atitti 提升稳定性的艺术之程序代码级别稳定性的艺术 attialx著 艾提拉著 s420.docx

[1. 前言 1](#_Toc18319)

[2. 为什么会发生稳定性问题 1](#_Toc6506)

[2.1. 单点故障(单点故障率较高) 1](#_Toc1029)

[2.2. 复杂 1](#_Toc2017)

[2.3. 资源耗尽 1](#_Toc17588)

[2.4. 死锁与等待 1](#_Toc15034)

[2.5. 崩溃 1](#_Toc537)

[3. 常见措施 2](#_Toc25300)

[3.1. Atitit 提升稳定性 错误处理 全局错误捕获 2](#_Toc17039)

[3.2. Trycatch error not ony ex 2](#_Toc27212)

[3.3. Try catch throwable解决异常 2](#_Toc30780)

[3.4. 保证数据正确性，增加key效验等 2](#_Toc28965)

[3.5. 线程池连接校验等 借出连接测试有效性 2](#_Toc18344)

[3.6. 注意null的处理 null忽略 2](#_Toc17433)

[3.7. 增加定时释放资源机制防止资源占用过多Gc自动释放资源 2](#_Toc17355)

[3.8. 增加future timeout机制，防止死锁 阻塞 2](#_Toc18130)

[3.9. Treadlocal机制隔离死锁 2](#_Toc21464)

[3.10. 进程隔离 2](#_Toc21798)

[3.11. 检测机制 定时检测主要服务问候 与状态切换 2](#_Toc22542)

[3.12. 重试机制 如果conn无效，在线程池娶一个 2](#_Toc31503)

[3.13. 更好用的第三方框架类库 3](#_Toc3664)

[3.14. 奔溃日志总结 wanging 3](#_Toc8558)

[3.15. 心跳机制 3](#_Toc31839)

[3.16. 按照稳定性最佳实践流程走， 3](#_Toc32221)

[3.17. Futuretask get 线程池超时时候，可能线程池堵塞，使用备用模式直接thrad执行 3](#_Toc5342)

[3.18. 线程池最起码设置为2，防止一个万一堵塞就麻烦 3](#_Toc17948)

[3.19. 看门狗技术随时恢复正确状态 3](#_Toc13711)

[4. 稳定性测试 3](#_Toc27614)

[4.1. Throw ex测试 3](#_Toc20588)

[4.2. 断开连接测试 3](#_Toc3070)

[4.3. 多线程测试并发 3](#_Toc24197)

[5. Atitit.提升稳定性之道 3](#_Toc21608)

# 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

# 常见措施

## [Java8新的异步编程方式 CompletableFuture -](https://www.baidu.com/link?url=5z3wArynTm-8FqOvSP4LFtzj8rAMdnOUz5h07QEsAK6oHpdflH0V2W6GlH_obeQLOLy-QlNkB_j_itkVGGR0_dzBXery8QH-sN7BdZIKPiW&wd=&eqid=9b9a1c7200005a42000000035b14a2c1" \t "https://www.baidu.com/_blank)

## Atitit 提升稳定性 错误处理 全局错误捕获

## Trycatch error not ony ex

Jonjye not import jar miss not take effect

## Try catch throwable解决异常

## 保证数据正确性，增加key效验等

## 线程池连接校验等 借出连接测试有效性

## 注意null的处理 null忽略

## 增加定时释放资源机制防止资源占用过多Gc自动释放资源

## 增加future timeout机制，防止死锁 阻塞

## Treadlocal机制隔离死锁

## 进程隔离

## 检测机制 定时检测主要服务问候 与状态切换

## 重试机制 如果conn无效，在线程池娶一个

## 更好用的第三方框架类库

## 奔溃日志总结 wanging

## 心跳机制

## 按照稳定性最佳实践流程走，

## Futuretask get 线程池超时时候，可能线程池堵塞，使用备用模式直接thrad执行

## 线程池最起码设置为2，防止一个万一堵塞就麻烦

## 看门狗技术随时恢复正确状态

# 稳定性测试

## Throw ex测试

## 断开连接测试

## 多线程测试并发

# Atitit.提升稳定性之道

1. 前言 2

2. 为什么会发生稳定性问题 2

2.1. 单点故障(单点故障率较高) 2

2.2. 复杂 2

2.3. 资源耗尽 2

2.4. 死锁与等待 2

2.5. 崩溃 2

3. 常见措施 3

3.1. Try catch throwable解决异常 3

4. 第一章 资源耗尽 3

4.1. 资源占比过高 （ 3

4.2. 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

4.3. /io读写 3

4.4. /cpu/内存/ 3

4.5. 网络等占用过高） 3

4.6. 文件句柄数量 3

4.7. 线程数量 3

4.8. 句柄 网络链接数量 3

4.9. 非托管资源的释放 3

4.10. 病毒造成网络句柄耗尽 3

5. 第二章死锁与等待 4

5.1. 等待与超时 单点故障 单进程 单线程等 4

5.2. 网络死锁 4

5.3. 文件死锁，文件并发读写 4

5.4. 数据库连接死锁 4

5.5. 代码死锁 4

5.6. Cli命令行死锁 4

5.7. GUI界面死锁（文件格式错误，有确认框弹出等情况） 4

5.8. 类库冲突 5

5.9. 热更新热部署(业务可用性 5

5.10. 程序崩溃 5

5.11. Timeout 5

6. 程序崩溃 5

6.1. 解决方案：多进程 5

6.2. 进程隔离 5

6.3. Rest接口隔离 5

6.4. 守护进程 5

6.5. 类库冲突,造成部署问题 5

6.6. 热更新的支持不足,部署比较麻烦 5

6.7. Web服务跟数据库服务崩溃 5

6.8. 其他的潜在隐患: 5

6.9. 子线程异常造成主线程崩溃(java不影响,.net有这个问题) 5

6.10. 别的网络socket连接释放问题... 6

6.11. 直接内存读写 6

6.12. Stream的关闭释放. 6

6.13. native method调用的内存 6

7. #----解决方法 大方向 6

7.1. 杀毒 木马查杀软件 6

7.2. 适当冗余 故障集群 6

7.3. 守护模式开启 守护各大资源的监控 循环守护机制 6

7.4. 检测机制 定时检测主要服务问候 6

7.5. 重试机制(包括自动重连) 6

7.6. 奔溃日志总结 6

7.7. 尽可能的减少单点并发与读写 6

7.8. Gc自动释放资源 与超时自动释放 6

7.9. 热更新热部署) 更简化的开发架构(.. 7

7.10. 更好用的第三方框架类库 7

7.11. 最佳推荐流程( 7

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