1:Cocos Creator项目结构：

ProjectName（项目文件夹）

├──assets

├──library

├──local

├──settings

├──temp

└──project.json

assets资源文件夹，放置游戏中所有的本地资源，脚本和第三方库文件，只有在该目录下的内容才能显示在资源管理器中，

library是将assets中的资源导入后生成的，在这里文件的结构和资源的格式将被处理成最终游戏发布时需要的形式。如果您使用版本控制系统管理您的项目，这个文件夹是不需要进入版本控制的。当 library 丢失或损坏的时候，只要删除整个 library 文件夹再打开项目，就会重新生成资源库。

local 文件夹中包含该项目的本地设置，包括编辑器面板布局，窗口大小，位置等信息。您不需要关心这里的内容，只要按照您的习惯设置编辑器布局，这些就会自动保存在这个文件夹。一般 local 也不需要进入版本控制。

settings 里保存项目相关的设置，如 构建发布 菜单里的包名、场景和平台选择等。这些设置需要和项目一起进行版本控制。

project.json 文件和 assets 文件夹一起，作为验证 Cocos Creator 项目合法性的标志。只有包括了这两个内容的文件夹才能作为 Cocos Creator 项目打开。而 project.json 本身目前只用来规定当前使用的引擎类型和插件存储位置，不需要用户关心其内容。这个文件也应该纳入版本控制。

2:项目设置：设置会保存在项目的 settings/project.json 文件里。如果需要在不同开发者之间同步项目设置，请将 settings目录加入到版本控制。分组管理主要为 [碰撞体系统](http://docs.cocos.com/creator/manual/zh/physics/) 提供分组支持；

3：对布局的修改放在项目所在文件夹下的 local/layout.windows.json 文件中。

4：脚本属性：Cocos Creator 规定一个节点具有的属性都需要写在 properties 代码块中

5：场景资源的释放；资源加载的策略（延迟加载与否）；

6：cocosCreator目前支持的图片格式包括JPG和PNG两种；

7：创建预制资源：直接将节点从 层级管理器 拖到 资源管理器。

8：图集资源：使用多张图片合成的图集作为美术资源的优势：减少游戏包体和内存占用，减少cpu运算时间，提高效率。使用专门的图集制作软件生成图集时选择cocos2d-x格式的plist文件，最终得到的图集文件是同名的plist和png。将这两个文件同时拖到资源管理器中，调用即可。使用cocosCreator生成图集：在 资源管理器 中右键，可以在如下菜单中找到 新建 -> 自动图集配置 的子菜单，点击菜单将会新建一个类；自动图集资源 将会以当前文件夹下的所有 SpriteFrame 作为碎图资源，以后会增加其他的选择碎图资源的方式。 如果碎图资源 SpriteFrame 有进行配置过，在打包后重新生成的 SpriteFrame 将会保留这些配置。

9：资源导出的文件包括场景文件和预制文件。会导出一个zip的压缩文件。

10：字体：动态字体：支持TTF格式的动态字体，将其拖到资源管理器中即可完成字体资源导入。位图字体：务必将fnt文件和png文件同时拖到资源管理器。

11：粒子资源：将cocos2d支持的粒子.plist文件直接放到工程资源目录下。可以直接将粒子文件拖到层级管理器或者拖到另一个节点的file属性中。注意：不支持 .plist 文件中的 sourcePosition 属性的导入。

12：声音资源：比较大的音频和背景音乐建议使用DOM Audio模式。

13：sprine骨骼动画资源：是由Sprine所导出的数据格式，导入所需资源：.json骨骼数据；.png图集纹理；.txt/.atlas图集数据。创建骨骼资源动画：1：从资源管理器拖到层级管理器，2：从资源管理器拖到场景；3: 从 **资源管理器** 里将骨骼动画资源拖动到已创建 Spine 组件中 Skeleton Data 属性中.

14: DragonBones 骨骼动画资源是由 [DragonBones](http://dragonbones.com/) 编辑导出的数据格式。导入所需资源：.json骨骼数据；.json图集数据；.png图集纹理。创建方式同上，只是第三种方式变为 将动画资源拖到DragonBones 组件的 Dragon Asset 属性中。从 **资源管理器** 里将图集数据拖动到 DragonBones 组件的 Dragon Atlas Asset 属性中。

15：瓦片图资源（地图资源）：所需资源：.tmx地图数据；.png图集纹理；.tsx tileset数据配置文件（部分tmx文件需要）；创建方式同上，只有第三种方式：从 资源管理器 里将地图资源拖动到已创建 TiledMap 组件的 Tmx File 属性中;

16:锚点（Anchor）：它决定了节点以自身约束框中的哪一个点作为整个节点的位置；（0.5，,05）表示约束框的中心；

17：场景编辑器的渲染节点的根节点是canvas；只负责挂载脚本执行逻辑的逻辑节点不包含任何渲染相关内容，与canvas并列，

18：脚本开发：

name: "sprite"给cc.class命名

print: function () { }创建实例方法；

extends: Shape 继承Shape；

properties{}：要声明属性，仅需要在 cc.Class 定义的 properties 字段中，填写属性名字和属性参数即可;属性检查器中可以看到定义的属性；

cc.Float：属性类型指定模式；

this.node获取组件所在的节点；

this.getComponent(cc.Label);获取其他组件；

player: {

default: null,

type: cc.Node

}利用属性检查器设置节点：将层级管理器上的任意节点拖到属性检查器中的控件，这样就可以直接在脚本中访问该节点。

this.node.children;将对象放在一个统一的父物体下，然后通过父物体来获得子物体。或者使用this.node.getChildByName("Cannon 01");还可以使用cc.find("Cannon 01/Barrel/SFX", this.node);根据传入路径进行逐级查找对象组件。

cc.find("Canvas/Menu/Back");全局通过名字查找对象组件。

this.node.active = false;关闭节点；当一个节点关闭，所有组件都会被禁用，其子节点和子节点上的组件也被禁用。

this.node.parent = parentNode;更改节点的父节点；

this.node.children 将返回节点的所有子节点数组。不会返回子节点的子节点  
this.node.childrenCount 将返回节点的子节点数量。不会返回子节点的子节点

节点和组件接口http://docs.cocos.com/creator/manual/zh/scripting/basic-node-api.html

this.enabled：是否每帧执行该组件的 update 方法，同时也用来控制渲染组件是否显示

update(dt)：作为组件的成员方法，在组件的 enabled 属性为 true 时，其中的代码会每帧执行

onLoad()：组件所在节点进行初始化时（节点添加到节点树时）执行

start()：会在该组件第一次 update 之前执行，通常用于需要在所有组件的 onLoad 初始化完毕后执行的逻辑，通常用于初始化一些中间状态的数据，这些数据可能在 update 时会发生改变，并且被频繁的 enable 和 disable。

lateUpdate：要在动效（如动画、粒子、物理等）更新之后才进行一些额外操作，或者希望在所有组件的 update 都执行完之后才进行其它操作；

onEnable：当组件的 enabled 属性从 false 变为 true 时，或者所在节点的 active 属性从 false 变为 true时，会激活 onEnable 回调。倘若节点第一次被创建且 enabled 为 true，则会在 onLoad 之后，start 之前被调用。

onDisable ：当组件的 enabled 属性从 true 变为 false 时，或者所在节点的 active 属性从 true 变为 false时，会激活 onDisable 回调。

onDestroy ：当组件或者所在节点调用了 destroy()，则会调用 onDestroy 回调，并在当帧结束时统一回收组件。

cc.director.loadScene("MyScene");场景进行加载和切换;

cc.game.addPersistRootNode(myNode);引擎同时只会运行一个场景，当切换场景时，默认会将场景内所有节点和其他实例销毁。如果我们需要用一个组件控制所有场景的加载，或在场景之间传递参数数据，就需要将该组件所在节点标记为「常驻节点」，使它在场景切换时不被自动销毁，常驻内存。(myNode变为常驻节点)；

cc.game.removePersistRootNode(myNode);取消常驻节点；

cc.director.loadScene("MyScene", onSceneLaunched);加载场景时回调onSceneLaunched函数；在场景加载后可以用来进一步的进行初始化或数据传递的操作。常用来配合常驻节点。

cc.director.preloadScene场景预加载；

cc.Asset：资源属性声明；所有继承自 cc.Asset 的类型都统称资源，如 cc.Texture2D,

 cc.SpriteFrame, cc.AnimationClip, cc.Prefab 等。可在脚本属性中定义一个Asset属性，之后就可以在属性检查器中看到这两个属性，之后从资源管理器中将资源拖到属性检查器对应属性。

cc.loader.loadRes：动态加载资源：所有需要通过脚本动态加载的资源，都必须放置在

 resources 文件夹或它的子文件夹下。resources 需要在 assets 文件夹中手工创建，并且必须位于 assets 的根目录；

cc.loader.releaseAsset(spriteFrame);释放特定的Asset实例；

var remoteUrl = "http://unknown.org/someres.png";加载远程资源；还可以用绝对路径加载本地资源；这种加载方式只支持图片、声音、文本等原生资源类型

this.node.on()：注册监听事件；除了使用 on 监听，我们还可以使用 once 方法。once 监听在监听函数响应后就会关闭监听事件。使用off关闭监听

emit 和 dispatchEvent：发射事件。两者的区别在于，后者可以做事件传递。在发送用户自定义事件的时候，不要直接创建 cc.Event 对象，因为它是一个抽象类，创建 cc.Event.EventCustom

 对象来进行派发。

event.stopPropagation()：终止事件传递；

<http://docs.cocos.com/creator/manual/zh/scripting/events.html>：事件对象的API

<http://docs.cocos.com/creator/manual/zh/scripting/internal-events.html>：鼠标和触摸对象

<http://docs.cocos.com/creator/manual/zh/scripting/player-controls.html>：键盘和重力事件

<http://docs.cocos.com/creator/manual/zh/scripting/actions.html>：动作系统

<http://docs.cocos.com/creator/manual/zh/scripting/action-list.html>：动作列表以及API

<http://docs.cocos.com/creator/manual/zh/scripting/scheduler.html>：计时器

schedule：开始一个计时器scheduleOnce：开始一个只执行一次的计时器

unschedule：取消一个计时器unscheduleAllCallbacks：取消这个组件的所有计时器

executionOrder：设置组件的执行顺序；executionOrder 越小，该组件相对其它组件就会越先执行。executionOrder 默认为 0，因此设置为负数的话，就会在其它默认的组件之前执行

<http://docs.cocos.com/creator/manual/zh/scripting/network.html>：网络接口

<http://docs.cocos.com/creator/manual/zh/scripting/pooling.html>：对象池

var Rotate = require("Rotate");引用模块；每一个单独的脚本文件就是一个模块；

extends: Rotate表示继承自Rotate；

module.exports = cfg（module.exports 的值可以是任意 JavaScript 类型）;普通js文件导出模块必加；creator脚本的默认为导出。

插件：所有插件脚本都会在普通脚本之前加载，可以用来定义一些全局变量；

<http://docs.cocos.com/creator/manual/zh/scripting/reference/class.html：CCClass>构造方法，对象参数，静态变量，重写，属性

默认情况下，是否显示在 属性检查器 取决于属性名是否以下划线 \_ 开头。如果以下划线开头，则默认不显示在 属性检查器，否则默认显示。如果要强制显示在 属性检查器，可以设置 visible 参数为 true:

editor 只能定义在 cc.Component 的子类。

<http://docs.cocos.com/creator/manual/zh/scripting/reference/attributes.html>：属性参数

（不断增加中，直到没办法加....有点乱）

1.场景加载  
cc.director.loadScene('场景名称');//场景跳转  
cc.director.preloadScene('场景名称');//预加载场景

cc.director.getScene();//获取当前场景  
  
2.获取节点  
var node = cc.find("Canvas/bg");//通过访问路径来获取节点

var a = this.node.getChildByName('name');//通过名字获取子节点  
node.getComponent(cc.Label).string = 'abc';//获取节点上的组件值

var a = cc.find("Canvas/bg").getComponent(cc.Sprite);//通过访问路径来获取节点，及获取该节点的指定组件

this.node .getChildByName('节点名称').getComponent(cc.Label)//通过节点名获取子节点，获取该节点指定组件

var a = this.node;//获取当前脚本所在的节点  
var a = this.node.parent;//获取父节点  
var a = this.node.getChildByTag(1001);//通过标签获取子节点  
var a = cc.find("bg/score",this.node);//通过指定节点下的路径获取节点

var a = this.node.children;//获取所有子节点  
var a = this.node.childrenCount;//获取子节点数量  
var a = cc.director.getScene();//获取场景主节点

var a = cc.instantiate(node);//克隆节点  
this.node.parent = cc.find('Canvas');//绑定父节点  
this.node.addChild(nodeName,zIndex,tag);//添加子节点,可设置层级和标签  
this.node.removeChild(nodeName);//通过名字移除子节点  
this.node.removeChildByTag (nodeTag);//通过标签移除子节点  
this.node.destroy();//销毁节点  
this.node.isValid;//判定节点是否可用

this.node.removeChild(newNode);//移除节点中指定的子节点  
this.node.removeChildByTag(1001);//通过标签移除节点中指定的子节点  
this.node.removeAllChildren();//移除所有子节点  
this.node.destroyAllChildren();//销毁所有子节点  
this.node.cleanup();//停止所有正在播放的动作和计时器  
var sprites = this.node.getComponentsInChildren(cc.Label);//递归查找自身及所有子节点中指定类型的组件

3.获取节点位置，设置节点  
var a = node.getPositionX();或 getPositionY() //获取节点的X轴或Y轴坐标  
var a = node.getScaleX(); 或getScaleY() //获取节点的X轴或Y轴缩放比例  
node.x = 100;//设置节点x轴坐标  
node.y = 100;//设置节点y轴坐标  
node.setPosition(x,y); //设置节点坐标  
node.rotation = 90; //设置节点旋转角度  
node.scaleX = 2; //设置节点x轴缩放倍数  
node.scaleY = 2; //设置节点y轴缩放倍数  
node.setScale(2); //设置节点整体缩放倍数  
node.width = 100; //设置节点宽度大小  
node.height = 100;  //设置节点高度大小  
node.setContentSize(100, 100); //设置节点宽高尺寸大小  
node.anchorX = 1; //设置节点x轴锚点坐标  
node.anchorY = 0; //设置节点y轴锚点坐标  
node.setAnchorPoint(1, 0); //设置节点锚点坐标  
node.opacity = 128; //设置节点透明度大小（0-255）  
node.setOpacity(20); //设置节点透明度（0~255）  
node.color = new cc.color(100,100,100,255); //设置节点颜色（R,G,B,透明度）  
if (cc.isValid(this.label.node) ) //判定节点是否存在  
node.destroy(); //销毁节点  
this.cannons = [];  
this.cannons = node.getChildren(); //获取所有子节点  
this.cannons = node.getChildrenCount(); //获取子节点数量  
node.active = false; //关闭节点(隐藏节点)  
cc.game.addPersistRootNode(myNode); //常驻节点（全局变量）  
cc.game.removePersistRootNode(myNode); //取消常驻节点  
  
4.动作操作

cc.show()//立即显示  
cc.hide ()//立即隐藏  
cc.toggleVisibility()//显隐切换  
cc.fadeIn(1)//渐显效果  
cc.fadeOut(1)//渐隐效果  
cc.delayTime(1)//等待1秒  
node.runAction(cc.moveTo(1,0,0)); //移动到当前节点（时间（s），X轴坐标，Y 轴坐标）  
node.runAction(cc.scaleTo(1,0.7,0.8));//缩放到当前倍数节点（时间（s），X轴倍数，Y 轴倍数）  
node.runAction(cc.rotateTo(1,160,160));//旋转到指定角度（时间（s），X轴角度，Y 轴角度）  
node.runAction(cc.skewTo(1,5,-5));//变化节点倾斜度（时间（s），X轴倾斜度，Y 轴倾斜度）

node.runAction(cc.fadeTo(2,0));//变化当前节点的透明度（时间（s），透明度）

node.runAction(cc.tintTo(2,255,255,0));//变化当前节点颜色（时间，R,G,B）  
node.stopAllActions();//停止所有动作  
//自定义动作  
var action = cc.moveTo(2, 100, 100);// 创建一个移动动作  
node.runAction(action);// 执行动作  
node.stopAction(action);// 停止一个动作

cc.sequence(action1,action2); //按顺序连续执行，先action1，后action2

cc.spawn(action1，action2); //同时执行，action1和action2一起执行

cc.repeatForever(cc.sequence(action1,action2)); //一直重复括号里的动作

5.计时器  
//只用1次的计时器,2秒后执行  
        this.scheduleOnce(function(){  
            //一条或多条执行语句

        },2); //(function(){},时间（s）)

//每隔5秒执行1次  
        this.schedule(function(){  
            //一条或多条执行语句

        },5);

//计算多次的计时器（1秒后，以0.1秒的执行间隔，执行10次）  
        this.schedule(function(){  
            //一条或多条执行语句

        },0.1,10,1); //(function(){},间隔时间，次数，多久后开始)

this.unscheduleAllCallbacks(this);//停止某组件的所有计时器

//自定义定时器执行内容（相比常规使用的定时器优势是：方便随时开启或关闭）  
var cb= function(){  
    //do something  
};  
this.schedule(cb,1);//启动定时器  
this.unschedule(cb);//取消定时器  
  
  
6.事件监听  
  
(开始：'touchstart'，移动：'touchmove'，结束：'touchend'，取消：'touchcancel')  
node.on('touchstart',function(event){  
    this.doSomething();  
},this);    
var a = event.getID();//获取触点的ID  
var a = event.getLocationX();//获取触摸点的坐标X  
var b = event.getLocationY();//获取触摸点的坐标Y  
  
cc.eventManager.addListener({  
            event: cc.EventListener.KEYBOARD/TOUCH\_ONE\_BY\_ONE,myfunction},self.node);  
  
  
7.定义全局变量  
  
window.DEFAULT\_IP = "192.168.1.1";//任意脚本里可定义全局变量  
//任意脚本里可定义全局变量  
window.G = {  
    a: null,  
    b: null,  
};  
//任意脚本里可访问全局变量（切记定义全局变量的那个脚本已执行过）  
G.a = 0;  
G.b = 0;

var something = require（‘something’）；

cc.game.addPersistRootNode(myNode);//常驻节点,必须位于层级的根节点（也可算全局节点吧）

module.exports = {

         config: 123

}

8.分辨率  
//获得设备分辨率  
var b = cc.director.getWinSizeInPixels()  
var bx = b.width  
var by = b.height

cc.view.getCanvasSize().width;//获得设备分辨率的宽度  
cc.view.getCanvasSize().height;//获得设备分辨率的高度  
cc.director.setDisplayStats(true);//显示帧数信息

9.音频控制  
cc.audioEngine.playMusic(this.BGAudio,true);//播放音乐（true代表循环）  
cc.audioEngine.stopMusic()//停止播放背景音乐  
cc.audioEngine.playEffect(this.ClickAudio,false);//播放音效（false代表只播放一次）  
cc.audioEngine.stopEffect(音效变量名);//停止指定音效（需要先把音效赋值给变量）  
cc.audioEngine.AllEffects();//停止所有音效  
cc.audioEngine.setMusicVolume(参数);  //设置背景音乐的音量（该参数范围是0到1）  
cc.audioEngine.setEffectsVolume(参数);  //设置音效的音量（该参数范围是0到1）  
  
10.存档操作  
cc.sys.localStorage.setItem('存储标识名',变量名);//存储存档数据  
var a = cc.sys.localStorage.getItem('存储标识名');//读取存档数据  
cc.sys.localStorage.removeItem('存储标识名');//擦除存档数据  
userData = {  
    name: 'Tracer',  
    level: 1,  
    gold: 100  
};  
cc.sys.localStorage.setItem('userData', JSON.stringify(userData));//存取复杂对象数据  
var userData = JSON.parse(cc.sys.localStorage.getItem('userData'));//读取复杂对象数据

11.判断平台

cc.sys.isNative  //是否是本地  
cc.sys.isBrowser  //是否是网页  
cc.sys.isMobile  //是否是移动系统  
cc.sys.platform  //正在运行的平台  
cc.sys.language  //当前运行系统的语言  
cc.sys.os  //当前正在运行的系统  
cc.sys.OS\_IOS  //是否是IOS系统  
cc.sys.OS\_ANDROID  //是否是android系统  
cc.sys.OS\_WINDOWS  //是否是windows系统  
cc.sys.openURL('Http://www.baidu.com');  //打开网页

12.监听和发射事件

this.node.pauseSystemEvents(true);//暂停节点系统事件  
this.node.resumeSystemEvents(true);//恢复节点系统事件  
this.node.targetOff(this);//移除所有注册事件

触摸监听：开始'touchstart',移动'touchmove',结束'touchend',取消'touchcancel'  
var pos = event.getLocation();//获取触摸点的坐标(包含X和Y)  
var x = event.getLocationX();//获取触摸点的X坐标  
var y = event.getLocationY();//获取触摸点的Y坐标  
var a = event.getID();//获取触点的ID

鼠标监听：鼠标按下'mousedown',移入节点'mouseenter',节点中移动'mousemove',移出节点'mouseleave,'松开鼠标'mouseup'  
var a = event.getScrollY();//获取滚轮滚动的 Y 轴距离，只有滚动时才有效  
var a = event.getLocation();//获取鼠标位置对象，对象包含 x 和 y 属性

输入框监听：获得焦点'editing-did-began',文字变化'text-changed',失去焦点'editing-did-ended',按下回车'editing-return'

属性变化监听：位置'position-changed',宽高 'size-changed',旋转'rotation-changed',缩放'scale-changed'

ScrollView控件监听：滚动中'scrolling',停止滚动'scroll-ended'

用户自定义事件:  
this.node.on('事件名',function,this);//注册监听  
this.node.emit('事件名');//发送监听广播  
this.node.off('事件名',function,this);//关闭监听

//注册带参数监听  
this.node.on('事件名',function(event){

“具体方法函数内容”

},this);  
//发送带参数的监听  
this.node.emit('事件名',{id:1001});  
cc.eventManager.addListener(listener, node);//添加事件  
cc.eventManager.removeListener((listener);//移除事件

13.其他操作  
cc.director.pause();//暂停  
cc.director.resume();//继续  
cc.director.end();//退出整个应用  
cc.log(变量)  或 console.log(something);//输出想要的信息  
let self = this;//锁定当前使用的this指向  
node. \_localZOrder;//层级获取  
node. \_localZOrder;//层级改变  
cc.find('canvas/map' + num)//读取带变量的路径  
cc.sys.openURL('Http://www.baidu.com');//打开网页