**技术准备**

能够做这个MMO的触发点是通过某些途径得到了某个大公司使用的一款3D引擎，其他的都是白手起家。当时大家还不知道有“分布式服务器端”一说，服 务器端框架参考了《剑3》：剑3内测的时候经常服务器crash，但是每次只crash一个地图，所以可以推知他们是一个地图一个server；加上自己 对服务器端的认识，需要Gate当防火墙，需要GameServer来总管MapServer，需要DB来存储，那么最初的服务器端框架就定下来 了：Gate、GameServer、MapServer、DBServer。想让服务器之间的连接方式最简化，所以确定GameServer是中心，其 他Server都连接并且只连接GameServer。MapServer和GameServer上面准备加脚本，脚本直接选择了python，因为 python语法清晰一点。开发平台选择windows，因为当时公司内没有一个人了解linux。

**策划定调**

做一个3D版的征途，这个就是最终确定的整个项目的方向。为什么这个方向？因为：1.征途火，大家都想学。2.征途的开发技术难度相对较低。3.当时很多运营游戏的公司的产品线上都没有类似产品。

**项目启动**

启动时team中只有5人。一共有3件事：1. 服务器端先行开发，服务器端3个月的计划是构建所有服务器并且封包走通。2.研究引擎并且改造引擎，确保自己会使用，同时没有抄袭痕迹。3.开始策划招 聘，对这个策划的定位很明确：要能够扛得起旗帜，有领导力，有大局观，最好是征途公司的。

服务器端这边的目标是达到每秒30万个封包的效率，封包大小在30byte到50byte。30万这个数据来自于估算：服务器端300人互相可见的情况下，每人每秒动3次，那么就是300\*300\*3，差不多30万封包了。

**服务器端第一次重构**

服务端这边开始是2人开发，我做大局，另一个做网络层，一个月后封包效率仍然上不去，于是大家一起检查代码，发现网络层的封包pool结构不是用 list存储，而是用map，还发现很多其他常识性问题，于是决定把他调离服务器端，去做场景编辑器。于是从此我开始了一个人的服务器端的旅程，花了1个 星期的时间完成重写他的代码，然后继续前进。

**优化经验**

这个项目的优化难度不太高，源于2点：1.整个项目设计都遵循一条指导原则：能不用第三方库就不用，所以优化过程中不需要理解第三方库的用法。2. 封包定义的结构是struct，定义封包的方式是先写伪代码，然后再用根据根据伪代码生成每个封包的struct和ReadBuffer、 GetBuffer函数。一般封包都是用memcpy赋值，只有动态封包，比如封包中如果不得不使用sting，才会通过比较慢的方式读取，这样的话在封 包赋值、转移操作上面没有碰到任何性能问题。

基于以上的定义，实际上只有2条优化经验：1.读取封包的时候，一次要尽可能多的读取封包，不要先read一个封包长度，再根据长度read实际内 容。这种方式会让性能大打折扣。2.发送封包的时候，需要稍微拼接一下再发送，可能会造成一定的封包延时，但是效率会上升至少10倍。

**策划启动**

理想的主策过来了，是以前征途的策划，纯市场导向。过来后没有急着开始招策划，而是先风格定调，这个过程就像RPG游戏里面给一个角色配点一样，总 共属性点是有限的，我们必须强调哪些东西而舍弃哪些东西。在这个游戏中，强调的是PK，舍弃的是娱乐性、PVE。我们坚决不做魔兽世界，这个是当时的口头 禅。事实证明这句话是对的，魔兽世界不是谁都可以做出来的。最终确定了游戏背景是战国时间，以秦始皇为蓝本。

世界观确定的时间大概有1个月，之后就开始招聘执行策划了。

**美术启动**

主美也过来了，是老牌工作室的人。主美过来的同时就开始招聘美术，这个时候，对于美术各个职位，待遇最高的是原画。为了激励大家的士气，老板公开定了3个月一次MileStone，每个MileStone拿一次奖金的制度，所有人都很开心。

**项目危机**

现在有几个问题：

问题1：内部不协调。程序是公司自己培养起来的，是公司的主动力，而普通策划和普通美术都是新来的，但是一来工资却比程序高很多，因为公司开的工资 在业内是偏低的，要招新人，必须按照业内标准来招才能找到人。也就是说新来的策划美术都是业内标准工资，而程序拿的是很低的工资。

问题2：过来的人的工资超过了老板的预期，资金准备不够了。于是老板不得不让出部分股权寻找天使投资人。

**建立基本开发流程**

主策招聘策划的标准很明晰：1.不做魔兽世界。2.要有血性。所以进来的策划全是热血的新手。主策对招人的点抓得很不错，对一个前途未卜的公司来讲，最需要的是热情。

“策划主导开发”，这个是征途的思想，也带到我们这里。功能策划负责整体功能的实现，服务器、客户端沟通，程序加班策划也要加班，功能失败也是先找 策划。当时程序还在打基础，策划有大量的时间，但是也没闲着，主策让他们写功能文档，越详细越好，经常几个策划无聊在比谁的功能文档写的全。

**服务器端第二次重构**

重构源于分布式服务器端，碰到问题：如果保证玩家在所有服务器登陆的一致性？也就是说：如何不卡号。一开始的登陆流程是：1.玩家登陆Gate。 2.登陆GameServer，GameServer读取DB数据。3.GameServer通知MapServer玩家登陆并且把DB数据给 MapServer。

我们的客户端测试工具是我自己写的模拟机器人，模拟器没有读条过程，所有很容易暴露问题，我们发现了以下问题：

问题1.因为读DB需要时间，MapServer会在没有收到GameServer玩家登陆消息的时候，收到玩家发来的移动消息。当玩家频繁重登的时候，这个玩家甚至有可能收到他上次登录发来的封包。

问题2：切换地图逻辑异常麻烦。切换地图逻辑需要做的事情：1.询问对方地图是否接受。2.传送到对方地图。3.完成传送，改变玩家在GameServer上面注册的MapID。这个逻辑是异步的，任何环节都要处理异常，所有封包都走GameServer。

因为所有封包都会通过GameServer，GameServer负责了太多的状态变量导致逻辑很不清晰，不把逻辑梳理清晰，肯定会引发重大问题。

于是有了第二次重构，重构的技术改动是：

改动1. GameServer拆分为RouterServer和GameServer，所有服务器都会连接RouterServer，GameServer只负责世界逻辑。

改动2. RouterServer增加协层的概念，目的是处理像登陆或者切换地图这种需要多台服务器互动的关系，把切换地图做成“事物”，其他服务器再也不用关系切换地图的异常处理问题。

因为增加了一台Server，服务器改动比较大，前后大概有2个星期的时间成型。

**美术部关系**

主美不是一个领导力很强的人，在安逸的环境中不会暴露他的问题，但是在公司现在这种生死未卜的环境中，他不能给他的美术以信心和力量，导致美术部集 体抗议项目奖金发得过慢的问题，最终导致集体“请假”。对此，公司老板非常火大也非常无耐，主策和老板泡了一会茶，大概意思是说：要忍耐，要有信仰。对于 主美，这是能力问题，不是他的错，必须帮他扛起这块地，而现在来讲，只要美术都没辞职，就将就用吧。所以在项目开发日夜颠倒的灰暗岁月中，美术是从来不加 班的。程序每每都是在美术部正常下班的时候，双眼发绿光地盯着离去的人发呆，因为程序还将继续战斗8个小时左右。

这一段开发是一段黑暗的日子，对程序的整体认识都被翻了一遍，公司的很多库函数也因为无法满足性能需求而优化重写，作为技术长进来讲，的确是一段脱胎换骨的日子，作为回忆来讲，只会感受到痛苦。

现在程序基础基本OK，可以开始大规模开发。此时客户端还停留在引擎阶段，没有真正的做游戏，服务器端一直用自己做的模拟器测试。

**第二MileStone开始**

程序的情况渐渐明晰，策划，美术都已到位。预计第一MileStone3个月搞定，实际上用了5个月。第二MileStone的任务很简单：做到10级。其中包括：客户端启动开发，制作一个地图，主线任务，怪物AI，AOI，寻路，物品，商人，玩家和玩家交易。

当时开发流程中没有项目管理软件，开发流程很简单：贴条。策划自己给自己贴条，然后给程序贴条。客户端程序由于刚刚启动，前途未卜，还无法真正开发，所以开发程序也就只是指我一个人。后来大家觉得因为只有一个程序，贴条好像没啥意义，所以统一直接找我口头沟通了。

**客户端开始，状态机的悲剧**

客户端程序也是一个人，老板对客户端程序的评价是：不撞南墙不回头。所以当老板做了错误的决定的时候，客户端程序也拼命冲上去做了。这个错误是：使 用状态机来管理主角状态。当时的状态有接近30个，比如：待机、走路、跳跃准备、跳跃中、落下准备、攻击待机、攻击中、攻击关键帧等等……，用了一个自己 做的工具来生成状态机代码，客户端程序在不断地考虑状态转换关系，而且每新增一个动作，就需要考虑这个动作和所有其他动作的转换关系。客户端程序本身就不 是一个逻辑很清晰的人，于是发生了一次又一次的悲剧，在黑白颠倒的工作压力下，客户端程序已经完全崩溃，会出一些非常低级的问题。有一次客户端程序随意的 给enum定义变量赋值而且赋值重复了，结果导致查了一通宵的问题。还有一些关联性问题：比如我们跳跃的格子数是2格，为什么每次玩家都会跳4格？一个通 宵后找到问题：因为美术做跳跃动画的时候，向前移了2格。

最后，终于客户端程序主动辞职。由制作场景编辑器的人来继续客户端的开发，事实上项目没有因为客户端程序的离职导致太大问题。

**策划数值定调**

这个游戏的面向用户不会很高端，所以一定不能做得太麻烦，于是定死了几条规则：1.做攻减防体系。2.不做任何强控制技。攻减防体系是说计算伤害的 时候，直接用攻击方的攻击力减去防御方的防御力得到总伤害，而不是像魔兽体系一样，防御做成百分比。后来的经验证明攻减防体系是不对的，因为数值会飘，比 如一个100点攻击力的人攻击防御50点的人，伤害是50点；但是加了双倍伤害之后，攻击方变成200点攻击力，对防御50的人造成的伤害是150点，总 伤害翻了3倍，所以一般pk的时候，谁先造成致命谁赢。强控制技是说击晕等让对方完全无法控制的技能，不做强控制技是为了防止若干个免费玩家通过操作能成 功玩死一个终极装备玩家。

**python框架定调**

采用的方法是数据在c++中，用swig做转换接口转换给python用，每一个封包到达会调用相应的python函数，python从c++的临 时数据中取出当前封包并且处理。整个项目都是基于这个在做下去，但是很后来的时候性能测试才发现，最消耗的地方不是在pyhton运行层，而是在 python和c++接口层，因为数据全部在c++中，python大量使用c++接口，导致整体性能低下，至今为止，这个问题还无法完美解决。

**任务框架制定**

制定任务框架的时候参考了2个源代码：一个是天堂II服务器端源码，一个是征途泄露出来的源码，它们都有一个特点：任务系统使用xml体系来描述。 天堂II的任务是html的，可以直接用浏览器打开并且能够看到对话和链接；征途的简化一点，是xml描述的。这个发现给了人很大的启发：任务系统和做网 页本质上是一样的。于是我们的任务系统也决定这么做，最终写起来的感觉让人觉得在做asp页面。

任务体系最终的形态是：一个任务一个目录，目录下面是事件目录，事件目录里面是以NPCID为文件名的xml处理，比如如果玩家有ID为1的任务， 需要和NPCID为100的NPC对话才能触发事件，那么目录结构是：目录（名称1）下面有子目录（名称on\_npc），下面有文件（100.xml）， 这么做的好处是：如果以后有某个任务坏了导致会有死链或者刷任务，我们可以直接删除任务这个目录，不会影响其他的任务的开发。

**python程序招聘和整个系统的重新设计**

我们从游戏开发培训学校招了2个学员来试用，虽然这2个人当时没有对开发产生多大的推动作用，但是引入了多人开发这个概念，整个框架定调都和以前完 全不同，这对整个项目有很积极的作用。比如以前我们写一个技能，考虑的实现方法是尽量重用，尽量使代码简化并且对编码者易维护，但是这样的副作用是过于抽 象会让代码变得难懂，只用开发者自己才能够维护，而且很难找到某个功能的负责人，只能由我全权负责。后来的技能写法转变为一个技能一个python文件， 如果某个技能出问题了，也可以很轻易的找到功能负责人，更好的是，如果一个技能有bug，直接删除这个技能脚本即可，不会对整个系统有任何负面影响。上面 说的任务框架，技能系统，物品系统全部是根据一个原则来设计：出了错误可以马上被替换。这个思想的转变给人的感觉是打war3对战和打星际对战的区别，一 个是英雄战，讲究个人风范，一个是群战，讲究分兵布阵。

**客户端python化**

服务器python化的经验证明，开发速度上面，python速度比c++速度快4倍，并且写出来的东西清晰易懂，所以这个经验也用到了客户端上 面，一开始是用在做界面上，客户端的界面框架是自己实现的，要增加一个界面，首先需要用c++ builder画好界面，然后把界面代码dfm文件放到客户端中，客户端解析这个界面文件，然后找到相应的python代码来处理逻辑，客户端大的思想是 一个界面一个py，并且各个界面存储各个界面自己的数据，相互独立。这么做的好处是功能划分清晰，坏处是对于同一个数据刷新，不同的界面如果想取得同样的 数据的话，需要2份一样的读取代码，比如如果有2个组队信息界面，每个界面都需要单独读取组队信息封包，管理自己的组队数据。但是，总的说来，当时的这个 选择是好处大于坏处

**MileStone3**

目标是做到40级，并且做副本系统。就是说一个MapServer可能会跑多个副本，每个函数的处理都需要加上GameWorld参数，标识自己在 哪个副本中，当时的做法很简单：多个副本在同一个线程中跑，每跑一个副本的时候，都会改变current\_game\_world变量，在逻辑处理的时候， 通过current\_game\_world可以得到当前的游戏世界。这么做的好处是所有逻辑都不用改，坏处是跨服逻辑的处理：其他服务器无法知道这个玩家 在哪个副本中，所以只好统一在0号副本处理跨服逻辑，这个变成了一个潜规则，而且每个开发者都会因为这个潜规则而糊涂一阵。

**python程序继续招聘**

python程序已经开始大量招聘，都是像那一家游戏培训学校要人，工资开得都很低，但是这些人都发挥了很重要的作用，他们特点总结说来：1.听 话。2.热情。笔者始终认为这2个特点是一个合格新手的必备条件。后来在这些python程序中成长起来一批人，至今笔者还非常认同，不比任何一个大学的 毕业生差。

至此，项目各项流程都已经初步确定，工作可以有序展开，下一步是最重要的：推向市场。

**寻找买家**

为了找到好的买家，公司上了一次CJ展会，果真发现了不少商机，有很多大公司都提出意向，但是开出的条件非常苛刻，不过老板还是去周边走了一趟，顺 便见了见国内的各个大小开发商。比较有印象的是3家公司：盛大，约18号去面谈，最终谈失败了，因为还不想加入18计划；腾讯，因为有个巨大的平台，所以 几乎没有开出什么利润条件；巨人，这个公司对我们很有帮助，产品总监过来看了一下，直说我们的产品还很次，不够档次。于是我们郁闷了很久，最终下决定美术 工作重翻。

在美术工作重翻的时候，我们也放弃了在大陆能够找到好买家的期望，因为毕竟是一个很新的公司，没人了解。于是找到了台湾的10年“合作”伙伴：台湾网龙，网龙那边很爽快就答应了，但是要求任务系统重翻一次。于是生意一拍即合，我们开始繁体化界面了。

**大陆买家**

在确认了台湾网龙会代理后，在大陆这边找代理商就容易多了，看来各个代理商都是在等待对方先吃螃蟹。最终大陆确认由光宇代理，对公司来讲，终于，拖了1年多的奖金总算发下来了一点点。

**大陆封测**

光宇的地推很给力，封测总共5组服务器，他们保证每组服务器都满载6000人在跑，但是进来玩的人似乎不都是玩家，整个封测过程大家都是挂机自动打怪，没有任何社会活动在里面，也没出任何问题，封测非常顺利。

**大陆内测**

封测的顺利让光宇决定：内测直接开商城。大家都知道，现在的游戏实际上已经没有内测封测的说法了，开商城就意味着游戏正式运营。我们公司对这件事很 没底，服务器没有故障过更觉得没底，但是来不及思考，内测开始了，当时的感受只能用一句很流行的话来说：Hell, It's about time。

内测总共出的问题列举如下：

串号：一个玩家错误的登陆到另外一个玩家的号上面去了，是因为我们没有用sessionID作为封包标识符。

卡号：通常是因为MapServer报错，导致玩家在Router上面一直是登陆状态，后来在Router上面加了ping包踢人机制，但是仍然还有这个问题，因为当router判定需要踢人的时候，玩家在MapServer上出错就无法下线，否则回档。

回档：都是因为服务器出错而导致的，解决方法是每个MapServer每2分钟会把本地图的所有玩家数据写到本地存储文件。每次开发的时候，如果发现本地存储文件有玩家数据信息，则把玩家数据写回数据库。

覆档：这个是上面的解决方案导致的，如果2个地图都有同一个玩家的本地缓存数据，那么可能发生旧的玩家数据把新的玩家数据覆盖的情况。解决方法是每个玩家进入地图的时候，都必须触发一次本地缓存，这样玩家数据可以保证唯一。

清档：运营商那边要求：为了防止运营商内部人员修改游戏数据，数据库中的数据都必须做一个哈希标识，玩家登陆的时候，如果根据数据中的数据计算的哈 希标识和已经保存的哈希标识符不匹配，则出错处理。在出错处理中，如果玩家身上的物品哈希标识不对，那么就直接删除此物品。这个是一个非常严重的失误，某 次游戏更新，我们给物品表新增了新字段，但是忘记了哈希错误处理这回事，结果放到公网上，每个登陆的玩家的物品都被清空，服务器开了半个小时立即全部关 服，但是已经有4000多个玩家的数据遭到损害。更悲惨的是，上周运营商那边忘记了做数据备份，所以我们的解决方式是：1.根据上上周的数据备份恢复丢掉 装备的玩家数据。2.根据2周的系统日志来补偿玩家物品。3.给出相应补偿。整个过程持续了28个小时，提供了8个版本的恢复工具给运营商，最后回到家的 时候，我连开门的钥匙扣都找不到在哪里。

刷钱：当时有个bug：比如一个玩家的账号名是account，他用账号名Account和账号名account能够登入同时到router，也就 是说，router判定重登的时候没有区分账号的大小写，导致的问题是玩家可以同时登陆2个同样的角色到选人界面，Account账号进入游戏，把钱全部 给小号，下线，然后account账号再下线，这样后下线的账号保存数据会盖掉前面登陆账号的保存数据，达到刷钱的目的。这个bug的发现过程非常复杂， 很多情况下只能靠联想去思考，运营商那边一直怀疑自己有内鬼，当发现最后的真相的时候，我们都很汗颜。

很难登陆：我们物品表的组织结构是一个物品一条记录，导致一组服务器运营半年后，物品表有一千多万条记录，玩家登陆一次，读取30个物品需要的时间 竟然需要30秒。经过排查发现，物品表的索引字段定得过多导致读取写入物品都非常慢，去掉了索引字段后，问题缓解。不久之后问题继续，因为数据量实在太大 了，所以我们对物品表做了一个系统性的改动：用二进制形式存储玩家的所有物品，一个玩家一行记录，这样才根本的解决了读取过慢的问题，但是副作用是我们不 能随意的查询玩家物品数据。

每日任务重复触发：因为当时每日任务触发机制做在MapServer上，而各个MapServer硬件的时间有些微妙的不一样，所以会发生这样的情 况：一个玩家在Map1上面走道00:00分触发每日任务，然后切换地图去Map2，Map2的时间是11:59分，所以Map2走到00:00分的时 候，又会再次触发每日任务。这个问题的解决方法是：1.玩家数据中记录下来当前触发每日任务的日期，用作防止重复触发判定。2.服务器整体时间由 GameServer来同步，MapServer禁止取得本地时间，防止再次发生类似悲剧。

客户端本地文件损坏：这个bug也隐藏得非常深，因为客户端用内存文件映射打开本地文件的模块没有用只读方式打开，所以客户端某一个野指针可能会写 到本地文件然后保存，结果是客户端本地文件莫名其妙不能用了，只能完全重新下载重装。排查这种bug和推理小说中的侦探做事方法差不多，每次查到问题，总 会有一种自虐的成就感。程序员最大的敌人是自己。

整个测试过程中，因为我们没有提供很好的游戏查询方式给运营商，所以比如说要找到一个服务器花钱最多的人，都是由我在公网数据库中执行select操作实现的，我可以直接访问运营极，并且好几次是由我来公网开服，客服很多问题也是由我来解答，整个运营体系极其混乱。

**台湾内测**

台湾在内测前提供以下几个文档：log接口和log格式，充值接口。log接口总共定了200多种日志类型，规则很清晰，并且导致我们对所有代码都 翻了一遍才加上完备的log。对接成功之后，整个游戏上线了。我们这边没有服务器的访问权限，但是整个运营过程几乎没出什么问题，台湾那边很少打电话过 来，只是看到他们的电视广告一直打，整个游戏很火，据台湾说同时在线最高有3万人。唯一记得的一个bug是：一次我们改动了物品拖动逻辑，导致在某些情况 下会复制物品。台湾因这个bug损失很大，他们重整了自己的很多QC流程，后来才知道，台湾网龙的代理流程是把游戏版本拿过来，先QC测试，QC保证这个 产品的质量，如果出问题，第一责任人是QC。

台湾的成功来源于2点：1.大陆先上线，很多雷被踩过了。2.制度规范，接口规范，运营商很清晰，不会有恶性循环。

**台湾计费**

虽然台湾产品很火，老板却不太高兴，因为充值计费全部是走的运营商接口，并且玩家在不登陆游戏的时候也可以给游戏充值，所以我们这边完全无法知道玩家确切的充值数据，也就是说，运营分成上面完全没有主动权。

**大陆在线人数跳水**

这个是预料之中的，因为程序这边的crash已经让所有人都崩溃。

**人数稳步下滑**

在这个时候，玩家对这款游戏的新意已经过去，没有玩点，粘稠度再强、再强调社交都没有用，我自己来讲，也是每日任务做完就下线了，pvp只是有钱人 的玩意，而pve的boss又必须花钱去买才能杀，对于不喜欢用钱玩游戏的人，几乎无法生存，没有了羊的世界，狼也慢慢消亡了。这是我个人总结的经验。

**小结**

至此，一款网游完整的生死已经结束，通过这次开发得到了不少经验也成熟了不少，同时身体也垮了不少。总的说来，“我做成了一件事情”这句话还是挺有成就感的。