**Cocos基础**

**游戏引擎：工具的集合**

**策划设计——>开发——>预览——>调试——>发布**

**可视化 组件**

**Cocos的使用：**

1. **创建项目：{**
2. **空白项目**
3. **浏览 选择项目存放的路径**
4. **项目的命名 项目名称\_项目时间（数字、字母、下划线）**
5. **点击新项目**
6. **资源管理器 左下角 新建场景 scene——> 双击进入场景**

**}**

1. **层级管理器：设计图纸**

**存在父子级关系：**

**对子节点进行一些操作时，影响不到父节点以及同级节点**

**对父节点进行一些操作的时候，会影响它的所有子节点**

**子节点的层级在父节点的下面，下面的节点会覆盖住上面的节点，后者居上**

**注意:都放在 Canvas 下面**

1. **场景编辑器：施工现场**

**可视化窗口**

1. **属性检查器：图纸上数据的部分**

**锚点**

1. **动画编辑器：制作动画的地方**

**序列帧动画**

1. **添加Animation组件**
2. **创建Clip文件**
3. **把Clip文件拖到Anmation组件 默认播放位置**
4. **把自动播放勾选起来**
5. **资源管理器：管理资源 管理素材 仓库 存放的是项目需要的到的东西 图片、字体、音乐、音效、 场景、脚本等等…**

**Node 组件：**

**不需要手动添加，只要往层级管理器里丢资源，自动添加node组件**

**能直观的看到节点的属性，尺寸颜色角度锚点倾斜角度……**

**Sprite组件：**

**不需要手动添加，只要往层级管理器里丢资源，自动添加Sprite组件**

**管理操作图片资源**

**Type渲染模式：**

**九宫格**

**平铺**

**填充：{**

**左右填充**

**上下填充**

**扇形填充**

**}**

**Lable组件：**

**需要加在哪里，就往哪个节点上添加**

**锚点：字体往左或者往右延伸、以及中间往两边走，中间往两边扩散**

**Button组件：**

**交互状态选true（打钩）**

**监控的过渡类型：{**

**颜色变化**

**更换图片**

**按钮缩放**

**}**

**脚本{**

**注意：不要重复绑定，不管重复绑定的节点还是自身**

**命名格式：**

1. **需要挂载在某个节点时，首字母大写**
2. **不需要挂载在某个节点的时候，首字母小写**
   1. **就是方便区分脚本是否挂载在节点上的**

**游戏的主脚本：**

**名字和场景名一样，区别于首字母的大小写，场景名首字母小写，脚本名首字母大写，主脚本一般挂载在Canvas上**

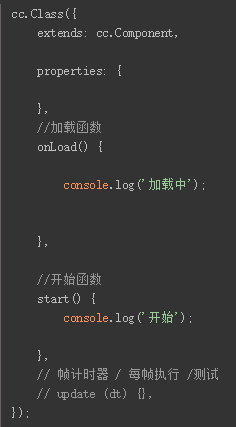
**}**

**脚本挂载的方法方式：**

1. **直接把脚本拖入指定的节点的属性检查器中**
2. **通过添加组件——>用户脚本组价**

**（无法主动被调用 7个）**

**脚本 生命周期函数 ：**

**on****Load 函数 加载函数**

**脚本挂载的节点被启动的时候立即执行**

**Start 函数 开始函数**

**被渲染的第一帧调用一次**

**Update(dt) 帧计时器 每帧执行、每帧调用一次**

**生命周期函数另外几个：**

**OnDestory 函数**

**在节点被销毁的时候调用一次**

**声明属性三步走：**

**第一步，在属性列表里声明出来**

**第二步：返回引擎，找到对应的节点找到对应的位置**

**第三步：通过脚本访问**

**节点之间的相互访问：**

**父节点能找到子节点，子节点能找到父节点**

**.parent 找父节点**

**.children[] 找子节点（根据下标找）**

**.getChildByName(‘’); 找子节点（根据名字找）**

**注意：但凡是我们自主命名的，通过一些方法去访问时，都是字符串类型‘’，**

**但凡两个单词以上命名的都是驼峰命名法**

**获取组件的方法：**

**GetComponent(cc.Label组件、cc.Sprite……).spriteFrame**

**获取组件时，组件的名称首字母大写**

**获取或修改属性时，首字母小写**

**按钮如何与脚本与引擎建立起联系**

**三个步骤：**

1. **先在脚本里写上按钮执行的函数**
2. **在属性检查器里面绑定按钮（**

**脚本所在的节点**

**脚本**

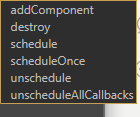
**脚本里的函数**

**）**

1. **才能实现相应的功能**

**实现GIF的效果**

**在制作动画时，属性选择Sprite Frame 把图片拖过去**



**cocos计时器**

**this.schedule(function(){**

**},0.1) 单位秒**

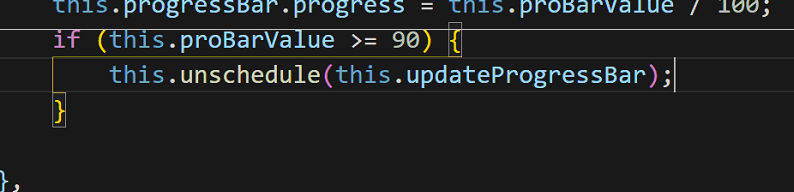
**与setInyerval一样属于，每隔多少秒执行一次**

**this.scheduleOnce(function(){**

**},0.1) 单位秒**

**与setTimeout一样属于，隔多少秒 执行一次**

**关闭计时器**

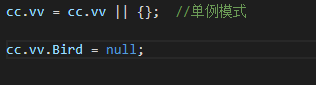
****

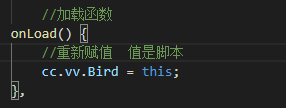
**跨脚本调用函数1，先获取到脚本所在节点2，通过节点获取到脚本组件3，调用该脚本下得函数。**

****

**跨脚本调用函数的第二种方法**

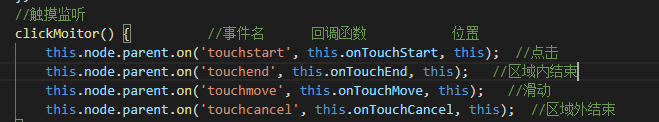
**先把变量设置为空，加载里面声明，**

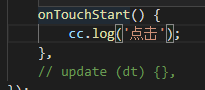
****

****

**点击监听：**

****

****

****



**播放动画**

**预制体生成：**

1. **把需要生成的预制体的模板搭建出来：**

**从资源管理器把用到的资源放到层级管理器**

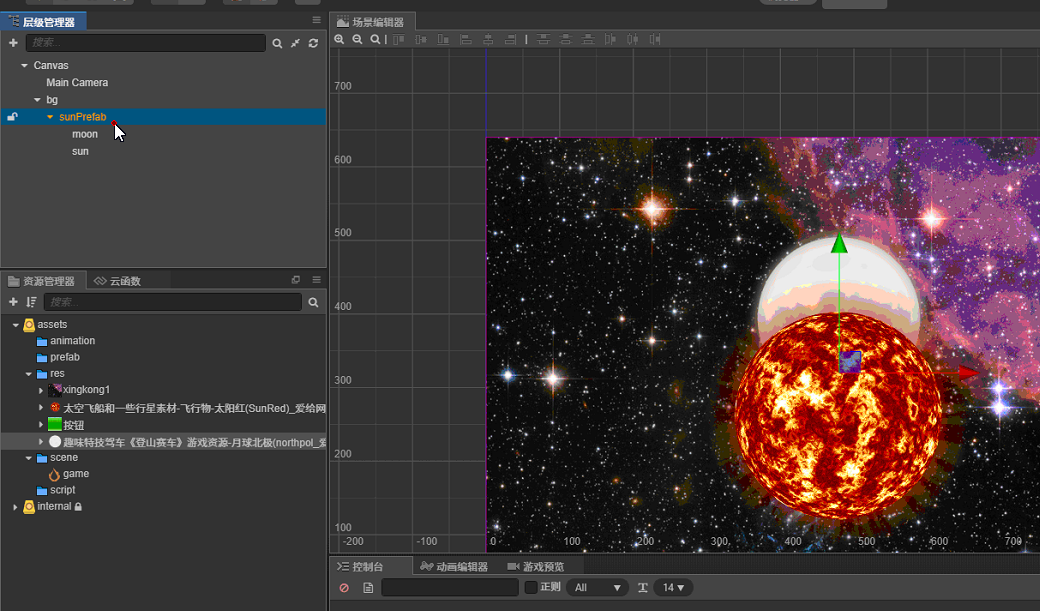
* 1. **从层级管理器 拖回到资源管理器**
  2. **在资源管理器里新建文件夹 prefab**
  3. **过河拆桥，把层级管理器里的删除（注意：widget对齐组件，如果后续需要修改坐标等等的数值时，两者会起冲突）**

**把预制体声明出来，与引擎建立起联系**

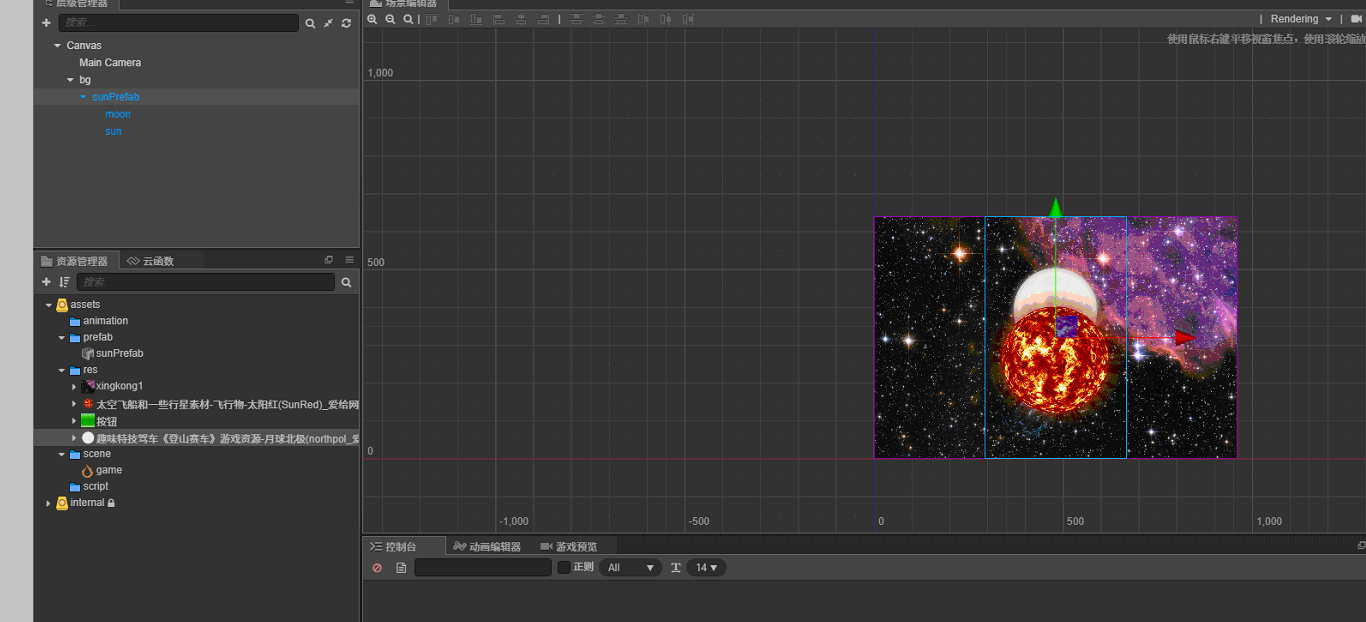
**2.cc.instantiate( 需要实例化出来的预制体)**

**通过实例化出来的节点，获取到预制体上挂载的脚本，调用里面的init函数，把实例化出来的节点做一个初始化**

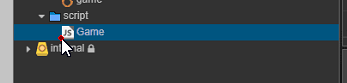
**3.添加到需要添加的子节点上**

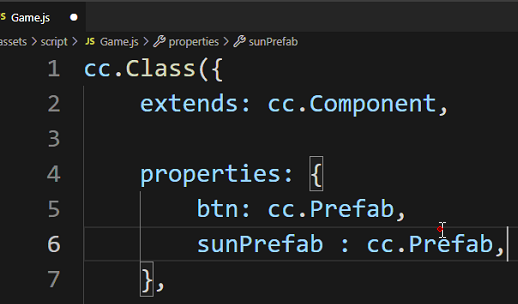


**空节点（sunPrefab不要添加wiget）**

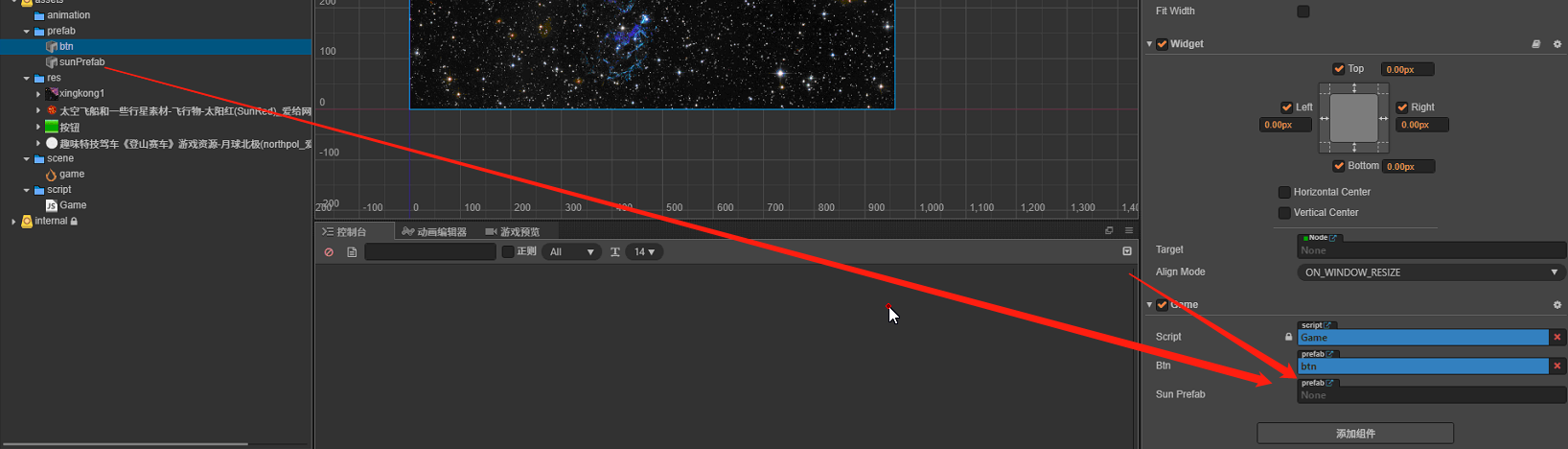


**拖回资源管理器并删除**

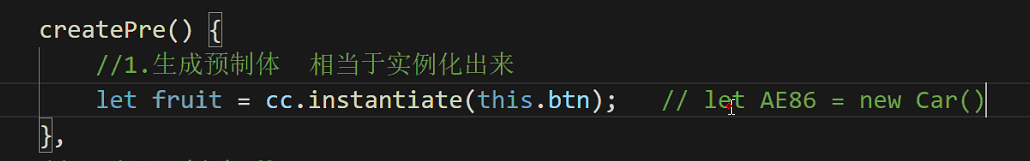
**新建主脚本****并挂载在Canvas**



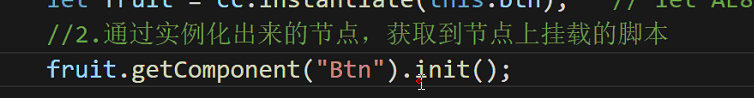
**声明预制体**



**预制体与引擎建立联系**



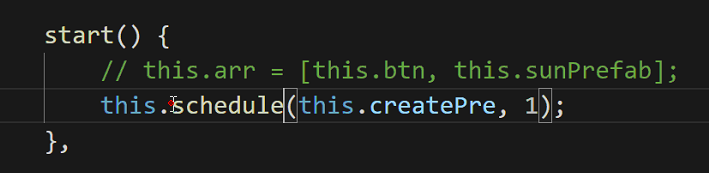
**声明一个变量来接住实例化出来的预制体。**



**通过实例化出来的节点获取到节点上挂载的脚本**

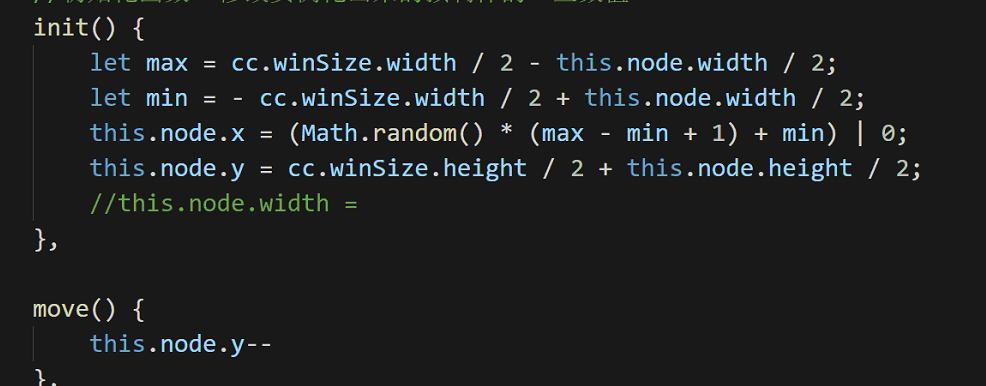


**添加到需要添加的节点上 把实例化出来的预制体添加为其它节点上的子节点**



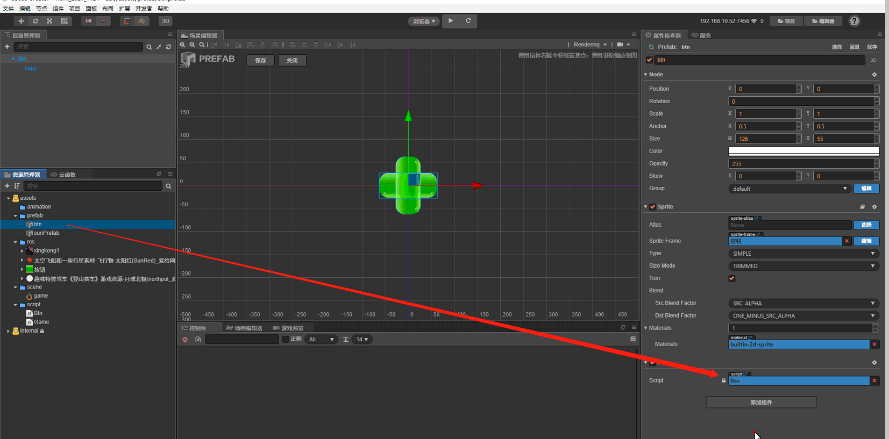
**多次调用预制体函数（计时器，1s生成一个）**

**新建预制体脚本**

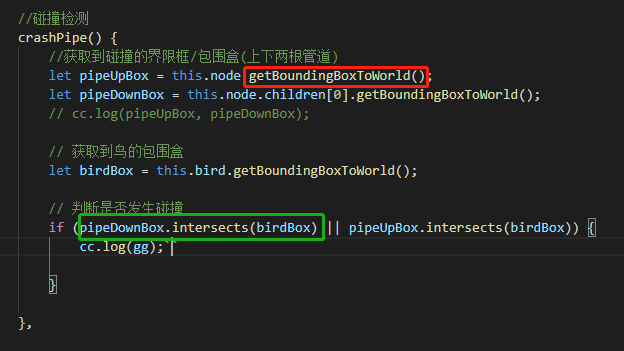


**初始化函数，修改实例化出来的预制体的一些数值**

**预制体的移动**

**预制体脚本挂载在预制体上。**

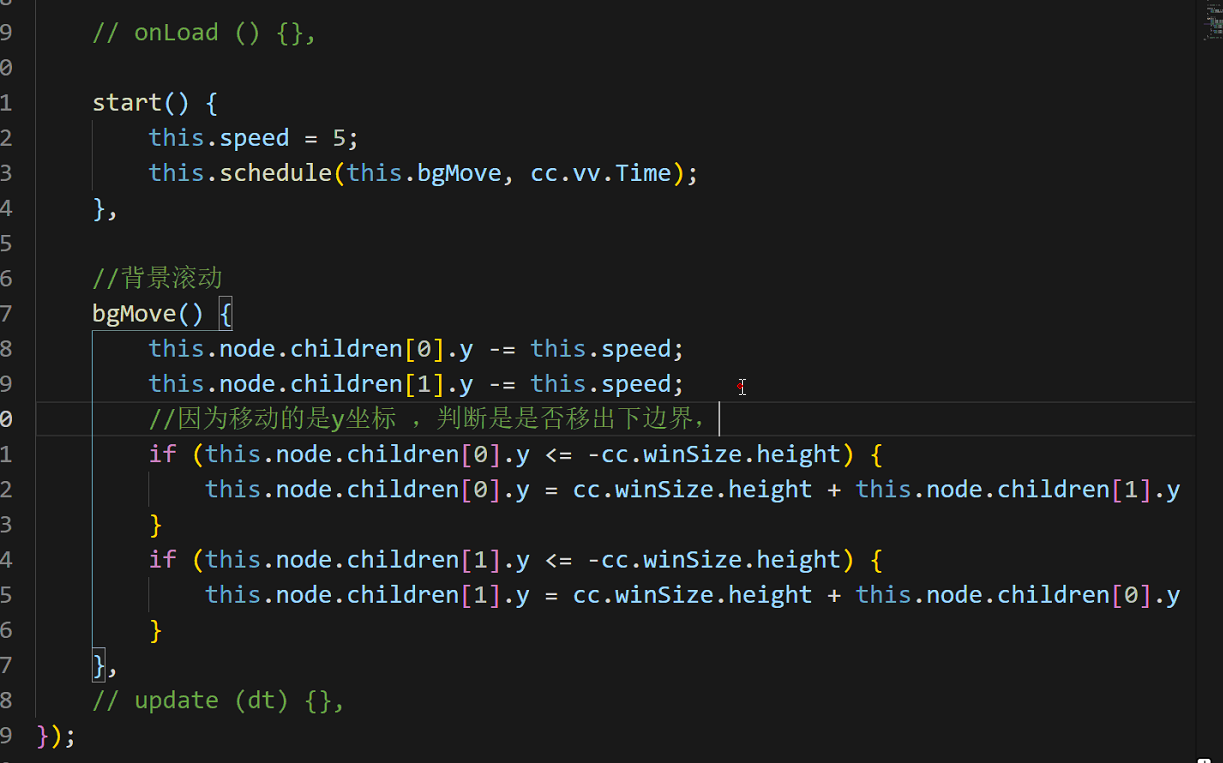
**碰撞函数（包围盒）；（重点）**



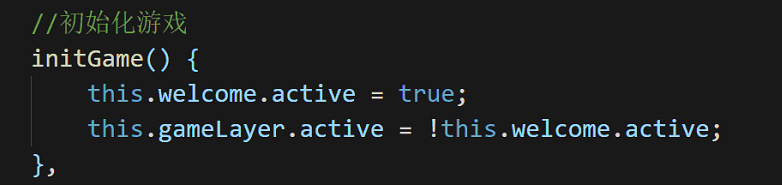
**红框为包围盒函数，直接调用就行**

**绿色为相交函数，两个东西相交**

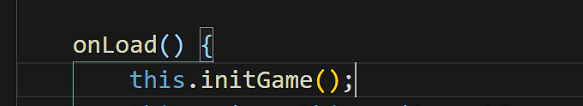
**背景的移动：例如从上往下循环**



**初始化游戏 欢迎界面与开始之后的界面的状态。**

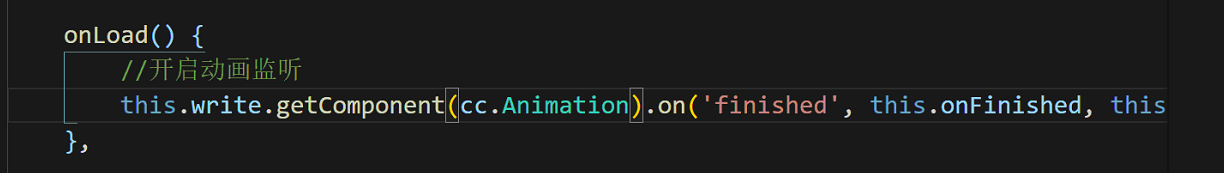


**进来就调用，**

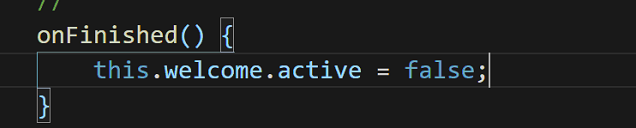


**动画回调**

**开启动画监听**



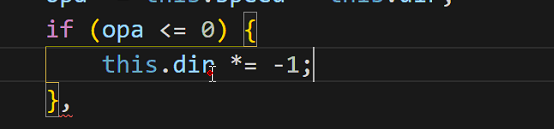
**动画播放完后执行的函数：**



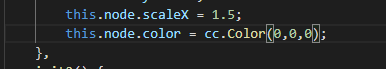
**改变方向 （满足条件：++变为--）**



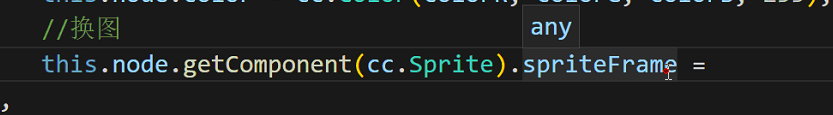




**改变颜色**



**换图**



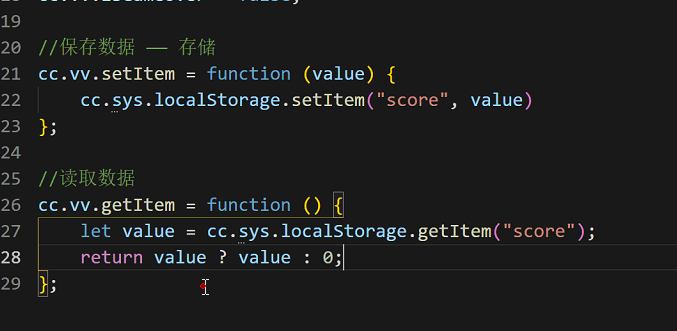
**这个节点需要先绑定图片数组，绑定**

answerSkins: [cc.SpriteFrame],

**调用：**

this.small.spriteFrame = this.answerSkins[this.levelDate.level - 1]

**存储与读取数据**



**场景跳转**

cc.director.loadScene('game');//跳转场景

**音乐音效：**

bgm: {

            default: null,

            type: cc.AudioClip

        },

**先声明，之后音乐资源丢过去**

**调用，值得注意的是，**

cc.audioEngine.playMusic(this.bgm, true, 0);



**PlayEffect为音效**



**PlayMusic为音乐**

cc.audioEngine.stopMusic(this.bgm);

**停止音乐或者音效；**

**设计分辨率**

**项目 >项目设置 >项目预览 >设计分辨率**

**粒子特效**

**还原本色：把颜色变化设置为白、黑、白、黑**

**每秒发射的粒子数目以及最大粒子数量 两着共同控制可见范围内粒子的多少**

**Emission Rate Total Particles**

**缓动系统**

**cc.tween( 需要操作的节点 ) 单位以秒记**

**.to(时间,对象{需要操作的属性：发生改变的量}) 到达某个值**

**.by(时间,对象{需要操作的属性：发生改变的量}) 改变某个值**

**.start() //启动的命令**

 cc.tween(this.star)

            .to(1, { position: cc.v2(100, 100), angle: 360 })

            .to(2, { scale: 2 })

            .to(2, { scale: 2 })

            .to(1, { position: cc.v2(-250, 420), angle: 360 })

            .start()

**.call的用法**

// 把上述所有的动作执行完毕之后，再执行下面的逻辑.call

            // .call(() => {

            //     console.log('All tweens finished.')

            // })

**同时执行两个缓动**

 starMove(pos) {

        // cc.tween(this.star)

        //     .by(0.5, { position: pos, angle: 72 })

        //     // .by(0.5, { angle: 72 })

        //     .start();

        cc.tween(this.star)

            // 同时执行两个 cc.tween

            .parallel(

                cc.tween().by(1, { angle: 72 }),

                cc.tween().to(1, { position: pos })

            )

            .start()

    }

**重复，永久执行缓动**

 cc.tween(this.yellowNode.children[0])

            //6,重复执行缓动

            .repeatForever(

                cc.tween(this.yellowNode.children[0])

                    .to(0.5, { scale: 1.5 })

                    .to(0.5, { scale: 1 })

            )

            .start();

**监听并获取到鼠标点击位置的X，Y**

onLoad() {

        this.clover.on('touchstart', this.onTouchStart, this)

    },

//触摸事件的回调

    onTouchStart(event) {  // 得到点击位置的X与Y，存入pos里

        let pos = event.getLocation(); //获取位置

        cc.log(pos.x, pos.y);

    }

**注意，这里获取到的是世界坐标系的位置，转换为节点坐标系需要在函数里输入：**

//触摸事件的回调

    onTouchStart(event) {  // 得到点击位置的X与Y，存入pos里

        let pos = event.getLocation();     //世界坐标系的位置

        // 将世界坐标系转换为节点坐标系:

        pos = this.clover.convertToNodeSpaceAR(pos);

        cc.log(pos.x, pos.y);

        //调用星星移动的函数

        this.starMove(pos);

    },

**键盘监听：**

 onLoad() {

        cc.systemEvent.on(cc.SystemEvent.EventType.KEY\_DOWN, this.onKeyDown, this);

    },

    start() {

    },

    onKeyDown(e) {

        switch (e.keyCode) {

            case cc.macro.KEY.a:

                // cc.log(this.guaiWu);

                this.guaiWu.scaleX = -1;

                this.guaiWu.x -= 10;

                // cc.log('fwefweefw');

                break;

            case cc.macro.KEY.d:

                this.guaiWu.scaleX = 1;

                this.guaiWu.x += 10;

                // cc.log('qwefasfa');

                break;

        }

    },

**获取节点大小（面积）**

//获取到游戏区域的大小

        let bigBox = this.bigBox.getContentSize();

**克隆复制**

//2每一次调用就复制原始的皮肤

        let newSkin = this.skin.clone();

**快速生成矩形 并切割裁剪**

**X Y 宽 高**

**cc.rect(rectX, rectY, this.cellSize.width, this.cellSize.height)**

//1设置一个矩形

        let rect = cc.rect(rectX, rectY, this.cellSize.width, this.cellSize.height);

        // 通过setRect(）方法，从克隆出来的皮肤上切割一块下来

        newSkin.setRect(rect);

**加载单个资源**

 autoLoadRes() { //这个函数加载单个资源

                                // 文件名       文件类型         错误  正确

        cc.resources.load('missionBranch', cc.SpriteFrame, (err, img) => {

            if (err) {           //如果错误就输出错误

                cc.log(err);

            } else {               //如果正确就输出img

                cc.log(img);

            }

        });

},

**批量加载一个文件夹下的所有资源**

**并且以数组形式输出**

autoLoadRes() { //这个函数加载文件夹下所有资源

        //                        文件夹名       文件类型         错误  正确

  cc.resources.loadDir('singleSprite', cc.SpriteFrame, (err, list) => {

            if (err) {           //如果错误就输出错误

                cc.log(err);

            } else {               //如果正确就输出img

                cc.log(list);

            }

        });

    },

**导出、**

module.exports

**引入全局**

调用 require 来访问你创作的模块脚本

cc.vv = cc.vv || {}

//把仓库里面的东西引入全局，方便使用

cc.vv.Warehouse = require('./warehouse');

**进度条：**

**拿到进度条所在节点的progress**

this.bar.getComponent(cc.ProgressBar).progress

**判断progress是否为1（1则进度条填满，反之0为空）**

if (this.jdt < 1) {

          this.jdt = (this.count / this.total).toFixed(2);

          this.bar.getComponent(cc.ProgressBar).progress = this.jdt;

} else {

          this.jdt = 1;

          this.jindu.string = this.jdt \* 100 + '%';

          this.bar.getComponent(cc.ProgressBar).progress = this.jdt;

        } cc.log(this.jdt);

**需要注意的是，计算完之后要重新赋值给progress进度条才会动**

**按钮的策略模式**



/\*\*

 @author   Vision             作者

 @time     2021/12/07         时间

 @duty     工具模块            作用

 \*/

module.exports = {//导出

    /\*\*

     \* //参数

     \* @param  targetNode  这个 node 节点是你的事件处理代码组件所属的节点

     \* @param  component  脚本名  点击了按钮的回调，所在的脚本

     \* @param  callbake  回调函数的函数名（通过传参的形式）

     \* @param  foobar 引擎当中按钮组件上的最后一个参数

     \* @param  buttonNode  组件所在的一个节点

    \*/

    addCallBackToButton: function (targetNode, component, callbake, foobar, buttonNode) {

        //

        let clickEventHandler = new cc.Component.EventHandler();

        //// 这个 node 节点是你的事件处理代码组件所属的节点

        clickEventHandler.target = targetNode;

        //这个是代码文件名 脚本名  点击了按钮的回调，所在的脚本

        clickEventHandler.component = component;//

        //回调函数（通过传参的形式）

        clickEventHandler.handler = callbake;

        //引擎当中按钮组件上的最后一个参数

        clickEventHandler.customEventData = foobar;

        //检查节点上有没有Button组件  如果没获取到组件

        if (!buttonNode.getComponent(cc.Button)) {

            // 获取不到的时候 就添加一个button组件

            buttonNode.addComponent(cc.Button)

        };

        let button = buttonNode.getComponent(cc.Button);

        //引擎里面   对应的回调事件

        button.clickEvents.push(clickEventHandler);

    }

};

start() {

        this.addButtonEvent();

    },

    //给按钮添加点击的回调事件

    addButtonEvent() {

        //获取到节点上的所有子节点

        let children = this.node.children;

        //遍历所有子节点（按钮节点）

        children.forEach(element => {

            //根据点击时传入的最后一个参数调用对应的回调函数，把节点的名字作为参数，传入

            let customEventData = element.name;

            //调用工具模块

            cc.vv.Tools.addCallBackToButton(this.node, 'ButtonLayer', 'onBUttonClicked', customEventData, element);

        })

    },

    //按钮的策略模式的回调函数

    onBUttonClicked(event, customEventData) {

        //根据点击时传入的最后一个参数来调用相应的函数

        this[customEventData]();

    },

**自制音乐的工具脚本**

/\*\*

@author   Vision             作者

@time     2021/12/07         时间

@duty     音乐模块            作用

\*/

//匿名函数 并且是一个立即执行函数    工厂模式

module.exports = (function () {

    //1.导入原材料

    let dj\_module = {};

    //5.导入完成开始之前并没有音乐在播放

    dj\_module.bgMusic = null;

    //2.背景音乐的播放

    dj\_module.playBgm = function (clip, value) {

        //6.如果有音乐正在播放，就停止原本正在播放的音乐

        if (dj\_module.bgMusic) {

            cc.audioEngine.stopMusic();

        }

        //3.进入函数，就开始播放音乐

        cc.audioEngine.playMusic(clip, true);

        //4.加个保险，判断音量这个参数是否存在，没有就给默认值

        value = value ? value : 10;

        cc.audioEngine.setMusicVolume(value);//设置音量，音量值为value

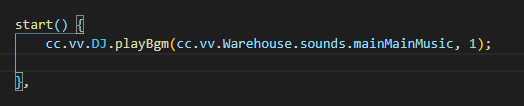
        dj\_module.bgMusic = clip

    };

    //得到成品

    return dj\_module;

})();

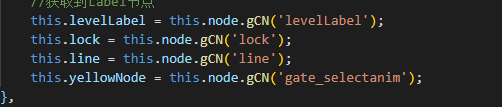
****

**新建函数快捷查找（声明之后可以快速使用）**

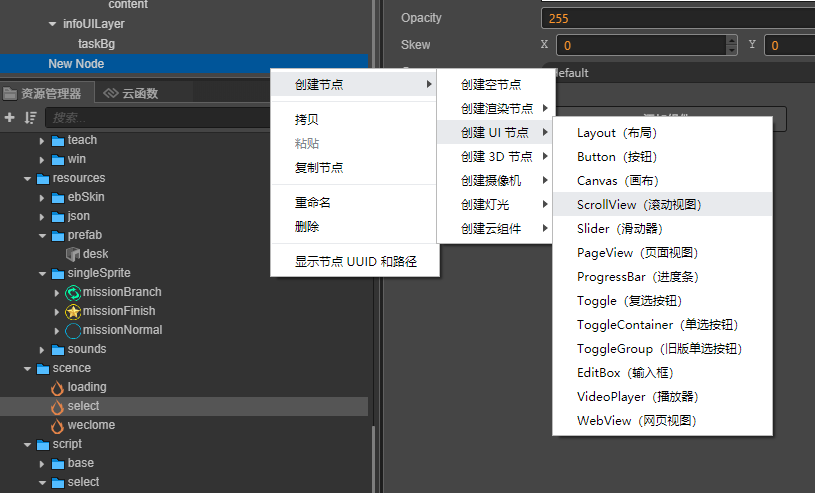
**全局脚本里声明：例如查找某个节点中根据名字查找子节点**

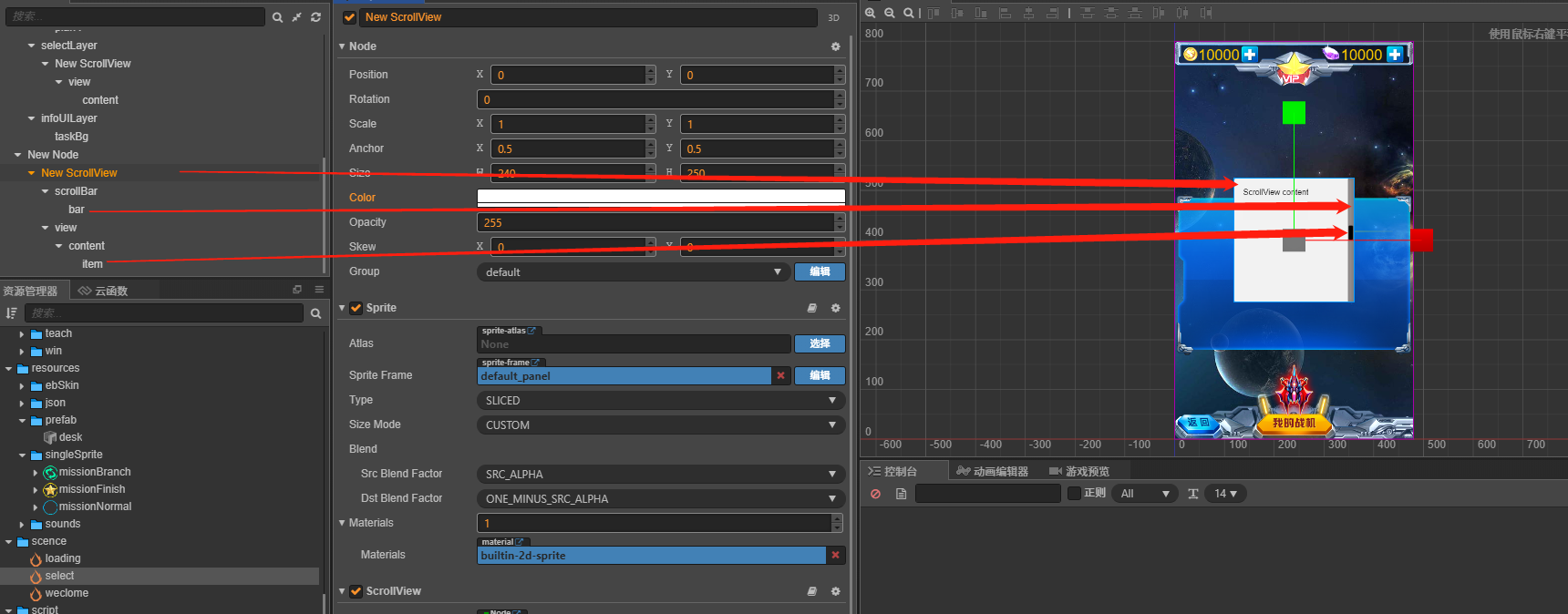
cc.Node.prototype.gCN = cc.Node.prototype.getChildByName;

**可以看到，查找的话方便很多（cocos其它自带函数也可使用此方法）**



**滚动视图的创建**

****



**重点（view是视野范围）**

**声明滚动组件**

 scro: cc.ScrollView,

**滚动停止与检测**

if (this.scro.isAutoScrolling()) { //检测是否正在依靠惯性滚动

            this.scro.stopAutoScroll()//停止滚动

        }

**自动布局组件：Layout**

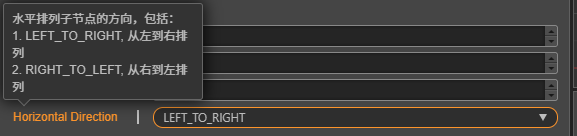
**三种对齐方式**

**Left Right为节点最左最右边距**

**SpacingX为节点之间水平距离（垂直布局的话为SpacingY）**

**Affected By Scate为子节点缩放是否能影响到布局**

****



**巧妙的替换位置的方法：**

children.unshift(children.pop());//children[0]和children[1]换位置（把第二个换到第一个的位置）