# Cocos2d-x: A cross-platform 2D game engine



## Abstract

Cocos2dx is an open source 2D game engine which support cross-platform and a wide range of languages like c++, javascript and lua. You can use cocos2dx to develop your game which can runs on windows, macOS, android, Linux and Web platform with only one set of code. It is supported by lots of people on github, you can easily download its source code and start building your application. The features of cocos are fast, free, easy to use. To find out how cocos achieves those features, we start looking at its architecture from the bottom. Below is the what we find about the architecture of cocos through different viewpoint and perspectives

## .Table of contents

1. Introduction
2. Stakeholder Analysis
3. Development View

## Introduction

Cocos2d-x is a cross-platform game engine. A game engine is a piece of software that provides common functionality that all games need. You might have heard this referred to as an API or framework but in this guide, we'll be calling it a **game engine**.

**Game engines** usually support multiple platforms thus making it easy to develop your game and then deploy it to multiple platforms without much overhead at all. Since **Cocos2d-x** is a **game engine**, it provides a simplified API for developing cross-platform mobile and desktop games. By encapsulating the power inside an easy to use API, you can focus on developing your games and worry less about the implementation of the technical underpinnings. Cocos2d-x will take care of as much or as little of the heavy lifting as you want.

The original version of Cocos2d was written in python, then use objectiveC, which was applied to iphone. After that, someone in China changed it to C++ version according to the design concept. This version can run on iphone, widnows and android platforms. The latest version of cocos is 1.0.0. The C++ version of the engine also introduces the lua scripting system. Cocos2d is a completely open source engine which is widely used including Zynga, Nangong Meng and other well-known game companies. Cocos2d is not only a 2D graphics engine library, but also provides some systems to support game logic. Cocos2d provides a simple sound engine that supports playing mp3, wave, ogg etc. cocos2d can be used together with the physics engine box2d to achieve physical effects such as collisions. The structure of the cocos2d engine is clear, and the documents and examples are rich which makes cocos very easy to learn.

This chapter,we discuss in details about the architecture about cocos from different stakeholders’ view. We divide the whole system into different components which have a specific function and look at how they work tighter. I believe that after reading our report, you will have a better understanding of cocos.

## Stakeholder Analysis

In the book *Software System Architecture* we learned how to analysis the stakeholders of a standard software system by identify the roles which are played by those people/group who are involved in the process of construction. For this section we are going to analysis and identify the significant roles that of great importance.

### Acquirers

Acquirers, or called as investors, is the first part of the chain which decides the survival of tech companies. Cocos2d-x was developed by @*Ricardo Quesada* who was born in Argentina, and later on established the @Los cocos which was the prototype of cocos2d-x. Now we would like to claim that the cocos2d-x is sponsored or invested by a Chinese company @Chukong Technologies, which is the role of sponsor. This company decides the business and technical decisions and provides funds to cocos2d-x.

### Assessors

Assessors are stakeholders that test bugs in the system which is game engine in Cocos case, and report or modify the bug-list to the developers. For cocos, the job is taken by the group inside the company rather than give out to other testers. In the test part, testers would test every detail in System, like Actions test, Billboard test, Box2D test, ClickandMove test, etc. Each part will be allocated to a tester or two to make sure the System goes well. @minggo(Zhang XiaoMing), @drelaptop are active bug-testers in Cocos2d-x github who donate significant number of bugs.

### Developers

As an open-source program on GitHub, dozens of developers would like to take part in this job. A few of them can join in the program at last, however. As developer of game engine, they need to master the framework, web service techs, androids programming, js language, 3d developing and other skills, so the number of core developers are 13 only. As for framework/web service support, each part gets no more than 2 individuals. For certain many individual developers also did a great job and made the cocos grow faster ever.

### Supports

We divide the support characters as tech support and platform support to identify the concrete role. Tech support grant developers deploy their program on different platforms that are supported, which makes cocos efficient and powerful.

ⅰtech support

①C++

②Lua

③JavaScript

④Cocos platform framework tools and IDEs

ⅱplatform support

①iOS

②Android

③Windows phone

④Windows

⑤Mac OSX

⑥Tizen

### Support Staff

Cocos has dedicated support staff spreading in several websites: [Facebook](https://www.facebook.com/cocos2dx/), [Twitter](https://twitter.com/cocos2dx), [Blog](http://blog.cocos2d-x.org/), and most accessed [community](https://github.com/CocosStructure/cocos2d-x-1#where-to-get-help). At these places, developers who ran into troubles that unable to solve can obtain the details of APIs and notes of program, and get help from other experienced developers. Or, they can have a look at the [guide](https://github.com/CocosStructure/cocos2d-x-1#learning-resources) on github to acquire the most knowledge/guide that he/she needs.

### Communicators

Communicators often play an important role on explaining the architecture of the system and the function of each part of system to those who may buy/use the product and those who may join in and contribute to the program. Usually they may also take the job which need them to create linkage between different classes. As an open-source program, the importance of communicators is inevitably lower than others. For the core developers, the structure is established by themselves, so they could be seen as the communicators in Cocos program.

### Users

Cocos in China is a perfect successful case in commercial, almost 70% of games in Chinese market is developed by Cocos rather that unity in foreign countries. Without considering the sponsor is a Chinese company, it would also be a marvelous great achievement for a game engine. The user is not only including individual developers and game lovers but great organizations like Zynga、Wooga、Gamevil、Glu、GREE、Konami、TinyCo、HandyGames、IGG and Disney Mobiles. Also for big enterprises the cocos is a great choice, engineers in these companies such as Google, Microsoft, ARM, Intel are active in cocos developing.

### Competitors

As a game engine, at no circumstances can Cocos take in charge of game developing market. For certain, the **unity,** **unreal,Frostbite,IW** are powerful competitors which are trying their best to grasp the market share.

## Power-Interest-Grid View



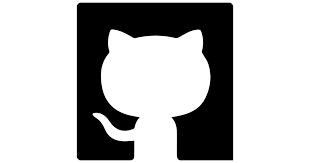


*power*



*Web structure developer*

*Core developers*





*C++*

*Ordinary developers*

*Sponsor*

*API programmers*



*competitors*



*individual contributors*



*Support service*

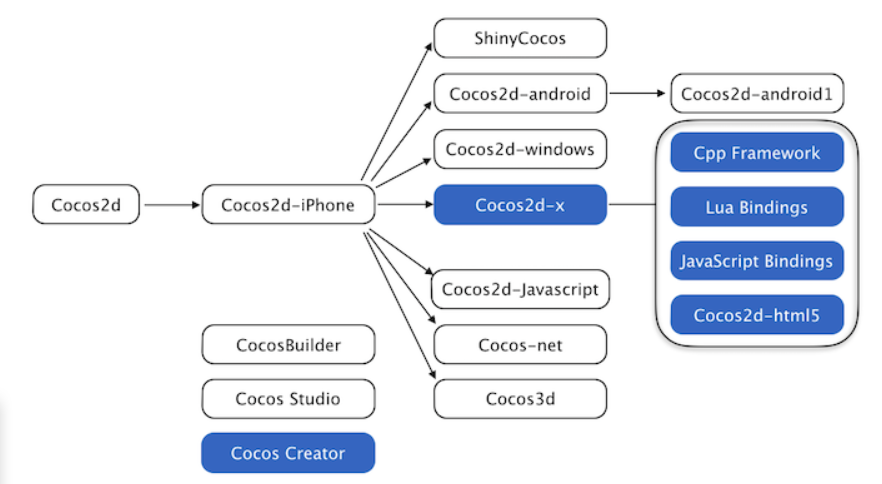
*users*

*Interest*

# Context View

Cocos2d-x is an open-source mobile 2D game framework released under the MIT license. This is a version of the cocos2d-iphone project.It's all written in C++. Cocos2d-x development focuses on the framework provided by Cocos2d cross-platform and cocos2d-x. Mobile games can be written in C++ or Lua, using an API that is fully compatible with the cocos2d-iphone version. The cocos2d-x project can easily be built and run on iOS, Android, Blackberry and other operating systems. Cocos2d-x also supports desktop operating systems like Windows, Mac and Linux.Therefore, the source code written by developers is easy to edit and debug in the desktop operating system.

Family members



Branch engines and editors in the blue box are most active.

Cocos2d-x allows developers to use visual editor Cocos Creator for rapid prototyping, debugging packaging, and cross-platform deployment.

Features：

1.Easy to use: game developers can focus on the game without spending a lot of time learning the arcane OpenGL ES. Cocos2d-x offers plenty of specification.

2.Efficient: cocos2d-x performs graphics rendering based on OpenGL ES to maximize GPU performance of mobile devices.

3.Flexibility: easy to extend and integrate third-party libraries.

4.Free: the free open source framework based on the MIT protocol allows users to use without worrying about commercial licensing.

5.Community support: developers set up a community organizations, easy to access various technical problem.

version control and issue tracker: Git and Github.

Supported platforms: The core advantage of cocos2d-x is that it allows developers to leverage C++, Lua, and Javascript for cross-platform deployment, covering platforms include iOS、Android、Windows Phone、Windows, Mac OS、Tizen、Linux and so on.

Language of IDE: completely written in C++. the core engine has the smallest footprint, yet the fastest speed of any other game engine, and is optimized for running on all kinds of the devices including low-end Android systems.

Software project development language: C++、Lua and Javascript



Users：Cocos2d-x users include not only individual developers and game development enthusiasts, but also many well-known large companies such as Zynga, Wooga, Glu, GREE, Konami, TinyCo and Disney Mobile.As of September 2013, more than 1.5 billion games were downloaded worldwide based on the cocos2d-x engine, many of which also dominated the apple AppStore and Google AppStore rankings.At the same time, many companies such as touch, Google, Microsoft, ARM, and Intel engineers are also very active in the cocos2d-x field.

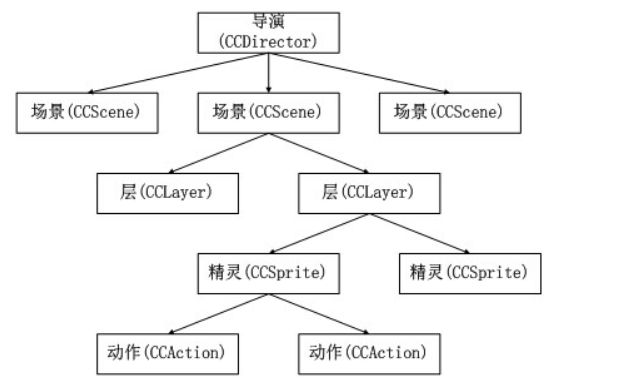
Developers: Wang zhe and his team of Beijing touch technology co. LTD、open source community

Market position:cocos2d-x mainly occupies the high-end and low-end markets.In the development of 2D mobile games in China, the cocos2d-x engine accounts for more than 70%.There are 45 domestic monthly income over 10 million mobile game, 30 based on cocos2d-x.

License: Cocos2d-x is released under the MIT license.

Cocos-2dx 主要类的功能

|  |  |
| --- | --- |
| Stakeholder class | Description |
| Ref | *Ref* class is the base class for the Cocos-2dx project. It contains the related functions that the project uses for automatic memory management, and many classes inherit from it. It should be noted that the constructor of the Ref class is protected, so it can only be inherited and cannot be instantiated. |
| Director | *Director* class determines what is displayed on the screen, that is, through its related functions to achieve the display and hiding of certain nodes. |
| Node | *Node* class is the most basic element in the project scene graph [on the screen or the collection of elements that have been displayed or may be displayed], and most of the classes related to rendering come from Node. It mainly determines the position, size, angle, etc. of the element, and the *Node* class object can be used as a child node, using timers and performing actions. |
| Scene | *Scene* class is the basic unit of scene switching, and provides a rich switching effect. The unit that the project renders on the screen is the scene. There are three types: game content scenes; option class scenes; display class scenes. It should be noted that there can be multiple scenes in a project, but only one scene is active at the same time. |
| Layer | *Layer* class is the basic unit that makes up the scene. Multiple layers form a scene, which is convenient for layered processing to show different effects. The *Layer* class is a container for displaying content. It is usually used as a background for games, and can receive operations such as clicks from users and make corresponding changes. |
| Sprite | *Sprite* class is the most basic unit that players can see. |
| Renderer | *Renderer* class mainly implements the rendering function, and internally, it also needs to classify and sort the Render Command. |
| Event | *Event* class identifies the different types of events and records the specific data, which determines which *EventListener* to handle this event. |
| EventDispatcher/EventListener | *EventDispatcher* class performs like the control center of all events [equivalent to a relay], which manages the *EventListener*. When an Event arrives, it determines the calling sequence of CallBack. *EventListener* class establishes the mapping relationship from Event to CallBack, and the *EventDispatcher* calls the corresponding CallBack according to the relationship. |
| Scheduler | *Scheduler* class is similar to Timer in other languages and is used to implement delayed calls and timed calls. Its other sub-methods also enable control of timing operations, such as pauses, resumes, and so on. |



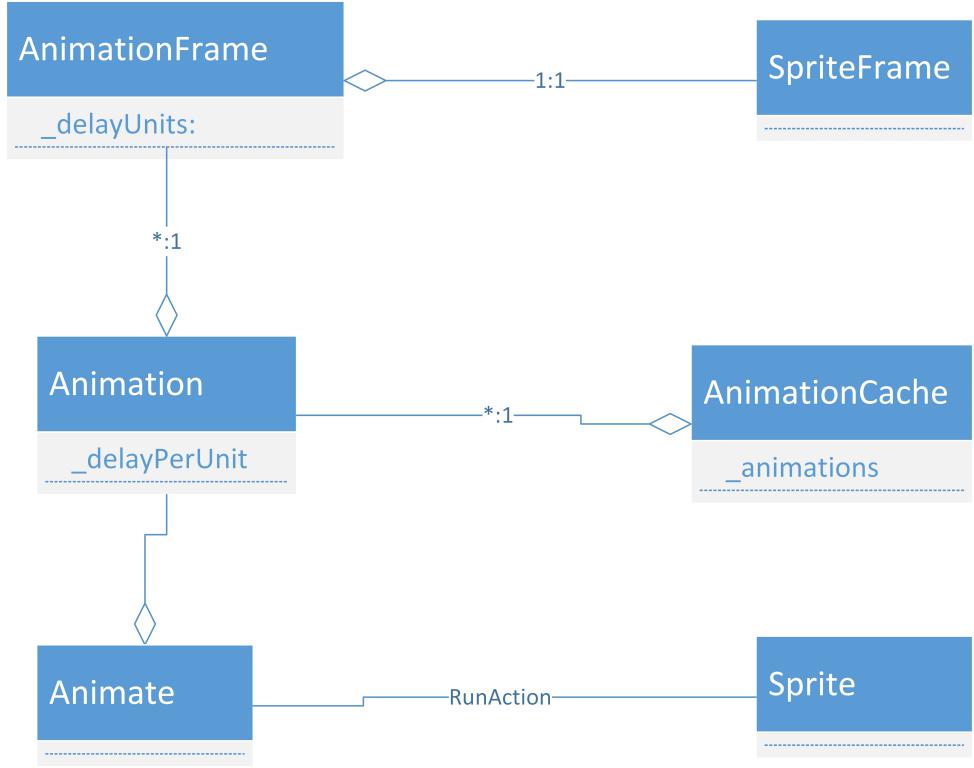
# Functional view

In this part, we focus on four important features that cocos has implemented. The four functions are animation support, cross-platform, node rendering and event-listenser. We will look deep into cocos to see how it achieve those goals.

### ⅠHow does cocos2d-x play animation for its characters?

As a well-known powerful game engine, cocos2d-x is capable of generating animations which will be applied to characters and effects to make them looked like they are “alive”, and for certain those developers who use marvelous skills to modify their animation in their 2d game always get payed times more than those who do not. The structure of frame animation in cocos2d-x provides the fitful way to update games.

### How does it work?



Brief description: Before applying the animation function into cocos system, the basic theory of animation or movies should be introduced. In 1835 the theory of *Persistence of Vision\*,* which claims that due to the react speed of vision cell in our brain, lights will stay on retinal for a short time(1/10 seconds), was found by a Belgium physicist *Joseph Plateau\*.* This brilliant discovery directly leads to the prosperity of film industry, also it inspires the authors of comics and animations. In a word, people can’t recognize pictures separately when they are played in a high speed one by one. The basic principle of structure of animation in cocos2d-x is not an exception.

#### Process:

1. Create the spriteFrame

At first the relationship among sprite, spriteFrame and texture2D need to be declared. Texture is created by single picture and is stored in GPU cache, and texture2D is a class that encapsulate the information of a texture. And a spriteFrame can be created by an object of class texture2d using the function **static SpriteFrame\* create (const std::string & filename,const Rect & rect)** which take a texture file and a rect object as parameters. And a sprite is the encapsulation of spriteFrame and texture2D which will generate a “character” which includes the information of the picture(texture), and this “character” can run kinds of action and communicate.

1. create animationFrame according to those spriteFrames

AnimationFrame is the basic unit in Animation. By the theory *Persistence of Vision* we have introduced above, we can compare cocos to movie player, in which the AnimationFrame is equal to the frame in movie. AnimationFrame can and only can be created by SpriteFrame. The Function is like **static AnimationFrame\* create(SpriteFrame \* spriteFrame, float delayUnits, const ValueMap & userInfo)**

1. create animation

According to the information file for cocos2d-x, Animation objects are used to perform animations on spectacular sprite objects. It contains AnimaionFrame objects and some kind of possible delay between two frames, the way to create animation object is different from the way to create spriteFrames: using a create function without any parameters could create an animation object, and developer should add spriteFrame to the animation object or he/she could add spriteFrame from a file which contains image information, and the animationFrames will be stored in a Vector array. When called, the getFrames() method will return the Vector which contains every single frame for this animation, and setLoops(), getLoops() function will decide how to load this animation.

1. Invoke

Sprites perform animations, that means when needed, animation will be invoked by a specific sprite, and it will be activated by using animate action: **sprite->runAction(Animate::create(animation))**.

1. AnimationCache

As a resource, animation always take time to be loaded into graphic memory, so if many sprites calls for their animations, problems show up. To resolve this, cocos developer decide to create a class that stores/registers the animations instead of saving animations on the Sprite since v0.99.5. This class satisfy the principle of singleton pattern which means there are only one instance of object that can be used in whole system. All needed animations will be stored into this instance. Methods like addAnimation/removeAnimaion/getAnimation provides manipulation on those stored/pre-loaded animations. Also developers could add animations from a plist file after load them in the spriteFrameCache.

### ⅡEvent listener

**Functional capabilities:**

Functional capabilities defines what the system is required to do and what it is not required to do. The following table shows the core functionalities and describes what their responsibilities are.

|  |  |
| --- | --- |
| Functionality | Description |
| Event Listener | Every Event Listener consists of a callback function, a subscriber type, and a listenerID.  Its main responsibilities are implementing various triggering logics, and add the event to the event dispatcher to realize other functions. |
| Event Dispatcher | This component is responsible for scheduling and managing all event listeners, and when an event occurs, it should scheduling the corresponding events. |
| … | … |

All event types must inherit Event. Event is the parameter that the event source passes to the event listener. It contains some information about handling the event. The subclass of Event consists of a type “Event:: Type” and some data.

* enum class Type {

TOUCH,

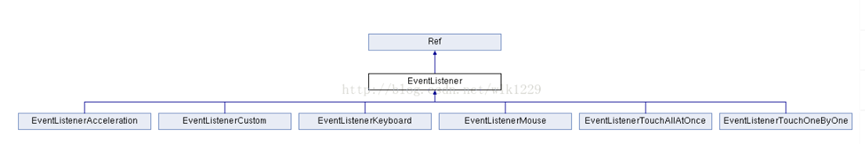
KEYBOARD,

ACCELERATION,

MOUSE,

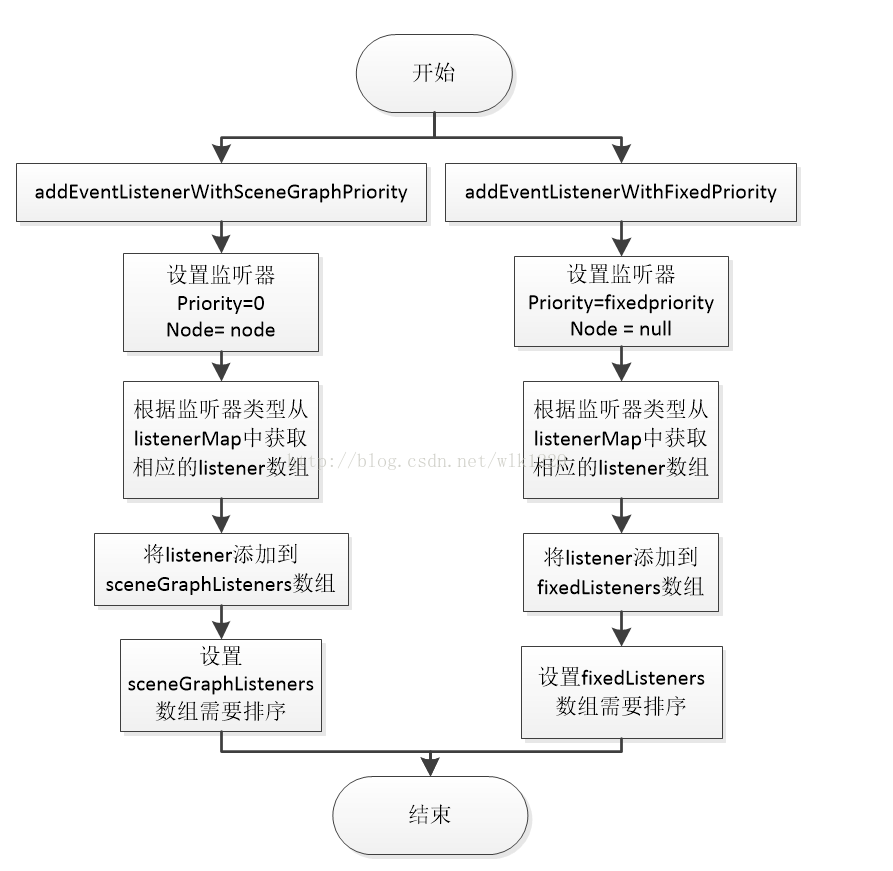
FOCUS,

CUSTOM

}

If a handler program is concerned about an event, create an instance corresponding to the EventListener subclass. For example, EventListenerTouch is corresponding to the Touch Event. The EventListener implements various triggering logics.

We use different methods to add EventListeners, then EventDispatcher has different processes as followed.



Process:

Firstly, we should register Event listeners. Usually, we use addEventListenerWithSceneGrphPriorith() or addEventListenerWithFixedPriority() to register. Then assign the callback functions to each response function of the event listener. We should use codes to implements every functions. The value of \_fixedPriority will help decide the order of execution of callback functions.

Finally, register the event listener to the event distributor \_eventDispatcher. The addEventListener() method will subscribe events. When an event occurs, it will use callback functions.

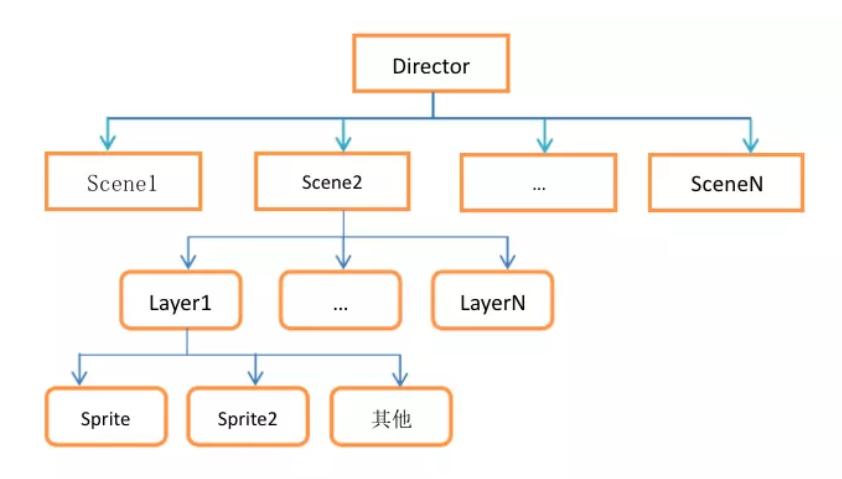
One tip: If one EventListener needs to be reused, we should use clone() method.

### Ⅲ Rendering function

### Rendering Tree

Cocos2d-x adopts a hierarchical management structure.As shown in the figure, the director class directly controls the root node of the game, namely the Scene.Scene controls the change of the Layer, and finally the Layer controls all the nodes displayed.Any two - dimensional game is created by controlling different images, positions, and presentation levels.

Render tree is a tree structure made up of various game elements according to hierarchical relations.Each node has its child nodes.All child nodes move with the parent.Drawing a parent node causes the drawing of a child node.The drawing method of the child node is also related to the attributes of the parent node.



### Rendering Principle

Past rendering systems were drawn by the parent node by calling the underlying OpenGL function through the visit function, causing two problems.First, the flexibility of drawing order is lost.Second, there is no separation between logic and rendering.

Cocos2d-x3.0 made the following changes to the rendering system.

1. Decouple rendering from scene tree.Instead of calling any OpenGL functions in the visit function, the render instructions are put into a queue.This separates the rendering from the game logic and gives the rendering more flexibility.

2. Separate the rendering logic code that calls OpenGL from the main thread and open a separate thread.

3. SpriteBatchNode and ClippingNode to improve efficiency.

4. Customize nodes.

Rendering order

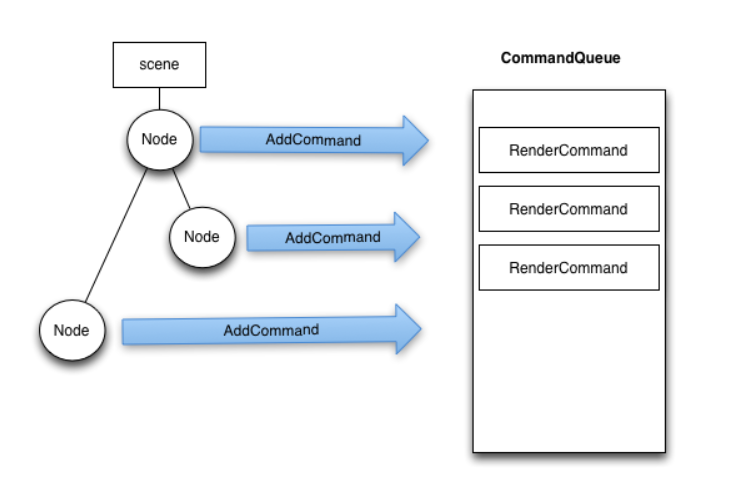
1. The main thread calls drawScene to start drawing the scene.

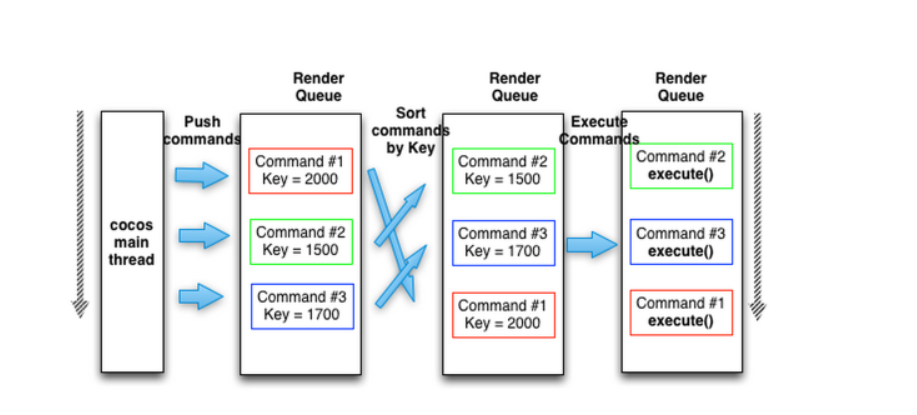
2. Recursively traverse the child nodes of the scene, that is, call the visit function.

3. Call the draw function of each child node.

4.Initialize the render command to put the QuadCommand object into the render queue.

Render logic: first, further process the render command, and execute the render command after the processing is completed





### Text Rendering

1. CCLabelAtlas class

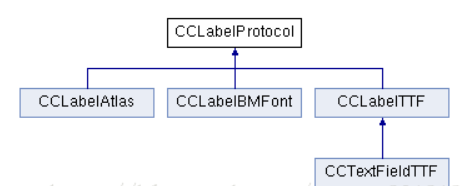
The CCLabelAtlas class is a way of using images as text, which can be defined directly by images.

2．CCLabelBMFont class

The CCLabelBMFont class is also the text rendering label class.Each word in the CCLabelBMFont class is a Sprite class, meaning that each word can have its own spin action and support FNT type files

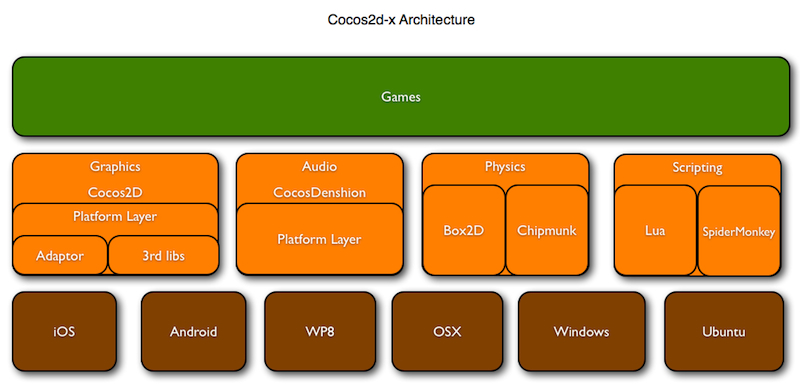
1. CCLabelTTF class

The CCLabelTTF class implements the font label through the system word



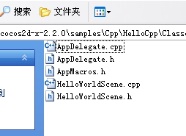
### Ⅳ Cross-platform

To see how cocos achieve cross-plarform, we first look in to the architecture of cocos.



In the graph we can see that there is a layer called platform layer to interact with different platforms. On top of that, is the graphic library of cocos2d-x, which provide us with uniform interfaces so we don’t need to care about the differences between different operation system. With only one set of code, we can compile different versions for different operation system.

Project directory:



We can see that for each platform, cocos has write the entry function. In win32, we use Application::run() to launch your application. In Android, the entry function is Java\_org\_cocos2dx\_lib\_Cocos2dxRenderer.\_nativeInit in platform\android\jni to launch your game.

How does the compiler know which entry point to use?

All code related to platform is in the /cocos/platform directory, in CCPlatformConfig.h we can see such code:

#if defined(\_WIN32) && defined(\_WINDOWS)

#undef CC\_TARGET\_PLATFORM

#define CC\_TARGET\_PLATFORM CC\_PLATFORM\_WIN32

#endif

If we define \_WIN32 and \_WINDOWS which we are in windows platform, the macro variable CC\_TARGET\_PLATFORM is defined as CC\_PLATFORM\_WIN32, so the complier knows where to find the entry point of the program.

This is how cocos implements cross-platform functionbility.

# Development View

This view focus on how cocos was developed. Devemop view include code structure and dependencies, build and configuration management of deliverables, system-wide design constraints, and system-wide standards to ensure technical integrity and so on. In this view, we focus on four aspect of development view which are module organization, release process, source code structure and design patterns.

### ⅠModules of cocos2d-x

By accessing the website of cocos2d-x organization, we obtain the higher-level view of the modules that classify the classes exists in cocos2d-x project. Here is the list of all modules which are independent from each other:



#### 2D nodes

See in this module, we would find

①the classes that actually implement the animation function of 2D games such as AnimationFrame/Animation/AnimationCache which are mentioned in the function view: animation part;

②the polygon classes which helps hold the required data to display Sprites.

③TMXTiledMap classes which are used to parse and render a TMX map

④Menu classes which would establish the menu system in a 2d game

⑤Node ,Scene,Particle, and transition classes which are used to establish the Scene;

#### 3D nodes

①the classes that are used to implement 3d animations are like Animaion3D which are just like those in 2D nodes;

②the classes that are used to establish a model which are like Bone3D, Skeleton3D, Mesh, MeshSkin, Sprite3D,Ray and textureCube classes

③the classes that are used to bound the other classes are like OBB

④the map classes that are like Skybox and terrain which would establish the 3D map/space;

#### Actions

This module is simple for the function it implemented. All of the actions/moves such as Follow, Ease, StopGrid, Waves3D, FlipX3D, MoveBy/MoveTo, FadeIn/FadeOut would be classified into the set of Action module.

#### Audio

This module controls the function of sound, or we called Audio officially. There are only two classes should be classified into the set of Audio module which are AudioEngine that offers a interface to play audio and the SimpleAudioEngine that plays background music and sound effects.

#### Base

This module is multi-functional; here are the classes that involved in this module;

①System release functions: the autoReleasePool class would manage autorelease objects; Configuration class would contain some variables; Controller class represents a connected physical game controller,etc;

②event classes: Event, EventAcceleration, EventCustom, EventListener, EventDispatcher etc classes would implement the event handle function which are mentioned in function view

③script classes: For Lua, Wrapper the script data that should be used to find the handler corresponding to the Lua function by the nativeobject pointer and store the value pointer which would be converted concretely by the different events,then the converted data would be passed into the Lua stack.

④colors, quad, vertex: system classes

#### Network

Network module handles the request and responds between clients and servers; The HttpRequest, SocketIO, Websocket classes would do the job.

#### Physics(Physics 2D)

Physics module implement the physical system of the whole cocos2d; The physicsBody class would create a body affect by physics, and other classes would do the operations on the body to make it perfect like it was affected by the actual physical system or following the physical laws

#### Platform Specific

This module is simple because it has only two classes which handles the file operations;

#### Renderer

This module is about the usage of command which are used to draw batches in one TextureAtlas class.

#### Storage

And this module is for local storage support for the JS bindings. The localStorageFree would allocated resources, and the Set and get classes would set/get item from the JS. Of course it has the release class.

#### UI nodes

This module involves all the components that would show up in UI. For example, Button, Checkbox, Hbix, EditBox, ImageView, etc would be classified into the UI node module.

#### Lua Binding

This module controls the function of building bridge between C and Lua

### Ⅱ Release process

* Since it first released in 2011, Cocos2d-x has released many versions. At the beginning, the primary version and the 2.X versions are maintaining the Objective-C interface style which is consistent with Cocos2d-iphone. However, after 3.0, Cocos began to abandon the Objective-C style, restore C++, and add some C++11 features to the engine interface, such as using clone() method to instead of copy(). There is also a new renderer and time distribution mechanism.
* It is worth noting that users not only can report debugs，but also can submitting patches. The procedure is as followed:
* Download the latest cocos2d-x develop branch from github:
* Apply your changes in the recently downloaded repository
* Commit your changes in your own repository
* Create a new branch with your patch: $ git checkout -b my\_fix\_branch
* Push your new branch to your public repository
* Send a “pull request” to user “cocos2d”

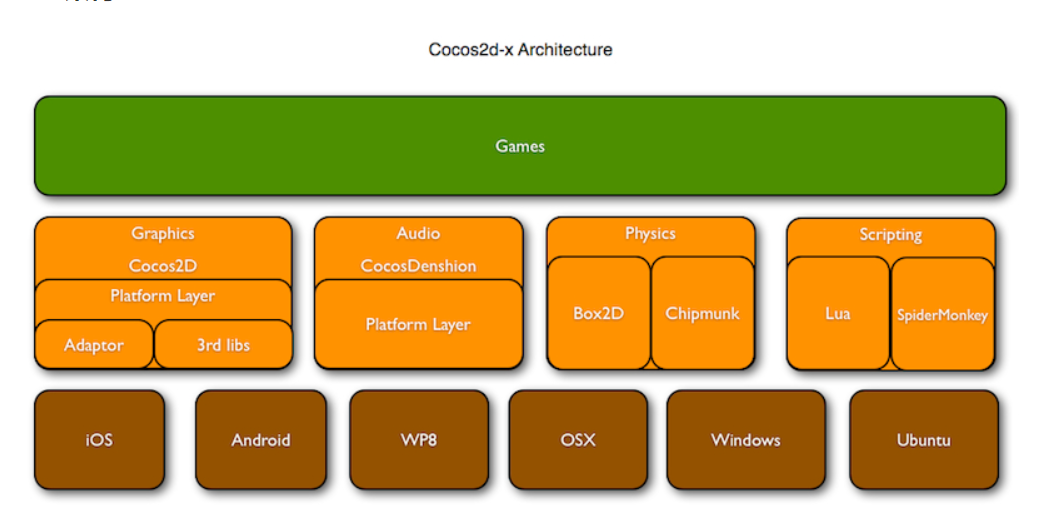
What's more, this mechanism motivates the external contributors as their contribution can be published once adopted. Lastly, it is a great opportunity for the contributors to develop techniques, tools and accumulate knowledge.

Releases are handled on GitHub using the release system and tags. The latest release V3.17 was released on may 17 2018 on GitHub.

Documentation updates are required to be created by the developer with the pull-request. Other documentation changes are handled with the same process as a code update with pull-request and issues.

### Ⅲ Source code structure

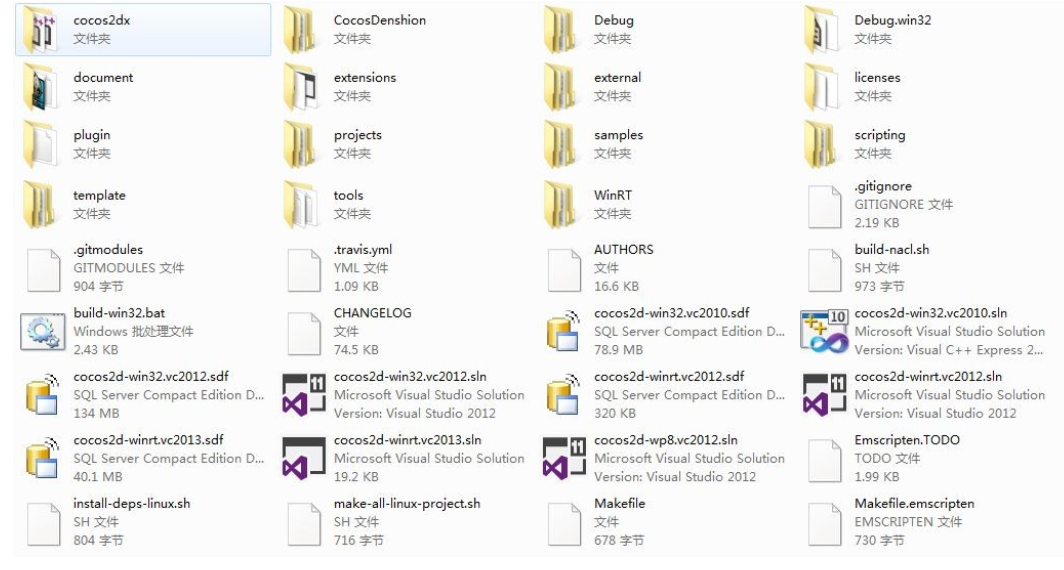
##### The Architecture of cocos2d-x



##### The overall structure of the directory hierarchy to store source code

Cocos2d-x has many updates, and the file directory structure is similar between versions. The following is just an example of a version.

Cocos2d-x 2.2.3 directory structure



cocos2d-x : Home directory of the cocos2d-x framework.

CocosDenshion： source files related with Sound module.

Debug: Debug output directory.

document: You can download the doxygen document system, use it to open the doxygen.config file in this document folder, and reproduce it into the offline API document.

extensions： If you need more control of the graphical user interface, network access, cocosBuilder support, or even 2.5d functionality, you can use using namespace cocos2d::extension.

external： Includes box2d and Chipmunk libraries.

licenses： cocos2d relies on many other open source projects.All the license files are in this directory.

plugin： Store some related plug-ins.

Project: store the project that you create.

samples: this is the file you should start using.Start with Cpp/HelloCpp, and you'll find all the class USES in TestCpp.Lua and js samples are also in this directory.

Scripting : offer Lua and Javascript.The Scripting folder contains the lua official engine from firefox and the SpiderMonkey engine.I know you don't like C++. It's too complex to write.No problem, we have Lua and Javascript.The Scripting folder contains the lua official engine from firefox and the SpiderMonkey engine.

template: includes templates for creating new cocos2d-x projects in different integrated development environments and platforms.Here is a huge collection of templates covering a variety of development environments and platforms!

Tools: includes script files that bind C++ to lua and javascript.

Build-win32.bat: compile the windwos project script for the cocos2d-x engine

CHANGELOG: author revises record document.

Cocos2d-win32. Vc2010. SLN: matching Visual Studio 2010 open.

Vc2012. SLN: matching Visual Studio 2012 open.

.......

##### Overall description of the cocos2d-x framework

As a 2d game engine, cocos2d-x is also composed of the following major concepts:

CCDiretor :

CCDirector class is the organization and control core of the whole game. The running rules of the game, the movement of CCScene (scene), scenery (CCLayer), characters (CCSprite) in the game are all managed by CCDirector.

It contains many funtions:

CCScene \* getRunningScene (void) // gets the currently running scene

Double getAnimationInterval (void)// gets the current FPS

Virtual void setAnimationInterval (double dValue)=0 // set FPS

Bool isDisplayFPS (void) // whether to display FPS at the bottom of the screen

Void setDisplayFPS (bool bDisplayFPS) // set to display FPS at the bottom of the screen

CC\_GLVIEW \* getOpenGLView (void) // get CCEGLView

Void setOpenGLView (CC\_GLVIEW \*pobOpenGLView) // set CCEGLView

Bool isPaused (void) // is of course appropriate for pausing

Unsigned int getFrames (void) // gets the number of all called Frames from director

CcDirectorProjection getProjection (void) // gets OpenGl's projection

Void setProjection (ccDirectorProjection kProjection) // sets Opengl projection

If the new scene is pushed, the old scene will not receive the cleanup event. If the new scene is replaced, the old scene will be able to receive the cleanup event.

CCSize getWinSize (void) // gets the size of Open gl view, in points

CCSize getWinSizeInPixels (void) // gets the open gl view size in pixels

CCSize getDisplaySizeInPixels (void) // gets the size of the display area of open gl view in pixels

Void reshapeProjection (const CCSize &newWindowSize) // change the size of projection

CCPoint convertToGL (const CCPoint &obPoint) // converts UIKIT coordinate system into opengl coordinate system

CCPoint convertToUI (const CCPoint &obPoint)// converts the opengl coordinate system to UIKIT coordinate system

Float getZEye (void) // gets the position of the default lens Z

Void runWithScene (CCScene \*pScene) // run scene

Void pushScene (CCScene \*pScene) //push scene

Void popScene (void)//pop scene

Void replaceScene (CCScene \*pScene) // replaceScene

Void pause (void pause) // pause the game

Void resume (void) // reply game

Virtual void stopAnimation =0 // stop the animation

Virtual void startAnimation (void)=0 // start the animation

Void drawScene (void) // each frame is called without having to call itself

Void purgeCachedData (void) // delete cached data

Void setGLDefaultValues (void) // sets the default for Opengl

Void setAlphaBlending (bool bOn) / / set whether to enable opengl alpha channel

Void setDepthTest (bool bOn)// sets whether to test depth of field

Virtual void mainLoop (void)=0// the game mainLoop

Void applyOrientation (void) // is the device orientation setting in effect

CcDeviceOrientation getDeviceOrientation (void) // getDeviceOrientation

Void setDeviceOrientation (ccDeviceOrientation kDeviceOrientation) // set the game landscape

Bool enableRetinaDisplay (bool enabled) // set whether to enable retinal screen support

Bool isRetinaDisplay ()// whether to enable retinal screen support

Void resetDirector () // reset game

CCCamera

Every node in the game needs a camera. When the node zooms, rotates and changes its position, it needs to cover the camera and let the node re-render through the camera.

CCCamera is applied to each CCNode.The gluLookAt function in OpenGL is used to locate the camera when viewed from different perspectives.

It contains funtions:

Char \* description (void) // returns the description of the current camera

Void restore (void) // reset the camera to the default position

Void locate (void) // place the camera at the set eye coordinates

Void setEyeXYZ (float fEyeX, float fEyeY, float fEyeZ) // sets the camera eye coordinates

Void setCenterXYZ (float fCenterX, float fCenterY, float fCenterZ)// sets the coordinates of the center

Void setUpXYZ (float fUpX, float fUpY, float fUpZ)// sets the coordinates of up

Void getEyeXYZ (float \*pEyeX, float \*pEyeY, float \*pEyeZ) // gets the coordinates of the eye

Void getCenterXYZ (float \*pCenterX, float \*pCenterY, float \*pCenterZ)// get the coordinates of the center

Void getUpXYZ (float \*pUpX, float \*pUpY, float \*pUpZ)// gets the coordinates of Up

(CCScene)

In the game, the scene is the level.The level consists of characters and backgrounds.In the cocos2d-x engine, the scene stores the scene that needs to be rendered, the task role and the menu, which can be rendered together, destroyed together and used together by scene switching.

It contains functions :

bool init()

static CCScene \* node (void)

CCLayer

The set is the background of the scene.This is the manual layering of the game scenes.Each game scene can have many layers, each layer has its own responsibilities and tasks, and different elements can be placed on each layer, including text, spirit and menu, etc. Through the combination of layers and layers, we can easily control and display various interfaces.

It contains functions :

Bool init () // initialize the function

Callback function when virtual void onEnter ()// enters

Callback function when virtual void onExit ()// leaves

Virtual void onEnterTransitionDidFinish () / / to enter after the callback function

Virtual bool ccTouchBegan (CCTouch \*pTouch, CCEvent \*pEvent)// single-touch screen event

Virtual void ccTouchesBegan (CCSet \*pTouches, CCEvent \*pEvent)// multi-touch events

Virtual void ccTouchesMoved (CCSet \*pTouches, CCEvent \*pEvent)// move events on the touch screen

Virtual void ccTouchesEnded (CCSet \*pTouches, CCEvent \*pEvent)// touch screen movement time ends

Virtual void ccTouchesCancelled (CCSet \*pTouches, CCEvent \*pEvent)// touchscreen event cancelled

Static CCLayer \* node (void)// generates layer

CCSprite

Sprites are the main objects of the whole game development and processing, including the protagonist and the enemy, NPC and so on, and even a piece of cloud or bird floating randomly. Technically, they are also sprites, because the sprites in the cocos2d-x are a picture that can change constantly. These changes include position change, rotation, zoom, movement and so on.

It contains functions :

Virtual void setTexture (CCTexture2D \*texture)// set the Sprite texture

Virtual CCTexture2D \* getTexture (void)// get the Sprite texture

Bool initWithTexture (CCTexture2D \*pTexture)// initialize the Sprite according to the texture

Bool initWithTexture (CCTexture2D \*pTexture, const CCRect &rect)// initialize the Sprite according to the region specified by the texture

Bool initWithSpriteFrame (CCSpriteFrame \*pSpriteFrame)// initialize SpriteFrame according to SpriteFrame

Bool initWithSpriteFrameName (const char \*pszSpriteFrameName)// initialize the spriteFrame by its name

Bool initWithFile (const char \*pszFilename)// initialize sprites according to the plist file

Bool initWithFile (const char \*pszFilename, const CCRect &rect)// initialize Sprite according to plist file

Bool initWithBatchNode (CCSpriteBatchNode \*batchNode, const CCRect &rect)// batch initialize sprites according to the node

Void setDisplayFrameWithAnimationName (const char \* animationName, int frameIndex)

Static CCSprite \* spriteWithTexture (CCTexture2D \*pTexture) // generate sprites according to the texture

Static CCSprite \* spriteWithTexture (CCTexture2D \*pTexture, const CCRect &rect)// generate sprites according to the region specified by the texture

Static CCSprite \* spriteWithTexture (CCTexture2D \*pTexture, const CCRect &rect, const CCPoint&offset)// generate Sprite based on the region and point offset specified by the texture

Static CCSprite \* spriteWithSpriteFrame (CCSpriteFrame \*pSpriteFrame)// generate SpriteFrame based on SpriteFrame

Static CCSprite \* spriteWithSpriteFrameName (const char \* pszSpriteFrameName) / / generated according to the name of SpriteFrame elves

Static CCSprite \* spriteWithFile (const char \*pszFileName)// generate CCSprite according to the plist file

Static CCSprite \* spriteWithFile (const char \*pszFileName, const CCRect &rect)// generate sprites according to the plist file

Static CCSprite \* spriteWithBatchNode (CCSpriteBatchNode \*batchNode, const CCRect &rect)// generate sprites in batches according to the node

CCAction

The action that the character has, use commonly when wizard produces action, be like move, release magic etc.

It contains functions :

Virtual CCObject \* copyWithZone (CCZone \*pZone)// clone object

Whether the virtual bool isDone (void)// action has been replaced

Virtual void startWithTarget (CCNode \*pTarget)// sets the object associated with the action, which is called before the action runs

Virtual void stop (void)// stop action

Virtual void step (ccTime dt)// sets the interval time of the action

CCNode \* getTarget (void)// gets the object associated with the action

Void setTarget (CCNode \*pTarget)// sets the associated object of the action

Int getTag (void)// gets the tag of the action

Void setTag (int nTag)// sets the tag of the action

Static CCAction \* action ()// generate action

### Design patterns

#### Sec construction mode

**What and why is sec construction mode?**

In C++ we all know that to construct an object, we need to use new operator to call the constructor of the object. But there is some problem, if we want to use a image to create a sprite, what if the image does not exist? Usually we use a try-catch to handle the exception, but the constructor does not have a return value, so it’s different to know whether the operation success. On this condition, cocos uses the sec construction mode. The first step is use new to allocate memory of the object , second step is call init method to initiate the object.

Here is the code of Sprite::create():

Sprite\* Sprite::create()

{

//分配内存

Sprite \*sprite =newSprite();

//init初始化

if(sprite && sprite->init())

{

//内存管理的工作

sprite->autorelease();

return sprite;

}

CC\_SAFE\_DELETE(sprite);

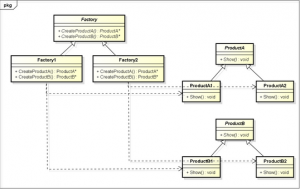
Return nullptr;

}

As we can see, use sec construction mode also provide us with a way to manage the memory, because if the init() return true, we will add the object the autorelease pool which will auto release memory if the object has no reference attach to it. This makes developer don’t have to worry about memory leak and focus the application.

#### Factory mode

UML diagram of factory mode



The above is a description of the factory mode. Let's take a look at the application of the factory model in cocos2dx. Refer to the book cocos2d-x advanced development tutorial: "The factory method is a classic design pattern in programming, referring to the base class. Only define the interface that creates the object, deferring the actual implementation to a subclass. Here, we will promote it a bit, referring to all static functions that generate and return an object." Everything that generates and returns an object's static function is a factory method. So, is there a lot of such methods in cocos2dx? For example, create a scene's createScene function, create a majority object's create function, a classic factory method like this

1Sprite\* factoryMethod()

2{

3Sprite\* ret =newSprite();

4//在这里对 ret 对象进行必要的初始化操作

5ret->autorelease();

6returnret;

7}

The application scenario that uses the factory pattern in our own program can be like this: we have to create a lot of bullets. If you use Sprite's create method to allocate memory each time, and the bullet is freed when the bullet is destroyed, the creation method is not efficient. If we use the factory method to accomplish this, our factory method facade maintains a container that holds the destroyed bullets and takes them out of the container when new bullets are needed. Depending on the type of bullet, change the texture, reset the position, reset the flight speed and direction, and then launch. If there are no bullets destroyed in the container, initialize one, so that you don't need new, delete each time, the number of bullets in the memory is certain, which can improve the efficiency of the