如何优化你的代码？

编写高性能的Java代码着重注意的四个部分：

- 并发
- 通信
- 数据库操作
- JVM

并发

Unable to create new native thread ……

问题1：Java中创建一个线程消耗多少内存?

每个线程有独自的栈内存，共享堆内存

问题2：一台机器可以创建多少线程?

CPU，内存，操作系统，JVM，应用服务器

结论：不要new Thread()，采用线程池

	非线程池	线程池
100次	16ms	5ms
1000次	90ms	28ms
10000次	1329ms	164ms

线程池

CreditEase
2006-2017
科技让金融更美好

使用线程池要注意的问题

- 死锁 请尽量使用CAS
- ThreadLocal

  ThreadLocalMap使用ThreadLocal的弱引用作为key，如果一个ThreadLocal没有外部强引用来引用它，那么系统 GC 的时候，这个ThreadLocal势必会被回收，这样一来，ThreadLocalMap中就会出现key为null的Entry，就没有办法访问这些key为null的Entry的value，如果当前线程再迟迟不结束的话，这些key为null的Entry的value就会一直存在一条强引用链：Thread Ref -> Thread -> ThreaLocalMap -> Entry -> value永远无法回收，造成内存泄漏。

- 线程之间的交互

线程不安全造成的问题

- 经典的HashMap死循环造成CPU100%问题
- 应用停滞的死锁，Spring3.1的deadlock 问题

基于JUC的优化示例

CreditEase
2006-2017
科技让金融更美好

一个计数器的优化

Synchronized

ReentrantLock

Atomic

通信

- 数据库连接池的高效问题
  - 一定要在finally中close连接
  - 一定要在finally中release连接

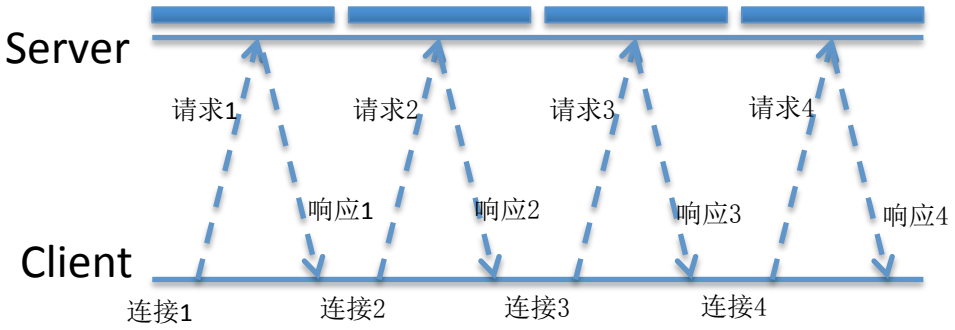
- OIO/NIO/AIO

	OIO	NIO	AIO
类型	阻塞	非阻塞	非阻塞
使用难度	简单	复杂	复杂
可靠性	差	高	高
吞吐量	低	高	高

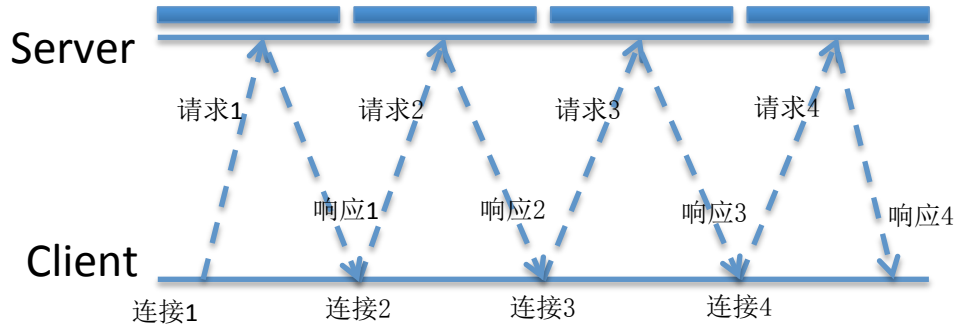
通信

- 串行连接，持久连接（长连接），管道化连接

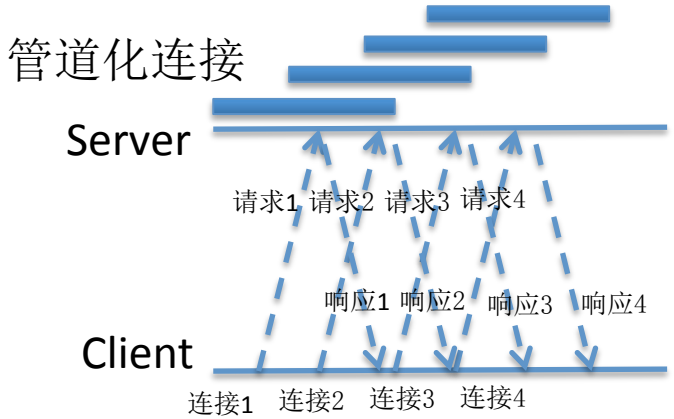
串行连接



持久连接



管道化连接



结论：

管道连接的性能最优异，持久化是在串行连接的基础上减少了打开/关闭连接的时间。

管道化连接使用限制：

1，HTTP客户端无法确认持久化（一般是服务器到服务器，非终端使用）；

2，响应信息顺序必须与请求信息顺序一致；

3，必须支持幂等操作才可以使用管道化连接.

通信

- TIME_WAIT(client)，CLOSE_WAIT(server)问题

  反应：经常性的请求失败

  获取连接情况 netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'

  - TIME_WAIT：表示主动关闭，优化系统内核参数可。
  - CLOSE_WAIT：表示被动关闭。
  - ESTABLISHED：表示正在通信

  解决方案：二阶段完成后强制关闭

数据库操作

- 必须要有索引（特别注意按时间查询）
- 单条操作or批量操作

```
private static String insertSql = "INSERT INTO skytrain_queue_message_info_history "
        +
"(queue_name,un_consume_message_num,publish_message_num,deliver_message_nu
m,worktime) "
        + "VALUES (?,?,?,?,?)";
PreparedStatement psts = getConnection().prepareStatement(insertSql);
for (QueueInfo queueInfo : queueInfos) {
   psts.setString(1, queueInfo.getQueueName());
   psts.setInt(2, queueInfo.getUnConsumeMessageNum());
   psts.setInt(3, queueInfo.getPublishMessageNum());
   psts.setInt(4, queueInfo.getDeliverMessageNum());
   psts.setTimestamp(5, ts);
   getStatement().execute(sql.toString());
}
```

```
private static String insertSql = "INSERT INTO skytrain_queue_message_info_history "
        +
"(queue_name,un_consume_message_num,publish_message_num,deliver_message
_num,worktime) "
        + "VALUES (?,?,?,?,?)";
getConnection().setAutoCommit(false);
PreparedStatement psts = getConnection().prepareStatement(insertSql);
for (QueueInfo queueInfo : queueInfos) {
   psts.setString(1, queueInfo.getQueueName());
   psts.setInt(2, queueInfo.getUnConsumeMessageNum());
   psts.setInt(3, queueInfo.getPublishMessageNum());
   psts.setInt(4, queueInfo.getDeliverMessageNum());
   psts.setTimestamp(5, ts);
   psts.addBatch();
}
psts.executeBatch();
getConnection().commit();
getConnection().setAutoCommit(true);
```

- CPU标高的一般处理步骤

1. top查找出哪个进程消耗的cpu高
2. top －H －p查找出哪个线程消耗的cpu高
3. 记录消耗cpu最高的几个线程
4. printf %x 进行pid的进制转换
5. jstack记录进程的堆栈信息
6. 找出消耗cpu最高的线程信息

- 内存标高（OOM）一般处理步骤

1. jstat命令查看FGC发生的次数和消耗的时间，次数越多，耗时越长说明存在问题；
2. 连续查看jmap –heap 查看老生代的占用情况，变化越大说明程序存在问题；
3. 使用连续的jmap –histo:live 命令导出文件，比对加载对象的差异，差异部分一般是发生问题的地方。

JVM

- GC引起的单核标高
- 常见SY标高
  - 线程上下文切换频繁
  - 线程太多
  - 锁竞争激烈
  - Linux2.6高精度定时器
- Iowait标高

抖动问题

原因：字节码转为机器码需要占用CPU时间片，大量的CPU在执行字节码时，导致CPU长期处于高位；

现象：“C2 CompilerThread1” daemon，“C2 CompilerThread0” daemon CPU占用率最高；

解决办法：保证编译线程的CPU占比。