www.qconferences.com
www.qconbeijing.com

QCon北京2014大会 4月17—19日

伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山

London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

QCon全球软件开发大会

International Software Development Conference

www.infoq.com/cn

InfoQ

@InfoQ

infoqchina

软件
正在改变世界！

InfoQueue
Enterprise Software Development Community

特别感谢
QCon上海合作伙伴

并发编程的思想变革

陈超强

HRM@4399.COM

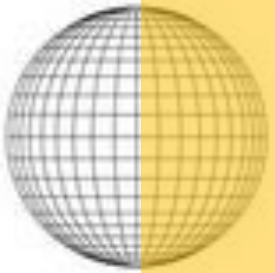
1 游戏并发编程的现状

2 WEB和游戏技术的对比

3 并发编程的总结和设计思路

4 并发编程与跨服设计

并发是趋势

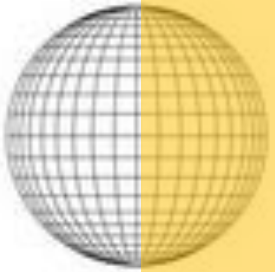


单个CPU的速度提升缓慢

多核时代到来

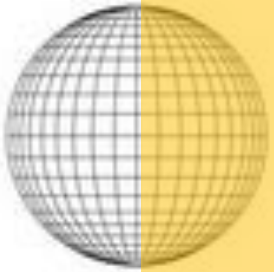
其实大家都知道，但是游戏开发上并没有太大的转变

并发发展障碍



- C++写的单进程游戏服务器也能撑很多人
- 进程的创建、通信、销毁、管理麻烦
- 数据原子性问题非常复杂
- 功能的快速迭代，无暇深入研究
- 游戏开发中没有对人数上限做硬性的要求

游戏并发设计案例



- 网关进程
- 数据库保存进程
- 寻路、AOI可视判断进程
- 聊天进程
- 场景进程

现有网游，有很多并发游戏服务器架构的尝试

但并没有固定的模式，发展相对缓慢

难以通过加机器来提高单服承载量

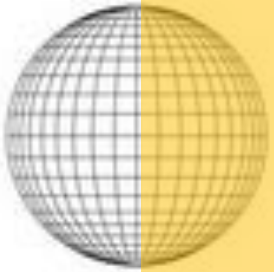
1 游戏并发编程的现状

2 WEB和游戏技术的对比

3 并发编程的总结和设计思路

4 并发编程与跨服设计

WEB负载均衡

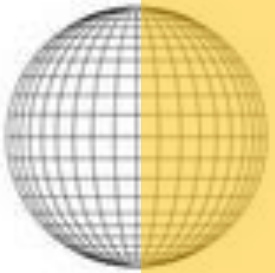


- DNS轮询
- 反向代理
- 各种缓存：MEMCACHE、REDIS
- 数据库与负载均衡
- IP层的负载均衡LVS
- 其他

能支撑百万数据量级的访问

加机器即可提高单个站点的承载量

WEB服务逻辑特点

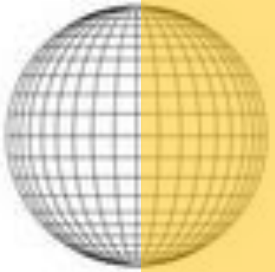


- 大部分逻辑是往客户端推送数据
- 业务基本上只跟单个用户有关
- 用户间的交互较少，或者交互逻辑简单
- 1个用户1个进程，增加机器即可提升承载

1. 思考

其实大量的游戏逻辑都只跟单个玩家有关，包括网络、数据库、心跳、任务、背包、各类养成玩法等等

思考



思考 2.

假如1个玩家1个进程，加机器数即可提高上述功能的承载（类似于数据库横向切分的优化思路）

3. 思考

交互类强的系统需要重新设计，比如聊天、战斗、交易、组队、帮会等

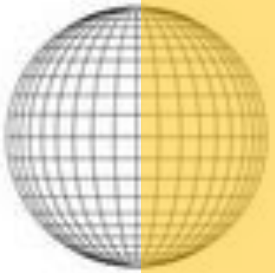
1 游戏并发编程的现状

2 WEB和游戏技术的对比

3 并发编程的总结和设计思路

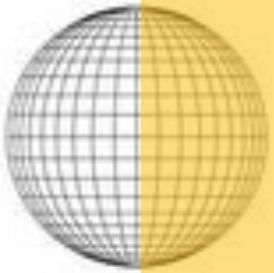
4 并发编程与跨服设计

释放你的并发思路



- 建立进程创建、管理方便的机制
- 一套简易的可遵循的并发设计思路

进程管理机制

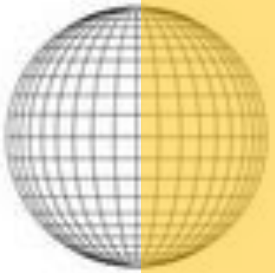


语言级别的进程管理机制（以ERLANG为例）

1. 轻进程.
2. 创建进程：SPAWN(FUNCTION)
3. 进程间的通信
   - ✓ 发送消息：PID!{DATA}
   - ✓ 接受消息：RECEIVE

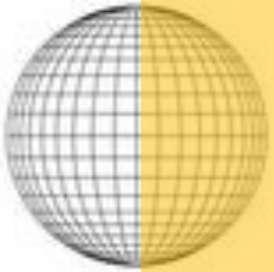
在语言层面减少进程创建、管理的工作（或者自己写库）

并发设计思路



- 单进程 与 多进程编程环境的区别
- 多进程的限制

单、多进程编程环境的区别



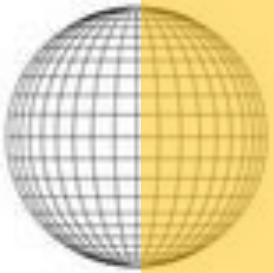
单进程

所有数据都在同一个进程空间里，所有代码都跑在同一个进程里

多进程

任何变量都可以放到任何进程，任何代码都可以在任何进程上跑。排列组合非常多

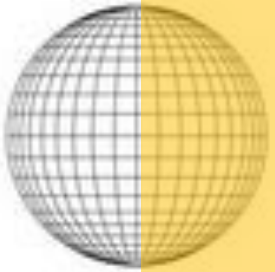
多进程的限制



多进程的限制

1. 数据的原子性。数据放到哪个进程、代码在哪个进程跑受这个限制
2. 数据操作可以分为两类
   - ✓ 一次性数据操作
   - ✓ 非一次性数据操作

引子

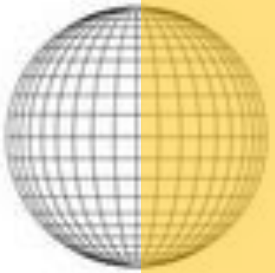


复印 10000 张开会资料

单进程做法：1台复印机复印10000张

多进程做法：1台复印机复印N张，
再分别给N台复印机复印

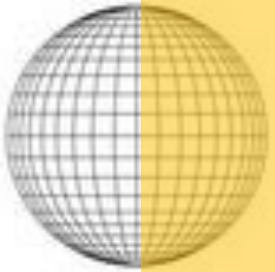
一次性数据操作



一次性的数据操作

1. 像打印这种逻辑，是不受原始数据变化而改变。这类操作有
   - ✓ 纯写操作
   - ✓ 纯读操作
2. 没有原子性的问题，可以开进程去做
   - ✓ 例子：足球比赛在球场上进行，通过广播，成千上万的观众在电视前看。（实际上场景同步属于这类操作）

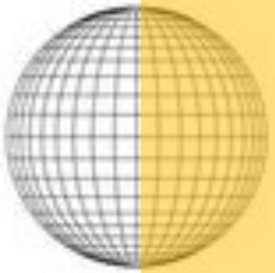
游戏中的例子



- SOCKET数据的收发
- 数据库的保存
- 聊天
- 地图信息的同步
- LOG输出

这些功能都属于一次性数据操作，类似于WEB的数据下载业务，可以放心的开多个进程去做

非一次性数据



数据既读又写的情况

对这个数据的操作，只交给一个进程，保证原子性

再细一点，可以分为以下操作

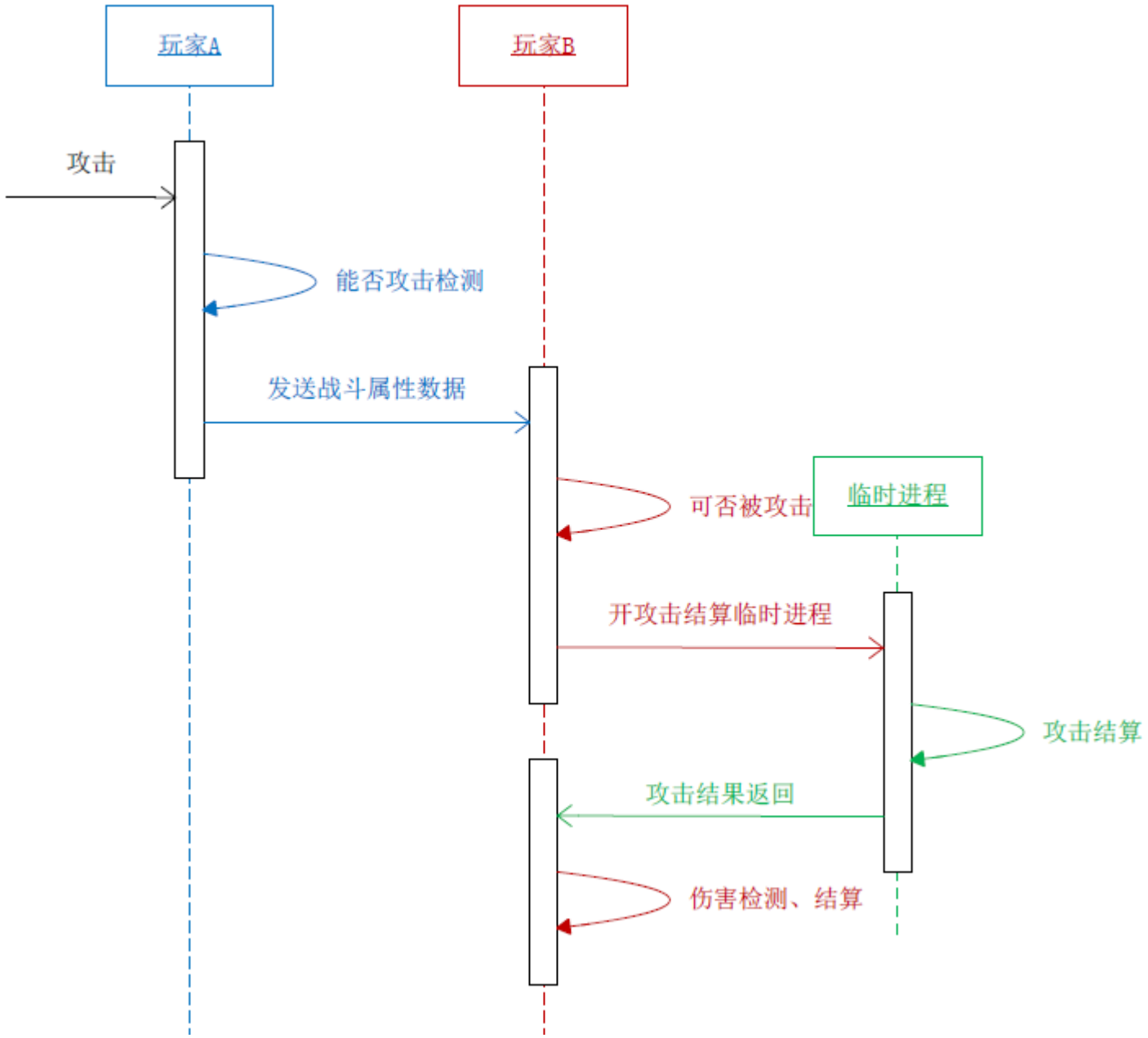
CHECK，CALC，SET（检查、计算、修改）

可以优化为

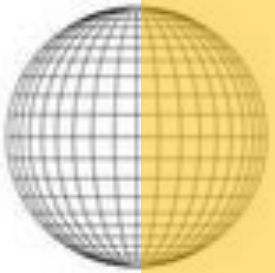
CHECK，SET放在一个进程，CALC可以放到别的进程进行

例子

HP


玩家A
玩家B
攻击
能否攻击检测
发送战斗属性数据
可否被攻击
临时进程
开攻击结算临时进程
攻击结算
攻击结果返回
伤害检测、结算

总结



1. 对所有业务流程进行梳理，确认每个业务对某个变量的操作类型（读写）

2. 确定数据放到哪个进程，CHECK和SET都交由同一个进程去做，CALC可以考虑新开进程去

3. 确定哪些业务对数据是一次性的操作，可以新开进程去处理这些业务

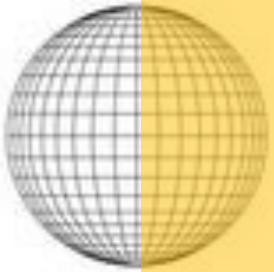
1 游戏并发编程的现状

2 WEB和游戏技术的对比

3 并发编程的总结和设计思路

4 并发编程与跨服设计

并发思路与跨服设计



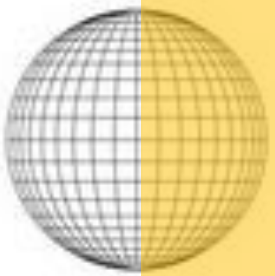
并发思路

任何变量都可以放到任何进程，任何代码都可以在任何进程上跑

跨服玩法

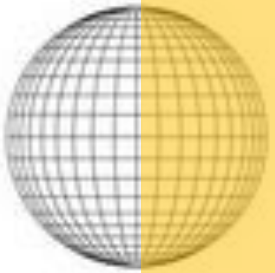
大部分的游戏功能跟跨服没有关系，实际上涉及到的功能只有：战斗同步相关、聊天、组队等等，以及跨服玩法本身。用并发的思路进行修改即可

页游的跨服设计案例



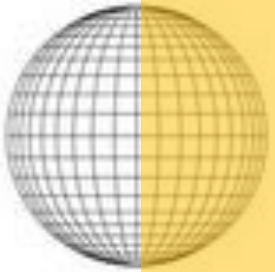
- 客户端始终和同一个服务进程保持着通信，SOCKET一直保持着连接
- 跨服玩法中，可以跟本服玩家聊天、发邮件、交易，跟没跨服的情况一致
- 一个玩家的大部分数据始终存在于同一个进程（机器），不存在着数据同步的操作
- 再加强跨服玩家之间的交互并不困难

补充



- 端游、页游、手游的服务器架构都是类似的
- 客户端也可以考虑，只是可用的CPU少很多

网络稳定性的影响



单台机器

单台机器多个核，进程间的通信基本不会失败，基本都会成功

多台机器

多台机器，进程间的通信可能失败的概率较高，编程时考虑的异常情况变得复杂

- ✓ 避免
- ✓ 要求IDC提供更好的服务，保证网络不断

THANK YOU!

谢谢聆听，欢迎交流