Atitit.软件开发提升稳定性总结

[1. 第一章 资源耗尽 3](#_Toc32012)

[1.1. 资源占比过高 （ 3](#_Toc24187)

[1.2. 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.. 3](#_Toc25979)

[1.3. /io读写 3](#_Toc11714)

[1.4. /cpu/内存/ 3](#_Toc24458)

[1.5. 网络等占用过高） 3](#_Toc774)

[1.6. 文件句柄数量 3](#_Toc9221)

[1.7. 线程数量 3](#_Toc15588)

[1.8. 网络链接数量 3](#_Toc25639)

[1.9. 非托管资源的释放 4](#_Toc10347)

[2. 死锁与等待 4](#_Toc15969)

[2.1. 等待与超时 单点故障 单进程 单线程等 4](#_Toc24938)

[2.2. 网络死锁 4](#_Toc25086)

[2.3. 文件死锁，文件并发读写 4](#_Toc6634)

[2.4. 数据库连接死锁 4](#_Toc8483)

[2.5. 代码死锁 4](#_Toc10928)

[2.6. Cli命令行死锁 4](#_Toc27599)

[2.7. GUI界面死锁（文件格式错误，有确认框弹出等情况） 5](#_Toc7371)

[2.8. 类库冲突 5](#_Toc17340)

[2.9. 热更新热部署(业务可用性 5](#_Toc7276)

[2.10. 程序崩溃 5](#_Toc30793)

[2.11. Timeout 5](#_Toc1944)

[3. 程序崩溃 5](#_Toc13029)

[3.1. 解决方案：多进程 5](#_Toc8067)

[3.2. 进程隔离 5](#_Toc24457)

[3.3. Rest接口隔离 5](#_Toc22921)

[3.4. 守护进程 5](#_Toc6721)

[3.5. 类库冲突,造成部署问题 5](#_Toc29985)

[3.6. 热更新的支持不足,部署比较麻烦 5](#_Toc20958)

[3.7. Web服务跟数据库服务崩溃 6](#_Toc26176)

[3.8. 其他的潜在隐患: 6](#_Toc4413)

[3.9. 子线程异常造成主线程崩溃(java不影响,.net有这个问题) 6](#_Toc24584)

[3.10. 别的网络socket连接释放问题... 6](#_Toc11215)

[3.11. 直接内存读写 6](#_Toc27062)

[3.12. Stream的关闭释放. 6](#_Toc12092)

[3.13. native method调用的内存 6](#_Toc30760)

[4. #----解决方法 大方向 6](#_Toc19295)

[4.1. 奔溃日志 6](#_Toc6262)

[4.2. 守护模式开启 7](#_Toc10993)

[4.3. Gc自动释放资源 与超时自动释放 7](#_Toc23664)

[4.4. 热更新热部署) 更简化的开发架构(.. 7](#_Toc19087)

[4.5. 更好用的第三方框架类库 7](#_Toc9)

[4.6. 最佳推荐流程( 7](#_Toc2315)

[4.7. 更简化的编程语言 7](#_Toc14053)

[4.8. 监测,warnning,跟自动恢复 7](#_Toc17189)

[4.9. 压力测试 7](#_Toc19377)

[4.10. 重试机制(包括自动重连) 7](#_Toc7704)

[4.11. 故障集群 7](#_Toc18161)

[4.12. 预警机制 与检测机制 7](#_Toc32684)

[4.13. 超时回收资源gc 7](#_Toc16730)

[5. #----解决方法总结细则 8](#_Toc26531)

[5.1. 类库冲突避免(ide,检测工具,开发时,运行时) 8](#_Toc26109)

[5.2. 引擎+脚本结构(c++,java+python,lua,php等) 8](#_Toc15804)

[5.3. 语言级的新的特性 8](#_Toc2765)

[5.4. php/.net 8](#_Toc160)

[5.5. 建立基于提升稳定性的内部封装框架/流程文档 8](#_Toc14914)

[5.6. Finalize/Dispose 8](#_Toc7975)

[5.7. 容错(包括自动重连) 9](#_Toc13579)

[5.8. SoftReference 9](#_Toc3035)

[5.9. 连接池的配置: 自动超时回收Connection+超时自动断开conn 9](#_Toc23140)

[5.10. 语句块回收资源/using块中自动调用Dispose 9](#_Toc24713)

[5.11. 崩溃时候儿core dump并且重启 9](#_Toc30923)

[5.12. 日志，缓存等文件，尽可能按时间生成多个文件。。 9](#_Toc28703)

[5.13. 重要业务服务和页面gui监测 10](#_Toc18759)

[5.14. 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) 10](#_Toc16914)

[5.15. 网络，文件操作使用wrap类库secury方式调用 10](#_Toc17884)

[5.16. 死锁自解除(数据库,文件等) 10](#_Toc16175)

[5.17. #----压力测试 10](#_Toc6033)

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

# 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

## 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

# 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

## 奔溃日志

## 守护模式开启

## 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,warnning,跟自动恢复

## 压力测试

## 重试机制(包括自动重连)

## 故障集群

资源耗尽 解决方案

## 预警机制 与检测机制

## 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*