Atitit 软件性能与数据库性能的提升attilax总结

[1. 性能的几个指标主要是响应时间（Response time） 2．吞吐量（Throughput） 4](#_Toc29491)

[2. 性能提升的5个原则与性能提升模型 5](#_Toc342)

[2.1. 分类优先级，与分区域，减少要操作或提取的内容与范围 5](#_Toc25589)

[2.2. div分而治之 聚沙成塔 分布式 并发 5](#_Toc6194)

[2.3. 大力提升单体处理能力 5](#_Toc7578)

[2.4. 异步 提升快组件与慢组件的同步模式，防止快速组件等待慢速组件 5](#_Toc17333)

[2.5. Cache法 空间换时间大法 “凡是看得见的,都是暂时的,只有看不见的,才是永远的 6](#_Toc6239)

[2.6. Hybrid 混合综合模式 6](#_Toc18337)

[3. 性能问题的检测方法与工具概览 6](#_Toc27531)

[3.1. 问题程序的全局资源占用统计 Os资源占用统计比如win的资源管理器，与linux的top等工具 6](#_Toc32362)

[3.2. 性能检测工具 Profile工具 7](#_Toc20198)

[3.3. 自制profile工具 7](#_Toc29523)

[3.4. 人工检查法 7](#_Toc28892)

[3.5. 常见性能故障排查表，根据性能异常情况，按图索骥，解决方法 8](#_Toc29317)

[3.6. 性能检测流程，防止遗漏重要的流程节点 8](#_Toc3761)

[3.7. 对常用涉及性能的api，使用团队内部api类库包装代替通用api，固化性能算法 8](#_Toc26345)

[4. 常用的性能问题标准化检测流程最佳实践bp总结 8](#_Toc26490)

[5. 编程语言的性能提升（以java项目为例子） 10](#_Toc23500)

[5.1. 性能检测Profile工具jvisualvm.exe JProfiler8 10](#_Toc10165)

[5.2. 常见的性能问题分类总结 11](#_Toc26768)

[5.3. 使用了性能较差的api或者类库，使用更快的api代替 11](#_Toc6830)

[5.4. 操作过多，使用事务模式 批量集中提交处理 12](#_Toc16119)

[5.5. 是否只使用了一个cpu，解决办法使用多线程异步模型 12](#_Toc6174)

[5.6. 多线程线程数设置不合理，导致cpu大量耗费在切换线程上。 12](#_Toc6357)

[5.7. 使用事件驱动模式，本质上属于同步倍频操作 异步 12](#_Toc6047)

[5.8. 缓存频繁使用的对象变量等。 使用对象池 cache等， 13](#_Toc26656)

[5.9. 同步锁频繁冲突，使用无锁结构 13](#_Toc24816)

[5.10. 项目编译性能变慢，改动一行代码可能需要一分钟编译，Eclipse ide增量编译是开启的 13](#_Toc1211)

[5.11. 使用了Spring框架后项目启动运行性能很慢的解决，大型项目可能需要好几分钟，十分影响进度 13](#_Toc18201)

[5.12. 改动一个东西，需要频繁启动web服务器比如tomcat，解决方案完全热部署 14](#_Toc11637)

[5.13. 其他 参考互联网xxx语言优化 14](#_Toc3566)

[6. 数据库的性能提升 14](#_Toc7690)

[6.1. 统计数据库的表记录数量，以及磁盘文件体积，评估是由于大数据量还是sql性能 数据库连接数量过多引起的。 14](#_Toc21644)

[6.2. 数据库的profile工具。找到性能较差的sql 15](#_Toc18992)

[6.3. Explain 当找到具体的sql语句了，使用Explain 来查看sql的具体过程，一般是索引没有引用到。 15](#_Toc29883)

[6.4. 索引原理 常见的类似于书籍目录，hashmap原理，散列表 dic字典 15](#_Toc3563)

[6.5. 如何设置正确的索引 16](#_Toc1624)

[6.6. 设置了索引，但是使用expian却发现，没有使用到索引 16](#_Toc26057)

[6.7. 使用存储过程大力减少数据网络往返 17](#_Toc28479)

[6.8. 移植其他数据库的性能高的特性功能 17](#_Toc5554)

[6.9. 优先操作 HIGH\_PRIORITY 滞后操作LOW\_PRIORITY 延时插入 INSERT DELAYED 17](#_Toc27463)

[6.10. 临时表模式与 强制使用临时表 SQL\_BUFFER\_RESULT 18](#_Toc7017)

[6.11. 设置全文索引代替like查询 18](#_Toc5452)

[6.12. 使用solr 等第三方全文索引代替数据库的全文索引，降低数据库的资源占用 19](#_Toc29179)

[6.13.](#_Toc347) **[使用数据库连接池和线程池](#_Toc347)** [19](#_Toc347)

[6.14.](#_Toc5539) **[使用cache redis等](#_Toc5539)** [19](#_Toc5539)

[6.15. 分区 分区是个很重要的提升性能操作 19](#_Toc14721)

[6.15.1. 分区的原理 19](#_Toc29688)

[6.15.2. 分区类型 19](#_Toc14795)

[6.15.3. 什么时候需要分区，基本一个是单表记录数量，达到几百万到一千万，或者数据表文件超过1g左右时 20](#_Toc4703)

[6.16. 分库分表 20](#_Toc11631)

[7. 多机情况下可以使用分布式，微服务msa等架构模式提升性能 20](#_Toc17673)

[8. 性能优化的一些误区 反模式与需要注意的地方 20](#_Toc32328)

[8.1. 过度设计 vs 事后优化策略 20](#_Toc25366)

[8.2. 可读性 vs 性能 vs可维护性 21](#_Toc4347)

[8.3. 性能指标响应时间 vs吞吐量 的矛盾 21](#_Toc2189)

# 性能的几个指标主要是响应时间（Response time） 2．吞吐量（Throughput）

其次还有3．资源使用率（Resource utilization） 5．并发用户数（Concurrent users） 并发连接数等。

这里的性能调整主要以缩短响应时间为主 兼吞吐量，有时他们之间是互相矛盾的。要根据实际项目需求调整

# 性能提升的5个原则与性能提升模型

## 分类优先级，与分区域，减少要操作或提取的内容与范围

## div分而治之 聚沙成塔 分布式 并发

工作多，怎么办，招募更多的人手了。 短期内，分而治之，可能是常见的解决方法。就像盖金字塔，在当时的技术条件下，聚合众多人力可能是唯一的方法。

## 大力提升单体处理能力

从长远趋势的来看，提升单体处理能力是长久解决之道。。就像盖大楼，不再使用众多人力的模式了，而是使用大型工具了。

## 异步 提升快组件与慢组件的同步模式，防止快速组件等待慢速组件

提升快组件与慢组件的同步模式，防止快速组件等待慢速组件。。比如常见的倍频同步技术，异步技术 buffer 减少不必要的状态切换和设备I/O **懒加载**

## Cache法 空间换时间大法 “凡是看得见的,都是暂时的,只有看不见的,才是永远的

所需要的结果已经预先计算好了。需要的时候，直接使用。。Cache是个很重要的性能提升模式。。

## Hybrid 混合综合模式

架构提升性能

# 性能问题的检测方法与工具概览

## 问题程序的全局资源占用统计 Os资源占用统计比如win的资源管理器，与linux的top等工具

查看 问题程序的资源占用统计监测工具，，统计资源占用情况

一般是统计出现性能问题程序的cpu占有率，线程数量，文件句柄数，内存使用率，磁盘与网络io使用率等指标。

## 性能检测工具 Profile工具

要解决问题，首先要找到问题。。工具可以大大提升准确度与检测效率。。

一般profile工具可以提供线程数量监测，内存对象占用，以及代码执行时间等监测功能。

## 自制profile工具

有时候，你使用的语言或数据库等没有提供自己的profile工具，网上也找不到第三方工具，与其他人沟通交流也得不到相关工具信息，就要自己动手，制作profile工具。。

一般调用time api，毫秒级，类似aop模式，计算每一个方法的执行时间，输出到日志或者数据库，输出到数据库更好，因为可以方便的按照执行时间排序，api调用层次等，优先调整占用时间长的代码片段方法。。

具体工具设计可以参照成熟的profile工具。。

## 人工检查法

优点是快速，很多代码片段一眼看过去就能看出所在问题。但需要一定的经验积累才可。。对技术要求较高。。其次，有些情况下，没有合适的工具或不能插入检测代码，只能人工检查。

## 常见性能故障排查表，根据性能异常情况，按图索骥，解决方法

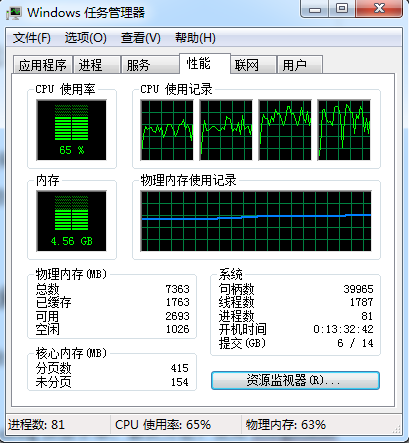
## 性能检测流程，防止遗漏重要的流程节点

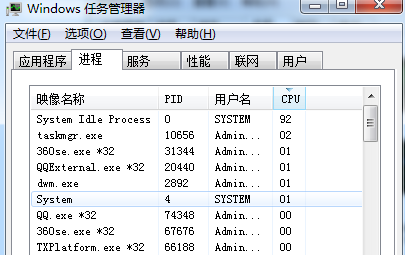
## 对常用涉及性能的api，使用团队内部api类库包装代替通用api，固化性能算法

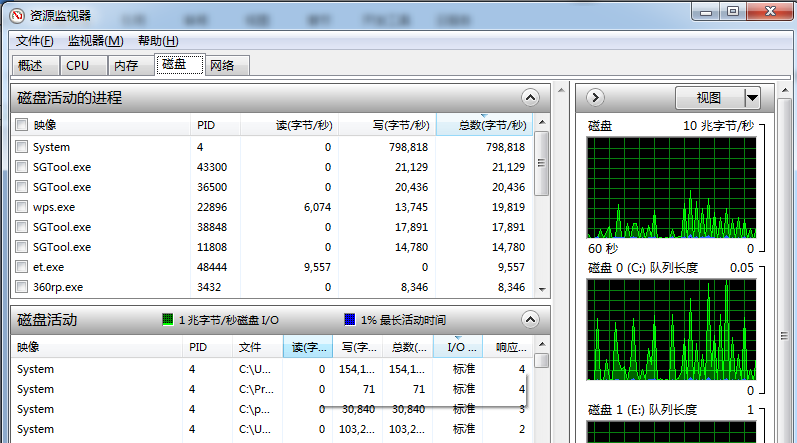
# 常用的性能问题标准化检测流程最佳实践bp总结

首先尽可能排除，os系统资源占用问题，可能是另外的程序占用了大量资源，导致本项目性能较慢。。这个一般使用win系统使用任务管理器即可。。Linux系统使用类似功能的top等性能检测工具。。。

主要检查四大资源占用，cpu，内存，磁盘io，网络io







如果确实是其他程序占用了大量资源，要么kill掉，要么就要移动到其他机器上部署了。。再次观察，如果可以了，则性能提升完成。否则就要继续下一步。

然后，使用相关语言具体领域的profile工具开始检测具体项目的资源占用。。

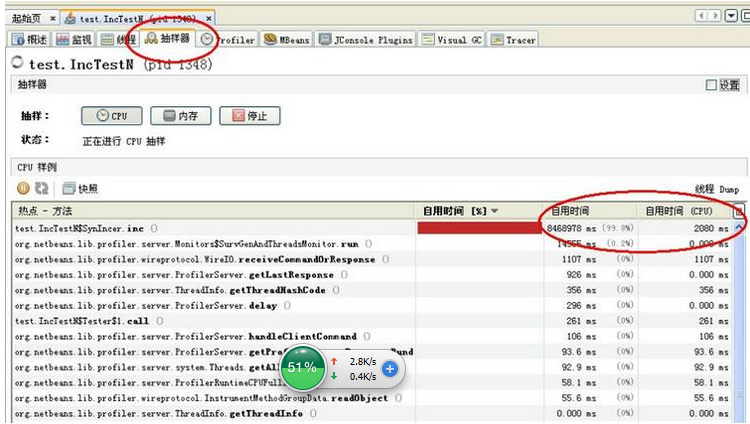
# 编程语言的性能提升（以java项目为例子）

## 性能检测Profile工具jvisualvm.exe [JProfiler8](http://www.baidu.com/link?url=CN7FBINS6nLVzgp2RUSYj50nfqrIrvBde8sPmwDxerOw97fUFYtXBYXMFgZq7yodScWrDi9YIPpD3uC_7eWmTa" \t "https://www.baidu.com/_blank)

C:\d\jdk1.8.0\_71\bin\jvisualvm.exe

Jvisualvm是jdk自带的。绝大部分场合已经够用。可以提供线程数量监测，内存对象占用，以及代码执行时间等监测。。[JProfiler8](http://www.baidu.com/link?url=CN7FBINS6nLVzgp2RUSYj50nfqrIrvBde8sPmwDxerOw97fUFYtXBYXMFgZq7yodScWrDi9YIPpD3uC_7eWmTa" \t "https://www.baidu.com/_blank)类似功能，但是更加强大

会把耗费时间长的api或方法显示出来，然后就可以根据这个提示调整代码。。其他语言一般也有相应的profile工具，功能类似

IMG_256

## 常见的性能问题分类总结

主要常见于大循环下会出常见的代码执行性能问题。。

## 使用了性能较差的api或者类库，使用更快的api代替

分词的时候，使用了string拼接字串，解决方法：使用stringbuilder代替。更快的api

循环内部多个数据库连接执行慢。。解决方法，存储过程等，减少网络io

某一个api慢,看看是否可以找到更快的类似api或者类库，比如使用更快的fastjson类库代替常用的netsf库。。

## 操作过多，使用事务模式 批量集中提交处理

循环内部使用了日志输出过多。。解决方法，批量输出，或者间隔输出日志，比如每隔多少条输出一个日志。。

## 是否只使用了一个cpu，解决办法使用多线程异步模型

如果所在语言不支持多线程，则可以使用多进程模式代替。

## 多线程线程数设置不合理，导致cpu大量耗费在切换线程上。

解决方法，首先判断操作是io密集型还是cpu密集型。。

一般cpu密集型，一般是纯算法，不涉及磁盘数据库网络io，设置的线程数是cpu核心数。当然如果其他程序占用了一些核心，则要相应的减去。。

Io密集型，则要根据io与cpu速度的倍数，一般要设置在cpu核心\*几十线程左右，具体数值根据profile工具测试来调整。

## 使用事件驱动模式，本质上属于同步倍频操作 异步

可以大大减少所需要的线程数，以便提升性能。

## 缓存频繁使用的对象变量等。 使用对象池 cache等，

## 同步锁频繁冲突，使用无锁结构

## 项目编译性能变慢，改动一行代码可能需要一分钟编译，Eclipse ide增量编译是开启的

这个问题有时频繁出现，网络上解决方法都测试无效。十分影响开发进度。后来经过不懈努力终于攻克。 Ide 的一个缺点，problem面板错误清除即可运行如飞。原因是触发了全面编译策略，导致编译性能变慢

## 使用了Spring框架后项目启动运行性能很慢的解决，大型项目可能需要好几分钟，十分影响进度

如何提升性能，使用lazy加载模式，不使用监听器模式。

此外，如果配置加载够多，即是懒加载，也会性能较慢，需要大力拆分服务配置文件，

除此之外，使用google guice等轻量级更快的ioc类库也是不错的解决方案

其次项目使用ServiceLocator模式，工厂模式也是不错的性能解决方案。。

可以达到几秒启动。提升性能几十倍。

## 改动一个东西，需要频繁启动web服务器比如tomcat，解决方案完全热部署

Tomcat jboss默认只支持很有限的热部署，和没有差不多，多年前我是使用jrebel框架实现完全热部署的，貌似效果还可。。

现在基本使用resin了，全面的支持热部署，几乎很少需要重启web服务了。

## 其他 参考互联网xxx语言优化

还有一些很细节的性能提升，就篇幅有限，暂时略过了

# 数据库的性能提升

## 统计数据库的表记录数量，以及磁盘文件体积，评估是由于大数据量还是sql性能 数据库连接数量过多引起的。

在任务管理器统计数据库进程占用的线程数，也可以使用sql命令统计。确保数据库进程的 线程数量，io，内存占用 网络io等

普通常见情况是sql性能问题，即是少量的sql语句线程就消耗了大量的cpu资源。

## 数据库的profile工具。找到性能较差的sql

此工具是查找到项目或产品的哪些sql语句会性能较差，一般会显示sql执行时间，并按照执行时间排序。

Mysql常用设置慢查询日志 ，会把执行时间长的sql记录到慢查询日志里面，一个文本日志文件。

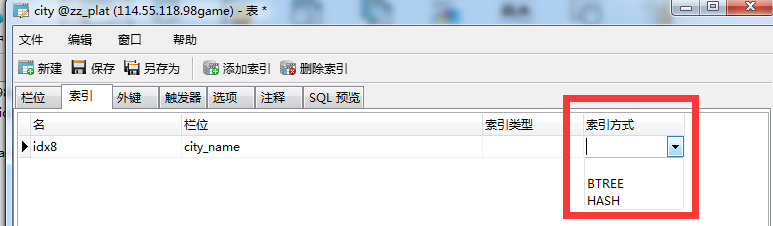
sql server profile 。。Oracle也有类似工具

## Explain 当找到具体的sql语句了，使用Explain 来查看sql的具体过程，一般是索引没有引用到。

具体的explain使用，也是一个比较内容较多的一块。。暂时略

## 索引原理 常见的类似于书籍目录，hashmap原理，散列表 dic字典

常见的索引类型有 hash btree索引，全文索引fulltext , bitmap索引等



Hash索引类似于书籍的索引，一个索引数据就是指明内容id，内容所在位置 key-value值。。

btree索引 类似于树形结构金字塔型，从大到小范围缩减。。常见的手机号，身份证号码，银行卡号码本质上就类似于一个btee的索引数据项，分节点不断缩小范围。

索引的原理也有大块内容，此处暂时略。

## 如何设置正确的索引

主要是根据比如where字段而设置相关字段的索引，有些数据库可能需要单独指明desc 还是asc顺序索引。

如果多个字段，可能需要设置多字段复合索引，有些数据库会自动走单索引，具体情况根据数据库版本与类型而有所不同，需要使用explain命令来精确判断。

这块展开来也有好多内容，此处暂略

## 设置了索引，但是使用expian却发现，没有使用到索引

这种情况下，往往是由于sql的特殊性，而导致数据库不能自动识别出需要引用的索引。。此时可以使用强制索引语法，指明要使用的索引mysql使用强制索引 FORCE INDEX 。。根据数据库类型和版本，大型数据库一般还是提供此类功能的。如果某些数据库不提供此功能，那就要自己实现强制索引功能，具体方法篇幅关系，暂略。

## 报表等复杂统计sql很慢，解决方案，业务索引

## 使用存储过程大力减少数据网络往返

多条数据表数据交互过程，可以使用存储过程大力减少数据网络往返。。注意存储过程可能会降低吞吐量，但是缩短响应时间。

## 移植其他数据库的性能高的特性功能

比如MySQL需要在表上增加一列,才能够实现类似Oracle的函数索引

或者使用编程语言来扩展数据库的一些性能特性。。

## ****优先操作 HIGH\_PRIORITY 滞后操作LOW\_PRIORITY 延时插入 INSERT DELAYED****

HIGH\_PRIORITY可以使用在select和insert操作中，让MYSQL知道，这个操作优先进行。

复制代码 代码如下:

SELECT HIGH\_PRIORITY \* FROM TABLE1;

**后操作 LOW\_PRIORITY**LOW\_PRIORITY可以使用在insert和update操作中，让mysql知道，这个操作滞后。

代码如下:

update LOW\_PRIORITY table1 set field1= where field1= …

**延时插入 INSERT DELAYED**

代码如下:

INSERT DELAYED INTO table1 set field1= …

INSERT DELAYED INTO，是客户端提交数据给MySQL，MySQL返回OK状态给客户端。而这是并不是已经将数据插入表，而是存储在内存里面等待排队。当mysql有空余时，再插入。另一个重要的好处是，来自许多客户端的插入被集中在一起，并被编写入一个块。这比执行许多独立的插入要快很多。

## ****临时表模式与 强制使用临时表 SQL\_BUFFER\_RESULT****

代码如下:

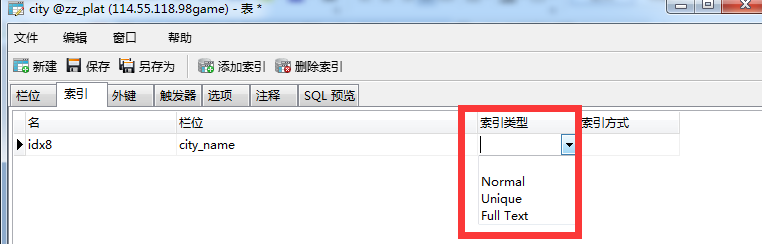
SELECT SQL\_BUFFER\_RESULT \* FROM TABLE1 WHERE …

当我们查询的结果集中的数据比较多时，可以通过SQL\_BUFFER\_RESULT.选项强制将结果集放到临时表中，这样就可以很快地释放MySQL的表锁(这样其它的SQL语句就可以对这些记录进行查询了)，并且可以长时间地为客户端提供大记录集

**分组使用临时表 SQL\_BIG\_RESULT和SQL\_SMALL\_RESULT**

## 设置全文索引代替like查询

目前的三大数据库基本都支持了全文索引，所以在较大数据量情况下，要对具体的字段做全文索引



全文索引的原理就涉及到了 倒排索引 ，Trie树,又称字典树,单词查找树或者前缀树,是一种用于快速检索的多叉树结构,如英文字母的字典树是一个26叉树,数字的字典树是一个10叉树

## 使用solr 等第三方全文索引代替数据库的全文索引，降低数据库的资源占用

普通项目下，可以优先使用数据库的全文索引，因为可读性，维护性开发效率，更高些

当数据量较大时，影响到数据库的资源占用，此时可以重构为Lucene等全文索引类库模式。

## ****使用数据库连接池和线程池****

## ****使用cache redis等****

## 分区 分区是个很重要的提升性能操作

### 分区的原理

分区的基本原理就是通过访问一个表或者索引的较小片断，而不是访问整个表和索引，以提高数据库的性能。如果将一个表的不同分区放置在不同的磁盘上，磁盘整体的吞吐量就会成倍上升

### 分区类型

范围分区：可以指定应当存储在一起的数据区间。这可能是中最常用的分区机制

散列分区：指在一个（或多个列）上应用一个散列函数，行会按这个散列值放在某个分区中

列表分区：指定一个离散值集，来确定应当存储在一起的数据。例如，可以指定STATUS列值在（’A’,’M’,’Z’ ）中的行放在分区1中，STATUS值在（ ‘D’,P’,’Q’ ）中的行放在分区2中

复合分区：是范围分区和散列分区或列表分区混合使用的一种分区方法

### 什么时候需要分区，基本一个是单表记录数量，达到几百万到一千万，或者数据表文件超过1g左右时

具体情况根据服务器与对应数据库的实际性能来决定。。大型数据库一般会比小型数据库同样多的数据下，更好的性能。。

## 分库分表

更大的数据量下就需要分库分表存储数据。。分库分表下就带来了一个跨机器节点join等联合查询的问题。。以及跨机器维护全局索引的问题。。少量机器情况下一般就可通过遍历节点即可解决。。大量机器下，就需要建立二级索引，定位数据所在的节点位置。

# 多机情况下可以使用分布式，微服务msa等架构模式提升性能

# 性能优化的一些误区 反模式与需要注意的地方

## 过度设计 vs 事后优化策略

一般来说，要根据项目的实际情况来决定是否要在设计阶段大力考虑性能问题。有些项目一开始就要在设计上考虑大性能设计，因为设计阶段考虑性能就会对项目前期进度造成一些影响。不过大部分中小型项目是事后优化策略，先解决有没有，再解决好不好，也是一种常见的策略。。

综合起来，适当的设计阶段考虑性能以及重构优化结合起来比较好，二八原则。

## 可读性 vs 性能 vs可维护性

有时候提升性能会大大降低可读性，或者可维护性。。一般要遵循首先面向人类编程，其次面向机器的原则。

软件的发展趋势，本身就是不断提升可读性，而语言性能相对不断下降的过程，语言性能的下降由硬件性能的提升所弥补。。

## 性能指标响应时间 vs吞吐量 的矛盾

性能指标包含响应时间 和吞吐量，有时他们之间是互相矛盾的。要根据实际项目需求调整

响应时间优先模式类似于我们日常生活的银行模式，一条龙办理，从业务开始到结束，不会中断，用户体验更好。

吞吐量优先模式类似于医院，挂号缴费诊断都是分离的，需要不断的跑上跑下，对于单个个体请求来说，响应时间大大延长。。但对于总体来说，吞吐量更大。