笔记

进程间通信管道底层ps grep如何通信，显示的是Java虚拟机的进程还是所有Java进程？

stringBuffer和stringbuilder底层，为什么不像string：

**String 字符串常量**

**StringBuffer 字符串变量（线程安全）**

**StringBuilder 字符串变量（非线程安全）**

简要的说， String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。

而如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用。所以在一般情况下我们推荐使用 StringBuffer ，特别是字符串对象经常改变的情况下。而在某些特别情况下， String 对象的字符串拼接其实是被 JVM 解释成了 StringBuffer 对象的拼接，所以这些时候 String 对象的速度并不会比 StringBuffer 对象慢，而特别是以下的字符串对象生成中， String 效率是远要比 StringBuffer 快的：

String S1 = “This is only a” + “ simple” + “ test”; StringBuffer Sb = new StringBuilder(“This is only a”).append(“ simple”).append(“ test”);

你会很惊讶的发现，生成 String S1 对象的速度简直太快了，而这个时候 StringBuffer 居然速度上根本一点都不占优势。其实这是 JVM 的一个把戏，在 JVM 眼里，这个 String S1 = “This is only a” + “ simple” + “test”; 其实就是： String S1 = “This is only a simple test”; 所以当然不需要太多的时间了。但大家这里要注意的是，如果你的字符串是来自另外的 String 对象的话，速度就没那么快了，譬如：String S2 = “This is only a”;String S3 = “ simple”;String S4 = “ test”;String S1 = S2 +S3 + S4;这时候 JVM 会规规矩矩的按照原来的方式去做

arrayList和vector扩容 为何是1.5倍，有什么性能问题操作系统？

分页和分片

TCP流量控制，慢启动

sql注入 防止sql注入的方式

创建虚拟机底层实现

XEN技术

四次挥手

输入一个网址之后发生了什么

thrift：首先写thrift文件（request，response，service，然后在plugin里面generate，会自动生成Java文件）然后写mapper.xml，在xml文件里写sql语句  
maven：http://maven.sankuai.com在子目录下，在非test模式下deploy，完成后在maven.sankuai.com中查看是否更新，最后提供pom中的更改的api的包的artifactId还有version版本  
  
plus：https://plus.mws.sankuai.com部署任务->如果构建过就可以包部署，不然就按照分支部署剩下的会自动填好；选好主机之后就可以发布了  
cargo : cargo.sankuai.com在cargo里面——新建应用编排，新建团队，在应用编排的修改配置——修改全局配置里面操作团队的权限。然后在应用里点击构建，填写分支，最后构建。查看泳道名称即可。具体发布情况到plus里查看  
Ops：ops.sankuai.com选择好机器，根据名称、IP选好，之后登陆即可查看发布的机器的日志  
Octo:http://octo.test.sankuai.com在Octo下面搜索服务，选择环境，然后根据主机端口号查看服务是否已经发布成功  
Pom文件版本要对齐，如果maven build失败可能是依赖冲突  
环境：/data/webapps下的appenv文件是环境配置，环境不对就改配置  
有很多坑都是环境的问题，比如数据库出问题，Oracle说找不到存储方法声明或者连不上什么的，可以先检查一下环境是不是对的！！  
  
数据库 找不到表名 很可能是代码对应的用户没有权限  
调用EBS过程记得加上apps.  
在使用JUnit时，如果获取不到bean，去看下conf.spring下的配置文件是不是写对了，service是否加了注解  
source code does not match the bytecode:是因为没有打包，所以要先clean一下，把涉及到的包都install一下，然后再调试  
提PR在git.sankuai.com里面点击New PR，source填写自己的分支，target填写master，然后点击create Pull Request，然后写标题和描述，Reviews填写负责人（caojianzhou），让负责人来通过这个PR  
  
上线部署在prod里面，分支一般选master，不基于包部署！！  
  
【上线通报】【标题】账务系统-基础数据【时间】2019年07月09日 18:00【内容】中途岛优化-在可选列表中只显示启用的段值规则【影响范围】基础数据-入账规则【pr】http://git.sankuai.com/v1/bj/projects/FIN/repos/finance-base/pull-requests/360/overview【发布分支】master【预计完成时间】2019年07月09日 18:10【联系人】caojianzhou  
  
如果发现同一个url，同一个参数，返回值不一样。可以考虑是不是有服务器没有部署上新提交的代码，在octo上看“所有”的服务器。可能出现plus在diff上只有一台服务器，但是实际上有两台，其中只有一台部署了新的代码的情况。  
mybatis #{clientId,jdbcType=VARCHAR}后面那个类型真是让人心累= =  
select \* from business\_account\_rule\_conf where system\_code = &apos;MobikeFA-IFRS&apos;  
update business\_account\_rule\_conf set segment1\_type = &apos;I&apos;, segment1 = &apos;RECORD\_COMPANY&apos; where system\_code = &apos;MobikeFA-IFRS&apos;;update business\_account\_rule\_conf\_unsaved set segment1\_type = &apos;I&apos;, segment1 = &apos;RECORD\_COMPANY&apos; where system\_code = &apos;MobikeFA-IFRS&apos;;  
  
1.accountInfo 数据没有导入，是因为ccid校验不通过。明天试一下正确数据使ccid校验通过，然后试一下是否能正常导入。联调完了，一脸血。。。。1.Oracle工具，显示不出来过长的数值类型，这个修改工具的配置即可2.同一个session>同一个事务。要先提交事务，才能调用ebs过程3.批量导入的幂等问题。不能只在外面校验新导入的一批里是否有与表中相同的数据。还需要判断新的一批本身里，是否有相同的数据。这个据说数据量很大，需要考虑大数据还有并发的问题。目前并没有解决并发问题（解决了，使用select for update。  
2.clientInfo这个数据写入了那张表CUX.CUX\_ZTD\_CLIENT\_TEMP client里的所有数据，但是调用EBS过程修改不成功。原因虽然是同一个session，但是同一个事务没有提交，所以调用ebs过程不行。解决方法：平级的两个事务即可。  
3.select for update 当这个语句的事务没有提交的时候，其他事务永远停滞等待。  
线程：在 set 方法中，首先获取该线程对应的 ThreadLocal 集合，然后向 map 集合中添加键值对，key 为该 ThreadLocal 对象，value 为 set 方法的入参。get 方法操作也是类似的，首先获取线程对应的 ThreadLocal 集合，再查询 ThreadLocal 对象对应的 value。  
注意：需要在线程执行完成之后，要通过 remove 方法来删除 ThreadLocal 里保存的值，否则会造成内存泄漏。  
  
  
事务的隔离级别：用例子说明各个级别的情况：  
未提交读——① 脏读: 脏读就是指当一个事务正在访问数据，并且对数据进行了修改，而这种修改还没有提交到数据库中，这时，另外一个事务也访问这个数据，然后使用了这个数据。（脏读是对于两个事务之间的）  
已提交读（可解决脏读）——② 不可重复读:是指在一个事务内，多次读同一数据。在这个事务还没有结束时，另外一个事务也访问该同一数据。那么，在第一个事务中的两次读数据之间，由于第二个事务的修改，那么第一个事务两次读到的的数据可能是不一样的。这样就发生了在一个事务内两次读到的数据是不一样的，因此称为是不可重复读。（不可重复读是一个事务之内两次查询由于另外一个事务的提交操作而受到的影响，比起脏读，人家好歹是提交的操作。）  
可重复读（可解决不可重复读）——③ 可重复读: 在一个事务之内的两次查询不会受到另外一个事务的提交操作影响。当第一个事务提交之后，再次查询，才会得到第二个事务的提交结果  
可序列化（可解决幻读）——④ 幻读:第一个事务对一个表中的数据进行了修改，这种修改涉及到表中的全部数据行。同时，第二个事务也修改这个表中的数据，这种修改是向表中插入一行新数据。那么，以后就会发生操作第一个事务的用户发现表中还有没有修改的数据行，就好象发生了幻觉一样。  
  
  
数据库引擎：Mysql有两种引擎，innodb和myisam。这两个主要区别是前者行级锁，后者表锁。表锁并发性能低。但是MySQL如果select for update的where条件不是主键或索引，就不会锁行而是锁表。Oracle倒是没有这种顾虑。  
两种引擎所使用的索引的数据结构是什么？  
答案:都是B+树!  
MyIASM引擎，B+树的数据结构中存储的内容实际上是实际数据的地址值。也就是说它的索引和实际数据是分开的，只不过使用索引指向了实际数据。这种索引的模式被称为非聚集索引。  
Innodb引擎的索引的数据结构也是B+树，只不过数据结构中存储的都是实际的数据，这种索引被称为聚集索引  
  
  
锁：可重入锁ReentrantLock（默认是非公平锁）Lock lock = new ReetrantLock();//默认为非公平锁//Lock lock = new ReetrantLock(true); //公平锁Lock.lock()最好写在try里面 为何？？？？？可重入锁，就是支持重进入的锁，它表示该锁支持一个线程对资源重复加锁。  
举例  
可重入：A线程当前持有锁lock，A线程再次要求获取锁lock时，A线程可再次获取锁成功。  
不可重入：A线程当前持有锁lock，A线程再次要求获取锁lock时，A线程此次获取锁的操作将被阻塞。  
除了可重入特性，该锁还支持获取锁的公平和非公平性选择。  
什么是公平锁，非公平锁？如果在绝对时间上，先对锁进行获取的请求一定先被满足，那么这个锁是公平的，反之，是非公平的。  
举例，线程A在t1时刻请求获得锁，线程B在t2时刻请求获取锁，且t1 < t2，如果为公平锁，则一定是线程A先获得锁，然后B先获得锁。公平锁保证了锁的获取按照FIFO原则，但代价是大量的线程切换。  
非公平锁虽然可能造成线程“饥饿”，但极少的线程切换，保证了其更大的吞吐量。  
ReentrantLock是排他锁，在同一个时刻仅允许一个线程访问资源。读写锁则不一样  
读写锁  
读写锁在同一个时刻刻允许多个读线程访问。  
当写线程访问时，所有的读线程和其他写线程均被阻塞。  
读写锁维护了一对锁，一个读锁和一个写锁，通过分离读锁和写锁，使得并发性相比排它锁有了很大提升。  
在try代码块之前调用Lock实现类的lock()方法，避免由于加锁失败，导致finally调用unlock()抛出异常。  
说明：在lock方法中可能抛出uncheck异常，如果放在try代码块中，必然触发finally中的unlock方法的执行，它会调用AQS的tryRelease方法，（取决于具体实现类）。根据Lock接口中的unlock描述，对未加锁的对象解锁抛出unchecked异常，如：IllegalMonitorStateException，虽然都是加锁失败造成程序中断，但是真正加锁出错信息可能被后者覆盖。