###### 大型PC端UI可能遇到的风险和问题

1. UI界面表现丰富，加载打开较慢
2. 同时打开的UI较多，显示层级管理复杂
3. 多个UI长时间打开后GPU内存占用过高导致表现卡顿
4. UI表现层与逻辑层高耦合
5. UI语言本地化工作量较大
6. UI资源较多，如果图集分类不合理会导致draw call较高
7. 某些界面数据较多时需要实时更新及刷新界面，可能导致卡顿
8. 界面较多，逻辑间可能耦合程度高，会导致实现热插拔困难
9. 界面较多，会有很多重复的组件，应制作通用组件
10. 分辨率较多，需要注意屏幕适配
11. 太多界面有时需要自行控制渲染顺序
12. 可能存在UI样式实时替换的需求
13. 热更新支持
14. 常用界面的生命周期管理及缓存管理
15. 抗锯齿
16. 项目过大，需要一些工具辅助，实现半自动化UI工作流
17. 日志系统的支持
18. 尽量使美术、策划、程序的工作可同时进行，并在最后可以方便的合并

###### 解决方案

1. **UI界面表现丰富，加载打开较慢**

* 我认为UI框架和资源加载的关系是非常紧密的，要解决加载打开较慢的问题，首先需要从资源管理开始
* 在制定资源规范时应定好，通用资源不应重复制作，能用九宫格的图片就制作成九宫格，资源的尺寸能否为2的n次方，压缩格式应该选什么等
* 图集划分的粒度问题，哪些资源应当打到一起，按什么粒度划分等，部分界面可以考虑动态图集，同时应当使用工具定期对项目内的资源进行冗余剔除
* 在制作UI时应当遵守一定规范，如通用组件的复用，组件间的层级管理等，并且需要考虑动静分离
* 有些不复杂的动效可以用代码实现
* 在执行资源加载时如有必要，需要对UI资源进行异步/同步加载的区别处理
* 一些较大资源是否可以考虑使用mini map，用空间换时间
* 当界面上有持续性的动效，可以考虑制作为单独的特效，使其脱离UI层，也可考虑缓存池
* 复杂的界面应该拆成多个界面，在点击不同tab页时再去动态加载界面，避免一开始就把所有资源加载出来，耗时过长
* 框架应控制单帧内加载的UI数量，做好分帧加载
* 引入界面缓存，减少复杂界面重复加载销毁的操作

1. **同时打开的UI较多，显示层级管理复杂**

* 在框架层面应该先划分好大致的窗口层级，先把所有窗口分成几个大类，一般为：常驻界面、普通窗口、弹窗、提示、警告，优先级依次递加。在开发不同界面时需要自行对应声明该UI所属的层级，以方便框架对其进行管理
* 管理类应维护一个栈去管理打开的UI界面显示层级，先按优先级决定界面的展示，同优先级的界面根据打开的时间先后决定展示顺序。这样做也可以方便后续拓展，比如已经打开的UI，但用户看不到，可以将其表现层生命周期暂停，提高性能，或是处理子父界面的交互问题等

1. **多个UI长时间打开后GPU内存占用过高导致表现卡顿**

* 这个问题在第2点里提到了一些解决方案，在框架设计时应该考虑到每个界面的生命周期管理，及整个UI的界面管理
* 框架设计时应将逻辑层和表现层分开，将耦合度降到一个合适的程度，以支持后续当逻辑和表现其中一个停掉后，另一个可以正常运行
* 当某个界面用户看不到时，应当停掉它的表现层，只保留逻辑层接收数据，当此界面被重新激活时，再以当时的最新数据刷新界面
* 不排除玩家开了很多UI界面，但处于挂机状态，并不会反复查看已打开界面，在框架设计时应该给不同类型的界面设置好一定的生命周期，比如一些弹窗、提示等，如果玩家长时间没有手动关闭，应当自行关闭

1. **项目过大，需要一些工具辅助，实现半自动化UI工作流**

* 资源冗余剔除工具，定期清理项目内的冗余资源
* 图集、assetbundle、addressable打包工具，按照自定的粒度进行自动划分及打包操作，并应做好重复资源提取操作
* 资源处理工具，对导入工程的不同资源进行初始化处理，去掉一些不需要的设置，并初步设定好一些参数
* 多语言配置工具
* UI编辑工具，支持美术导出资源后，策划可以先用资源在编辑器内查看效果，不需要等待程序开发完成
* 代码导出工具，一些组件绑定等重复的代码交给工具完成
* 等等

1. **热更新支持**

* 如果需要热更新，框架设计初期就应该将lua融入，确定使用的lua框架、设计模式等
* 划分哪些界面需要热更新，哪些不需要，要设计好对应的C#和lua框架，因为使用lua也会在一定程度上浪费性能，因此如果可以的话，部分常修复、更新的界面使用lua即可
* 热更检查及更新流程需要处理好，资源打包时也需要将此部分考虑进去
* 在制作及开发UI时需要考虑此界面是否需要热更，如果需要，要注意组件的引用及逻辑的编写，尽可能减少与其他界面的耦合