基于《琅琊榜》游戏框架需求及问题汇总（技术向）

基于琅琊榜项目需求，总结出来了一些目前的不靠谱、不合理、可优化的做法，希望可以在以后的项目中避免、完善

1.更新逻辑

问题：1.更新一点资源就需要下载完整的mainconfig.xml文件

2.文本资源、脚本代码加密后不能被压缩

3.ui改一点贴图就要整张更新

4.服务器端动更已经在琅琊榜中使用,目前没发现由这方面引起的问题，是否有隐藏的问题？

方案：1.资源、配置文件做diff，修改现有打包流程

2.需调研或修改打包流程

3.跟新的UI制作流程配套实施

4.需要继续完善，确定使用条件，如：配置资源、不涉及到数据的纯逻辑代码等

2.开服-服务器列表系统

问题：

1.目前的开服流程、配置繁琐，容易出错，出过一次事故，现在只是在现有流程中加了校验，整套流程需要重做

2.目前公司的区分系统，服务器列表维护更新时操作太多，太累，冗余信息太多

方案：web一键开新服，随后直接转入服务器列表系统

3.活动系统

问题：

1.需要前后端打表更新，不及时

方案：

1.中小活动动态配置、大活动再提前规划素材的前提下也可配置

2.能不能改成一个web系统，前后端都去web动态读取活动数据

4.合理数据监控、异常预警、排查

问题：

1. 遇到bug或者作弊都是有人反馈时才被动发现，这时候往往已经影响很大
2. 客户端崩溃堆栈无法获取
3. 服务器端有日志收集，但无配套的分析系统

方案：

1.由策划确定各种货币在各个系统的比较合理的产出，超出后报到预警系统里人工判定

2.前后端错误日志，必要的话可以接入第三方的错误收集sdk,如客户端崩溃分析：https://www.crittercism.com/

5.多平台sdk的接入、管理

问题：多接一个大的平台就搞个分支，无论版本管理还是接入都比较繁琐

方案：对底层sdk进行封装，通过枚举各种平台调用各自平台接口和资源

1. 整个游戏的代码包括引擎、资源、脚本和这个底层壳，根据不同平台加载对应的sdk资源，
2. 动态更新和整包强制更新分离，动更走统一一套cdn资源，需跟打包流程配套完成

6.查数据源的问题

问题：各种补偿、回滚处理时出现纰漏

方案：具体情况不同，暂无对策

7.打点系统

问题：1.太繁琐，甚至造成服务器端的压力，

2.需求信息量太大，每个点都需要带上海量的冗余数据，这些数据明明早已经打过了，只有数据搜集，基本没什么分析

方案：1.客户端能打的点，不要交给服务器端打

1. 参考dataeye，游戏数据属保密内容，由BI负责，目前不是项目组能控制的

8.各种编辑器

问题：资源编辑器、技能编辑器、战场脚本编辑器、地图编辑器等使用过程中不便

方案：需要提出完整、固定需求，重新架构（需大量时间做和维护）

9.聊天

问题:带宽压力大，出现恶意刷频时甚至可能阻塞正常游戏通讯

方案:单拆出聊天服务器，与游戏服务器剥离

------------------------以下是其他同事提出的比较具体的一些问题或分享------------------------

1. 频繁推翻重做且开发时间严重不足，导致上线前根本没有时间做各种优化，尤其是内存方面的优化，导致游戏前期IOS的内存问题
2. 组件问题，引擎比较老，侵入式修改严重且功能冗余，一些基本组件被重复实现
3. 活动条件考虑不周全，且各部门缺乏足够的沟通，缺少项目经理的角色
4. 着急修改bug，测试不够充分就紧急上线，导致事故
5. 压力测试方面的一些做法可以在日后用上，比如聊天和打点会造成比较大的压力
6. DB入库需要考虑到特殊字符的情况
7. 内存泄露查证不方便，需完善，只要是lua代买层面的泄露
8. 新手引导都存服务器的话可能会出现各种通讯问题，导致引导终止，能否先把步骤记到本地
9. 平台接入时ui、gl线程容易出现问题，需留意
10. 测试机覆盖不够充分，注意分析testin结果
11. 内存泄露的查找方式（xcode/贴图/lua/java）需要出专题普及
12. 良好的编程习惯，命名规范、注释清楚
13. 应用宝的各项处理方式不同，尤其是支付和消耗相关方面，需留意
14. 代码、思路要review，自己review、相互review、隔天review等各种体位的review
15. 消息对照表的策略是否需要加上UUID？是否需要动态组合通讯数据？是否需要重新设计？
16. 复杂逻辑运算、大量素材加载时候做分帧、延时、异步处理，可保证流畅度
17. 功能需求明显有问题时，需不断确认并留出修改的接口
18. 战斗服务器端防作弊一定要做