====2020/09/010====

一、java8

1. Stream API

a. 作用：通过Stream API，我们不需要for循环，就能对集合进行操作；

b. 使用：Stream.of或者lsit.Stream；

c. 方法：map遍历，filter过滤，forEach循环，limit，sort，Match。

二、mysql

1. #{}和${}的区别

a. #将传入的数据都当成一个字符串，会对自动传入的数据加一个双引号；

b. $将传入的数据直接显示生成在sql中，无法防止sql注入。

c. ~~#{}：使用预编译，对应JDBC的PreparedStatement；使用#{}时，具体的sql语句的参数会被加上单引号，这也是导致排序失败的原因~~

d. ~~${}：mybatis不会修改或者转义字符串，直接输出变量值~~。

====2020/09/010====

一、java基础

1. volatile、synchronized、lock（lock是类，其余两个是关键字）
2. voliate，保证了可见性、有序性，不能保证原子性，不能保证线程安全

a. 可见性

b. 有序性

3. synchronized，在多线程的环境下，只能由1个线程获得资源，是线程私有的。
4. 索引覆盖

a. 索引覆盖是指一个查询语句的执行只用从索引中就能够取得，不必从数据表中读取。

b. 索引包含了查询正在查找的所有字段（包括select、join、where用到的所有字段），通过索引就可以返回所需要的数据，而不必查到索引之后再进行回表操作，减少IO，提高效率。当发起了一个索引覆盖查询时，在explain查看执行计划，它的extra显示为using index。

5. 回表

a. 回表，在查询语句中，select所需获得列中有非索引列，那么数据库需要2次查询，

b. innodb时聚簇索引，在innodb的非主键索引中，非主键索引的叶子节点保存的是主键id，拿到主键后再进行主键索引，找到相应的数据，需要两次查找过程就是回表。

c. 解决方法：使用覆盖索引，覆盖索引是覆盖了多个字段的索引，建立一个组合索引，把需要的字段都放在里面。比如说查username和password：

i. 如果只有username索引，第一步根据username找到对应数据行的主键id，再根据id找到相应的数据；

ii. 建立组合索引后，根据username可以找到password，因为passwoed已经是索引的一部分，所以只要一步查找就可以找到。

6. 聚簇索引的优缺点

a. 优点：

i. 主键和行数据是一起被载入内存的，找到叶子节点，就可以立刻把行数据返回。行数据和叶子节点存储在一起，同一页会有多条行数据，访问同一数据页的不同行时，已经把页加载到了内存中，会在内存中完成访问；

ii. 辅助索引（非主键索引）使用主键作为指针，而不是使用地址，当修改的时候，出现行移动的时

候，无需更改辅助索引叶子节点的数据；

iii. 把相关的数据保存在一起，适用于排序的场景，适用于去除一定范围数据时；

b. 缺点：

i. 维护索引的代价高，在插入新行或者主键被更新导致要分页的时候。

7. ~~voliate~~

a. ~~可见性、有序性、不能原子性：~~

i. ~~如果一个字段被声明为volatile，那么可以保证多个线程看到的值是一致的；~~

ii. ~~禁止指令重排序；~~

iii. ~~volatile只能保证可见性，不能保证原子性。~~

b. ~~定义：Java编程语言允许线程访问共享变量（成员变量或静态成员变量），为了确保成员变量能够被准确和一致地更新，线程应该通过排它锁单独获得这个变量。JVM通过内存屏蔽来实现volatile。~~

c.

d. ~~类的共享变量（~~成员变量~~或者~~静态成员变量~~）被volatile修饰后，~~

i. ~~保证可见性，不保证原子性。保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这个新值对其他线程来说是立即可见的；~~

ii. ~~禁止进行指令重排序。~~

e. ~~使用volatile的场景：~~

i. ~~对变量的写操作不依赖于当前值；~~

ii. ~~变量中没有包含影响外部的其他变量。~~

f. ~~static表示属于类，不必创建对象就可以使用。~~

g. ~~成员变量（全局变量）和局部 变量~~

i. ~~成员变量是定义在方法外部，作用在整个类中；~~

~~成员变量会随着方法的生命周期结束而结束。~~

====2020/09/08====

一、框架

1. 消息队列

a. 消息队列中间件是分布式系统的重要组件，主要使用场景有异步处理、应用解耦、流量削峰，实现高并发、高可以、可伸缩、最终一致性。使用较多的消息队列有Kafka、RocketMQ、RabbitMQ、ActiveMQ；

b. 异步处理：比如在商场系统中，用户提交订单操做后，需要进行发送短信和记录积分的操做，不管是串行或者并行的方式都会花费较多时间，系统的吞吐量会有瓶颈，可以通过引入消息队列，将不必须的业务逻辑异步处理，此时用户的响应时间只有提交订单这部分时间，异步完成发送短信等操做；

c. 应用解耦：用户下单后，订单系统会调用库存系统，如果库存系统不能正常运行，则导致订单失败。加入中间件后，订单系统将消息写入消息队列，则返回用户下单成功。而库存系统则是去消息队列中获取下单信息，进行操做。订单系统写入消息队列，就不再关心后续操做了，实现了应用解耦；

d. 流量削峰：用在秒杀活动中，流量暴增导致应用挂掉。这是加入消息队列，可以控制活跃人数，可以缓解短时间内高流量压垮应用。用户请求，首先写入到消息队列，加入消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面；秒杀业务根据消息队列中的请求信息，再做后续处理。

2. spring中bean的作用域

a. 作用域scope：

i. singleton：单例的，默认；

ii. prototype：原型的，每次调用bean都会返回一个新的实例；

iii. request：在一次HTTP请求中有效，每个HTTP请求都会创建新的bean；

iv. session：在一次用户会话中有效，Session中共享一个Bean，不同的session使用不同的bean；

v. global session：在全局会话内有效，所有会话共享一个bean。

b. 单例作用：减少创建对象的消耗，减少jvm垃圾回收，可以快速获取到bean。

c. 线程安全：

i. 原型bean每次创建新对象，线程之间不存在bean共享，是线程安全的；

ii. 单例bean中，所有线程共享一个单例实例bean，因此是存在资源的竞争。如果单例bean是无状态的bean，也就是线程中的操做不会对bean的成员执行查询意外的操做，那么这个单例bean是线程安全的。

3. IOC

a. 将对象的控制器交给容器，由容器来完成对象的装配和管理。也就是说对象的创建过程，由主动的new变成了交给spring容器去完成，优点是降低了代码的耦合度。

4. 依赖注入DI

a. 依赖注入的三种：

i. 注解@Autowired自动装配；

ii. setter方法注入；类需要实例化一个Dao对象，可以定义一个private的成员变量，然后用创建set方法；

iii. 构造器注入；通过带参数的构造器完成注入。

二、Java基础

1. 程序计数器

a. 程序计数器用于存放指令的地址，在程序执行时，程序计数器的初值为程序第一条指令的地址，在顺序执行程序时，控制器首先按程序计数器所指出的指令地址从内存中取出一条指令，然后执行该指令，同时将程序计数器的值加1指向下一条要执行的指令；

b. 程序计数器是线程私有的，每个线程都有自己的程序计数器；

c. 如果线程正在执行的是Java方法，则程序计数器的记录是正在执行的虚拟机字节码指令地址；如果正在执行的是Native本地方法，则计数器的值为空；程序计数器是为Java字节码文件服务的，本地方法大多数会直接映射到其原生的平台上，和JVM无关。

d. 且程序计数器这一块区域，是在Java虚拟机规范中唯一没有规定OutOfMemoryError情况的区域。

e. ~~程序计数器，PC寄存器，是一块较小的内存空间，它可以看作是当前线程所执行的字节码指令的行号指示器，通过改变这个计数器的值来读取下一条执行的字节码指令，比如一些循环、跳转的指令。~~

2. 双亲委派机制

a. JVM定义了三种类型的类加载器：BootstrapClassLoader启动类加载器（Java核心库），ExtensionClassLoader扩展类加载器（Java扩展类），ApplicationClassLoader系统类加载器（用户定义在ClassPath中指定路径的类）；

b. 双亲委派机制可以描述为，某个特定的类加载器在接到加载类的请求时。首先检查加载器是否已加载这个类，如果有，已经加载过就直接返回不再加载；如果没有，将加载任务委派给父类加载器。

依次递归，直到BootStrapClassLoader。BootStrapClassLoader会在Java核心库中查找是否有该类，有则加载并返回，没有则抛出ClassNotFound异常给ExtClassLoader，依次下沉，直到底层没有任何加载器加载，抛出ClassNotFoundException。

c. 作用：

i. 安全，可以避免用户自己编写的类动态替换Java的核心类；

ii. 避免类重复加载，防止出现多份同样的字节码。

iii. ~~防止内存中出现多分同样的字节码。比如我们自己重写一个System类，经过双亲委派机制，它只会加载BootStrapClassLoader中的System类，就算我们重写，也总是加载父类的System类，自己写的System类根本没有机会得到加载。~~

3. 集合类

a. Collection

i. List

1. ArrayList
2. LinkedList
3. Vector

a. Stark

ii. Set

1. hashSet
2. treeSet

iii. Queue

1. priorityQueue

b. Map

i. hashMap

ii. hashTable

4. 线程安全的集合类：Vector，Stack，HashTable，枚举，ConcurrentHashMap。

====2020/09/07====

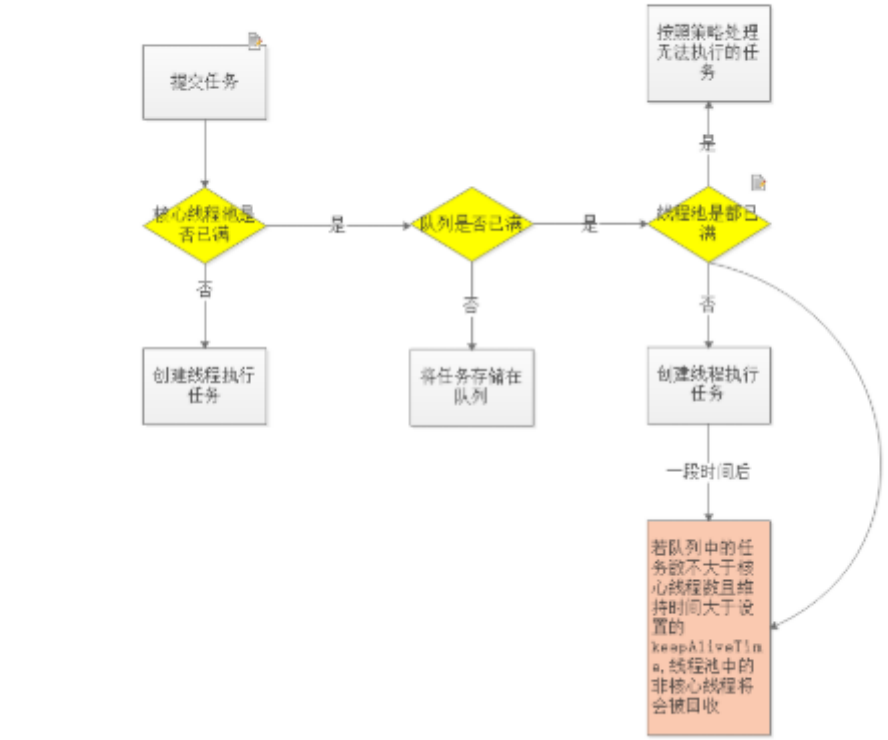
一、java基础

1. 线程池

a. 线程池执行过程

i. 图解

如图所示：



ii. 参数

1. 核心线程数corePoolSize

2. 最大线程数maximumPoolSize

3. 存活时间keepAliveTime

a. 线程池拒绝策略

1. jdk1.8的GC：

a. jdk1.8把方法区移除了，变成了元空间；

b. 把堆分成新生代、老年代。。。

c. 垃圾回收器：

i. CMS：使用的是标记清除算法，包括初始标记、并发标记、重新标记、并发清除4个步骤；

ii. G1：使用的是标记整理算法，包括初始标记、并发标记、最终标记、筛选回收4个步骤。优点，使用多cpu缩短停顿时间，空间整理，把区域分成多个region进行回收，停顿可预知，用户指定停顿时间，分代回收，最大回收价值，把每个region的价值按优先级排列，回收最大内存。

iii. jdk1.8 默认使用ParallelGC，采用多线程来扫描并压缩堆，目的是提高吞吐量。年轻代使用使用复制算法，老年代使用标记整理算法。

2. String

a. String类是不可变的类，任何对String的改变都会引发新的String对象的生成；

b. StringBuffer和StringBuilder是可变类，在原对象上进行操做，StringBuffer是线程安全的，StringBuilder是线程不安全的；

3. 四种引用

a. 强引用：无论如何都不会被GC回收，即使抛出OutOfMemory；

b. 软引用：新建对象时，内存溢出，JVM会进行一次GC，非强引用回收，如果仍然溢出，则抛出异常；

c. 弱引用：和软引用相似，但是不管是否内存溢出，都会被回收；

d. 虚引用：引用强度最弱的引用，并不影响对象的生命周期，如果虚引用和一个对象关联，那就跟没有引用与只关联一样，在任何时

候都可能被GC回收。虚引用一般配合引用队列使用，程序可以通过判断虚引用是否加入了引用队列，来判断关联的对象是否回收。

4. Throwable接口，异常分类
    a. Throwable
        i. Error
            1. VitualMachineError，虚拟机错误
            2. ThreadDeath ， 线程死锁
        ii. Exception
            1. 运行时异常
                a. NullPointException，控制在异常
                b. ClassCastException，类型转换异常
                c. ArithmeticException，算术异常
                d. ArrayIndexOfBoundsException，数组下标越界异常
            2. 非运行时异常
                a. IOException，IO异常

b. SQLException，SQL异常

c. 自定义Exception

5. IO流

a. 字符流：Reader、Writer；

b. 字节流：InputStream，OutputStream；

c. 输出文件FileoutputStream，输出json BufferedWriter。

6. 锁：

a. 共享锁、排它锁

b. 乐观锁、悲观锁：悲观锁考虑最坏的情况，认为每次去拿数据都会对数据进行修改，所以要加锁；乐观锁则不会加锁，在写入数据的时候，会判断数据是否被其他进程修改，如果修改了，先更新数据再写入。

7. 序列化

a. 序列化是指将对象转换成字节序列的过程，反序列化是指将字节序列恢复成对象；

b. 把对象的字节序列保存在硬盘上，或者在网络上传输。

二、Redis

1. 缓存穿透和雪崩

a. 缓存穿透：

i. 原因：用户大量并发请求的key对应的数据在redis和数据库中都不存在，导致尽管数据不存在但是还是每次都会进行数据库查询；

ii. 解决：从数据库中查询出来数据为空，进行空数据缓存。

b. 缓存雪崩：

i. 原因：大量的key同一时间点失效，同时又有大量的请求打进来，导致流量直接打在数据库上，造成数据库不可用。例如某个时间点内，系统预加载的缓存周期性集体失效。

ii. 解决：

1. 设置不同的国企时间，错开缓存过期，从而避免缓存集体失效；
2. 保证缓存服务的高可用性，部署在不同的节点，不同的机器上，还有不同的redis cluster上；
3. 采用多级缓存。
4. ~~设置热点数据，key永不失效；~~
5. ~~key的缓存失效尽可能错开；~~
6. ~~使用多级缓存。~~

三、框架

1. jpa：Java Persistence API，对象持久化。

====2020/09/05====

一、框架

1. springboot常用注解：

a. SpringBootApplication：启动类，项目启动的入口；(@SpringBootApplication实现原理)

i. SpringbootApplication是springboot中最核心的注解，写在启动类上面。它是

@SpringBootConfiguration、@EnableAutoConfiguration、@ComponentScan的组合注解；

1. @SpringBootConfiguration里面就是一个@Configuration，@configuration声明了这是一个配置类。springboot会通过@CompenentScan扫描带有@Configuration注解的配置了，并读取配置类的信息。~~@Configuration和@bean注入定义一个实体类；~~
2. @EnableAutoConfiguration启用上下文自动配置；
3. @ComponentScan扫描指定的包，默认扫描被注释的类所在的包。

b. @Controller：定义控制器类，~~由控制器复制将用户发来的URL请求转发到对应的服务器接口；~~

c. @RequestMapping：提供路由信息，负责URL到Controller中具体方法的映射；

d. GetMapping，PostMapping = @RequestMapping(method = RequestMothod.GET)

e. @ResponseBody：在控制器方法上加上ResponseBody注解，该方法的返回结果将直接写入HTTP ResponseBody中。一般配合@RequestMapping使用，使用@RequestMapping后，返回值通常解析为跳转路径，加上@ResponseBody后返回结果不会被解析为跳转路径，而实直接写入HTTP RequestBody；

f. @RestController：是@Controller和@ResponseBody的合集。在类上使用，表示这个类是控制器，并且方法返回值直接填入HTTP ResponseBody中，是RESTful风格的，控制器返回json数据；

g. @RequestParam和@PathVariable：注入参数;

h. @PathVariable：获取URL上的参数；

i. @Service：修饰service层的组件；

j. @Bean：用Bean注解标注方法，等价于spring的xml中配置的bean标签，产生一个bean交给springboot管理；

2. @PathVariable和@RequestParam

a. RequestParam注解是获取URL传入的参数，PathVariable注解是获取请求路径中的变量作为参数；

b. ~~RequestParam是从请求中获取变量，PathVariable是从路径中获取变量。~~

3. redis如何保证高可用，哨兵模式原理

a. 主从模型，哨兵模式，Cluster集群模式；99%的时间都能对外提供服务；

b. 主从模型，一个master主数据库和多个slave从数据库；master进行写操作，slave进行读操做，当master数据发生变化时，会自动将数据同步给slave。主备切换，slave切换成master。作用一是读写分离，分担master压力，二是容灾备份；

c. 哨兵模式：集群监控、消息通知、故障转移、配置中心。哨兵本身也是分布式的，通过选举来确定新的master。主从模式当master出现故障时，需要手动切换master。哨兵模式实现自动化监控和故障恢复。哨兵是一个独立的进程，当发现master宕机时，会将一个slave切换成master；

d. cluster集群模式，哨兵模式实现了读写分离加高可用，但是一个master会有性能瓶颈，如果需要支持更大的数据量，那么就需要横向扩容master节点，采用多个master节点；cluster集群模式是从redis3.0开始使用的，用的是服务器sharding技术，实现分布式存储。

4. 分布式锁实现，Redlock，超时时间应该远小于锁的失效时间。
5. 前后端分离
    a. 优点：
        i. 多端适应，PC、手机、Pad；
        ii.

二、redis
1. redis有哪些数据类型及应用场景：
    a. String、Hash、List、Set、Sorted set。
2. redis取出同一个key有多个value，Apache commons工具包，提供了MultiValueMap类，是在hashMap的value设置成ArrayList。

三、java基础
1. jvm调优
    a. 年轻代不能太小，会导致mirrorGC频繁；老年代也不能太小，会导致FullGC频繁；
2. jdk8垃圾回收机制
3. 索引一定是唯一的吗

====2020/09/04====

一、框架
1. springboot依赖包的x，y坐标：
    a. groupid和artifactid统称为坐标，是为了保证项目的唯一性而提出的；
    b. groupId一般分为多端，一般是域名+公司名称，artifactId是项目名称。
2. Mybatis的自增主键：
    a. Mybatis的insert和update标签中；

b. useGenerateKeys：设置自增主键，默认是false；

c. keyProperty：设置自增主键返回的字段；

d. keyColumn：数据库自增主键的列名，默认是数据库表的第一列。当主键列不是表中第一列时需要设置。

3. Mybatis设置表的关联：

a. 使用association和collection标签；

b. Mybatis中使用association标签来解决一对一的关联查，resultMap中的association标签的属性包括：

i. property：对象属性的名称；

ii. javaType：对象属性的类型；

iii. column：所对应的外键字段名称；

iv. select：使用另一个查询封装的结果。

c. 如果是对多的关联查询，使用collection标签，ofType属性指定集合中元素的对象类型。

d. ~~Mybatis多表关联查询的重点是，在resultMap中的association中传入resultMap的类型。~~

4. SQL执行计划：

a. explain查看sql执行计划。首先查看type列，如果出现all关键字，就是发生了全部扫描；再查看key列，查看是否有索引，null是没有使用索引；然后查看rows列，表示扫描的行数，行数多则耗时长。

5. 建立索引的规则：

a. 表的主键外键；

b. 数据量大的表和经常查询的字段；

c. 经常与其他表进行连接的表，连接字段建立索引；

d. 使用窄索引，索引应该建立在小字段上，磁盘中数据页的大小是固定为8k，字段越长，数据块就越长，数据页存放的数据项的数量就越少，索引层次就会变多，B+树的高度会增加。

e. 频繁修改数据的表，避免建索引；

6. mvn项目

a. 打包：mvn clean package；

b. package install deploy:

i. 打包区别：

1. mvn package：打包到本项目，在项目target目录下；

2. mvn install：打包到本地仓库；

3. mvn deploy：打包上传到远程仓库。

ii. 打包过程：

1. 依次执行：clean、resources、compile、testResources、testCompile、test、jar、install、deploy；

iii. 主要区别：

1. package命令：完成项目编译、单元测试、打包功能，打包的文件在target目录中；install把打包文件部署到本地maven仓库；deploy把打包文件部署到本地仓库和远程仓库。

c. mvn jar包版本冲突：

i. 排除加载，用exclusions标签；在用到这个jar包的地方加上说明，告诉它不用加载。

二、Java基础

1. mybatis一二级缓存

a. sqlsession：通过sqlsession实现对数据库的增删改查，mapper的里面也是通过sqlsession来指向SQL操作。

b. 一级缓存是sqlsession级别的缓存。在操做数据库时构建sqlsession对象，对象中有一块内存区域是用来缓存数据的，缓存的数据结构是hashmap。不同的sqlsession之间的缓存区域是互不影响的。一级缓存的作用域是同一个sqlsession，在一个sqlsession中执行查询语句后，结果会放入缓存，如果下次有相同的查询语句，则直接放回缓存中的结果集，而不用查询数据库。如果缓存对于的数据有了增删改操做，则缓存会自动清除。也可以在sql标签中设置flushCache="false"，查询结果不缓存。

c. 二级缓存是mapper级别的缓存。多个sqlsession去操做同一个mapper的sql语句，查询的结果集会存放在二级缓存区域，多个sqlsession可以共享二级缓存，二级缓存是跨sqlsession的，是多个sqlsession共享的，它的作用域是mapper的同一个namespace命名空间。二级缓存要手动开启，在配置文件中cacheEnable设置为true。

2. ArrayList、LinkedList、Vector

a. ArrayList是就动态数组，LinkedList是基于双向链表，vector和arrayList基本相同，只是vector是同步的，索引vecor开销大；

b. 对于随机访问的get/set，ArrayList效率高，因为LinkedList要移动索引指针；

c. 对于添加删除add/remove，LinkedList效率高，因为ArrayList需要移动元素。

3. concurrentHashMap：

a. concurrenHashMap是在hashMap的基础上增加了同步操做；

b. jdk1.7：分段锁，在整个bucket桶数组中分割成多个分段（segment），没一把锁值锁一个segment，也就是锁一部分数据，多线程访问容器中不同分段的数据时，不存在锁竞争，提高并发访问率；

c. jdk1.8：使用synchronized关键字+cas来实现，降低了锁的粒度，1.7锁的粒度是segment一个分段，1.8中锁的粒度就是一个

entry。

d. 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。hashtable是用synchronized关键字实现同步。

e. cas：compareAndSwap比较并替换，从内存中取值V，和预期值A想比较，如果内存值V和预期值A相等，则把新值B更新到内存，如果相等，重复上述操做直到成功。

i. ABA问题，CAS过程中知识进行简单的值校验，值虽然相同，但是不是原理的数据了。添加版本号对比，一个数据一个版本号。

====2020/09/03====

一、java基础

1. 策略模式
2. 设计一个数据库连接池
3. tomcat和jetty区别

二、java基础 -- 值传递和引用传递

1. 对象和对象引用

a. 在java中一切都被视为对象，对象的引用就是一个指向该对象的标识符；

b. String是final的，一旦被创建就不能修改，当对string进行修改时，引用会指向一个新的对象。

2. 值传递和引用传递（值引用和对象引用）

a. 按值传递不会改变实际参数的数值；按引用传递可以改变实际参数的内容，但不能改变实际参数的参数地址。

b. 引用传递（对象引用）是两个引用指向的对象地址相同，值传递（值引用）是指向不同的对象。

c. 基本数据类型和String是值传递（值引用），其他是引用传递（对象引用）；

d. 两个引用指向同一个对象，此时a==b，当b引用的值改变的时候，b指向的地址就发生了改变，此时a==b为false。

3. String实现值引用的方式，String对象在内存中的位置

a. String是一个final类，那么对于同样值的String实例，可以不用重复创建，JVM有一个StringPool的概念，里面存放这堆里面的String对象的引用；

b. JVM会在Stringpool里面通过equals()方法查找是否存在现成的String对象引用；如果没有，则在堆里创建一个String对象并将该对象的引用保存在StringPool中；如果有，那就直接返回该对象的引用；

c. new String("abc")的时候，JVM会像生成普通对象一样生成这个String，在堆里保存，直接返回引用，并不会与StringPool交互，可以通过string.intern()方法让StringPool管理这个对象；intern()方法的工作原理是，在堆中创建一个完全一样的String对象，并将该对象的引用放入StringPool中，最后返回给调用方。

4. ~~对象引用和值引用，对象和对象的引用，值传递和引用传递：~~

a. ~~八种基本数据类型是值引用，其他是对象引用~~

二、数据库

1. log

a. binlog和redolog：

i. redo log重做日志，用于记录事务操做的变化，记录的是数据修改之后的值，不管事务是否提交都会记录下来，所以不用每次去操做磁盘，可以先暂时记录在redo log中，等到机器空闲的时候再批量更新到磁盘中。如果发生介质失败（media failue），比如断电数据库down掉的时候，可以通过redo log来恢复到断电前的时刻，保证数据的完整性；

ii. binlog记录了对mysql数据库执行修改的所有操做，增删该不包括查。binlog的存在是因为redolog是innodb引擎特有的，binlog是mysql的server层实现，所有引擎可以用，serverceng也有一套日志记录，就是binlog。

iii. redolog在事务开始的时候，会记录每次信息变更；binlog在事务提交的时候，才会记录数据变更。

iv. redolog是循环写的，空间固定会用完；binlog是追加写入，不会覆盖原来的空间。

v. 流程：

1. sever层 + innodb引擎层；
2. sever层执行语句，innodb引擎层查看数据是否在内存中，不在则磁盘查找，返回数据集，sever层进行修改

操做，调用存储引擎接口写入新数据，innodb引擎层对内存数据进行更新，记录redo log重做日志并标记状态prepare，sever层生成binlog，调用引擎接口提交事务，redo log状态改为commit，结束。

====2020/09/02====

一、查询sql优化

1. sql优化
2. 避免索引失效场景
3. limit offset过大
4. 数据库索引为什么快
5. 主键的作用/好处：作为唯一的标识，且占用的存储空间少，4字节；UUID是数据库独立的，数据可以在不同的数据库中迁移而不受影响。
6. 外键：关联另一张表，保持数据的一致性；
7. 索引：优点：提高查找的速度；缺点：创建索引和维护索引需要消耗时间，并且索引需要占用一定的物理空间。

====2020/09/01====

一、数据库事务

1. 什么是事务：具有ACID的一组操作，原子性、一致性、隔离性、持久性，要么全部执行成功，要么撤销不执行。
2. Innodb实现事务：

a. 通过redo log 重做日志、undo log 回滚日志、还有锁或者MVCC实现；redo log 重做日志实现事务的原子性和持久性，undo log 回滚日子保证事务的一致性，锁或者MVCC实现隔离性。

b. 阐述：

i. 在执行事务的每条sql中，会先将数据原来的值写入undo log，然后执行sql对数据进行修改，最后将修改后的值写入redo log ；

ii. redo log 重做日志包括两个部分：1是内存中的重做日志缓存，2是重做日志文件。在事务提交时，必须先将该事务的所有日志写入到日志文件进行持久化，待事务commit操作完成后才算完成；

iii. 当一个事务中的所有sql执行成功后，会将redo log 缓存中的数据刷入到磁盘，然后提交；如果发送回滚，会根据undo log 恢复数据。

c.

特征	INNODB实现方式
原子性（A）	回滚日志（undo log）：用于记录数据修改前的状态；
一致性（C）	重做日志（redo log）：用于记录市局修改后的状态；
隔离性（I）	锁：用于资源隔离，分为共享锁和排它锁；
持久性（D）	重做日志（redo log） + 回滚日志（undo log）；

d. MVCC

i. MVCC多版本并发控制，查询需要对资源加共享锁（s），修改需要对数据加排他锁（X）；

ii. 利用undo log回滚日志使读写不阻塞，实现了可重复读。当一个事务正在对一条数据进行修改的时候，该资源会被加上排它锁。在事务未提交时，对加锁资源进行的读操作，读操做无法读到被锁资源，会通过一些特殊的标识符去读undo log 中的数据，这样读到的是事务执行之前的数据。

1. 回滚实现：在执行语句之前，会将原数据写入undo log 回滚日志，然后对字段进行更新，如果发送回滚，则利用undo log 回滚，撤销已完成的更新操做。

2. 当前读和快照读

a. 当前读是用在修改数据的时候，也就是更新插入删除的时候，读取的是最新的版本，并且对读取的数据加锁，阻塞其他事务，其他事务不能修改这条记录。

b. 快照读就是select操做，读取的是快照版本，innodb中是在开启事务后的第一个select的时候执行快照读，快照读的实现方式是uodolog和MVCC；

3. mysql中的各种日志

a. binlog：

i. 定义：binlog记录数据库表结构和表数据的变更，增删改会被记录，select查询不会记录；

ii. 作用：主从复制和增量恢复。

b. redo log：

i. 定义：当用户把数据加载到内存是，对数据进行修改之后，会同步一份到redo log文件中；记录这某页上的修改操做。

ii. 作用：数据恢复的时候，会把redolog的日志，同步到数据库中。

c. binlog和redolog：

i. redo log重做日志，用于记录事务操做的变化，记录的是数据修改之后的值，不管事务是否提交都会记录下来，所以不用每次去操做磁盘，可以先暂时记录在redo log中，等到机器空闲的时候再批量更新到磁盘中。如果发生介质失败（media failue），比如断电数据库down掉的时候，可以通过redo log来恢复到断电前的时刻，保证数据的完整性；

ii. binlog记录了对mysql数据库执行修改的所有操做，增删该不包括查。binlog的存在是因为redolog是innodb引擎特有的，binlog是mysql的server层实现，所有引擎可以用，serverceng也有一套日志记录，就是binlog。

iii. redolog在事务开始的时候，会记录每次信息变更；binlog在事务提交的时候，才会记录数据变更。

iv. redolog是循环写的，空间固定会用完；binlog是追加写入，不会覆盖原来的空间。

v. 流程：

1. sever层 + innodb引擎层；
2. sever层执行语句，innodb引擎层查看数据是否在内存中，不在则磁盘查找，返回数据集，sever层进行修改操做，调用存储引擎接口写入新数据，innodb引擎层对内存数据进行更新，记录redo log重做日志并标记状态prepare，sever层生成binlog，调用引擎接口提交事务，redo log状态改为commit，结束。

d. undolog：

i. 定义：记录的是数据被修改前的值，用于回滚。

4. 锁：

a. 为什么要锁：为了保证数据的一致性。

5. 索引（idnex）

a. 为什么快：建立索引，事先排好序，在查找的时候用二分查找来提高效率；

b. 不能建太多：如果所有很大，那么检索的时候开销会很大，索引本身的效率就会降低；

c. 索引缺点：

i. 在数据库中更新数据，也需要更新索引，降低了写入性能；

ii. 索引占用磁盘空间；

iii. 外键必须建立索引，提高多表查询的效率。

d. 聚集索引：表中数据的物理顺序与列的逻辑顺序相同，一般是和主键列相同。可以理解为，聚集索引就是数据节点，非聚集索引的叶子节点是索引指针，指向对应的数据块；

e. 索引失效：

i. 索引列上不能做计算操做；

ii. 最左匹配原则，范围查询右边的列索引会失效，联合索引是一颗b+树，它的键值数量不是一个，而是多个，构建一颗b+树只能根据一个值来构建，一次数据库依据联合索引最左边的字段来构建b+树；

iii. mysql在使用!=,<,>,between, is null, is not null,like 通配符开头；

iv. 尽量覆盖索引；

f. InnoDB只有通过索引条件检索数据，才会使用行锁，否则会使用表锁。也就是说InnoDB的行锁是基于索引的。

6. sql优化

a. 避免全表扫描：字段没有索引，则会全表扫描；

b. 避免索引失效；

c. 避免排序，避免查询不必要的字段。

7. 生成100万不重复的8位随机编号

a. 8位就是 1千万 到 1亿-1 ，假设在1千~9千9百..万 之间生成100万个随机数，使用9999..-1千万，等于89999..，除以100万，舍去小数位，得到89。然后从1000万开始，每次加89，这样就能生成100万个数；

b. 建立一个表，由两个字段组成。一个字段是从1000万开始，以89为步长的100万个数；另一个字段使用newid随机生成数。

c. 这样表中就有100万条记录，不过是顺序的，，由于newid生成的字段是随机的，只要按照这个字段排序，第一个字段上的值也就随机了。

d. 每次分批的生成记录，然后一起插入表中，一次产生1000条记录，只要循环1000次就能完成。

8. newid = 网卡号 + CPU时钟。

====2020/08/31====

一、Java基础

1. 物理内存：

a. 程序寻址用的都是物理地址，程序能寻址的范围是有限的，在32位系统下，寻址范围是2^32也就是4G。并且这是一个固定的值，如果没有虚拟内存，每次开启一个进程都给4g物理内存。因为物理内存是有限的 ，很快就会分配完；而且直接访问物理内存，这个过程可能修改其他进程的数据。

2. 虚拟内存：

a. 虚拟内存是计算机系统的内存管理技术，它使得应用程序认为自身拥有连续的可用内存，也就是说，虚拟内存能够提供一大块连续的地址空间，对于程序来说它是连续的、完整的，实际上虚拟内存是映射在多个物理内存碎片上，还有部分映射到了外部磁盘存储器上。

b. 进程运行时会得到4g的虚拟内存，而实际上用了多少内存，就会对于多少物理内存。

c. 进程访问一个地址，需要查看页表，页表有2个部分。第一部分是一个valid值，记录此页是否在物理内存上，0是不在，1是在；第二部分记录对应物理内存的地址。

d. 如果查看也表达的时候，发现对应的数据不在物理内存上，就是缺页。缺页的时候，操作系统会阻塞该进程，并将磁盘中对应的页换入内存，然后使该进程就绪。

二、数据库：

8. sql优化
9. 索引失效场景
10. limit offset过大
11. 数据库索引为什么快
12. 主键的作用/好处：作为唯一的标识，且占用的存储空间少，4字节；UUID是数据库独立的，数据可以在不同的数据库中迁移而不受影响。
13. 外键：关联另一张表，保持数据的一致性；
14. 索引：优点：提高查找的速度；缺点：创建索引和维护索引需要消耗时间，并且索引需要占用一定的物理空间。

====2020/08/30====

一、JVM类加载机制：

1. Java跨平台：

a. Java语言可以在不同的操作系统平台运行，因为Java语言的运行环境是在JVM，Java虚拟机中。只要操作系统安装了Java虚拟机，就能够加载.class二进制文件。

2. 类加载过程：

a. 类的生命周期：加载，验证，准备，解析，初始化，使用，卸载7个阶段。

3. 符号引用和直接引用的区别：

a. 解析阶段是JAVA虚拟机将常量池内的符号引用替换为直接引用的过程；

b. 符号引用（Symbolic Reference）：符号引用以一组符号来描述所引用的目标，符号引用可以是任何形式的字面量，只要使用时能无歧义的定位到目标即可，符号引用和虚拟机的布局无关。个人理解是：在编译的时候每个java类都会被编译成class文件，但在编译的时候虚拟机并不知道所引用类的地址，索引用符号引用来代替，而在解析阶段，就要把符号引用转换为真正的地址。

c. 字面量（literal）：是用于表达源代码中一个固定值的表示法，是数字或者字符串，它可以作为右值出现，也就是等号右边的值。变量可以用来保存字面量。

二、spring和springboot框架相关：

1. springboot和spring的区别：

a. springboot是spring框架的扩展，它消除了设置spring应用程序所需的XML配置，创建独立的spring应用；

b. 提供嵌入式的tomcat容器，有tomcat、jetty、Undertow；

c. 提供starter简化构建配置；

2. 对象的访问定位方式，句柄和直接指针：

a. 如果使用句柄的话，要在java堆中开辟一个句柄池，用来存放句柄地址，句柄地址包含堆中对象实例数据的地址信息，和方法区中对象的类型数据的地址信息。使用句柄的好处就是引用地址对应的句柄中存储的是稳定的句柄地址，当数据被移动的时候，只需要修改句柄中实例数据指针，而不需要修改引用地址。

b. 使用直接指针的优势就是速度快，节省了一次访问指针定位的时间开销，引用直接指向存放实例数据的堆内存。

====2020/08/29====

一、cookie和session

1. 区别：

a. cookie和session都是会话技术，cookie是保存在客户端的，session是保存在服务端的；

b. cookie有大小和数量限制，大小一般是4k，数量根据不同的浏览器，每个域名50个左右，chrome是53个；session则没有限制，和服务端的内存大小有关；

c. cookie有安全隐患，攻击者通过拦截或者在本地文件中找到cookie就可以进行攻击；

d. session在服务端上存在一段时间后会消失，session过多会增加服务器的压力，cookie在关闭浏览器后会消失。

二、java基础

1. oom：out of memory，jvm没有足够的内存来为对象分配空间，并且垃圾回收器业已经没有空间可以回收，就会抛出这个错误。

2. 内存泄漏：申请的内存在被使用完后没有释放，导致虚拟机不能再次使用该内存，就是内存泄漏；

3. 内存溢出：申请的内存超出了jvm能够提供的内存大小，溢出。

4. 索引字段长度的影响：会有影响，当索引在这一列的时候，由于数据页和索引页是固定8k大小的，字段越长，一个页面存放的数据就越少，索引的层次就越多，查找的范围也越大。一般使用“窄索引”，~~比如用smallint而不用int。~~

5. hashmap：

a. hashmap的底层数据结构是数组和链表，它是用于存储key-value键值对的集合，是一个entry数组，每个键值对是一个Entry，存储在bucket桶，每个元素的初始值是null。

b. hashmap是基于hash的，当我们通过put(key,value)存储对象到hashmap时，先调用hashCode()方法，返回hash值，找到bucket对应的位置来存储Entry对象。如果这个位置没有元素，则直接放入；如果这个位置已经有元素，则以链表的形式保存，在JDK1.8中，如果链表的长度达到8，则转为红黑树。

c. 在get()方法获取对象的时候，通过键对象的equals()方法找到正确的键值对，然后返回值对象。

6. 红黑树：

a. 节点必须是红色或者黑色；

b. 根节点是黑色；

c. 每个叶子节点是黑色（nil节点或空姐点）；

d. 每个红色节点的两个子节点都是黑色。

7. HashMap扩容

a. HashMap的默认初始长度（initialCapacity）是16，负载因子（loadFactor）是0.75，也就是说第一次扩容的阈值时12，因此在第一次存储第13个简直对时，会触发扩容机制，数组长度变为原来的2倍。

b. 扩容后的长度必须时2的幂次。在put()方法中，获取数组索引的计算方式为key的hash值和数组长度减一进行按位与，长度为2的幂次，那么2的幂次减一的值的二进制位数一定全为1，这样数组下标index的值完全取决于可以的hash值得后几位，使得元素在数组中分布均匀，发生hash碰撞的概率小。

8. SpringBean的作用域

a. 在Spring中可以在<bean>元素的scope属性里设置bean的作用域，包括singleton、prototype、request、session四种。

b. singleton：是默认的，在创建ioc容器的时候会创建bean实例，是单例的，每次得到的是同一个实例；

c. prototype：IOC容器创建的时候不实例化bean，在调用getBean()方法时返回一个新的实例；

d. request：每次http请求会创建一个新的bean；

e. session：同一次会话中共享一个bean，不同的session会话使用不同的bean。

9. ThreadLocal：

a. 线程本地变量，当使用ThreadLocal维护变量时，每个Thread拥有一份自己的副本变量，多个线程互补干扰，从而实现线程间的数据隔离。

b. ThreadLocal维护的变量在线程生命周期内起作用，可以减少同一个线程内多个函数或者组件之间公共变量的传递复杂度；

10. http协议：

a. 超文本传输协议，采用请求/响应模型，是用于服务器传输超文本到本地浏览器的协议，是基于TCP/IP协议来传递数据的，常用的请求方法有GET/POST，是无连接的（每次连接只处理一个请求，服务器处理完客户端的请求，收到应答后，即断开连接，~~可以节省传输时间~~），是无状态的，默认端口是80。

b. HTTP请求 / 响应的步骤：

i. 客户端连接到web服务器；

ii. 发送http请求；

iii. 服务器接收请求并返回http响应；

iv. 释放TCP连接；

v. 浏览器解析HTML内容。

c.

HTTP 请求/响应的步骤如下：

1、客户端连接到Web服务器

```
1 一个HTTP客户端，通常是浏览器，与Web服务器的HTTP端口（默认为80）建立一个TCP套接字连接。例如，http://www.baidu.com。
```

2、发送HTTP请求

```
1 通过TCP套接字，客户端向Web服务器发送一个文本的请求报文，一个请求报文由请求行、请求头部、空行和请求数据4部分组成。
```

3、服务器接受请求并返回HTTP响应

```
1 Web服务器解析请求，定位请求资源。服务器将资源复本写到TCP套接字，由客户端读取。一个响应由状态行、响应头部、空行和
2 响应数据4部分组成。
```

4、释放连接TCP连接

```
1 若connection 模式为close，则服务器主动关闭TCP连接，客户端被动关闭连接，释放TCP连接;若connection 模式为keepalive，则
2 该连接会保持一段时间，在该时间内可以继续接收请求；
```

5、客户端浏览器解析HTML内容

```
1 客户端浏览器首先解析状态行，查看表明请求是否成功的状态代码。然后解析每一个响应头，响应头告知以下为若干字节的HTML
2 文档和文档的字符集。客户端浏览器读取响应数据HTML，根据HTML的语法对其进行格式化，并在浏览器窗口中显示。
```

d.

4、举例：

```
在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1、浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址；

2、解析出 IP 地址后，根据该 IP 地址和默认端口80，和服务器建立TCP连接；

3、浏览器发出读取文件（URL中域名后面部分对应的文件）的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器；

4、服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器；

5、释放 TCP连接；

6、浏览器将该 html 文本并显示内容；
```

e. ~~HTTP协议定义Web客户端如何从Web服务器请求Web页面，以及服务器如何把Web页面传送给客户端。HTTP协议采用了请求/响应模型。客户端向服务器发送一个请求报文，请求报文包含请求的方法、URL、协议版本、请求头部和请求数据。服务器以一个状态行作为响应，响应的内容包括协议的版本、成功或者错误代码、服务器信息、响应头部和响应数据。~~

f. 参考资料：https://blog.csdn.net/N1314N/article/details/94154643?ops_request_misc=%257B%2522request%255Fid%2522%253A%252215986712661972521992527 1%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=159867126619725219925271&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-94154643.ecpm_v3_rank_business_v1&utm_term=%E8%AF%B4%E4%B8%80%E4%B8%8Bhttp%E5%8D%8F%E8%AE%AE&spm=1018.2118.3001.4187

1. localStorage和sessionStorage:

a. 都是window对象提供的全局属性，在浏览器中存储key/value键值对的数据。

b. ~~区别在于时效性，sessionStorage在关闭窗口之后会删除这些数据，而localStorage则没有时效性，除非手动删除它。~~

c. ~~如果在浏览器窗口关闭之后还要保留数据，使用localStorage；如果用于临时保存一个窗口的数据，使用sessionStorage。~~

d. 生命周期，licalstorage存储的数据，如果不手动删除将会一直存在，关闭浏览器再次打开，数据依然存在；sessionStorage存储的数据，关闭浏览器窗口就会删除。

e. 容量区别，localStorage容量是20M，sessionStorage是5M。

2. httpservlet中的对象：

a. HttpServletRequest；

b. HttpServletResponse；

c. ServletConfig；

d. ServletContext。

3. java8新特新：

a. Lambda表达式；

b. StreamAPI；

c. Optional类，减少空指针异常；

d. hashmap。

4. hashset，是基于hashmap实现的。

5. HTML的全局属性：id，class，style，hidden，dir，lang，title。

====2020/08/28-1====

一、mysql实现可重复读

1. 通过MVCC实现，Innodb的MVCC是通过在每行记录的后面保存两个隐藏的列来实现的，这两个列分别保存了这一行的创建时间和删除时间，这里存储的不是时间值，而是系统的版本号（可以理解为事务的ID），没开始一个新的事务，系统版本号都会自动递增，事务开始时刻的系统版本号，会作为事务的版本号。

2. 在进行select查询的时候，会对每行记录检查两个条件：

a. InnoDB只会查找版本小于等于当前事务版本的数据行，也就是说事务读取到的行在事务开始前已存在，或者是事务自身插入或修改的；

b. 数据行的删除版本号，要么未定义，要么大于当前事务的版本号，也就是说事务读取到的行在事务开始之前未删除；

二、java基础

1. java访问修饰符，private，默认，protect，public；类中，包中，子类，非子类；
2. 基本数据类型：byte，short，int，long，float，double，boolean，char；
3. spring事务处理：

a. 事务是指在一系列的操作中，一旦其中一个操作出现错误，则全部回滚到事务开始的状态；

b. spring中的声明式事务，通过@Transactional注解实现，是建立在AOP上的，其本质是对方法前后进行拦截，然后在目标方法开始之前创建一个事务，在执行完目标方法之后，根据执行情况提交或者回滚代码。

4. 多线程：多个线程并发执行；优点：充分利用CPU资源；缺点是可能存在线程安全问题；

====2020/08/28====

一、面向对象和面向过程

1. 面向过程：分析出解决问题所需的步骤，然后用函数把这些步骤一步步实现，使用的逐个依次调用；
2. 面向对象：把构成问题的事务分解成各个对象，每个对象都有自己的属性和方法。建立对象来描述某个事务解决问题的步骤和行为。

二、cms和g1

1. cms：

a. cms（concurrent mark sweep）是以获取最短回收停顿时间为目标的收集器，是基于“标记-清除”算法实现的，它的运行过程包括：初始标记，并发标记，重写标记，并发清除；

b. 初始标记、重新标记，需要stop-the-world。初始标记是标记GCRoots直接关联的对象，速度很快；并发标记是从GCRoots出发，进行可达性分析，搜索引用链，找到需要回收的对象；重新标记阶段是为了修正并发标记阶段用户线程产生的对象；

c.

图 3-10　Concurrent Mark Sweep 收集器运行示意图

d. 优点：并发收集、低停顿；

e. 缺点：

i. 无法处理浮动垃圾：最后的并发清理阶段，也会产生垃圾对象；

ii. 基于“标记-清理”，产生大量的内存空间碎片；

1. g1：

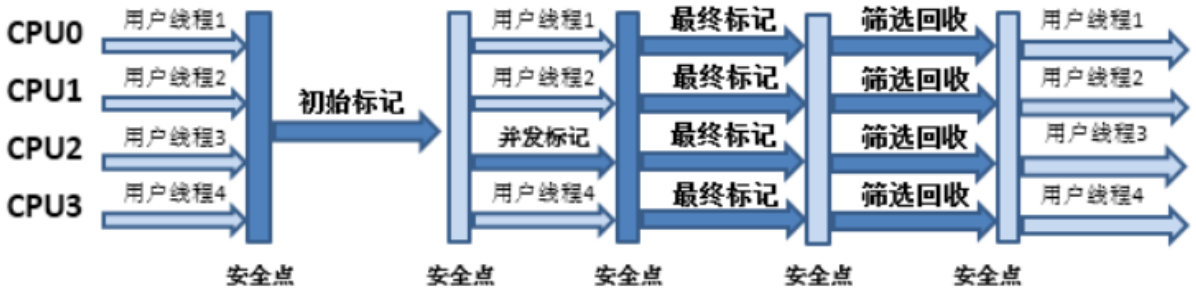
a. g1是面向服务端应用的垃圾收集器，是基于“标记-整理”算法实现的，它的运行过程包括：初始标记，并发标记，最终标记，筛选回收；

b. 特点：

i. 空间整理，内有内存碎片产生；由于G1使用的是独立区域（region）的概念，G1从整体来看是基于“标记-整理”算法实现的，从两个region上看是基于“复制”来实现的，G1在运行期间不会产生内存空间碎片；

ii. 可预测的停顿：可以对每个region中的回收价值进行分析，也就是回收内存和回收时间的比值，在最后的筛选回收阶段，对每个region的对象回收价值进行排序，用户可以自定义停顿时间，获得一个最大的回收价值。

c.



G1收集器运行示意图

3. stop-the-world： 在垃圾回收过程中涉及到对象的复制移动，进而导致需要对对象的引用进行更新，为了保证引用更新的正确性，将暂停所有的其他线程，导致全局停顿。

4. GCRoots：

a. 虚拟机栈中的引用对象；

b. 方法区中类静态属性引用的对象；

c. 方法区中常量引用的对象；

d. 本地方法栈中JNI（java本地接口）引用的对象。

e. 总结：方法区和栈中的引用对象。

====2020/08/27====

一、redis的zset实现有序

1. redis：是一个基于内存的高性能kv数据库；速度快，数据存在内存中；支持丰富的数据类型，String，hash，list，set，zset；支持事务；可以按照key设置过期时间。

2. redis和mencache的区别：

a. memcache把数据存储在内存中，断电后会挂掉；redis有部分数据保存在硬盘中，保证了数据的持久性

b. redis支持的数据类型丰富。redis的value最大可以存1GB的数据，mencache只能存1MB。

3. zset的内部数据结构：

a. zset给每个member成员维护一个分值score，根据分值从小到大进行排序；

b. 有序集合zset的数据结构可以使用压缩列表zipList或者跳跃表skipList来实现；

c. 压缩列表：

i. 使用压缩列表的情况：

1. 集合元素小于128个；
2. 集合每个元素都小于64字节。

ii. 使用压缩列表的有序集合，使用紧挨在一起的压缩列表节点来保存，第一个节点保存member，第二个节点保存score。由于数据元素较少，所以并不影响效率；而且可以充分利用压缩列表的连续内存和紧凑的数据结构来节省内存

iii. ~~也就是在集合元素较少、元素类型较小的时候，使用压缩表，这样由于数据元素较少，虽然压缩表效率低，但是基本不影响，而且可以充分利用压缩表的连续内存和紧凑的数据结构来节省内存空间。~~

d. 跳跃表：

i. 跳跃表的zset底层是一个zset结构体，包含一个字典和一个跳跃表。跳跃表按score从小到大保存所有集合元素。字典则保存从member到score的映射，这样可以用O(1)的时间复杂度来查找member对应的score。字典和跳跃表通过指针来共享相同元素的member和score，不会浪费额外内存；

ii. 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logn+1层索引指针，查找、插入、删除的时间复杂

度是logn。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。

4. zset操作命令

a. zadd(key, score, member)：向名称为key的zset中添加元素member，score用于排序。如果该元素已经存在，则根据score更新该元素的顺序。

二、类加载机制

1. 软件生命周期：提出需求，需求分析，软件设计，程序编码，软件测试，运行维护；

2. 类加载：虚拟机把描述类的数据从Class文件加载到内存，并对数据进行验证、解析、准备、初始化，最终形成可以被虚拟机直接使用的Java类型，这就是虚拟机的类加载机制；

3. 类加载时机：new、初始化、反射、启动时主类、invoke对应的句柄。

4. 类加载的7个阶段：加载、验证、准备、解析、初始化、使用、卸载。

====2020/08/26====

一、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，只要给定类的名称，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所具有的方法和属性；

iv. 在运行时调用任意一个对象的方法；

v. 生成动态代理。

1. 给一个对象提供一个代理对象，通过代理对象来控制该对象的引用；

2. 开闭原则，中介隔离作用；

3. JDK和CDLib区别：

a. JDK动态代理只能对实现了接口的类生成代理，使用反射完成；

b. CDLib则不需要接口，对指定的类生成子类，覆盖其中的方法，不能对final类进行继承，使用了动态生成字节码技术。

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")；

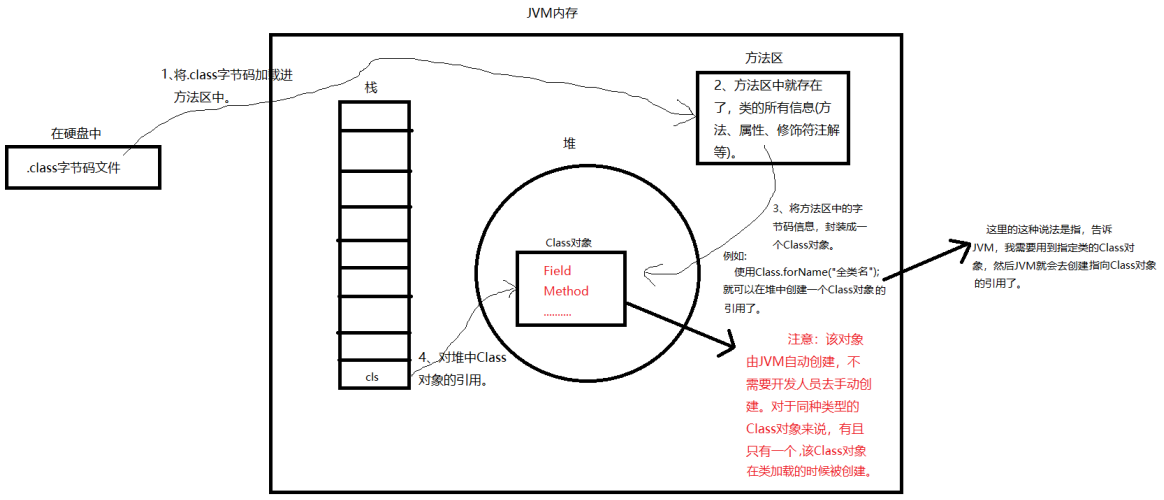
b. 调用某个类的class属性来获取该类对应的Class对象，这个属性保存的是Class对象的内存地址，Class c2 = Employee.class；

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.



c. 反射原理：

i. java文件经过编译会生成class字节码文件。~~开发人员保存的.java文件，进过编译生成.class字节码文件；~~.class字节码文件保存在硬盘中。

ii. 首先将class字节码文件加载到JVM内存中，class字节码文件会加载到方法区，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，保存在堆中，对同一类型的class对象，有且只有一个，JVM自动创建该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，也就是

说，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. 拿到这个引用，然后对这个class对象进行操作。

二、redis持久化

1. RDB（Redis DataBase）

a. 执行机制是基于快照，直接将数据库中的key-value以二进制形式存储在rdb文件中；

b. 优点：性能高，rdb文件存储的是key-value的二进制形式，恢复快；使用单独的子进程来进行持久化，不会占用主进程的IO；

c. 缺点：RDB是间隔一段时间进行持久化，如果两次save保存之间redis发生故障，这中间的发生数据丢失。所以这种方式适合数据要求不严谨的时候。

2. AOP（Append-only file）

a. 执行机制是将对数据进行的每一天修改命令追加到aof文件中；

b. 优点：数据不容易丢失；如果追加文件的时间是1s，那么只会丢失1s的数据。

c. 缺点：性能较低，每一条修改操作都会追加到aof文件，执行频率高，而且aof文件存储的是命令，恢复数据需要逐条执行，恢复慢，且aof文件比RDB文件大，恢复速度慢。

====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，数据库的字段优化，用合适的数据类型，加索引；
2. 加缓存，用redis缓存；
3. 主从复制，读写分离；
4. ~~分区表；~~
5. ~~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（logn），分块查找（介于折半查找和二分查找之间），跳跃表（logn）；
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logn+1层索引指针，查找、插入、删除的时间复杂度是logn。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reciver 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：发送发维护一个拥塞窗口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，发送发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rebanlance，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。

====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时无状态的，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；优点时保证所有连接都是有效的连接，缺点时创建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片没有运行完，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：

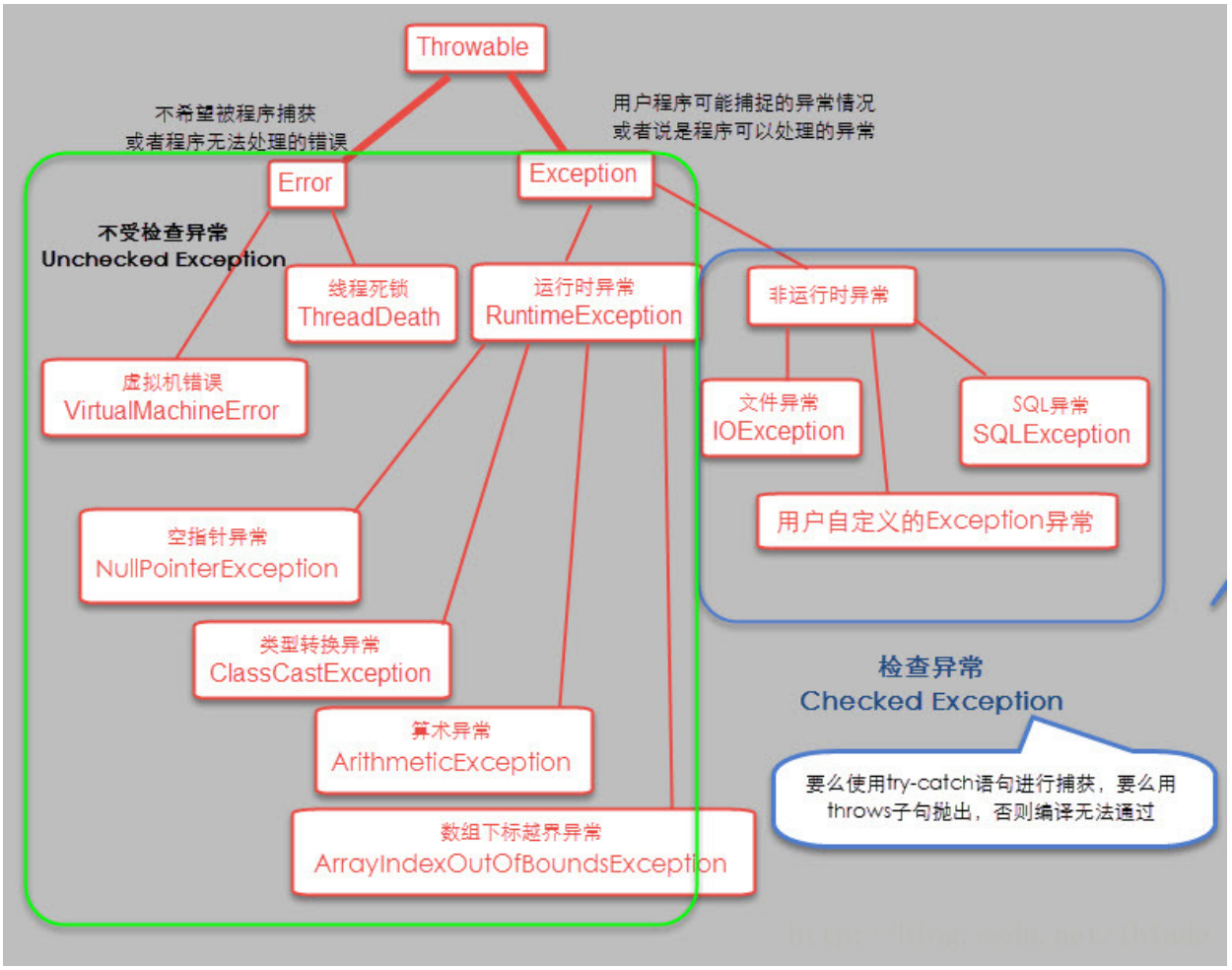
1. 构造一个对象，不必关心构造对象的细节和过程，把细节和过程交给工厂类；

2. 简单工厂顾明思议，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selectCilrd()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 ofController）：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 Programing）：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；

c. spring的aop和ioc是为了解决系统代码耦合度过高的问题，使代码重用度高、易于维护。

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. 一个表只能又一个聚集索引，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socet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(logn)；

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/mencache；

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(log2n)

分块查找：介于顺序查找和二分查找之间

跳跃链表：O(log2n)

基于树的查找：

二叉排序树：O(log2n)

平衡二叉树：O(log2n)

B-树：O(log2n)

B+树：O(log1.44n)

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. 装饰者模式是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：**给一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. SptingAOP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。

e. JDK和CDLib区别：

i. JDK动态代理只能对实现了接口的类生成代理，使用反射完成；

ii. CDLib则不需要接口，对指定的类生成子类，覆盖其中的方法，不能对final类进行继承，使用了动态生成字节码技术。

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0（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（logn）
4. Nlogn
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logn，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. 一个表只能又一个聚集索引，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，写读用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，客户端和服务器都可以主动释放，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. 服务的收到FIN报文，发送一个ACK报文，表时确认收到，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，表时确认，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显示所以进程 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. 只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，在到磁盘数据库操作。

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。缺点时文件大，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。永久代在方法去，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to区，放不到放老年代，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Miror 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GCRoots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. <max线程，创建线程
2. >=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RejectedEcexutionException

b. DiscardPolicy：丢弃任务不抛出异常

c. DiscardOrdestPolicy：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SYN1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. WSAsyncSelcet（异步选择）：网络事件以消息形式通知应用程序

iii. WSAEvenSelect（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logn+1层索引指针，查找是O（logn）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

====2020/09/15====

一、面试错题

1. robc
2. 数据库连接池
3. 实现ssm框架
4. mysql造一个死锁
5. 写一个服务器
6. 秒杀系统设计
7. udp实现可靠传输
8. 哨兵选举策略

二、ISO七层模型

1. 物理层，
2. 数据链路层，
3. 网络层，
4. 传输层，
5. 会话层，
6. 表示层，
7. 应用层。

三、错题解答

1. ucp实现可靠传输
    a. 方案：
        i. 将实现放在应用层，类似TCP实现确认机制、重传机制和窗口确认机制；
        ii. 解决丢包和包无序问题；
        iii. 给数据包编号，按顺序接收并存储，接收端收到数据包后发送确认信息给发送端，发送端接收到确认信息后继续发送，若接收端接收的数据不是期望的顺序编号，则要求重发。
    b. 已经实现的可靠udp：

i. RUDP，RTP，UDT。

c. tcp可靠传输：

i. 序号、校验、确认机制、超时重传、流量控制、拥塞避免。

2. 数据库连接池

a. 概念：

i. 程序启动时建立足够的数据库连接，并将这些连接组成一个连接池，由程序动态地对池中的连接进行申请、使用、释放。创建数据库连接是一个很耗时的过程，也容易造成安全隐患，所以，在程序初始化的时候，集中创建多个数据库连接，并把他们集中管理，供程序使用，可以保证较快的数据库读写速度。

b. 运行机制：

i. 程序初始化时创建连接池，使用时向连接池申请可用连接，使用完毕将连接返还给连接池，程序退出时，断开所有连接并释放资源。

c. 应用：

i. C3P0，阿里开源的Druid。

3. 哨兵模式

a. 作用：

i. Redis哨兵模式是Redis高可用架构，主服务器挂了，在从服务器中选举出一个作为主服务器。

b. 工作流程：

i. 哨兵每隔1s会ping一次主服务器，如果一个服务器ping的时间超过了设置的down-after-miliseconds时间，这个服务器就会被哨兵标记为主观下线（SDOWN）；

ii. 哨兵会询问其他哨兵，如果有一定数量（半数以上）的哨兵认为主服务器挂了，会把主管下线改为客观下线（ODOWN）；

iii. 选举出新的从服务器作为主服务器。

c. 选举策略：

i. 过滤故障节点；

ii. 根据优先级进行选择，priority；

iii. 选择复制偏移量最大的，也就是同步的数据最多的为主服务器；

iv. 选择runid最小的从为主，runid值越小说明重启时间越靠前。

4. 1000瓶药水，有1瓶是毒药，给很多小白鼠，如何找出毒药：

a. 1000瓶毒药和10只小白鼠；

b. 将10只小白鼠按二进制排序，2^10=1024，能够表示1000瓶药水；

c. 将1000瓶药水的编号转换为二进制，如果位数是1，喂给对应的小白鼠吃；

d. 找到对应死亡的小白鼠的二进制，转为十进制就是毒药编号。

5. 根据成绩分割查询每个成绩段的人数

a. select count(*) as num from (select case when score >= 90 and score <=100 then "1" when score >= 80 and score < 90 then "2" end as score from score_table) a group by score;

6. springboot的优点

a. springboot优化了配置文件，用stater优化了搭建项目，springboot内嵌了Tomcat服务器。

7. springboot设计模式

a. 工厂设计模式

i. 对应的工程来创建bean对象；

b. 单例

i. 保证bean是单例的；

c. 代理设计模式

i. AOP，JDK动态代理，CGLib字节码生成技术代码。原理就是使用代理模式对类进行方法级别的切面增强，也就是，生成代理类，并在代理类的方法前，设置拦截器，通过执行拦截器中的内容增强了代理方法的功能，实现了面向切面编程。

8. 饿汉式和懒汉式的区别

a. 饿汉式在类加载的时候初始化，加载慢获取对象快。

b. 饿汉式是线程安全的，在类加载的时候创建好静态对象提供给系统使用，懒汉式线程不安全，可以用双检锁优化。

9. Java实现多继承

a. 内部类；

b. 在子类里面，定义一个内部类继承A类，再定义一个内部类继承B类，创建AB的实例，通过实例调用A类中的方法。

10. 接口是一个公共的规范。

一、Redis

1. Redis为什么不是多线程?

a. Redis的瓶颈不在CPU，而在网络带宽和内存大小，它是基于内存读写数据，IO很快，单线程占用CPU的时间不会很长，且多线程有上下文切换的开销；

====2020/09/15====

一、java基础

1. ThreadLocal

a. 线程局部变量，只能在本线程访问，不能在线程之间共享的变量；

b. 原理是通过一个ThreadLocalMap来实现，map中key是ThreadLocal实例，value是需要保存的值，这个key所指向的实例是一个弱引用，随时会被回收。

2. ArrayList扩容

a. newCapacity = oldCapacity + oldCapacity >> 1；扩容为原来的1.5倍；

b. 如果1.5倍小于mincapacity，则newcapacity = mincapacity，再判断newcapacity和MAX_ARRAY_SIZE大小，大于则将maxarraysize设置为mincapacity。

c. 调用Arrays.copyof()，复制原来数组内容到新数组；Arrays.copyof()实际调用的是System.arraycopy()方法。

3. 线程池的作用：

a. 线程不需要每次创建和销毁，可以节约资源，响应实际更快；

4. 抽象类和接口：

a. 继承：抽象类只能单继承，接口可以多实现；

b. 成员属性：抽象类可以有普通属性，也可以有常量，接口中的成员变量默认是常量；

c. 代码块：抽象类可以有初始化代码块，接口不能初始化；

d. 构造函数：抽象类可以有构造函数，用来初始化，接口不能有构造函数；

e. 方法：抽象类可以有抽象方法和普通方法，接口只能是抽象方法。

f.

g. ~~多个实现；~~

h. ~~抽象类和接口都不能直接实例化；~~

i. ~~抽象类被继承，接口被实现；~~

j. ~~抽象类中可以有具体的方法实现；~~

k. ~~抽象类可以包含非抽象方法，接口中的所有方法必须是抽象的，不能有非抽象方法；~~

l. ~~接口的抽象方法必须是public的，抽象类的首先方法可以是publicprotectd。~~

5. HashMap加载因子

a. 0.75加载因子这个值是为了触发扩容，减少冲突发生的概率；

b. 因为hashmap的初始容量大小默认是16，所以当hashmap的数组长度达到一个临界值16x0.75=12时，会触发扩容操作；

c. 0.75是基于泊松分布和指数分布。

二、http

1. http1.0/1.1

a. 长连接，header信息，host域，缓存策略和错误码；

b. http1.1支持长连接，减少tcp三次握手开销；

c. http1.1支持只发送header信息，收到服务器的返回状态码之后，再发送请求体；

d. http1.1支持host域；~~通过不同的host可以区分访问服务器不同的站点~~；

e. http1.1有缓存控制策略，有更多的错误状态码；

2. http2.0

a. 多路复用，二进制分帧层，首部压缩，服务器推送；

b. 多路复用技术，允许同时通过单一的连接发送多重的请求响应信息。http1.1在同一时间对同一域名的请求数量是有限的，超过就会阻塞请求。多路复用采用二进制分帧层，将信息分割为更小的帧，用二进制进行编码，多个请求在同一个tcp连接上完成，有效使用tcp连接。

c. 多路复用：同一个连接并发处理多个请求。

d. 二进制分帧层，应用层和传输层之间；

e. 首部压缩，对header进行压缩，体积更小传输更快；

f. 服务器推送，服务器可以向客户端的一个请求发送多个响应。

3. https

a. SSL证书，携带公钥。

====2020/09/14====

一、java连接数据库

1. 利用java原生jdbc连接数据库

a. jdbc连接数据库的流程；

2. 通过properties配置文件，利用jdbc连接数据库
3. 利用c3p0（配置文件）连接数据库
4. 利用dbcp（properties配置文件）连接数据库
5. 利用mybatis连接数据库
6. 利用hibernate连接数据库

二、TCP/UDP

1. 三次握手的第三次ACK可以携带数据吗?

a. 可以携带数据，用的是fast-open技术，需要内核支持。

====2020/09/13====

synchronized volatile lock原理
ReentrantLock，ThreadLocal
线程的状态
java连接数据库

一、synchronized volatile lock

1. synchronized（关键字，jvm层面）：

a. synchronized用来锁住对象、方法、代码块，互斥、共享、可重入性；

b. 确保线程互斥访问同步代码，保证共享变量的修改能够及时可见。

c. synchronized的底层是通过一个monitor对象来完成的，notify/wait 这些方法都是依赖于monitor对象。每个对象有一个monitor监视器锁，当monitor被占用时就会处于锁定状态，线程执行monitorenter指令，尝试获取monitor的所有权；如果monitor的进入数为0，当前线程进入monitor，然后将进入数设置为1，该线程时monitor的所有者；如果线程已经占有了monitor，知识重新进入，则monitor的进入数加1；如果其他线程占有了monitor，这当前线程进入阻塞状态，直到monitor的进入数0，再尝试获取monitor的所有权；

d. 当monitor的所有者线程执行monitorexit时，monitor的进入数减1，如果减1后进入数为0，那么线程退出monitor，不在是这个monitor的所有者，其他线程可以尝试获取monitor的所有权。

e. ~~互斥：synchronized代码块中，只有一个线程能够获得锁，其他线程会阻塞；~~

f. ~~共享：类中的多个synchronized代码块，共享同一个对象锁；~~

g. ~~可重入性：同一线程可多次获得对象上的锁，通过计数器加锁加1，退出减1。~~

2. volatile（修饰符）：

a. 保证了可见性、有序性，不能保证原子性，用来修饰被不同线程访问的变量。

b. 原理：

i. 在对volatile变量进行写操作的时候，JVM会向处理器发送一条Lock前缀的指令，将这个变量所在工作内存的数据写回到主内存，在多处理器下，为了保证各个处理器的缓存是一致的，会实现缓存一致性协议，每个处理器通过嗅探找到主内存中的数

据，来检测自己的数据是否过期，当处理器发现数据被修改，会将当前处理器的缓存行设置为无效状态。当前处理器要对这个数据进行修改操作的时候，会强制重新从系统内存里把数据读到处理器缓存里。

c. 原理：在volatile变量修饰的共享变量进行写操作时，会增加一个lock前缀的指令，lock前缀的指令会有2个操作，

i. 将当前工作内存的数据写到主内存中；

ii. 更新到主内存的操作会引起其他线程对应的工作内存设置为无效。

iii. lock前缀指令相当于一个内存屏蔽，volatile的底层就是通过内存屏蔽来实现的。

d. 可见性原理：

i. 线程对volatiel变量进行修改时，JVM会把该线程对应的工作内存中的共享变量值刷新到主内存中，刷新到主内存的操作会引起其他线程的工作内存设置成无效，JVM会把该线程对应的工作内存设置为无效，那么该线程只能从主内存中重新读取共享变量，保证变量值最新。

3. Lock（类），ReentrantLock实现了Lock接口，可以通过该类来实现线程同步。

a. 关键字：State状态值，双向链表，CAS+自旋；lock.lock()，lock.unlock()；

b. 存储结构：包括1个state状态值，是int类型的，和一个双向链表，state状态值用于表示锁的状态，双向链表用于存储等待中的线程；

c. 获得锁：线程通过CAS来对状态值修改，如果没有修改成功，会把线程放入双向链表中等待，尾进头出；当线程到头部时，尝试

cas更新锁的state状态值，如果更新成功则表示获取锁成功，从链表头部移除。

d. 释放锁：修改状态值，调整等待链表。

二、可见性、原子性、有序性

1. 可见性：变量被操作之后，能够快速的写入内存，并提醒其他线程更新；
2. 原子性：过程属于原子操作；
3. 有序性：在Java内存模型中，允许处理器对指令进行重排序，重排序过程不会影响单线程程序的执行，却会影响到多线程并发执行的正确性。

三、synchronized和lock

1. 存在层面：synchronized是Java关键字，是jvm层的；lock是一个类；
2. 锁的获取：如果线程A获得锁，线程B会等待；
3. 锁的释放：synchronized中线程执行完或出现异常，线程释放锁；Lock中必须在finally中释放；
4. 锁的状态性能：sychronized中锁的状态是无法判断的，适用于少量同步；Lock锁的状态可以判断，适用于大量同步。

====2020/09/14====

一、get/post

1. get在浏览器回退时无害，post会重新提交；回退操作会从之前的缓存中拿结果；
2. get的url是可见的，会主动缓存，get在url上进行传参，url有长度限制，且只能接收ASCII码；
3. post通过requestbody传输参数，所以post更加安全；

二、TCP/UDP

1. TCP是基于连接的可靠传输，UDP是无连接的不可靠传输；
2. TCP有三次握手四次挥手，安全性好；UDP实时性好；
3. TCP首部开销20个字节，UDP8个字节；
4. TCP是端到端的。

三、线程/进程

1. 进程是资源分配的最小单元，线程是程序执行的最小单元；

2. 进程有自己独立的内存地址空间，进程中的多个线程共享内存空间。

3. 进程间通信：信号、管道、消息队列、共享内存；线程通信：volatile关键字，wait/notify，ReentrantLock，LockSupport。

====2020/09/12====

一、锁

1. 作用：数据库使用锁是为了支持对共享数据进行并发访问，提供数据的完整性和一致性。

2. 分类：悲观锁、乐观锁、共享锁、排它锁、记录锁、间隙锁、临键锁。

二、锁的具体分类

1. 乐观锁和悲观锁：

a. 乐观锁：

i. 乐观锁是基于乐观的概念，每次去读数据的时候都认为其他事务不会对数据进行修改操作，但是在更新数据的时候，会先判断这个值是否被其他事务修改，如果发生冲突，就进行回滚，cas就是一种乐观锁；

ii. 优点：避免了加锁解锁的开销；

iii. 缺点：增加了冲突。

iv. 乐观锁是在提交的时候检验冲突，悲观锁是加锁避免冲突。资源竞争小，用乐观锁；资源竞争大，用悲观锁。

b. 悲观锁：

i. 认为数据随时会修改，所以在数据处理过程中需要对数据加锁，这样别的事务拿到这个数据就会block阻塞，直到它拿到锁。关系型数据库中的行锁、表锁、读锁、写锁都是悲观锁，在读之前先加上锁；悲观锁的实现方式，依赖于数据库；

ii. 优点：使用阻塞的方式，所以避免冲突；

iii. 缺点：并发性能不好，因为未获得锁会阻塞。

2. 共享锁和排它锁：（按锁的类型分类）

a. 共享锁：

i. 共享锁，S锁，也是读锁，事务A对数据加了共享锁，其他事务也只能对这个数据加S锁，多个用户可以同时读，但不允许有写操作，直到除了某个事务自身外，其他事务都放掉共享锁，这个事务才能获得排他锁。

b. 排它锁：

i. 排它锁，X锁，事务A对对象加X锁以后，只有事务A可以读写该对象，其他事务不能对该对象加任何锁，直到事务A释放X锁。

c. 意向锁：

i. 意向共享锁、意向排它锁；

ii. 意向锁是用来表示事务接下来想要获得的锁的类型；

iii. 意向排它锁和共享锁是不兼容的。

3. 一致性非锁定读和一致性锁定读

a. 非锁定读：

i. 通过MVCC实现，读取的是快照版本。

b. 锁定读：

i. 用法：s锁lock in share mode, x锁for update；

ii. 防止死锁。因为使用共享锁的时候，修改操作，需要先获得共享锁，读取数据，再升级为排它锁，然后进行修改操作。这样如果同时有多个事务对同一个数据对象申请共享锁，然后再同时升级为排它

锁，这些事务都不会释放共享锁，而是等待其他事务释放，就造成了死锁。

iii. 因此，在数据修改前的select 语句中直接申请排它锁，其他数据就无法获取共享锁和排它锁，避免死锁。

MVCC

i. MVCC多版本并发控制，查询需要对资源加共享锁（s），修改需要对数据加排他锁（X）；

ii. 利用undo log回滚日志使读写不阻塞，实现了可重复读。当一个事务正在对一条数据进行修改的时候，该资源会被加上排它锁。在事务未提交时，对加锁资源进行的读操作，读操做无法读到被锁资源，会通过一些特殊的标识符去读undo log 中的数据，这样读到的是事务执行之前的数据。

4. 外键和锁

a. 在Innodb中，对于一个外键，如果没有显示的加索引，会自动地加上索引，避免表锁；

b. 对于外键的修改，需要对父表加S锁。这是，如果父表有X锁，则子表上加S锁的操作会阻塞；如果对父表加上S锁之后，则父表上不会加上X锁，保证了数据的一致性。

5. 行锁的三种算法（记录锁、间隙锁、临键锁）

a. 在可重复读的事务隔离级别下解决幻读的问题。幻读：可重复读的事务隔离级别下可能出现幻读，因为，可重复度保证了当前事务不会读取到其他事务已提交的update 操作。但同时，也会导致当前事务无法感知到其他事务的Insert或delete操作，这就产生了幻读。

b. 记录锁（Record lock）：

i. 锁住的是一条记录，它必须命中索引并且索引是唯一索引，否则会使用间隙锁；

c. 间隙锁（gap lock）：

i. 锁定的是一个范围，但不包含记录本身，多个事务可以同时持有间隙锁，但任何一个事务都不能在

锁范围内进行插入修改操作。

ii. 间隙锁是基于非唯一索引的，它锁定一段范围内的索引记录。

iii. ~~非唯一索引的记录之间的间隙上加的锁，锁定的是尚未存在的记录，间隙锁是基于临键锁的~~；

d. 临键锁（Next-key lock）：

i. 临键锁是记录锁+间隙锁，也是用在非唯一索引的，锁定的是一个范围，并且锁定记录本身，是一个左开右闭的区间，只会出现在可重复读隔离级别，是为了防止幻读。

ii. ~~在根据非唯一索引对记录进行加锁修改操作时，会获得该记录的临键锁，和该记录下一个区间的间隙锁。~~

iii. ~~临键锁是记录锁+间隙锁，锁定的是一个左开右闭的索引区间。间隙锁和临键锁只会出现在可重复读隔离级别，可以避免幻读~~。