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？**

首先，vector为什么不是固定大小扩容：由数学公式得知，对比可以发现采用采用成倍方式扩容，可以保证常数的时间复杂度，而增加指定大小的容量只能达到O(n)的时间复杂度，因此，使用成倍的方式扩容。

vector最关键在于查询，使用移位（2的幂）直接得到哈希链以及节点长度，然后相减直接得到键值，复杂度为O(2)，性能近似于数组，插入删除可动态，这就是vector设计的基本目的。

**它们的相似点和非相似点？**

**相同之处**

都实现了List接口

底层数据结构都是数组

API的实现几乎都是一样的

扩容机制是一样的，由于数组这种数据结构，在创建时它的容量就已经确定，因此想要随着数据项的插入数组容量发生动态变化，就需要创建一个新容量的数组，把旧数组中的所有元素拷贝到新容量的数组中，这就是扩容机制，但是它们的扩容策略稍有不同

查询和修改都比较快，增删都比较慢。有索引的话查询和修改都会很快，但是扩容的话，需要创建新的数组并复制旧数组的元素到新数组中，因此添加会比较慢。移除元素的话，需要后面的元素前移，因此也比较慢

**不同之处**

扩容策略稍有不同，ArrayList对象创建后会初始化一个容量为0的数组，添加第一个元素才会扩充容量为10，并且是1.5倍扩容，而Vector对象一创建后就会初始化一个容量为10的数组，并且是2倍扩容

ArrayList是线程不安全的，Vector是线程安全的

**分页和分片：**

分页就是将进程的逻辑地址空间分成若干大小相等的片（即页），然后装入内存。

分段就是用户可以把自己的作业按逻辑关系划分为若干个段，每个段都是从0开始编址，并有自己的名字和长度。这就相当于程序里边的主函数段、各个子函数段、数据段、栈段等等。

要知道在引入分页和分段之前，是通过连续分配方式来管理存储器的，就是说一个进程在内存中是连续存放的。

可以这样理解吧，内存中有进程1、2、3……进程2先执行完成了，然后释放了所占有的内存空间，而之后，

如果新调入的进程内存需求大于之前2所占有的内存空间，那么不可能利用这块内存，相对于内存需求更大的进程来说，之前2所占有的内存空间就是不能利用的碎片，

如果新调入的进程内存需求小于之前2所占有的空间就会留下空隙，也会带来碎片。

虽然可以通过“紧凑”的方法进行碎片整理，但开销很大这就产生了与连续分配方式相对的离散分配方式，便先后引入了分页和分段存储管理。

所以相同点就是，两者都属于存储器管理方式中的离散分配方式。都要通过地址映射机构来实现地址变换。

不同点就在于，离散分配方式的基本单位的不同，是页还是段。

页是信息的物理单位，段是信息的逻辑单位。

分页的作业地址空间是一维的，线性的，程序员只需利用一个记忆符表示一个地址；而分段的作业地址空间是二维的，程序员在表示一个地址的时候既要给出段名，又需要给出段内地址。其中，段名可以理解为函数名等，段内地址可以理解程变量等的地址。

值得一提的是其中页的大小要和物理块或者页框一样，而物理块是操作系统定好了的。也就是说你的操作系统装完之后，物理块的大小是定了的，从而页的大小也是定了的。而段的长度是不定的。

引入分页存储管理方式的目的是提高内存利用率，而引入分段存储是为了满足用户（程序员）在编程和使用上多方面上的要求。还有将两者结合一起的段页式系统。

**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：**

为什么连接的时候是三次握手，关闭的时候却是四次挥手？

关闭连接时，当Server端收到FIN报文时，很可能数据信息没有传完并不会立即关闭连接，所以只能先回复一个ACK报文（告诉Client端，"你发的FIN报文我收到了"）。只有等到Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步挥手。

**输入一个网址之后发生了什么**

第一步 浏览器查找该域名的 IP 地址

第二步 浏览器根据解析得到的IP地址向 web 服务器发送一个 HTTP 请求

第三步 服务器收到请求并进行处理

第四步 服务器返回一个响应

第五步 浏览器对该响应进行解码，渲染显示。

第六步 页面显示完成后，浏览器发送异步请求。

下面对每个环节做进一步分析：

01 浏览器查找该域名的 IP 地址

浏览器缓存 首先是查找浏览器缓存,浏览器会缓存DNS记录一段时间，不同浏览器保存的时常不等（2分钟到30分钟不等）。

系统缓存 如果在浏览器缓存里没有找到需要的记录，浏览器会做一个系统调用来查找这个网址的对应DNS信息。

路由器缓存 如果在系统缓存里没有找到找到对应的IP，请求会发向路由器，它一般会有自己的DNS缓存。

ISP DNS服务器 如果在路由器缓存里还是没有对应的IP，请求会被发送到ISP。

根域名服务器 如果还是没有，请求将发向根域名服务器进行搜索。找不到就说明此域名不存在。

**SpringMVC工作流程**

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。

Thrift

Mafka：

Mafka是基础架构-MQ团队从2016年开始自研的消息队列产品，底层基于Apache Kafka，增加了自研的基于机房粒度的中心化调度、时间回溯、粘性分配、死信、延迟队列、适用于美团自用的同步/异步客户端、机房容灾等高阶特性

Mafka普通的消息队列，可以满足传统的消息收发功能，**发送方**可以将消息发送给**Mafka队列**，其他各**接收方**可以来消费这个队列，接收方A和接收方B都可以收到a、b、c、d四个消息，接收上游业务方发送过来的消息。

在一些情况下，你可以先将消息发送给Mafka队列，等经过一定的时间后，再由Mafka将消息投递给下游消费方，这就是Mafka延迟队列。

Mafka可以支持最大7天的消息回溯，简单来说，就是你可以多次重复消费最大7天的消息(默认7天，根据业务需求可以调节)。

Zebra：

数据库访问层中间件，主要提供动态数据源、读写分离、分库分表、端到端监控等功能

zebra-dao是一套基于mybatis框架做了一个轻量的封装，本质上的用法就是mybatis的用法

zebra中主要包括三个实现了JDBC协议的数据源，分别是：

* SingleDataSource：负责抽象底层使用的连接池类型（c3p0,druid,tomcat-jdbc等），然后直连每一个数据库实例。每一个MySQL Master或者Slave，都对应一个SingleDataSource。
* GroupDataSource：负责读写分离的连接池，它主要负责判断SQL的读写操作，然后把相应的SQL发送给SingleDataSource。它负责连接一个数据库集群，因此它会包含若干个SingleDataSource。
* ShardDataSource：负责分库分表的连接池，它主要判断SQL的落到哪个分片上，然后把相应的SQL经过处理后发送给GroupDataSource。它负责连接多个数据库集群，因此它会包含若干个GroupDataSource。

**KMS 加解密服务是什么？**

KMS 密钥管理服务 2015 年正式上线，主要解决的问题是**消除公司内所有的明文密码**。但是随着公司的发展，原有特性已不足以满足公司现状，很多业务数据也需要做加密存储。

KMS 加解密服务可以帮助用户创建、管理加密密钥，对敏感数据进行保护，避免数据泄露造成用户、企业利益受损。



KMS 加解密服务主要提供了以下功能：

1. 密钥管理
   1. 实现密钥分层管理机制，主密钥不下发到业务系统，主密钥存储时使用根密钥加密保护，安全性更高。
   2. 提供密钥轮换机制，进一步提升安全性。
   3. 记录密钥访问审计日志，方便安全审计。
2. 两种加解密模式
   1. 远程加解密：安全性更高，适合对数据安全要求特别高的应用。
   2. 近似本地加解密：借助“信封加密”机制实现近似本地加解密的性能，在不大幅降低安全性的前提下，提升了可用性。
3. 数仓 UDF 函数：方便在数据仓库里进行批量加解密，方便做数据分析。

|  |  |  |
| --- | --- | --- |
| **name** | **简义** | **含义** |
| MK | 主密钥 | 用于对较短的数据或数据密钥 DK 进行加密。 |
| DK | 数据密钥 | 概念来源于“信封加密”，另外在“本地加解密”里通过缓存 DK 以及定时更新机制还起到了提升性能、分散风险的作用。 |
| RootKey | 根密钥 | 是用来对 MK 进行加密的密钥。目前是独立存储的，后续考虑引入硬件加密模块 HSM，进一步提升安全性。 |
| EMK、EDK | MK、DK 的密文 | 在 DB 里存储 MK、DK 时，保存的是密文，而非明文，避免密钥泄漏。 |
| kms-server | 加解密服务端 |  |
| kms SDK |  | 对加解密服务和算法进行封装，简化使用。 |
| 审计日志 |  | 记录业务机器 ip 访问密钥的记录，用于做安全审计。 |

#### 3.相同的明文多次加密得到的是相同的密文吗？

不是，目前仅提供 AES\_GCM 算法，在加密时会加盐，确保相同密文多次加密得到不同密文。不加盐或使用固定的盐都会增加数据密文泄漏后被破解的风险。

#### 4.相同明文多次加密得到的是不同的密文，在数据仓库里要和其他业务的表 join 查询时怎么办呢？

类似不能满足的需求还有：根据明文在数据库里查询、需要给加密字段建立唯一索引等。

上面几个问题可以通过新增一列 token 值来解决，明文和 token 是一对一的，可以解决唯一索引、多表联查的问题。token 的生成不是通过 SHA256 这样的哈希算法生成的，可以抵抗“彩虹表”攻击。Token 服务文档：[Token 服务介绍](https://km.sankuai.com/page/127442561)。密文 + Token 也是目前业界比较常见的用法(比如亚马逊)，该方案目前可以满足大多数业务的需求。