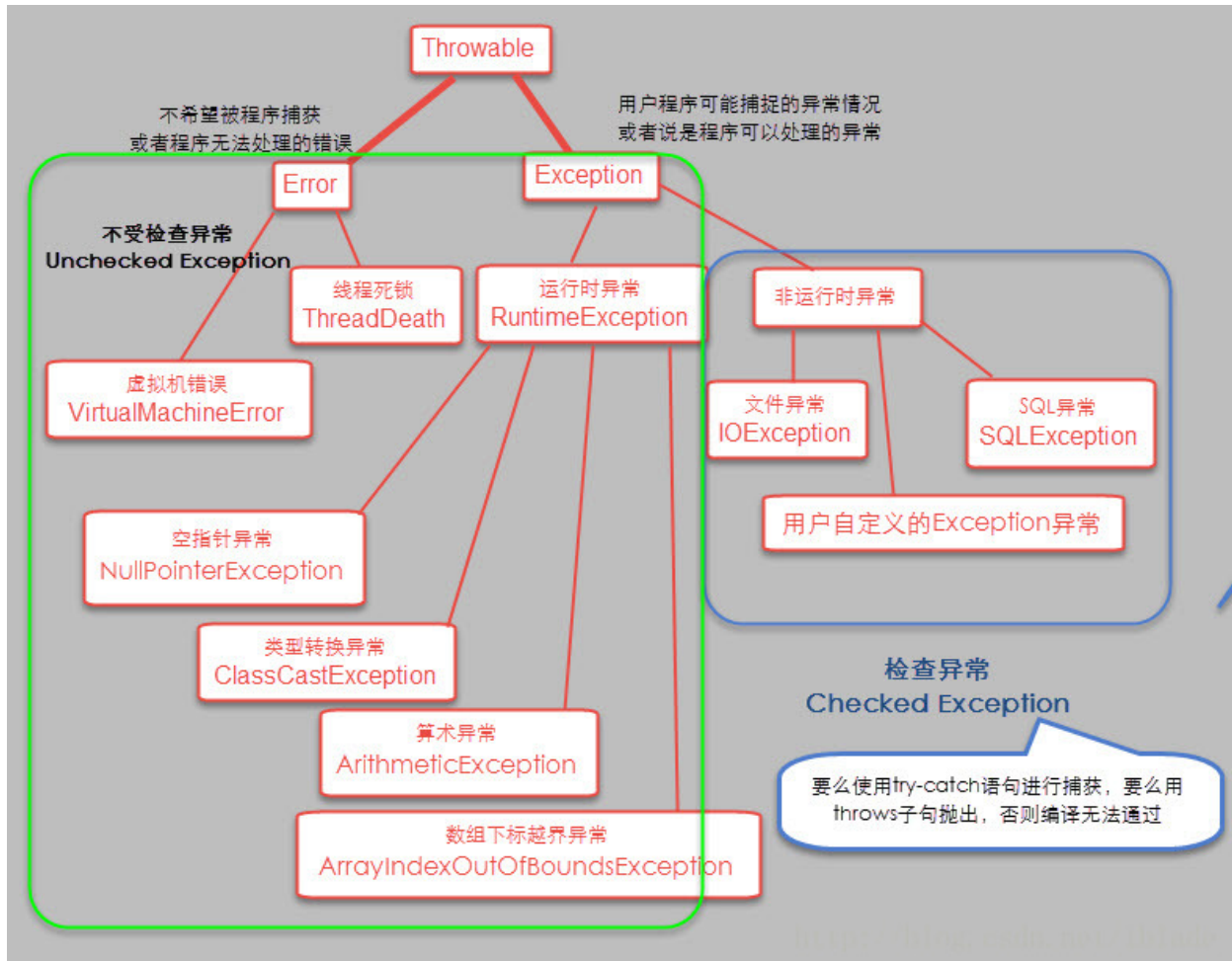
一、工厂设计模式：

1. 构造一个对象，不必关心构造对象的细节和过程，把细节和过程交给工厂类；
2. 简单工厂顾名思义，实现比较简单，只需要传入一个特定参数即可，不用知道具体的工厂实现类名，缺点就是违背了开闭原则。

3. 工厂方法和抽象工厂，遵守开闭原则，解耦了类关系。但是，扩展比较复杂，需要增加一系列的类。

4. 工厂方法和抽象工厂的区别就在于，抽象工厂关注于某一个产品族，当产品对象之间是有关联关系的一个产品族时用这种方式，而工厂方法没有产品族的概念。

二、Throwable, Error, Exception



====2020/08/18=====

一、索引的创建删除，基于mysql (<https://www.jb51.net/article/73372.htm>)

1. 索引作用：提高查询效率，适用于数据量大、查询涉及多个表。利用索引加速了where子句满足条件行的搜索，在多表连接查询时，在执行连接时加快了与其他表中的行匹配的速度。
2. 唯一索引，保证没有重复的值；主键索引，是一个唯一索引，一个表只有一个主键索引。

====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。

2. 聚簇索引和非聚簇索引:

a. 聚簇索引的索引和数据保存在一个文件, 索引顺序和数据物理存放数据一致, 一般使用主键作为索引, 主键自增使索引结构紧凑;

====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、hashTable、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. 孤儿进程是指, 父进程退出, 子进程仍在运行, 子进程将将为孤儿进程, 孤儿进程被init进程收养, 由init进程对他们完成回收工作; 僵尸进程是指, 子进程退出, 父进程没有调用wait或者waitpid去获取子进程的状态信息, 子进程的进程描述符仍然保存在系统中, 他就是一个僵尸进程。
- b. 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。
- c. 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。这种进程称之为僵尸进程。

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. 进程和线程：
 - a. 进程是程序的一次执行，是资源分配的基本单元；
 - b. 线程是cpu调度和分派的基本单元。
- 2. 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，结束；
- 3. 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；
- 4. 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；
- 5. 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。
- 6. IO多路复用模型：
 - a. 多个io复用一个线程；
 - b. select：轮询；

- c. poll: 最大连接数无上限;
- d. epoll: 事件通知。

7. 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;

8. 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。

9. 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。

10. 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 $O(\log n)$ ；

11. 集合线程安全：vector、stack、hashtable、枚举；

12. 适用于查找的数据结构：

13. 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;

14. 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。

2020/08/03=====

1. 自我介绍
2. 哈希表怎么实现，冲突怎么解决
 - a. 哈希表的底层数据结构是数组，很多地方也叫Bucket。首先通过将key的值传给hash函数，求出对应的索引，找到相应的下标进行存储，时间复杂度是 $O(1)$ 。
 - b. 解决方法：
 - i. 开放定址法
 - ii. 再hash法
 - iii. 链地址法（HashMap）
3. 维护一个堆（ $\log n$ ）
4. $N \log n$
5. B树和B+树
 - a. B树是多路平衡搜索树，它类似于普通平衡二叉树，区别是允许每个节点有多个子节点。B树为外部存储器（读写磁盘）设计，用于读写大块数据。
 - b. 空间局部性原理：存储器的某个位置被访问，它附近的位置也被访问。

c. B+树

i. 叶子节点存储数据，非叶子节点并不存储数据

ii. 叶子节点增加了链指针

d. 区别

i. B树的非叶子节点保存key和value，而B+树的非叶子节点只保存key的副本，叶子节点保存value（data值）。B+树查询时间复杂的 $\log n$ ，B树则与位置有关。

ii. B+树叶子节点数据是用链表连起来的，可以做到区间访问性，访问磁盘某个位置，附件位置也被访问。

iii. B+树适合外部存储，key小，磁盘单次IO信息量大，IO次数少。

e. Mysql的数据结构是B+树

6. 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. 二级索引：

7. MVCC多版本并发控制（Multi-Version Concurrency Control），实现对数据库的并发访问。MVCC是通过保存数据在某个时间点的快照来实现的，也就是同一时刻不同事物看到的相同表里的数据可能不同。从而实现并发控制。写写用锁，读写用mvcc。

8. DB日志

a. 重做日志：持久性，记录事务执行后的状态。

b. 回滚日志：原子性，保证事务发生前的版本

c. 二进制日志：实现备份，是增量备份，只记录改变的数据。（备份）

d. 错误日志：启动停止以及运行过程中的错误信息

e. 慢查询日志：查询时间长或无索引的查询语句

f. 通用查询日志：记录所有查询

g. 中继日志：主从复制，读取主服务器的二进制日志，本地回放，实现同步。（复制）

9. 事务用在并发操作多张表时，保证数据的完整性。一方发生错误，回滚数据，保证两次操作的安全。特征：

- a. 原子性：同一个事务是一个不可分割的操作单元，要么全部成功，要么全部失败。重做日志。
- b. 一致性：事务操作的前后状态是一致的，符合逻辑运算。
- c. 隔离性：并发执行多个不同的事务之间互不干扰。
- d. 持久性：事务一旦提交，对数据库的改变是永久性的。回滚日志。

10. 事务的隔离级别

a. 并发问题

- i. 脏读：一个事务读取了其他事务未提交的数据，读的是未提交。
- ii. 不可重复读：事务两次读取数据修改，读的是已提交。
- iii. 幻读：幻读是插入或删除操作，是已提交的。

b. 隔离级别

- i. 未提交读：事务A写数据，事务B不可写但可读修改未提交的数据。
- ii. 提交读：事务A写数据，禁止事务B读未提交的数据。
- iii. 可重复读：事务A写数据禁止事务B任何操作，事务A读数据禁止事务B写事务，。
- iv. 序列化：事务被定义为串行执行。

- c. 隔离级别对比
- d. Mysql: 可重复读; Oracle: 提交读。

11. TCP和UDP

12. TIME_WAIT

- a. 四次挥手，客户端和服务端都可以主动释放，以客户端为例：

- i. 客户端提出释放TCP请求，进入FIN_WAIT_1状态，向服务器发送FIN报文段。
- ii. 服务器的收到FIN报文，发送一个ACK报文，表示确认收到，此时处于半关闭状态，服务器进入CLOSE_WAIT状态。客户端收到ACK报文进入FIN_WAIT_2状态。
- iii. 服务器没有要发送数据时，发送FIN报文，由LAST_ACK状态，转为LISTEN状态。
- iv. 客户端收到FIN报文，向服务器端发送ACK报文，表示确认，客户端进入TIME_WAIT状态，待2个最长报文寿命MSL后进入CLOSE状态。
- v. 图示：

- b. TIME_WAIT状态作用：

- i. 主动关闭方发送的ACK包可能有延迟，从而触发被动关闭方重传FIN包，这样极端情况是2个MSL。

ii. 延迟发送的数据段会干扰新建立的连接，所以要等待。

13. URL、长度、字符、安全

14. Head请求，head方法和get方法相同，只不过服务器返回时不会返回方法体，用来检测超链接的有效性，和最近的修改。

15. HTTP

a. Http和Https

i. Http是超文本传输协议，Https增加了SSL（安全套接字层）协议用于加密传输。利用非对称加密实现对称加密。

ii. Http80端口，https443端口

b. Https请求过程

i. 客户端向服务端请求https连接；

ii. 服务端向客户端返回SSL证书，包含公钥；

iii. 客户端对证书进行验证，一般和本地的证书做比较，如果是信任的，客户端生成密钥，通过公钥加密发送给服务器；

iv. 服务器通过私钥解密得到对称加密的密钥

v. 通过对称加密的密文通信。

c. http1.0和http1.1

i. 长连接：http1.1默认使用长连接，维持一个长连接，不需要每次建立TCP3次握手连

接;

ii. 节约带宽: HTTP1.1支持只发送header信息, 在收到继续响应后, 在发送body;

iii. HOST域: web server上多个虚拟站点可以共享一个ip和端口。

d. http1.1和2.0

i. 多路复用: 同一个连接并发处理多个请求;

ii. 二进制分帧: 应用层和传输层之间, 加入二进制分帧层;

iii. 首部压缩: 对header数据进行压缩, 网络传输更快;

iv. 服务器推送: 客户端的一个请求, 服务器可以发送多个响应。将客户端需要的资源一起推送, 避免创建多次请求。

16. 进程、线程、协程

a. 区别:

i. 进程是程序运行和资源分配的基本单元

ii. 线程是CPU调度和分派的基本单元

iii. 协程是一个函数, 可以暂停执行过程, 类似于多线程调度。一个进程包含多个线程, 一个线程包含多个协程, 协程不是操作系统内核控制, 是程序控制, 所以不需要线程切换的资源消耗。

b. 语言

i. Go: 函数前加上go关键字, 这次调用就会在一个新的协程中并发执行;

ii. Python: 通过yield/send实现协程。

17. Linux

a. 查看进程、端口 ps显示进程 netstat显示端口

i. ps -ef 显示所有进程

ii. ps -ef | grep 进程名 进程名
查pid

iii. netstat -nap | grep 进程pid
进程pid占用端口

iv. netstat -nap | grep 端口号
端口查进程

b. 杀死进程

i. kill -9 进程号

ii. 默认状态是 kill -15 (sigterm先释放资源, 再停止, 会被阻塞)

-9 (singkill) 该信号不能被捕捉或忽略。

c. 杀死进程原理

i. 执行kill命令, 默认就是kill -15, 系统发送sigterm信号给程序, 程序释放资源, 然后停止。但是程序再做其他事情, 比如正在

处理IO的时候，不会立刻停止，sigterm信号阻塞。

ii. kill -9命令，系统发送sigkill信号给程序，强制杀死该进程。会留下不完整状态的文件。

d. ps中 -A/-e显示所有进程 f显示程序间关系 grep 查找

18. 方法的重载（overload）和重写（overwrite）

a. 重载是一个类定义多个同名方法，他们的参数不同。

b. 重写是子类继承父类，子类定义一个方法与父类有相同的名称和参数，子类对象使用这个方法，会调用子类中的定义。

c. C++的多态：在基类的函数前加上virtual关键字，在派生类中重写该函数。

19. 使用iterator进行遍历，erase进行删除

a. 对于vector，erase返回下一个iterator，用while去循环删除；

b. 对于map，删除iterator只影响当前iterator，所以for就行了。

20. auto_ptr, shared_ptr, weak_ptr, unique_ptr

a. 智能指针的作用是自动释放内存空间，避免内存泄漏

21. Redis

a. Redis是基于内存的、高性能的非关系型数据库。

b. 内存的速度比磁盘快很多，系统访问数据库，先访问内存的缓冲区查数据，如果缓冲区没有，再到磁盘数据库操作。

c. Redis持久化

- i. Redis是基于内存的，一旦重启数据会丢失，所以需要进行持久化操作，RDB（Redis DataBase）、AOF（Append Only File）。
- ii. RDB是基于快照的，把所有数据保存到RDB文件中，SAVE、BGSAVE、config配置文件3种方式实现，SAVE会阻塞，BGSAVE不会阻塞，创建子线程，由子线程创建RDB，缺点是若父线程修改，则会丢失数据。
- iii. AOF是Redis服务器执行写命令时，会将写命令保存到AOF文件中。命令追加到aof缓冲区，确认缓冲区写入文件，一般是1s同步一次，丢失数据少。缺点是文件大，恢复慢。默认aof恢复。

d. BIO、NIO、AIO

- i. 系统IO分为两个阶段：等待和操作
- ii. BIO同步阻塞IO，数据的读取写入必须阻塞在一个线程内等待其完成；
- iii. NIO同步非阻塞IO，多路复用机制，一个线程复制轮询，查看IO操作状态并进行操作；
- iv. AIO异步非阻塞IO，无需轮询，IO操作状态改变时，系统会通知对应的线程来处理。

e. Redis和java的hash区别

- i. 都是数组加链表，Java的hashMap对于长度大于8的链表，转为红黑树；
- ii. Redis有rehash操作，对于大量数据，Redis的hash性能高；
- iii. Redis的hash冲突时从头部插入， $O(1)$ ，HashMap在1.6时头部插入，1-8是尾部插入，所以是 $O(n)$ 。

22. 单例模式

2020/08/07=====

1. 自我介绍

2. 浅拷贝和深拷贝

a. 浅拷贝指向已有内存

b. 深拷贝指向新内存

3. 空间换时间，加大CUP的吞吐量，内存的IO以64位为单位进行。如果64位的数据，从第1位开始读，而不是从第0位，CUP要进行两次IO。

4. 是

5. 自旋锁和互斥锁

a. 自旋锁：不释放cpu，别的线程会自旋，尝试获得自旋锁，超过次数，挂起；

b. 互斥锁：释放cup，其他线程挂起，知道操作系统唤醒它。

c. 阻塞不释放cpu，挂起释放cpu。

d. 悲观锁和乐观锁：

i. 悲观锁：先锁定，再操作；分为共享锁和排他锁，多个事务公用一把共享锁，只能读不能写，如写数据时，变为排它锁，其他事务不能读写。

ii. 乐观锁假设数据一般情况不会冲突，所以在数据提交更新时加锁检测。实现方式：数据库版本号，乐观锁控制的标准增加时间戳。

6. 服务端开发能力：

- a. 语言的能力，从语法到编程能力，还有编程思维的理解；
- b. 数据库，熟练数据库的交互；
- c. 基础知识，数据结构，算法，网络，操作系统
- d. Linux编程，服务器的使用，主流的tomcat、apache这些，的使用和拓展；
- e. 应用方面，开发工具的使用，编译器，git、svn版本控制，其他的一些插件
- f. 个人能力，交流，项目经验

2020/08/07=====

1. 自我介绍。

2. JVM内存模型：

- a. 程序计数器：每条线程有独立的计数器，记录线程执行的位置；
- b. Java虚拟机栈：基本数据类型和对象的引用；
- c. 本地方法栈：调用的本地方法，也就是非Java语言的方法，和Java虚拟机栈类似；
- d. 堆：对象的实例；新生代（eden+2个survivor（from,to），8: 1: 1），老年代（15次gc）
- e. 方法区：常量和静态变量，class文件。

3. GC回收：

- a. Java堆中的新生代和老年代，新生代进程被gc回收，老年代较少。永久代在方法区，Jdk1.8取消了永久代，改为元数据，也是方法区。
- b. 回收机制：年轻代Eden区和2个survivor区，from和to；
 - i. 新对象分配在Eden区；

- ii. Eden区满了，存活的对象复制from区，放不下则全放老年代，Eden区内存回收；
- iii. Eden区又满了，Eden和from存活的对象复制到区，放不下放老年代，Eden和from回收；
- iv. 复制15次，放老年代，老年代满，调用Full GC（老年代满，永久代满，主动调用），前面时Minor GC。

4. 调用Minor GC和Full GC；

5. G1: Garbage First: 把堆分为很多区域（Region），初始标记，并发标记，最终标记，筛选回收。

- a. 初始标记：标记GC Roots直接关联的对象，方法区、JVM栈、本地栈引用的对象；
- b. 并发标记：从GC Roots出发，找出存活对象，和用户线程并发执行；
- c. 最终标记：用户线程变动的对象；
- d. 筛选回收：对各个region区域回收价值和成本排序，指定回收计划。

6. 线程池：可以创建若干个线程放在池子中。

a. 参数：

- i. corePoolSize：核心线程数
- ii. queueCapacity：阻塞队列大小
- iii. maxPoolSize：最大线程数
- iv. keepAliveTime：线程空闲时间
- v. rejectedExecutionHandler：任务拒绝处理器

b. 线程池执行过程：

- i. 当前任务 < 核心线程数，创建线程；
- ii. 大于core线程，队列不满，任务放入队列
- iii. 大于core线程，队列满，

1. $< \text{max线程}$, 创建线程

2. $\geq \text{max线程}$, 抛异常

RejectedExecutionException, 拒绝任务。

c. 四种池:

i. newCachedThreadPool: 可缓存线程池, 灵活回收线程

ii. newFixedThreadPool: 固定长度线程池, 控制最大并发

iii. newScheduledThreadPool: 定时线程池

iv. newSingleThreadExecutor: 单线程线程池, 唯一工作线程, 指定顺序执行

7. 拒绝策略:

a. AbortPolicy: 丢弃任务并抛出异常
RejectedExecutionException

b. DiscardPolicy: 丢弃任务不抛出异常

c. DiscardOldestPolicy: 丢弃队列最前面的任务, 重新提交

d. CallerRunsPolicy: 由调用者处理该任务

8. hashMap

9. Spring相关

10. TCP和UDP区别

a. 面向连接, 可靠性, 实时性, 首部开销, 点到点

11. TCP和UDP, 假设客户端主动发起。

a. TCP三次握手

i. 客户端发送编号SYN=J和随机序列号seq=J, SYN_SEND状态;

ii. 服务端由SYN=1, 就知道要建立请求, 标志位SYN和确认值ACK = 1, 确认值编号

$ack=seq+1=J+1$ ，随机产生序列 $seq=K$ ，发送给客户端，SYN_RCVD状态；

iii. 客户端检查 $ack=J+1$ ，ACK=1，正确则标准ACK=1， $ack=k+1$ ，发送给服务端，服务端检查ACK=1， $ack=k+1$ ，建立连接，ESTABLISHED。

iv. 3次

1. 客户端发送请求，SYN=1， $seq=x$ ；
2. 客户端发送请求，SYN=1，ACK=1， $ack=x+1$ ， $seq=y$ ；
3. 客户端发送请求，ACK=1， $ack=y+1$ 。

b. TCP四次挥手

- i. 客户端发送断开请求FIN=1、 $seq=x$ ，FIN_WAIT状态；
- ii. 服务端发送确认ACK=1， $ack=x+1$ ， $seq=y$ ；CLOSE_WAIT状态；
- iii. 服务器发送断开请求FIN=1，ACK=1， $seq=z$ ， $ack=y+1$ ，LAST_ACK状态；
- iv. 客户端确认断开，TIME_WAIT状态，等待2个MSL，ACK=1， $seq=z+1$ 。

12. TCP流量控制和拥塞控制

- a. 流量控制：滑动窗口大小，是端到端的，发送方发送太快，接收端无法接收，通过滑动窗口大小来控制接收窗口值，控制发送方发送数据的速率；

b. 拥塞控制：解决过多的数据注入到网络，超负荷，是全局的；慢开始，拥塞避免、快重传、快恢复4个策略；

i. 慢开始：窗口值设为1，每次增大一倍，触发门限限制则进入拥塞避免阶段；

ii. 拥塞避免：窗口每次加1，触发拥堵，窗口大小和门限限制变为一半；旧版本回到慢开始，新版到快恢复；

iii. 快重传：数据M2丢失，接收方重复发送M1的确认数据，发送发收到3次M1的确认以后，会立刻重发M2，同时触发快恢复；

iv. 快恢复每次加1。

13. 协程

a. 进程是程序运行和资源分配的基本单元；

b. 线程是CUP调度和分派的基本单元；

c. 协程是一个函数，可以暂停执行过程。

14. 进程通信方式：

a. 同步通信（管道、共享内存），异步通信（信号、消息队列）；

b. 低级通信（信号、信号量）传递少量数据，高级通信（消息队列、管道、共享内存）传递大量数据；

c. 管道：类似于缓存，一个进程吧数据放入缓存区域，另一个进程拿，单向传输，效率低；

d. 消息队列：也类似于缓存，如果数据大，拷贝花费时间多；

e. 共享内存：两个进程各自拿出一块虚拟内存空间，映射到物理内存；

f. 信号量：本质是计数器，实现进程间的互斥和同步。

15. NIO和IO

a. NIO同步非阻塞IO；多路复用：多个IO复用一个线程。

b. IO面向流、NIO面向缓冲区；IO阻塞，NIO非阻塞；NIO的选择器允许一个单独的线程来监控多个输入通道。

c. Linux

d. 触发:

- i. 水平触发: 默认工作模式, `epoll_wait`检测到文件描述符就绪, 通知程序, 程序不会立刻处理, 未处理的下次`epoll`还会通知;
- ii. 边缘触发: `epoll_wait`通知会被立刻处理, 下次不会再通知。

e. Window:

- i. Select (选择) : 轮询
- ii. `WSAsyncSelect` (异步选择) : 网络事件以消息形式通知应用程序
- iii. `WSAEventSelect` (事件选择) : 监听
- iv. OverLapped I/O (事件通知) : 设置缓冲区。

16. InnoDB、MyISAM、Memory、Archive。

17. 存储结构、事务、锁

a. 存储结构:

- i. MyISAM: .frm表结构, .MYD数据文件, .MYI(index)索引文件;
- ii. InnoDB: .frm表结构, idb数据和索引。
- iii. MyISAM非聚集索引, 引用和数据分开存储, 索引查找时, 叶子节点存储数据所在地址; InnoDB是聚集索引, 叶子节点存储整个数据行所有数据。

b. 事务: MyISAM不支持事务; InnoDB支持事务, 安全。

c. 锁: MyISAM表锁; InnoDB行锁, 索引查找失败, 行锁会转为表锁。

18. MVCC

19. 索引

a. 索引创建:

- i. 经常查询的字段;
- ii. 索引不是越多越好, 占用空间, 写操作会造成性能差;
- iii. 表更新索引也更新, 索引经常更行不索引;
- iv. 数据量小不索引;
- v. 定义有外键, 建立索引。

b. Sql优化:

- i. 在表中建立索引, where、group by使用到的字段建索引;
- ii. 避免select *
- iii. 避免in和not in (全表扫描), 使用between、exists;
- iv. 避免or, like, 判断null, =前面表达式, 1=1 (放弃索引, 全表扫描)

20. 用explain关键字查看执行计划, update、delete在5.6之后可以查看。

- a. Id: 操作表的顺序。Id值大优先, id相同, 从上往下, 顺序执行;
- b. select_type: select子句的类型;
- c. type: 访问类型;
- d. possible_key: 可能索引
- e. key: 索引
- f. key_len;
- g. ref、rows、extra;

21. 1

22. 1

23. 分布式

- a. 集群: 同一个业务, 高性能和高可用性, 一组服务器连接在一起协作完成任务, 可被看作一台计算机, 任务分配到每个节点上;

- b. 分布式，一个业务拆分成不同的子业务，部署在不同的服务器上。
 - c. 微服务：可以部署在同一个服务器上。
24. 一致性hash算法：（服务器IP、 2^{32} 取模、hash环、顺时针、虚拟节点）
- a. 普通hash算法，缓存服务器发生变化时，几乎所有缓存位置改变，大量缓存同一时间失效，服务器发生雪崩，服务器压力过大崩溃。
 - b. 对文件名称进行hash，一致性hash算法，对 2^{32} 取模，构成hash环，服务器IP地址对 2^{32} 做hash，文件和服务器的节点在hash环上对应位置，文件按照顺时针存在最近的服务器上；
 - c. 某个服务器故障时，只会影响该服务器的缓存内容，其他服务器可通过缓存找到文件，服务器节点映射虚拟节点，避免服务器节点密集时。
25. String、Hash、List、Set、zSet
26. 跳跃表：（有序、二分查找、关键节、多层索引、抛硬币）
- a. 跳跃表（SkipList），基于有序链表的扩展；
 - b. 对于n个元素的链表，采取 $\log n + 1$ 层索引指针，查找是 $O(\log n)$ 的时间复杂度。
 - c. 利用抛硬币的形式，决定该节点是否在上一层建立索引。
 - d. 跳跃表结构随意，二叉查找树需要rebalance实现平衡。

2020/08/09=====

[https://www.nowcoder.com/discuss/455369?
type=post&order=time&pos=&page=3&channel=666&source_id=search_post](https://www.nowcoder.com/discuss/455369?type=post&order=time&pos=&page=3&channel=666&source_id=search_post)