**后端优化方案**

**优化一：**

.经过三天的压力测试，结合测试结果，发现服务器主要瓶颈在场景代理广播压力上，看了一下代码，发现不少地方是这样实现的

圈红的方法是往名为ets\_online的ets表中读取数据，当并发起来时候此处会出现两处瓶颈：

1. 当几十个进程同时操作一个ets时会出现严重锁竞争，严重影响效率
2. 该接口取出来的是完整的玩家记录，玩家记录是游戏服务器中体积最大的“结构体”，但广播逻辑只需要用到玩家的socket进程，这样不合理的操作大大降低了广播的性能

**优化方案：**

1.因为服务器的广播是以场景为单位，所以每次的广播只需要操作对应场景的ets就可以了

2.当逻辑只涉及的广播的时候，可以考虑用额外的地方存放玩家的通信进程，单单一个进程标识只需要几个字节，但一个玩家记录需要上千字节

3.有并发需求的话，可以打开ets的并发读写开关，让ets更好的支持读写

**优化二：**

Gen\_server的call方法是游戏服务器的大忌，假如有50个进程同时call一个进程，那这50个进程就会同时堵塞，如果对call的超时处理不当还可能引发服务器的逻辑错误。我在走读怪物模块的代码时发现如下代码

怪物状态机在每次处理逻辑时，都要call场景进程获取怪物实例，这种做法会造成极大的消耗，如果场景的怪物数量多起来就会严重堵塞每个怪物状态机以及场景进程

**优化方案：**

可以模仿玩家进程，进程自己维护自己的玩家状态，代码如下

这样就可以显示避免每次都call了，这对场景进程的压力也能在一定程度上缓解

**优化三：**

在eprof的测试报告中，网络io也是服务器的瓶颈之一，erlang的send方法平均每次占用8个单位的cpu时间去执行，这个消耗还是不少的。

**优化方案：**

之前与技术中心的同事交流过，那边提出了粘包的概念，具体就是让当发送消息时，服务端不立马发送，而是等缓冲区的消息满200个字节或者等待200毫秒时才统一发送，这样可以减少服务器的io消耗。具体的做法我记得端游一组的一位姓萧的同事都提过，他们也在自己的项目中使用以上框架，但他们的io框架是基于c++ 的libevent的，虽然语言不同，但方法是可以借鉴的

**优化四：**

****发现代码中有多处的ets:match或者select操作，个人建议非必要的情况下尽量用ets:lookup代替，原因有二：

1. Ets只会在主键上添加索引，match/select系列方法如果搜索的条件不含有主键的话，erlang vm将遍历ets的每一个元素，一个个去比较查找，
2. match/select经测试性能上是不如lookup的

**优化方案：**

1. 尽量使用lookup代替match/select

**优化五:**

项目中大部分的ets插入操作都使用ets:insert，但Ets的insert方法涉及到记录的复制，效率不高，而ets:updateelement方法会在ets的内存中直接修改数据,不涉及数据复制问题

**优化方案：**

条件允许的话使用ets:updateelement代替ets:insert