使用红色涂色了的，表示需要再详细看

1. c++

c语言与c++的各自的特点； (c面向过程，c++面向对象，后者是前者的拓展)

c++语言特点： (面向对象，继承封装多态) 多态的好处：一个接口，多种方法。创建基类对象，指向不同的派生类，方便拓展和满足各种需求。

c++基础  <https://blog.csdn.net/kuweicai/article/details/82779648>

构造、析构函数调用顺序:有很多情况需要分析<https://blog.csdn.net/jfuck/article/details/8761425>，总结如下：构造函数调用顺序：基类构造函数(按继承顺序而不是初始化顺序，先虚基类，再非虚基类，另外虚基类必须初始化)—>成员的构造函数(按声明顺序)—>此对象的构造函数体。 析构函数和构造函数调用顺序相反！！

静态方法和非静态方法之间的互调：同一个类中，静态方法可直接调用静态方法，可通过对象来调非静态方法；非静态方法可直接调用非静态方法，可通过类来调用静态方法。在不同类中，无论主动方是谁，只能通过类来调静态方法，只能通过对象来调非静态方法。

指针和引用：指针是一个变量，指向的是一个地址，引用是别名，实质和原来的变量是一个东西；指针有多级，引用只有一级；指针可以只定义不初始化，引用在定义的时候必须初始化；可以const 指针，不能const引用；指针初始化能修改值，即修改指向的地址，引用初始化后不能修改值；sizeof(指针)得到指针自身的大小，32位上是4字节，sizeof(引用)得到原来变量的大小；指针自增表示指向的地址加1，指针的值没有变，引用自增表示原来变量的值加1；一般取地址、函数传参、用new关键字创建的内存、使用链表时，都需要用指针。

指针数组&数组指针：前者是数组，后者是指针

如何避免一个头文件被include多次：

1、使用宏定义，在.h中加入宏定义,这样无论在一个类中被引入几次,都只会定义一次.

#ifndef \_NAME\_H

#define \_NAME\_H

class Student{//....};

#endif

1. 使用#pragma once

#pragma once

class Student{//...};

1. 使用\_Pragma操作符，c99标准中新增了一个和#pragma指令类似的\_Pragma操作符,是#pragma的增强版，不仅可以实现#pragma的所有功能，还能和宏搭配使用。

\_Pragma(“once”)

class Student{//...};

define、const：define宏定义一个不带类型的常数，const定义一个带类型的常量；前者在预处理阶段执行，后者在编译阶段执行；前者只是简单的字符替换，没有类型检查，后者有类型检查。后者可以修饰很多地方(函数、函数参数、函数返回值等)，在函数后面加const表示此函数不可修改类的成员变量，且只能在非静态成员函数后加const，在函数前面加const表示此函数的返回值不可修改。

extern：修饰变量、函数的声明，表示该变量或函数是在别的文件中定义的

volatile：用来修饰变量，用它声明的类型变量表示可以被某些编译器未知的因素更改，所以编辑器对访问该变量的代码不进行优化，从而提供对特殊地址的稳定访问

new/delete、malloc/free：前者是关键字(需要编译器支持)，后者是库函数(需要头文件支持)；前者除了申请内存还会调用构造器完成初始化，后者只会申请内存；前者能重载，后者不能；

内存分配：堆 栈 全局区/静态区 常量区 代码区。其中栈和数据结构中的栈类似，也是线性存储，堆就不一样了，此处的堆结构是链表。

数据结构中的堆栈：与内存分配中堆栈不太一样，栈是线性表；堆是特殊的完全二叉树，堆的根结点可能是最大节点(大顶堆)，也可能是最小节点(小顶堆)

内联函数：函数代码放入符号表中，使用时直接替换，没有函数调用的开销，同时会进行类型检查，比宏要安全，但是函数体不能太大，会很耗内存

各种数据类型在32位、64位系统上占多少字节  int 4 4   char 1 1  char\* 4 8   float 4 4  double 8 8  long 4 8

虚函数和纯虚函数：虚函数是多态的体现，有纯虚函数的类是抽象类，抽象类不能实例，只能被继承并重写其虚函数才能被使用。区别：1、纯虚函数只有定义，没有实现，虚函数有定义也有实现

虚函数和虚继承：前者对于函数而言，后者对于类而言，都用了虚指针(均占用类的存储空间)和虚表(均不占用类的存储空间)。后者的原理是每一个虚继承的子类中都有一个虚基类指针，指向虚基类表，表中记录虚基类和本类的偏移地址，前者的虚表中存储的是虚函数地址(按声明顺序放入表中，)。 <https://blog.csdn.net/bxw1992/article/details/77726390>  https://blog.csdn.net/qq\_33508523/article/details/89503546

重载和重写：前者是指在一个类中，函数的参数列不同(类型、个数、顺序不同)，根据参数列表确定调用哪个函数，后者指子类中存在重新定义的函数，其函数列表与父类一致，且父类的此函数必须有virtual修饰；

c++ 和 lua、js如何通信

lua 和 c++ 交互：  c++调用lua：通过虚拟栈，把c++要用的变量或函数名入栈，如果是函数则要把参数入栈，然后函数的话调用lua\_call(PL, 1, 2),然后从lua中查找到变量或者函数返回值会放到栈顶，直接取出即可。      lua调用c++：1、直接将函数写入源码(lua.c)中并注册   2、使用静态依赖的方式

STL标准模板库：算法、队列、栈、模板类、迭代器、vector、list、map、set等

进程和线程：一个程序至少有一个进程，一个进程至少有一个线程；线程执行时是同步和互斥的，他们共享该进程下的资源，但一个线程结束不会影响同一进程下的其他线程；一个进程不能创建无数个线程，受内存的限制，比如操作系统给一个系统进程提供2GB的空间，一个线程堆栈空间默认是1MB，理论上最大线程数=2GB/1MB=2048，且内存不能完全拿来做线程的栈，所以实际数比2048要小。

进程间如何通信：1、管道(无名管道，是一种特殊的文件，可用read和write来读写，是半双工的，数据只能在一个方向上流动，只能用于有亲缘关系的进程间通信，比如父子和兄弟进程间，速度慢)；2、FIFO(命名管道，是一种文件类型，用I/O函数操作他，在无关的进程间交换数据，速度慢)；3、消息队列(容量收系统限制，且第一次读要考虑上次没读完数据问题)；4、信号量(用于实现进程间的互斥和同步，而不是存储通信数据)；5、共享内存(能很好控制容量，速度快，但要保持同步，一个进程在写的时候另一个进程要注意读写问题)

2、js

数据类型：Number、String、Boolean、null、undefined、Object(这个是引用类型，前5个是基本数据类型)

es6 新特性(promise awit) <https://www.jianshu.com/p/ac1787f6c50f>   1、const和let   2、模版字符串  3、箭头函数  4、函数的默认参数  5、promise  6、… 和 解构赋7、类   8、set、Map  9、Symbol(一种新的数据类型) 10、模块(用import取代require)

原型和原型链 https://blog.csdn.net/cc18868876837/article/details/81211729

\_\_proto\_\_和constructor 是对象和函数都有的，prototype仅函数有，函数也是一种对象

js实现类的几种方式：构造方法；原型链；es6里有类这个概念；call用法也能相当于继承。

let 和 var 区别：作用域不一样，let是块级作用域，var只有函数作用域和全局作用域；let变量不能重复声明，var可以；let不能在定义前访问该变量，var可以。

Null和undefined的区别：null和undefined是js的两个基本数据类型；null是关键字，表示空值，undefined是一个全局变量，以下情况会是undefined(使用未定义变量 or 已定义但未声明的变量 or 一个对象属性不存在 or 函数该提供的参数没有提供 or 函数没有返回值默认返回undefined)； null== undefined 为true，null === undefined为false。

静态作用域和动态作用域：前者也叫词法作用域，是指函数的作用域在函数定义时就确定了，后者指函数的作用域在运行时才确定。Javascript c c++ Java python lua都是静态作用域，常见用动态作用域的语言是bash。

闭包：能调用别的函数内变量的函数。 变量会一直保存，比较耗内存。应用场景：setTimeOut(js的setTimeOut传递的第一个函数不能带参数，通过闭包可以实现传参)、事件回调、封装私有方法、forEach和find等方法。

匿名函数和箭头函数：前者的this由调用者确定，call和apply可以更改调用对象；后者的this为词法作用域，谁调用的就是谁，不支持call和apply的使用。应用场景：数组最大值(Math.max.apply(null,arr))、将类数组对象转化为数组(Array.prototype.slice.call(arguments))、事件监听回调(bind)

接口和抽象类区别：前者需要被实现，后者需要被继承

js异步机制、事件循环：js是单线程的，但js的宿主环境(浏览器、node)是多线程的，通过事件驱动使js具备异步属性。异步的函数会放到任务队列里，每次事件循环结束后会去任务队列里看有没有需要执行的，es6中任务队列优先级比事件循环中的任务要高，Promise就是使用了es6中这一特性。

call、apply、bind的区别：call传的参数列表，apply传的参数数组，bind传的参数列表，返回的是函数，不会立马执行

setTimeout和 setInterval：前者是一定时间后将函数添加至执行队列，真正的执行时间受之前函数代码执行时间影响，前面阻塞了，这儿也会阻塞；后者是不管前一次是否执行完毕，每隔一定时间重复执行。

内存机制(下面有笔记?? 详1)

  3、  网络 <https://www.jianshu.com/p/e05632cea0c9>

长连接、短连接：长短连接的区别在于在一次请求后是否马上关闭连接，即tcp的长短连接。HTTP1.0默认是短连接，HTTP2.0默认是长连接，但每个请求仍要单独发header但HTTP仍然是面向连接且无状态的。Websocket的长连接是真正的长连接，后续请求不需要发请求头，而且C和S可以相互发。

http：几种请求方法：GET、POST、HEAD、PUT、DELETE、CONNECT、OPTIONS、TRACE、PATCH、MOVE(前3种是1.0定义的，其他是1.1定义的) <https://www.jianshu.com/p/e05632cea0c9>      https是在http下加入了SSL层(安全套接层) https://www.cnblogs.com/klb561/p/9221754.html

socket、websocket：socket是传输控制层的接口，通过socket来操作TCP/IP；websocket是一个完整的应用层协议，和http一样。两者都能做即时通讯

tcp、udp、rudp  数据包：tcp可靠(3次握手 4次挥手以及数据传输时有确认、窗口、重传、拥塞控制机制，传完后会断开连接节约系统资源)但是速度慢； udp不可靠易丢包但是快且可以广播。rudp是可靠的udp。

握手和挥手的c和s的状态变化：https://www.jianshu.com/p/6b2e35fdaf2c

握手：C和S都是CLOSED状态 —>S创建传输控制块TCB，处于LISTEN状态，等待C请求—>C创建TCB，发送连接请求(SYN=1,seq=x),进入SYN-SENT状态—>S收到连接请求，向C发送确认(SYN=ACK=1,确认号ack=x+1,初始序号seq=y),进入SYN-RCVD状态—>C收到S的确认，给S发出确认(ACK=1,ack=y+1,seq= x+1),A进入ESTABLISHED状态—>S收到C的确认后，进入ESTABLISHED状态。

挥手：C和S都是ESTABLISHED状态—>C发出连接断开报文(FIN=1,seq=u),进入FIN-WAIT-1状态—>S收到C的请求后发出确认报文(ACK=1,seq=v,ack=u+1),进入CLOSED-WAIT状态—>C收到确认后，进入FIN-WAIT-2状态，等待S的断开连接报文—>S发出连接断开报文((FIN=1ACK=1,seq=w,ack=u+1),进入LAST-ACK状态—>C收到S的断开请求后发送确认报文(ACK=1,seq=u+1,ack=w+1),进入TIME-WAIT状态—>S收到后进入CLOSED状态—>C经过等待计时器时间2MSL后，如果S没有发送消息过来，进入CLOSED状态。

udp如何实现可靠传输：<https://www.jianshu.com/p/6c73a4585eba>  rudp是可靠的udp

URI 和 URL：前者是统一资源标识符，用来唯一标识一个资源，后者是统一资源定位符；URI包括URL和URN(统一资源名称)

Tcp如何进行流量控制，避免网络堵塞：滑动窗口、流量控制(返回的ack中包含自己的接收窗口的大小)、拥塞控制(慢开始、快重传、快恢复)

断点续传实现：请求端在HTTP头部加入Range字段，比如Range:bytes=44445555-，告诉服务端需要续传的片段，服务端收到后，从44445555字节开始传，同时还会新增一个字段Content-Range=bytes 44445555-66667777告诉请求端发的是哪段数据，同时返回的状态不是200了，而是206。

Lua

基本数据类型：nil、boolean、string、number、table。非基本数据类型：function、userdata、thread线程。

如何实现面向对象：通过元表(\_\_index用来查找 和 \_\_newindex用来赋值)和元方法(setmetatable())，setmetatable(a,b) 设置b为a的元表，即b是a的父类,查找一个元素时会先在a中查找，a中找不到再通过\_\_index去b中查找。

协同程序和线程的异同：同：都拥有独立的堆栈，独立的局部变量，与其他线程/协同程序共享全局变量和其他大部分东西；异：可以同时运行几个线程，他们独立互斥。但协同程序需要彼此写作运行，在任一时刻只有一个协同程序在运行。协同程序有点类似在等待同一个线程锁的几个线程。

4、cocos

引擎

基础

creator生命周期：onload、start、update(每帧渲染之前调)、lateUpdate(每帧渲染结束之后调)、onDestroy、onEnable、onDisable

Cocos2dx生命周期：init、onEnter、onEnterTransitionDidFinish、onExitTransitionDidStart、onExit、cleanup。如果两个场景用replaceScene切换：进入scene1: s1:init->s1:onEnter->s1:onEnterTransitionDidFinish；进入scene2: s2:init->s1:onExitTransitionDidStart->s1:onExit->s1:cleanup->s2:onEnter->s2:onEnterTansitionDidFinish。如果两个场景用pushscene切换，没有cleanup，其他和replacescene一样。

new 和 create的区别：create包括new 和 init，create能把对象放入自动释放池中

    渲染

  RGBA8888一个像素占多少内存  (8bit+8bit+8bit+8bit) / 8 = 4byte

    内存管理：引用计数机制，自动释放池。通过create创建对象时，引用计数为 1，且会自动添加到自动释放池中，每一帧结束后自动释放池会自动释放掉引用计数为0的对象，可通过retain使引用计数加1，release使引用计数减1。

渲染流程：

Cocos2d-x2.0版本：每个node的draw函数直接调用opengl接口，有几个图形就渲染几次。

Cocos2d-x3.0版本：通过RenderCommand封装起来，然后添加到CommandQueue队列中，再通过ID进行排序再渲染，相对2.0提升了渲染速度。具体流程是：导演类的mainLoop中会调用drawScene,在drawScene中会调用场景类的render,render中会递归执行节点类的visit，visit中会调用精灵类的draw,draw中会执行渲染类的addCommand。对所有节点执行完addCommand后，会执行渲染类的processRenderCommand，接下来执行渲染类的drawBatchedTriangles，最终在drawBatchedTriangles内会调用多个openGL API完成渲染。

creator版本：creator1.0版本的渲染流程和cocos2dx3.0差不多，creator2.0以上有区别，摒弃了渲染树，直接使用节点和渲染组件数据来组装渲染数据，渲染组件都继承自RenderComponent组件，该组件持有\_assembler和\_renderData,入口在engine/cocos2d/core/renderer/index.js中render-flow.render方法——>forward-renderer.render方法——>base-renderer.\_render方法——>forward-renderer.\_opaqueStage方法——>forward-renderer.\_drawItems方法——>base-renderer.\_draw方法——>device.draw方法 后面就是调opengl的接口了.

    遇到过的引擎bug

webgl 和 canvas：前者支持3D，后者只支持2D。webgl底层还是OpenGL https://www.jianshu.com/p/e3d8a244f3d9

屏幕适配

性能优化：降低drawcall、减少内存占用、提高帧率

降低drawcall：??

减少内存开销：

图片压缩使用无损压缩png(因为jpg在加载的时候会实时地转化为png格式，或者使用有损压缩：Android用etc2(etc1不支持alpha通道，不支持有透明度图片的压缩，etc2支持alpha通道，但它是opengl ES3.0标准,(软件android4.3以上支持ES3.0,iOS7以上支持ES3.0,此外还需要硬件的支持)，支持此标准的设备市场占有率少),ios用pvrtc)、

合图

及时释放

延迟加载分步加载、避免断层渲染(同一批次的渲染不被打断)

帧数优化：分帧(time-slice)计算

热更新机制：creator中可用AssetsManager来更新，Cocos2d-x的话需要自己写，原理和AssetsManager一样。

状态同步和帧同步：前者是client把操作发给server，server把结果计算完后同步给每个客户端；后者是client把操作发给server，server不进行计算，只转发操作给每个客户端，客户端收到后再进行操作。前者优缺点：安全性高、网络要求宽松抖动丢包适应性强、断线重连快、游戏逻辑性能优化有优势，但是开发效率低，准确性差、网络流量随游戏复杂度增加。后者的优缺点与前者相反。

帧同步的延迟卡顿优化：tcp换成udp(为了快速重连)；逻辑与表现分离(避免卡顿)；本地插值平滑(避免卡顿抖动)。

骨骼动画：DragonBones、Spine。Spine的Attachment 类型有：region(纹理区域,默认这种)、mesh(贴了纹理的网格，顶点的位置会被多个骨头通过权重来影响)、linkedmesh(和另一个mesh共享uv、顶点和权重数据)、BoundingBox(一般用于物理的碰撞检测)、path、point、clipping。关于spine的详细知识可见知乎地址<https://zhuanlan.zhihu.com/p/118942376>。

4、算法

排序算法原理及时间复杂度

A\*算法 可以用堆排序提高效率

消除类算法: https://blog.csdn.net/sinat\_39291423/article/details/78089828

5、数据结构

ProtoBuf

set、list、map、vector：set是无序不重复的集合，list是允许重复的数组或者链表，map是无序不重复的键值对集合，vector是连续存储的动态数组。

hashMap的底层是哈希表，数据结构是数组+链表，通过hash算法计算key对应的hash值h，然后通过h计算出索引，把对应的<k,v>存储到此索引下，如果有冲突，需要通过开放寻址法(继续在此数组中找到一个适合的位置)或者链地址法来解决冲突(把key相同的存储在一个单链表中，链表长度大于8时，链表会转换成红黑树，小于6时又会退化成链表)。https://www.jianshu.com/p/67b825e08d17

map的底层是红黑树，能自平衡

map的查找速度是O(logn)级别，hashmap是O(1)，但是内存占用大，如果想要快就用hashmap，如果对内存有要求就用map。

跳表：即对链表加n级索引，只能用于元素有序的情况，插入删除搜索时间复杂度是O(logn)，对标平衡树和二分查找。

6、设计模式

观察者模式：NotificationCenter

装饰者模式：Action和他的继承类之间的关系

单例模式：Director

工厂方法：Create方法

7、项目中遇到的复杂问题

1.8.2上 spine局部换肤

内存释放问题，只release资源并没有用，需要把用到此资源的节点置空，且内存回收有延迟，有时候需要手动gc，此时会导致卡顿。

毛入口es6语法没有完全转成es5语法，然后在Android4.0以下上map等用不了，会报错。处理方法：当时项目着急且简单，就自己手写了一个对象，后来人物换装还可以用webpack打包就解决了。

龙骨在creator1.8.2上有内存泄漏，我们龙骨动画比较多，引擎创建了很多frame对象，然后没有释放，处理方法：在析构函数里强制释放了一下。

8、渲染管线

总结：

        渲染管线是一种计算机从数据到最终形成图像的形象描述

VBO、VAO：前者Vertex Buffer Object顶点缓冲区对象，后者Vertex Array Object顶点数组对象；前者是显卡存储空间里一个缓存区，存储顶点坐标/uv/法线/颜色等，后者是状态容器，记录VBO的状态。

IBO索引缓冲区对象，即展示上屏幕上的图形需要画的点的顺序(比如正方形四个点从左到右从上到下分别是1，2，3，4，那索引是[1,2,4,2,3,4])

shader是图形可编程方案的程序片段。主要分为定点着色器和片段着色器

材质可以理解为商品，shader是加工这种商品的方法，贴图是加工过程中需要的原材料

opengl渲染管线流程：顶点数据->顶点变换(各种矩阵变换)->图元装备(画线、三角形等)->纹理着色->光栅化，输出到帧缓存。顶点shader(vertex shader)发生在顶点变换阶段，通过顶点shader可修改基本的图元属性，如颜色、光照等；片段shader(fragment shader)发生在纹理着色阶段，主要对上阶段输出的数据进行再次加工。

shader  <https://blog.csdn.net/xufeng0991/article/details/72973664> <https://www.cnblogs.com/maple-share/p/5395097.html>

shader三种变量  uniform（全局变量）、attribute(只在定点着色器中使用)、varying(在定点着色器和片段着色器的传递变量)

webgl、opengl、canvas：opengl是底层的驱动级的图形接口，类似DirectX，webgl使用js调用封装过的opengl。Canvas只支持2D，webgl支持3D，且比canvas性能好。

向量点乘、叉乘：a.b = |a||b|cos@ 计算两个向量之间的夹角，以及b在a方向上的投影； aXb 得到垂直于a和b向量构成的平面的法向量，在二维空间中，也等于a和b构成的平行四边形的面积

python的内存管理和垃圾回收原理：内存管理是用引用计数器，回收引用计数为0的，且找到循环引用对象，释放其内存 https://blog.csdn.net/zx870121209/article/details/81363311

9、h5游戏优化：https://blog.csdn.net/zy835940305/article/details/90764549

1、优化文件结构(最先展示的先加载，后展示的后加载)

2、减少请求：减少文件数量(不需要预加载的不要放到resource下，避免产生不必要的import下的文件；图片使用图集)、减小单个文件的体积 (图片tinypng压缩)

3、减少drawcall：如下详3

4、延迟加载：在creator中设置

  5、使用对象池

10、其他

webpack/gulp/rollup构建工具：

webpack如何压缩代码：需要AST(抽象语法树)支持  1、去除多余字符：空格，换行、注释 2、压缩变量名、函数名、属性名 3、更简单的表达：合并声明以及布尔值简化

详1:

—————————————详1  start       js内存相关—————————————————

js内存回收：标记清除和引用计数 https://blog.csdn.net/yingzizizizizizzz/article/details/77333996?depth\_1-utm\_source=distribute.pc\_relevant.none-task&utm\_source=distribute.pc\_relevant.none-task

回收时，会暂停执行代码

v8内存分配(1.4G)：新生代内存空间(from、to)64M、老生代内存空间1.4G

回收算法：新生代：复制。把from中能复制到to中的变量复制到to中(因为有些变量已经不存在了，或者晋升到老生代空间去了)，然后把from中全部清空，下一次回收时把to中能复制的变量复制到from中，然后to中全部清空，以此循环往复

  老生代：标记删除整理。先标记存活的变量，然后把无用的变量删除，最后整理整个空间，换得一个连续的可用空间

新生代如何晋升到老生代：经历过一次回收(即从from里复制到to里过)   &&  to空间已经使用了25%(大的变量经过一次回收就会直接放到老生代空间)

js回收变量

局部变量：当作用域结束且没有引用的时候就会回收

全局变量：会存活到程序运行结束

优化内存技巧：

1、尽量不要定义全局变量

2、全局变量记得销毁，并不是所有的都得等程序结束，不推荐delete来销毁(1、有些浏览器的解析和支持有问题，2、严格模式下delete有bug)，推荐使用重新赋值，a = undefined or  a = null；(趣谈：null是一个保留字，undefined是一个变量)

3、用匿名自执行函数把全局变量变成局部变量，(function(){})()

4、尽量避免闭包引用。关于闭包的错误认知：使用闭包一定会内存泄露(错误认知)。    但是！闭包被引用了，那就会出问题，可能导致内存不释放。

防止内存泄露：

1、避免滥用缓存。不要用v8来缓存(exp：redis、localstorage)，一定要用的话可以加锁(比如：限制缓存的长度，如果超过长度，就删掉最老的缓存)

2、避免大内存量操作。即不要一次性操作大量文件，导致

—————————————详1  end—————————————————

详2:

————————————详2  start  排序算法————————————

常见排序算法：

插入排序： 直接插入(时间复杂度 O(n² )   空间复杂度O(1))  希尔排序(O(n^1.3  O(1))

选择排序： 选择排序(O(n² )  O(1))  堆排序(O(nlogn)  O(1))

交换排序： 冒泡排序(O(n²)  O(1))  快速排序(O(nlogn)  O(lgn))

归并排序： 归并排序(O(nlogn)  O(n))

————————————详2 end  排序算法——————————————

详3:

—————————————详3  start      游戏优化—————————————————

1、包体优化(压缩 或者Android用etc2,ios用pvrtc)、

2、内存优化(合图、及时释放、延迟分帧加载)

3、性能优化

drawcall优化  一个spine因为只有一张大图 所以drawcall是1

静态sprite：合图

label：尽量不要用系统字，尽量使用BMFont位图字体

减少mask和富文本组件的使用

避免跨层切换贴图 (一个节点下的多个sprite，用的是同一张图集，那么这些sprite会合为一个批次渲染哦，即drawcall为1)

优化节点树，减少节点数量，优化场景结构(结构上尽量sprite紧挨着，再放label)

场景中不要挂载过多prefab，可适当动态加载

材质不同、颜色或者透明度不同、渲染模式不同都会打断drawcall

对象池

—————————————详3  end      游戏优化—————————————————

详4: 虚函数 虚函数指针 虚函数表指针 虚函数表

1、函数一旦声明成了虚函数，在所有子类中都是虚函数，如果子类要覆盖，那行参和返回类型必须和父类一致，virtual关键字可写可不写。

2、每个类的对象，都有一个虚函数表指针，存在于对象实例中最前面的位置，指向该类的虚函数表，虚函数表属于类，存储所有虚函数地址(即虚函数指针，怎么存的见4)，虚函数指针占用类的内存空间，虚函数表不占用。

3、如果一个父类指针指向了子类对象：

3.1 此子类对象会做裁剪，会没有子类中独有的成员函数(A中有f()、g()，B继承A，B中有f()、g()、h()，A指针指向B对象，即 B b; A \*a = &b，此时a是调用不到h的，无论h是否为虚函数)

3.2 对于从父类继承过来的非虚函数(如果不是从父类继承过来的，参数3.1)，此子类对象会调用父类的

3.3 对于从父类继承过来的虚函数(如果不是从父类继承过来的，参照3.1)，如果有重写，那会调用重写的，如果没重写，那就会调用基类的

3.4 父类的析构函数必须为虚函数，不然就会出现3.2的情况，可能导致子类没有被析构

4、每个类的虚函数表中如何存放虚函数指针的  https://www.cnblogs.com/yinbiao/p/10987640.html

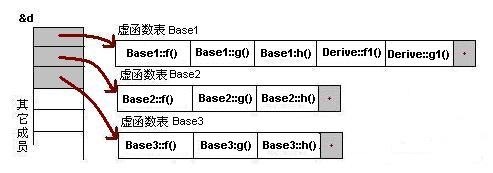
4.1 按虚函数声明顺序依次存放，父类在前，子类在后，以”.”作为虚函数表结尾标识符，不同编译器下使用不同标识符

|  |  |  |  |
| --- | --- | --- | --- |
| A::f() | A::g() | B::f1() | B:g1() |

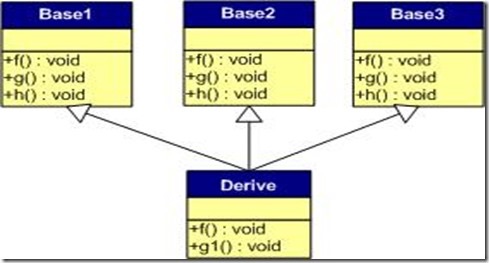
4.2 若子类中有覆盖父类的虚函数，那在子类的函数表中，子类中覆盖的虚函数会放到原父类虚函数的位置，没有被覆盖的函数没有变化

|  |  |  |  |
| --- | --- | --- | --- |
| B::f() | A::g() | B::f1() | B::g1() |

4.3 若多重继承 且 没有覆盖，每个父类都有自己的虚函数表；子类的虚函数表被放到第一个父类的虚函数表中



4.4 若多重继承 且 有覆盖，会结合4.3 和4.2(与4.2不同的是，所有父类虚函数表中被覆盖的虚函数都会替换成子类的虚函数)





虚函数、纯虚函数、虚继承补充(可能与上面有重复)

虚函数和纯虚函数：

静态函数不能定义为虚函数，因为static函数不属于任何对象；构造函数不能定义为虚函数，原因c++附1；析构函数必须为虚函数；

1、虚函数是多态的体现，是为了允许用基类的指针来调用派生类的这个函数(注：此函数必须是基类中存在的，例如A类中有f、g函数，B继承A类，B类中有f、g、h函数，A \*a = new B();此时a是调不到h函数的，因为会做裁剪)；     纯虚函数的目的是使派生类仅仅只是继承基类的函数接口，具体实现还得在派生类中。如果派生类中没有实现此函数，那此派生类仍然是抽象类

2、虚函数可被实现也可不被实现，纯虚函数不可被实现，有纯虚函数的类叫抽象类，抽象类不能生成对象，只能被继承，继承此抽象类的子类必须实现纯虚函数，此函数返回值和参数类型必须和基类的一样，virtual可写可不写。

3、虚函数：virtual void fun(){};    纯虚函数：virtual void fun()=0;

4、析构函数必须是虚函数，可以不用非的是纯虚函数。因为如果是从父类继承过来的非虚函数，子类对象会调用父类的，会导致子类没有被析构

5、友元函数不能是虚函数，但是友元函数可以调用虚函数

解决二义性的办法：

1、使用作用域限定符“::”，  A类中有fun函数，B类中有fun函数，C继承A和B，C的实例c如果想调用A中的fun函数，可以c.A::fun()来调用

2、子类中定义同名成员，即1中的例子，让C也定义fun函数，那么c.fun就是调的C类自己的fun，不存在二义性

3、使用虚继承

虚继承的作用

1、减少内存浪费，只会保证有一份虚基类在内存中

2、解决菱形继承导致的二义性

虚函数和虚继承：都有虚指针(前者是虚函数指针，后者是虚基类指针，分别指向各自的表。均占用一个指针的存储空间 4字节)和虚表(前者是虚函数表，存的是虚函数地址；后者是虚基类表，存的虚基类相对直接继承类的偏移地址。均不占用内存)。  特别说明：对于虚基类而言，虚基类依旧会在子类里存在拷贝，只是最多存在一份(即菱形继承时，B和C虚继承A，D继承了B和C，在D中只有一份A)，当虚继承的子类被当作父类继承时，虚基类指针也会被继承。

虚基类的初始化：

1、一般情况下，派生类的构造函数只需负责对其直接基类初始化，再由直接基类负责对间接基类初始化

2、对虚基类的派生类而言，不仅要负责对其直接基类进行初始化，还要对虚基类初始化。

虚函数表和虚基类表：

每一个有虚函数的类都有一个一维的虚函数表，每个对象都有一个虚函数指针指向虚函数表(注：纯虚函数的类即抽象类也有虚函数表，按理说抽象类不能被实例化所以不需要虚函数表，但是为了保证被重写的函数能找到正确的偏移地址所以虚函数表还是有必要的)，虚函数指针位于对象存储空间的开始位置，虚函数表存储的是虚函数地址(按声明顺序放入表中)；

每个虚继承的子类都有一个虚基类指针和虚基类表，虚基类表记录虚基类和本类的偏移地址。

虚函数表：详4

按虚函数声明顺序存放，父类在前，子类在后，子类中覆盖的虚函数会放到原父类虚函数位置；若只继承一个父类，则虚函数表是一维的，若多重继承，则是二维的，[[基类1的虚函数表],[基类2的虚函数表],[基类3的虚函数表]]，且子类的虚函数表放到第一个父类的函数表中

虚基类表：

虚基类指针是虚继承的子类对象才有。

虚基类指针的第一条内容表示的是该虚基类指针距离所在的子对象的首地址的偏移（即虚基类指针相对于自己对象的首地址偏移量，一般是-4？）

虚基类指针的第二条内容表示的是该虚基类指针距离虚基子对象的首地址的偏移（即虚基类指针相对于基类对象的首地址偏移量，一般得根据情况算）

Exp:

Class Base{int var; virtual void fun(){}}

Class A:virtual public Base{int varA; virtual void fun(){}   virtual void funA(){}}

Class B:virtual public Base{int varB; virtual void fun(){}   virtual void funB(){}}

Class C:public A,public B{int varC;  virtual void fun(){}   virtual void funB(){}  virtual void funC(){}}

每个有虚函数的类都有虚函数表，每个虚继承的子类都有虚基类表。

C对象模型如下：

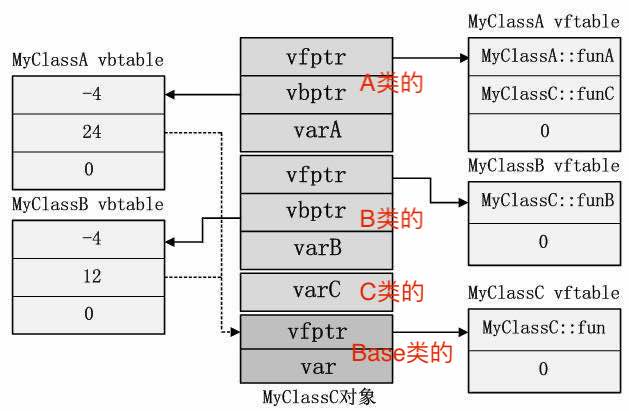
vfptr表示虚函数指针，vftable表示虚函数表； vbptr表示虚基类指针，vbtable表示虚基类表。

C类的虚函数会放在第一个基类A虚函数表中，所以A的vftable中有func函数。

根据虚基类指针的第一二条内容规定得出：

A的vbtable第一条即A中vbptr和vfptr的偏移，所以为-4，第二条即A中vbptr和Base中vfptr的偏移，所以为24。

B的vbtable第一条即B中vbptr和vfptr的偏移，所以为-4，第二条即B中vbptr和Base中vfptr的偏移，所以为12。



C++附1

为什么构造函数不能是虚函数？

1. 创建一个对象必须明确指出它的类型，否则无法创建。虚函数是在运行才能确定其调用哪个类型的函数，具体调哪个类型是编译器通过对象类型确定，此时对象还未创建也就无法知道其真实类型，从而形成了死锁。
2. 虚函数对应一张虚函数表，虚函数表是属于类的，但是每个对象有个虚函数指针指向虚函数表，如果构造函数是虚函数那就需要通过虚函数表来调用，可是对象还没实例化，虚函数指针还没有创建，找不到虚函数表

//todo

1、protobuf https://www.cnblogs.com/creator-star/p/11584504.html

2、家具遮挡层级计算

3、hashmap、TCP4次挥手时client和server的状态变化

4、观察者模式

5、是否是字串

6、cc.load的底层实现

7、自动释放池什么时候调用

8、缓动作和runaction区别

9、HTTP有哪些请求方式，post方式能在URL里带参数吗

10、ts是如何转换成js的