Atitit.提升稳定性之道

[1. 前言 2](#_Toc18905)

[2. 为什么会发生稳定性问题 2](#_Toc23307)

[2.1. 单点故障(单点故障率较高) 2](#_Toc7243)

[2.2. 复杂 2](#_Toc950)

[2.3. 资源耗尽 2](#_Toc28386)

[2.4. 死锁与等待 2](#_Toc8264)

[2.5. 崩溃 2](#_Toc28099)

[3. 第一章 资源耗尽 2](#_Toc28925)

[3.1. 资源占比过高 （ 2](#_Toc12864)

[3.2. 磁盘空间 磁盘空间不足,造成许多的莫名其妙的问题.也许提示连接耗尽.. 2](#_Toc9175)

[3.3. /io读写 3](#_Toc11361)

[3.4. /cpu/内存/ 3](#_Toc17032)

[3.5. 网络等占用过高） 3](#_Toc22960)

[3.6. 文件句柄数量 3](#_Toc3210)

[3.7. 线程数量 3](#_Toc15224)

[3.8. 句柄 网络链接数量 3](#_Toc1770)

[3.9. 非托管资源的释放 3](#_Toc20431)

[3.10. 病毒造成网络句柄耗尽 3](#_Toc6047)

[4. 第二章死锁与等待 3](#_Toc11302)

[4.1. 等待与超时 单点故障 单进程 单线程等 3](#_Toc28766)

[4.2. 网络死锁 3](#_Toc3386)

[4.3. 文件死锁，文件并发读写 3](#_Toc27980)

[4.4. 数据库连接死锁 3](#_Toc3986)

[4.5. 代码死锁 4](#_Toc28157)

[4.6. Cli命令行死锁 4](#_Toc7412)

[4.7. GUI界面死锁（文件格式错误，有确认框弹出等情况） 4](#_Toc9115)

[4.8. 类库冲突 4](#_Toc15987)

[4.9. 热更新热部署(业务可用性 4](#_Toc25908)

[4.10. 程序崩溃 4](#_Toc16392)

[4.11. Timeout 4](#_Toc25464)

[5. 程序崩溃 4](#_Toc14802)

[5.1. 解决方案：多进程 4](#_Toc8052)

[5.2. 进程隔离 4](#_Toc29807)

[5.3. Rest接口隔离 4](#_Toc25608)

[5.4. 守护进程 4](#_Toc2320)

[5.5. 类库冲突,造成部署问题 5](#_Toc16427)

[5.6. 热更新的支持不足,部署比较麻烦 5](#_Toc25395)

[5.7. Web服务跟数据库服务崩溃 5](#_Toc12349)

[5.8. 其他的潜在隐患: 5](#_Toc14634)

[5.9. 子线程异常造成主线程崩溃(java不影响,.net有这个问题) 5](#_Toc22260)

[5.10. 别的网络socket连接释放问题... 5](#_Toc31189)

[5.11. 直接内存读写 5](#_Toc27304)

[5.12. Stream的关闭释放. 5](#_Toc24892)

[5.13. native method调用的内存 5](#_Toc25098)

[6. #----解决方法 大方向 5](#_Toc13037)

[6.1. 杀毒 木马查杀软件 5](#_Toc910)

[6.2. 适当冗余 故障集群 5](#_Toc20548)

[6.3. 守护模式开启 守护各大资源的监控 循环守护机制 6](#_Toc21763)

[6.4. 检测机制 定时检测主要服务问候 6](#_Toc21008)

[6.5. 重试机制(包括自动重连) 6](#_Toc5437)

[6.6. 奔溃日志总结 6](#_Toc18473)

[6.7. 尽可能的减少单点并发与读写 6](#_Toc30651)

[6.8. Gc自动释放资源 与超时自动释放 6](#_Toc5878)

[6.9. 热更新热部署) 更简化的开发架构(.. 6](#_Toc23200)

[6.10. 更好用的第三方框架类库 6](#_Toc27132)

[6.11. 最佳推荐流程( 6](#_Toc12686)

[6.12. 更简化的编程语言 6](#_Toc11228)

[6.13. 监测(线程 句柄),warnning,跟自动恢复 6](#_Toc29714)

[6.14. 压力测试 6](#_Toc14205)

[6.15. 预警机制 6](#_Toc5322)

[6.16. 超时回收资源gc 6](#_Toc21288)

[7. #----解决方法总结细则 7](#_Toc28272)

[7.1. 类库冲突避免(ide,检测工具,开发时,运行时) 7](#_Toc19937)

[7.2. 引擎+脚本结构(c++,java+python,lua,php等) 7](#_Toc15971)

[7.3. 语言级的新的特性 7](#_Toc6220)

[7.4. php/.net 7](#_Toc16498)

[7.5. 建立基于提升稳定性的内部封装框架/流程文档 7](#_Toc15826)

[7.6. Finalize/Dispose 7](#_Toc24041)

[7.7. 容错(包括自动重连) 7](#_Toc2028)

[7.8. SoftReference 7](#_Toc12859)

[7.9. 连接池的配置: 自动超时回收Connection+超时自动断开conn 7](#_Toc9400)

[7.10. 语句块回收资源/using块中自动调用Dispose 8](#_Toc31160)

[7.11. 崩溃时候儿core dump并且重启 8](#_Toc13123)

[7.12. 日志，缓存等文件，尽可能按时间生成多个文件。。 8](#_Toc25390)

[7.13. 重要业务服务和页面gui监测 8](#_Toc27447)

[7.14. 监测程序(cpu,内存占用, io队列深度, 磁盘空间,数据库连接数,数据库死锁监测) 8](#_Toc17263)

[7.15. 网络，文件操作使用wrap类库secury方式调用 8](#_Toc707)

[7.16. 死锁自解除(数据库,文件等) 8](#_Toc25092)

[7.17. #----压力测试 8](#_Toc19704)

# 前言

# 为什么会发生稳定性问题

## 单点故障(单点故障率较高)

## 复杂

## 资源耗尽

## 死锁与等待

## 崩溃

# 第一章 资源耗尽

## 资源占比过高 （

## 磁盘空间 磁盘空间不足,造成许多的莫名其妙的问题.也许提示连接耗尽..

解决:添加监测程序

## /io读写

## /cpu/内存/

内存泄漏溢出

有时gc不起生效..可以调用native方法释放内存.

new memory().start();监测内存占用，当物理内存占用超过此值M时，调用SetProcessWorkingSetSize方法回收内存。

## 网络等占用过高）

## 文件句柄数量

## 线程数量

## 句柄 网络链接数量

数据库连接泄漏

连接池自动关闭连接,简化开发,,同时提升性能..

http链接泄露等

## 非托管资源的释放

托管资源交给GC就好，非托管资源则必须使用框架来自动回收 或者 亲自写代码回收

## 病毒造成网络句柄耗尽

# 第二章死锁与等待

## 等待与超时 单点故障 单进程 单线程等

## 网络死锁

## 文件死锁，文件并发读写

## 数据库连接死锁

数据库死锁

避免多个线程/请求/事务修改同一个记录..

不使用事务或者使用单语句事务

要是必须使用事务,需要调整代码.

Dbms 可以探测到死锁,但是不能自动释放死锁,需要监测程序自动解锁锁死的连接..(要是数据库被多个应用使用,要修改驱动/或者使用反射尝试,记录此应用打开的连接端口,到数据库端过滤,在执行解锁)

## 代码死锁

多线程并发读写死锁

压力测试解决.

## Cli命令行死锁

使用timeout机制解决。。其次要读取std,err,

## GUI界面死锁（文件格式错误，有确认框弹出等情况）

## 类库冲突

## 热更新热部署(业务可用性

## 程序崩溃

## Timeout

# 程序崩溃

## 解决方案：多进程

## 进程隔离

## Rest接口隔离

## 守护进程

#-----影响稳定性的因素

## 类库冲突,造成部署问题

需要工具检测

## 热更新的支持不足,部署比较麻烦

Classloader?? Resin glassfish等web服务器检测...jboss支持有限的热部署.

## Web服务跟数据库服务崩溃

数据库服务启用服务监测,自动恢复..Web服务单个的进程,需要寻找个监测程序或者安装为服务.

## 其他的潜在隐患:

## 子线程异常造成主线程崩溃(java不影响,.net有这个问题)

抛出线程，线程体内要TRY CATCH。。否则抛出EXP导至主程序OUT。。特别重要，一定要做.

## 别的网络socket连接释放问题...

## 直接内存读写

## Stream的关闭释放.

## native method调用的内存

finalize()中可以用本地方法来调用它。以释放这些“特殊”的内存空间。

# #----解决方法 大方向

## 杀毒 木马查杀软件

## 适当冗余 故障集群

资源耗尽 解决方案

## 守护模式开启 守护各大资源的监控 循环守护机制

## 检测机制 定时检测主要服务问候

## 重试机制(包括自动重连)

## 奔溃日志总结

## 尽可能的减少单点并发与读写

## Gc自动释放资源 与超时自动释放

避免死锁跟解除)

自动资源释放池

## 热更新热部署) 更简化的开发架构(..

## 更好用的第三方框架类库

提升稳定性的内部封装框架/类库

## 最佳推荐流程(

## 更简化的编程语言

## 监测(线程 句柄),warnning,跟自动恢复

## 压力测试

## 预警机制

## 超时回收资源gc

需要建立框架,比较简单的超时自动回收资源.可以解决大部分问题...使用code template配合ide自动import 自定义类库代替系统类库.

# #----解决方法总结细则

## 类库冲突避免(ide,检测工具,开发时,运行时)

## 引擎+脚本结构(c++,java+python,lua,php等)

## 语言级的新的特性

## php/.net

Php的自动释放资源做的非常好,几乎所有的的问题都解决了...同级的脚本语言ruby几乎和php同时起步,python更是早好几年,,最终市场php应用最广泛(c系列的语言风格也很重要,跟c++,java 一脉相承)...ruby/python解决了热更新跟[类库冲突](#_Toc10285),但是好像都没解决自动释放资源的问题.

Java 也可以使用Quercus类库内嵌python/Php/js,内嵌方式能不能自动释放资源还没有检验

.net也解决了部分稳定性问题.(主要是热更新跟[类库冲突](#_Toc10285),但是没解决资源自动释放的问题) ,不过ide vs的强大大大提升了2倍以上的开发效率.

## 建立基于提升稳定性的内部封装框架/流程文档

全面代替系统默认库和常使用第三方库,从框架级角度解决一些问题,,会损失一点儿性能跟灵活性..需要的时候儿也能直接使用系统库...

建立api文档已便查看..

## Finalize/Dispose

finalize()的主要用途是释放一些其他做法(non--new法)开辟的内存空间，以及做一些清理工作

使用code template配合ide自动生成Finalize框架方法

## 容错(包括自动重连)

## SoftReference

java .lang.ref 包，其中定义了三种引用类。这三种引用类分别为SoftReference、 WeakReference和

## 连接池的配置: 自动超时回收Connection+超时自动断开conn

c3p0.checkoutTimeout=10000

c3p0.unreturnedConnectionTimeout=25

c3p0.maxConnectionAge=20

## 语句块回收资源/using块中自动调用Dispose

## 崩溃时候儿core dump并且重启

Java的调用oom自动恢复脚本..

PRPGRAM。CS内要TRY CATCH，发现主程序出问题，重启。

PROGRAME。CS内增加UnhandledException 的捕获..

## 日志，缓存等文件，尽可能按时间生成多个文件。。

可以防止万一个哪个文件句柄没被释放，也不会影响后面的文件写入。

## 重要业务服务和页面gui监测

可以及时发现服务out service

## 监测程序(cpu,内存占用, io队列深度, 磁盘空间,数据库连接数,数据库死锁监测)

提前发现不稳定性因素...

## 网络，文件操作使用wrap类库secury方式调用

默认的sdk库使用一定要TRYCATCH。

## 死锁自解除(数据库,文件等)

## #----压力测试

当前项目虽然并发不大(当前200左右,默认的配置可支持5000左右)...

但是压力测试可以提前测试出稳定性方面的问题..

常常使用工具jmeter,*LoadRunner等*

作者 老哇的爪子 Attilax 艾龙， EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax