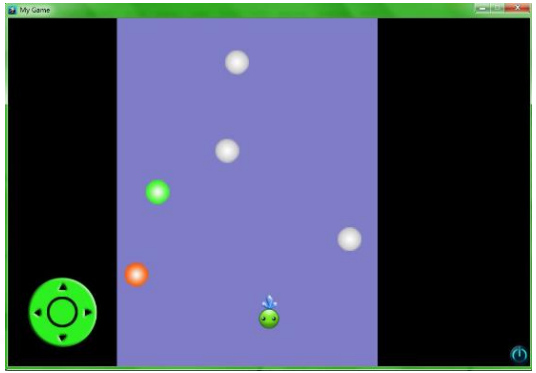
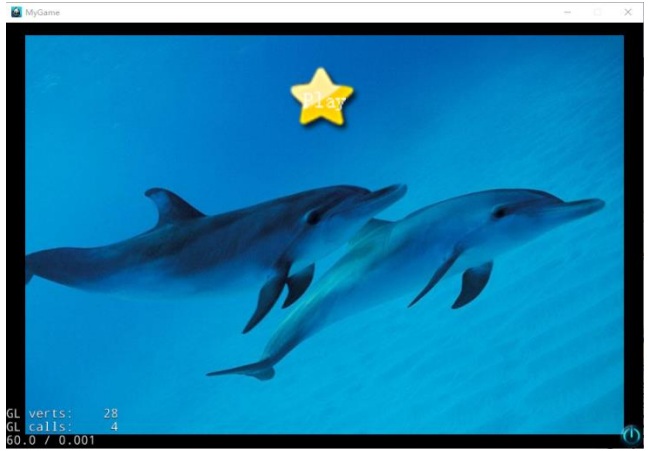
第七章 二维游戏动画合成

上节回顾

- 游戏界面设计实例----贪食豆
- UI在Cocos2d-x中的应用
- Cocos2d-x中的场景切换
- 布置实验2
  - 游戏录屏
  - 实验报告
  - 2021.5.4





本节内容

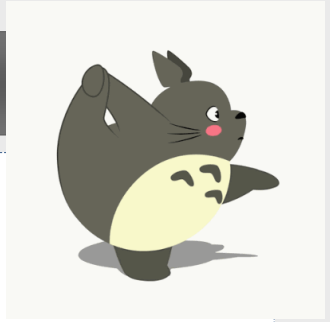
- Chapter 7
  - 计算机动画概述
  - 常见计算机动画技术
  - Cocos2d-x中的动作类
  - 课后作业

计算机动画概述

英国著名动画艺术家John Halas曾指出：

“运动是动画的本质”

计算机动画概述



- 动画
  - 把动态物体的运动过程人为地制作成静态画面
  - 以逐格摄影、逐帧录制、存储的形式记录
  - 利用人眼视觉特点，以一定速度连续播放

8 fps　12 fps　24 fps　25 fps　30 fps

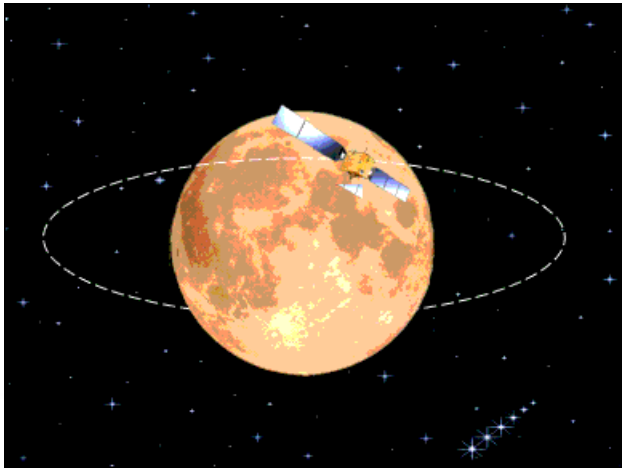
计算机动画概述

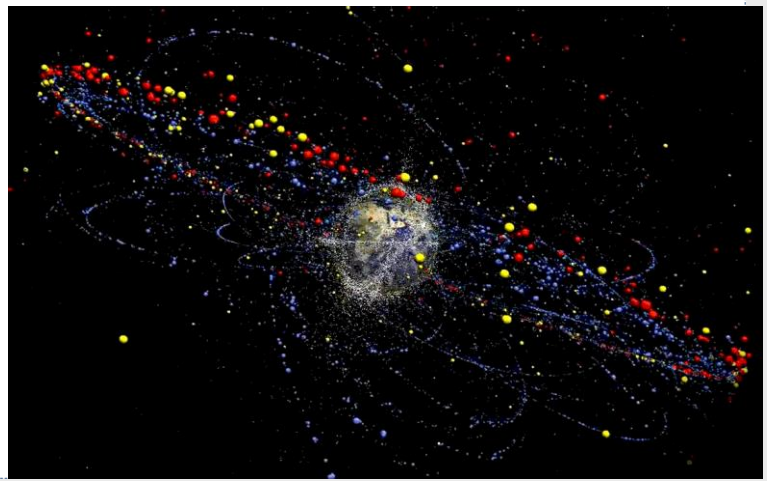
拍摄动画片大闹天宫时，几十位动画工作者花了近两年的时间才完成，总共绘制了600多万张图画。

美国迪士尼公司于1937年创作的第一部长达83分钟的大型动画片白雪公主和七个小矮人，共绘制了两亿张草图，最后用来拍摄的图画有250000张。

计算机动画概述

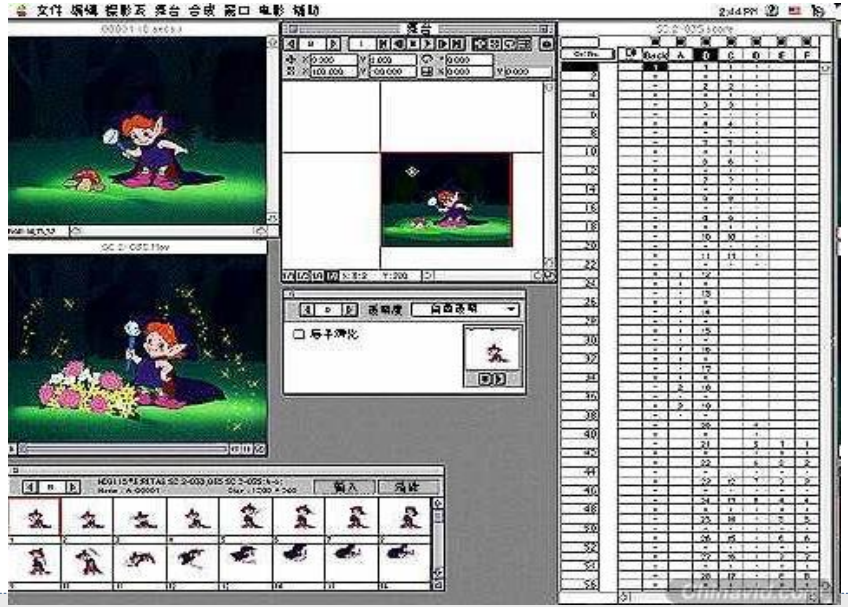
- 计算机动画
  - 采用图形图像处理技术，借助编程或动画制作软件生成一系列动态连续图像
  - 第一个计算机动画是 贝尔实验室 的扎伊克（Zajac）在1961年制作的
  - 它通过线框图形展示了地球卫星在太空运行时的景观





计算机动画概述

– 1971年，被称为计算机动画之父的Nestor Burtnyk 和 Marceli Wein 提出了“计算机产生关键帧”动画技术，并应用该技术开发了MSGEN二维动画系统

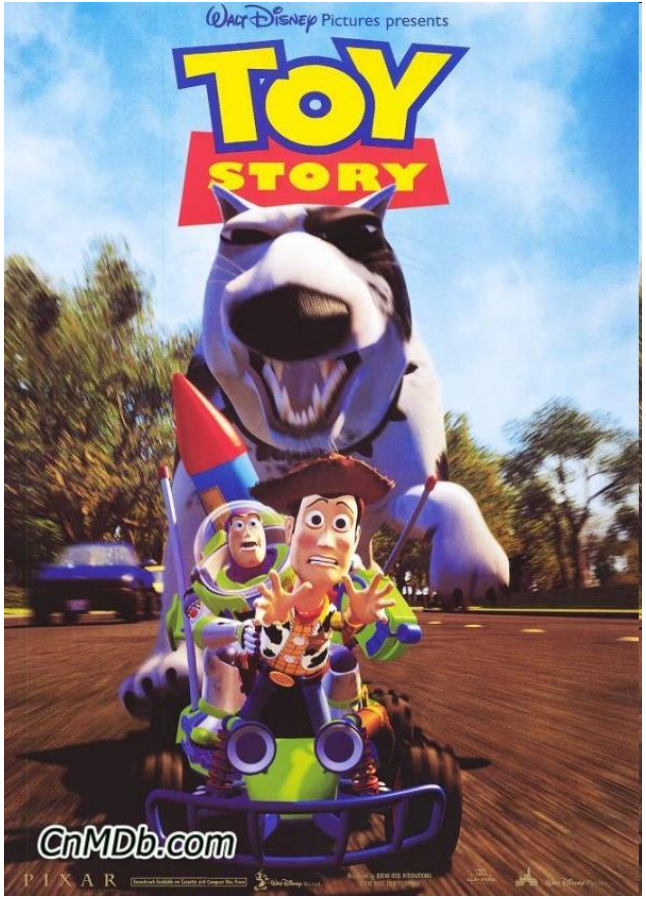


计算机动画概述

- 1995年，Disney公司和Pixar公司联合推出两家公司第一部三维动画片《玩具总动员》，并从该动画片获得3.6亿美元的票房收入。

计算机动画概述

- 《玩具总动员》全部采用计算机制作，片长约81分钟，共包含76个角色和114000个画面
- 主角安迪（Andy）共有12384根头发，小狗头的毛发多达15977根，而且每一根毛发都能活动
- 牛仔胡迪（Woody）全身有700多个控制点，仅仅脸部就有212个可活动的控制点



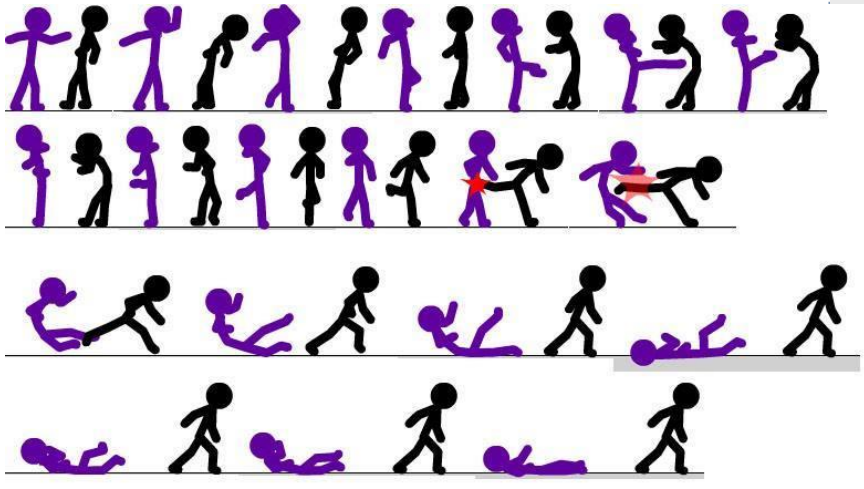
本节内容

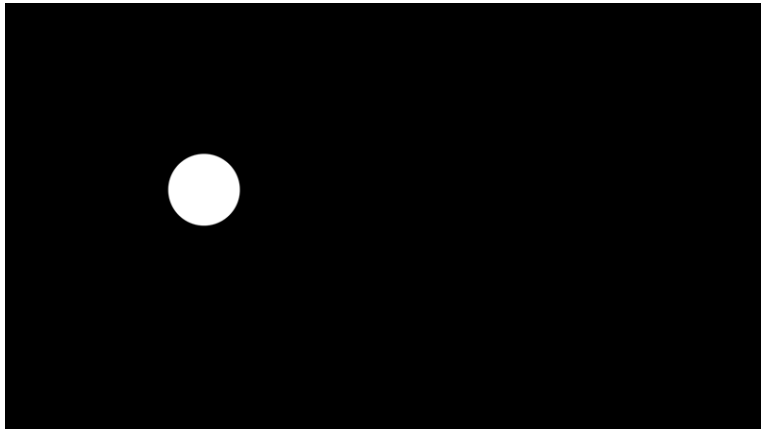
- Chapter 7
  - 计算机动画概述
  - 常见计算机动画技术
  - Cocos2d-x中的动作类
  - 课后作业

常见计算机动画技术

常见计算机动画技术

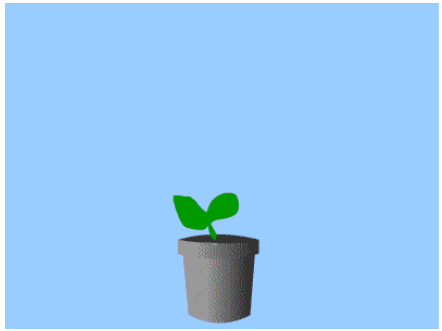
- 基本动画技术
  - 逐帧动画
    - 将组成动画所需一系列画面存放在不同帧中
    - 一帧一帧顺序播放
  - 关键帧动画
    - 从连续的画面中，选出少数几帧作为关键帧
    - 在关键帧基础上，使用计算机自动生成期望帧序列





常见计算机动画技术

- 基本动画技术
  - 逐帧动画
    - 逐帧保存、文件体积大
    - 制作麻烦、费时
    - 灵活性高、画面细腻
  - 关键帧动画
    - 仅保存关键帧
    - 核心
      - 关键帧的选取
      - 中间帧的插值算法



常见计算机动画技术

- 脚本驱动的动画技术
  - 主要应用于游戏开发
  - 使用脚本语言
    - 面向美工和游戏设计人员
    - 外部的、具有良好可读性的操作指令集
    - 不属于游戏代码
    - 易于理解、可读性强、交互性友好

可控制游戏中各玩家的互动过程

可对动画人物的运动路径进行规划

常见计算机动画技术

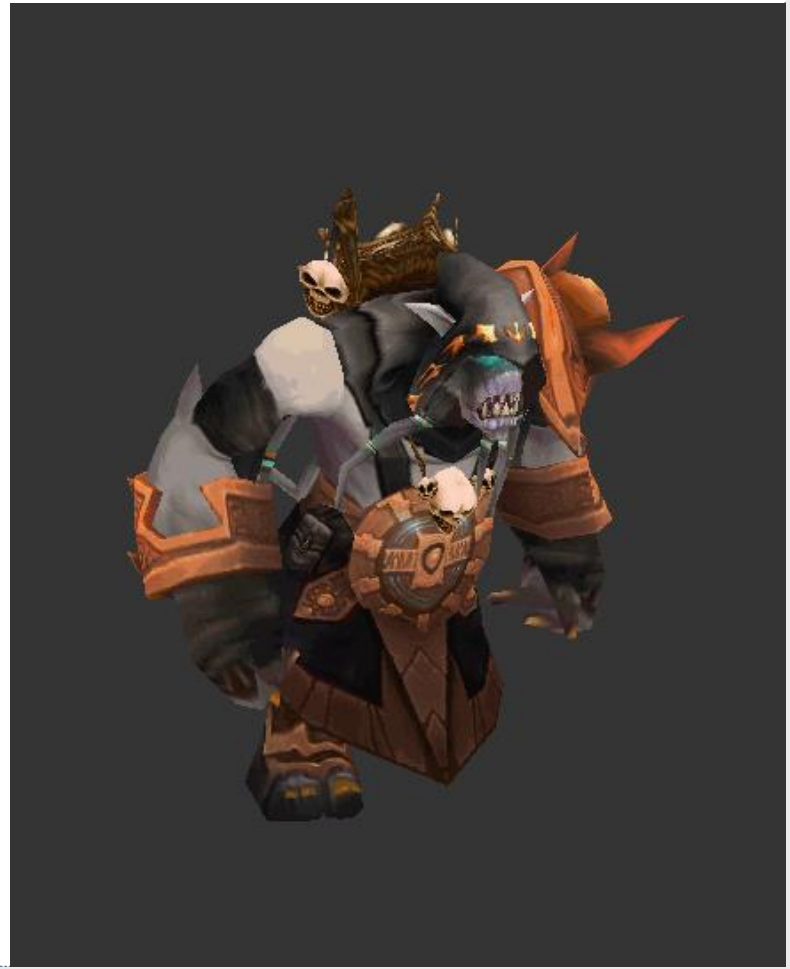
- 骨骼动画技术
  - 表现动画中角色的运动过程
  - 与人体运动方式相似，把角色各部分身体部件图片绑定到互相连接作用的“骨头”上，通过骨骼的位置变化而生成动画
  - 只需少量图像资源，骨骼动画数据保存在Json文件中，占用空间小

常见计算机动画技术

- 骨骼动画技术
  - 易于混合使用
    - 使角色同时完成
    - 走路、转头、挥手等动作

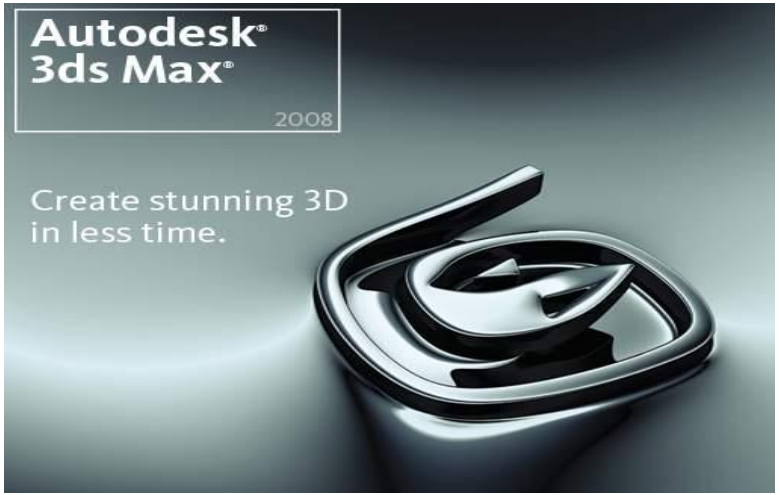
常见计算机动画技术

- 骨骼动画技术



常见动画制作软件

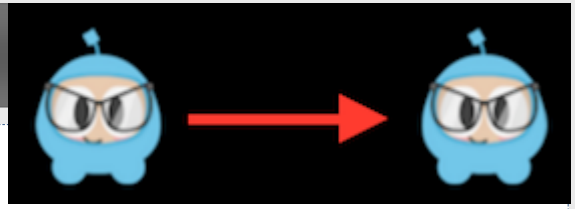
- Flash
- 3ds Max
- Maya



本节内容

- Chapter 7
  - 计算机动画概述
  - 常见计算机动画技术
  - Cocos2d-x中的动作类
  - 课后作业

Cocos2d-x中的动作类



动作(Action)

动作(Action) 的功能就和字面含义一样，它通过改变一个 `Node` 对象的属性，让它表现出某种动作。动作对象能实时的改变 `Node` 的属性，任何一个对象只要它是 `Node` 的子类都能被改变。比如，你能通过动作对象把一个精灵从一个位置移动到另一个位置。

通过 `MoveTo` 和 `MoveBy` 方法:

```
// Move sprite to position 50,10 in 2 seconds.
auto moveTo = MoveTo::create(2, Vec2(50, 10));
mySprite1->runAction(moveTo);
```

```
// Move sprite 20 points to right in 2 seconds
auto moveBy = MoveBy::create(2, Vec2(20,0));
mySprite2->runAction(moveBy);
```

By 和 To 的区别

你能注意到，每一个动作都会有两个方法 **By** 和 **To**。两种方法方便你在不同的情况使用，**By** 算的是相对于节点对象的当前位置，**To** 算的是绝对位置，不考虑当前节点对象在哪。如果你想动作的表现是相对于 `Node` 当前位置的，就用 **By**，相对的想让动作的表现是按照坐标的绝对位置就用 **To**。看一个例子:

Cocos2d-x中的动作类

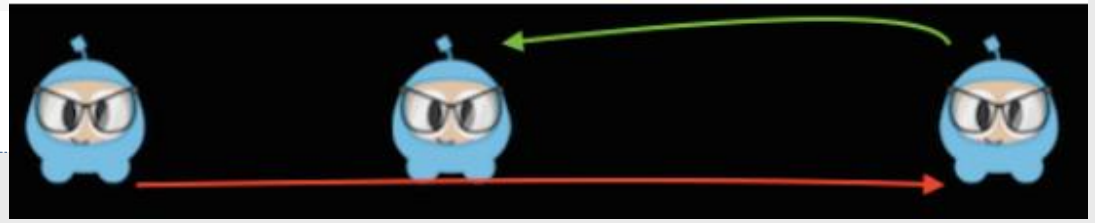
```
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(Vec2(200, 256));
```

```
// MoveBy - lets move the sprite by 500 on the x axis over 2 seconds
// MoveBy is relative - since x = 200 + 500 move = x is now 700 after the move
auto moveBy = MoveBy::create(2, Vec2(500, mySpri etPositionY()));
```

```
// MoveTo - lets move the new sprite to 300 x 256 over 2 seconds
// MoveTo is absolute - The sprite gets moved to 300 x 256 regardless of
// where it is located now.
auto moveTo = MoveTo::create(2, Vec2(300, mySprite->getPositionY()));
```

```
// Delay - create a small delay
auto delay = DelayTime::create(1);
```

```
auto seq = Sequence::create(moveBy, delay, moveTo, nullptr);
mySprite->runAction(seq);
```



Cocos2d-x中的动作类

旋转

使用 `RotateTo` `RotateBy` 完成节点对象在一个设置的时间后顺时针旋转指定角度。

```
auto mySprite = Sprite::create("mysprite.png");
```

```
// Rotates a Node to the specific angle over 2 seconds
auto rotateTo = RotateTo::create(2.0f, 40.0f);
mySprite->runAction(rotateTo);
```

```
// Rotates a Node clockwise by 40 degree over 2 seconds
auto rotateBy = RotateBy::create(2.0f, 40.0f);
mySprite->runAction(rotateBy);
```

Cocos2d-x中的动作类

缩放

使用 `ScaleBy` `ScaleTo` 完成节点对象的比例缩放。

```
auto mySprite = Sprite::create("mysprite.png");
```

```
// Scale uniformly by 3x over 2 seconds
auto scaleBy = ScaleBy::create(2.0f, 3.0f);
mySprite->runAction(scaleBy);
```

```
// Scale to uniformly to 3x over 2 seconds
auto scaleTo = ScaleTo::create(2.0f, 3.0f);
mySprite->runAction(scaleTo);
```

Cocos2d-x中的动作类

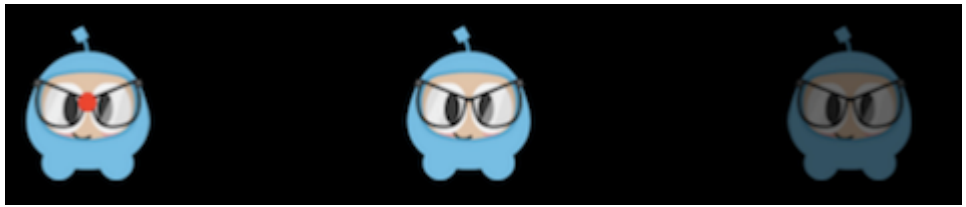
淡入淡出

使用 `FadeIn` `FadeOut` 完成节点对象的淡入，淡出。 `FadeIn` 修改节点对象的透明度属性，从完全透明到完全不透明， `FadeOut` □相反。

```
auto mySprite = Sprite::create("mysprite.png");
```

```
// fades in the sprite in 1 seconds
auto fadeIn = FadeIn::create(1.0f);
mySprite->runAction(fadeIn);
```

```
// fades out the sprite in 2 seconds
auto fadeOut = FadeOut::create(2.0f);
mySprite->runAction(fadeOut);
```



Cocos2d-x中的动作类

色彩混合

使用 `TintTo` `TintBy`，将一个实现了 `NodeRGB` 协议的节点对象进行色彩混合。

```
auto mySprite = Sprite::create("mysprite.png");
```

```
// Tints a node to the specified RGB values
auto tintTo = TintTo::create(2.0f, 120.0f, 232.0f, 254.0f);
mySprite->runAction(tintTo);
```

```
// Tints a node BY the delta of the specified RGB values.
auto tintBy = TintBy::create(2.0f, 120.0f, 232.0f, 254.0f);
mySprite->runAction(tintBy);
```

Cocos2d-x中的动作类

序列

动作序列(Sequence) 是一种封装多个动作的对象，当这个对象执行时被封装的动作会顺序执行。

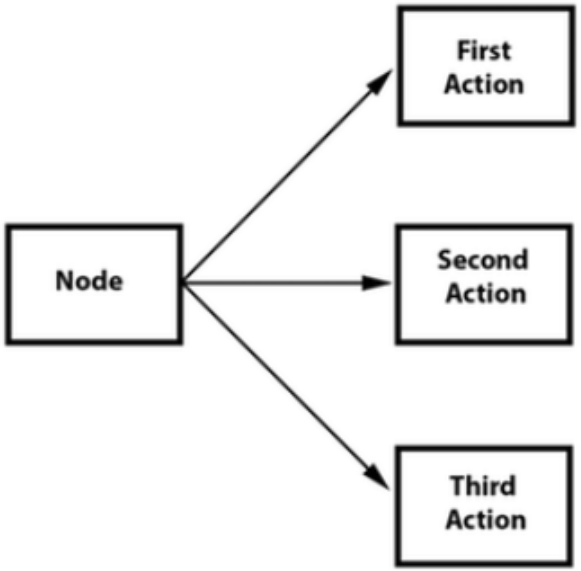
一个 `Sequence` 可以包含任何数量的动作对象，回调方法和其它序列。可以包含回调方法? 没错! Cocos2d-x 允许把一个方法添加进去 `CallFunc` 对象，然后将 `CallFunc` 添加到 `Sequence` ，这样，在执行序列的时候就能触发方法调用。因此，你能在一个序列中添加一些个性化的功能，而不仅仅是添加 Cocos2d-x 提供的有限动作。下面是一个序列的动作执行示意图：


```
82      auto moveTo = MoveTo::create(rand() % 5, Point(ball1->getPositionX(), -10));//移动动作
83      //当小球移动到屏幕下方时回调removeBall函数，移除小球
84      auto actionDone = CallFunc::create(CC_CALLBACK_0(HelloWorld::removeBall, this, ball1));
85      auto sequence = Sequence::create(moveTo, actionDone, nullptr);
86      ball1->runAction(sequence);//执行动作
```

Cocos2d-x中的动作类

Spawn

`Spawn` 和 `Sequence` 是非常相似的，区别是 `Spawn` 同时执行所有的动作。 `Spawn` 对象可以添加任意数量的动作和其它 `Spawn` 对象。



`Spawn` 的效果和同时运行多个动作的 `runAction()` 方法是一致的，但是它的独特之处是 `Spawn` 能被放到 `Sequence` 中，结合 `Spawn` 和 `Sequence` 能实现非常强大的动作效果。

Cocos2d-x中的动作类

例如，创建两个动作：

```
// create 2 actions and run a Spawn on a Sprite
auto mySprite = Sprite::create("mysprite.png");

auto moveBy = MoveBy::create(10, Vec2(400,100));
auto fadeTo = FadeTo::create(2.0f, 120.0f);
```

使用 `Spawn`：

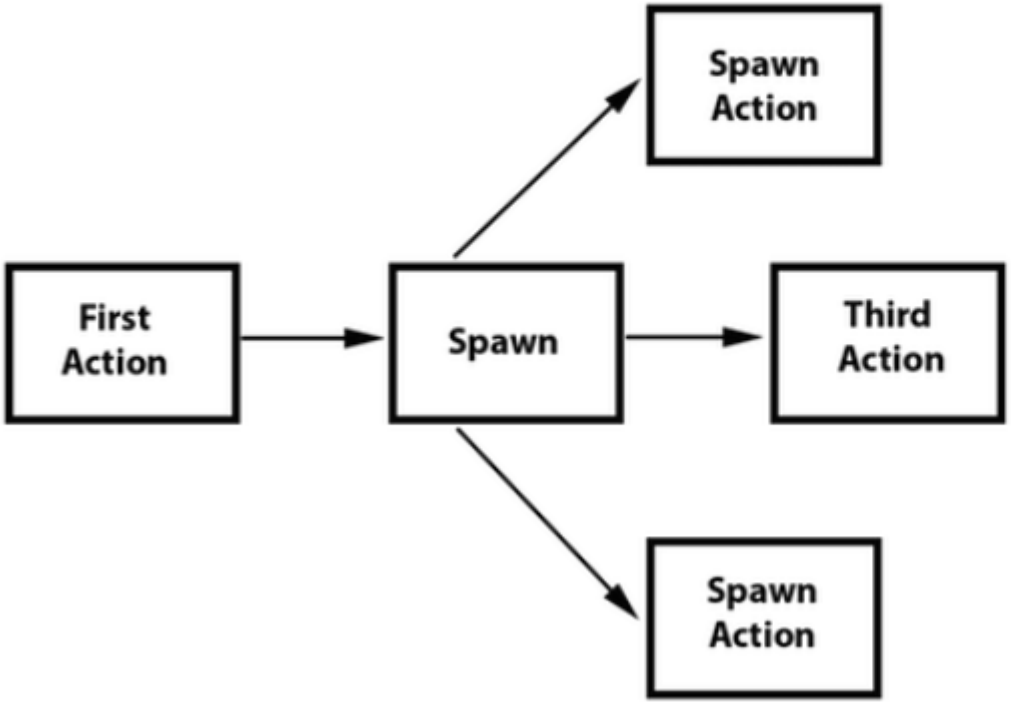
```
// running the above Actions with Spawn.
auto mySpawn = Spawn::createWithTwoActions(moveBy, fadeTo);
mySprite->runAction(mySpawn);
```

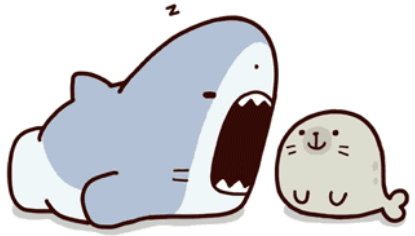
同时调用方法 `runAction()`：

```
// running the above Actions with consecutive runAction() statements。
mySprite->runAction(moveBy);
mySprite->runAction(fadeTo);
```

Cocos2d-x中的动作类

上面两种方式产生的效果是一样的，现在看把一个 `Spawn` 添加到一个 `Sequence` 中是怎样的一种情景，动作的执行流程会看起来像这样：





Cocos2d-x中的动作类

```
// create a Sprite
auto mySprite = Sprite::create("mysprite.png");

// create a few Actions
auto moveBy = MoveBy::create(10, Vec2(400,100));
auto fadeTo = FadeTo::create(2.0f, 120.0f);
auto scaleBy = ScaleBy::create(2.0f, 3.0f);

// create a Spawn to use
auto mySpawn = Spawn::createWithTwoActions(scaleBy, fadeTo);

// tie everything together in a sequence
auto seq = Sequence::create(moveBy, mySpawn, moveBy, nullptr);

// run it
mySprite->runAction(seq);
```

Cocos2d-x中的动作类

Cocos2d-x中的动作类

动作的克隆

克隆(Clone) 的功能和字面含义一样，如果你对一个节点对象使用了 `clone()` 方法，你就获得了这个节点对象的拷贝。

为什么要使用 `clone()` 方法?

从代码中学习用法吧，先看看错误的情况:

```
123     auto moveBy = MoveBy::create(10, Vec2(400, 100));
124     // run it on sprite
125     sprite->runAction(moveBy);
126     // run it on mysprite
127     mysprite->runAction(moveBy);

128     mysprite->runAction(moveBy->clone()); // correct!
```

Cocos2d-x中的动作类

Cocos2d-x中的动作类

动作的倒转

倒转(Reverse) 的功能也和字面意思一样，调用 `reverse()` 可以让一系列动作按相反的方向执行。 `reverse()` 不是只能简单的让一个 `Action` 对象反向执行，还能让 `Sequence` 和 `Spawn` 倒转。

倒转使用起来很简单：

```
// reverse a sequence, spawn or action
mySprite->runAction(mySpawn->reverse());
```

思考下面这段代码在执行的时候，内部发生了什么？

Cocos2d-x中的动作类

```
// create a Sprite
auto mySprite = Sprite::create("mysprite.png");
mySprite->setPosition(50, 56);
```

```
// create a few Actions
auto moveBy = MoveBy::create(2.0f, Vec2(500,0));
auto scaleBy = ScaleBy::create(2.0f, 2.0f);
auto delay = DelayTime::create(2.0f);
```

```
// create a sequence
auto delaySequence = Sequence::create(delay, delay->clone(), delay->clone(),
delay->clone(), nullptr);
auto sequence = Sequence::create(moveBy, delay, scaleBy, delaySequence, nullptr);
// run it
mySprite->runAction(sequence);
```

```
// reverse it
mySprite->runAction(sequence->reverse());
```

Cocos2d-x中的动作类

思考起来可能有点困难，我们将执行的每一步列出来，或许能帮助你理解：

1. `mySprite` 创建
2. `mySprite` 的坐标位置设置成(50,56)
3. `sequence` 开始执行
4. `sequence` 执行第一个动作 `moveBy` ，2s 中 `mySprite` 移动到了坐标位置(550,56)
5. `sequence` 执行第二个动作， 暂停 2s
6. `sequence` 执行第三个动作， `scaleBy` ，2s 中 `mySprite` 放大了2倍
7. `sequence` 执行第四个动作， `delaySequence` ，暂停 6s
8. `reverse()` 被调用，序列倒转，开始反向执行
9. `sequence` 执行第四个动作， `delaySequence` ，暂停 6s
10. `sequence` 执行第三个动作， `scaleBy` ，2s 中 `mySprite` 缩小了2倍(注意：序列内的动作被倒转)
11. `sequence` 执行第二个动作， 暂停 2s
12. `sequence` 执行第一个动作 `moveBy` ，2s 中 `mySprite` 从坐标位置 (550,56)，移动到了 (50, 56)
13. `mySprite` 回到了最初的位置

我们能发现 `reverse()` 方法使用起来很简单，内部逻辑却一点都不简单。因为 Cocos2d-x 封装了复杂的逻辑，为你留下了简单易用的接口！

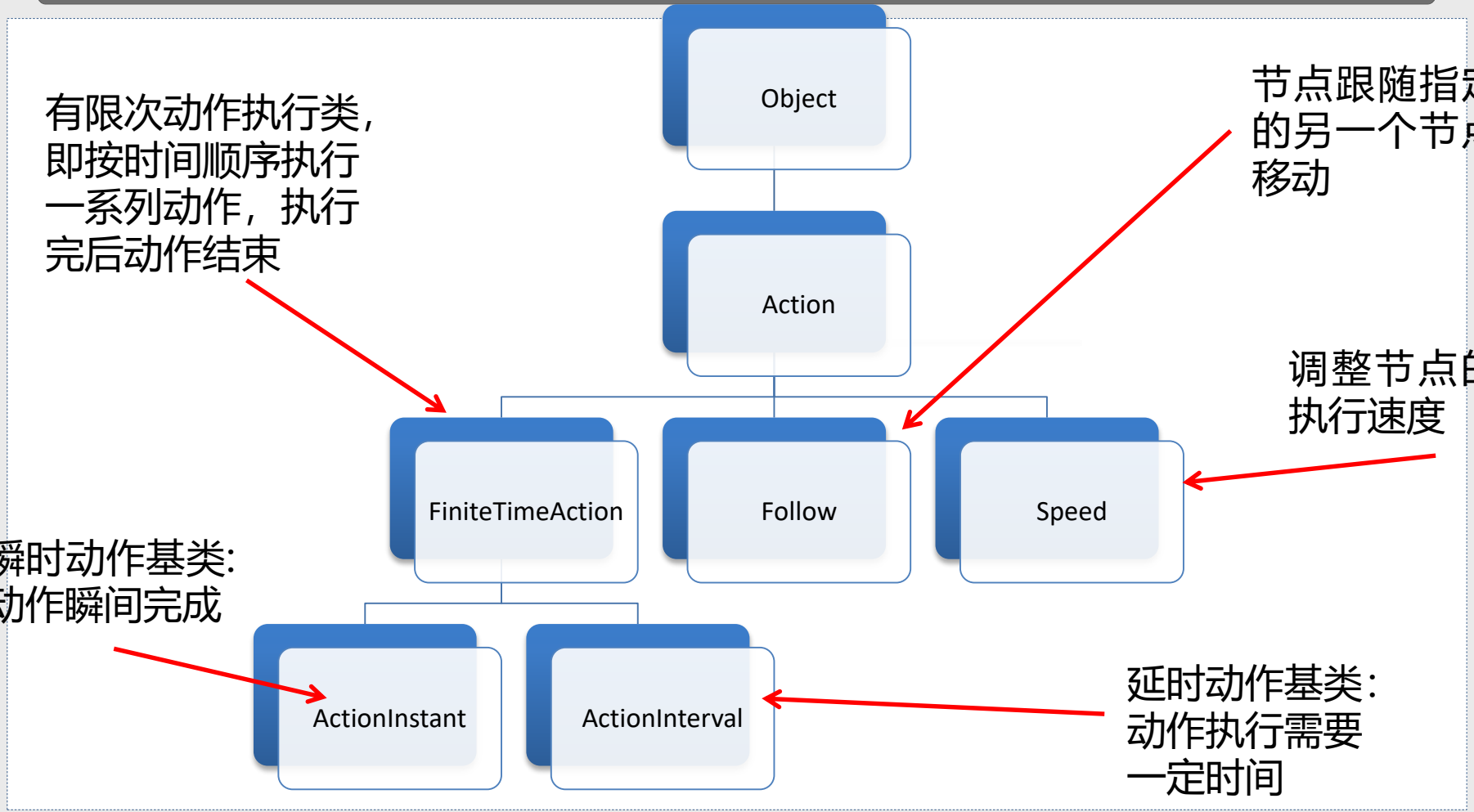
Cocos2d-x中的动作类

Cocos2d-x中的动作类

- Action：动作特效
- 引擎内部封装了很多Action的子类，它们分别实现各种各样的特效，如移动、旋转、跳动、缩放、闪烁等。
- 每个Node都可以通过runAction(Action* action)来播放一个特效。

- 动作特效代码示例：
- //创建一个精灵
- auto sprite = Sprite::create(“fish.png”);
- //创建一个移动的动作特效
- auto moveAction = MoveTo::create(2, Point(0, 0));
- //让精灵执行这个动作
- sprite->runAction(moveAction);

Cocos2d-x中的动作类



延时动作

MoveTo/MoveBy：移动动作。继承自ActionInterval

```
//参数: 持续时间，移动到的坐标
auto actionTo = MoveTo::create(2, Point(80,80));

//参数: 持续时间，移动的位移
auto actionBy = MoveBy::create(2, Point(80,80));
```

延时动作

RotateTo/RotateBy：旋转动作。继承自ActionInterval。

```
//参数: 持续时间，x和y轴上旋转到的角度
auto actionTo = RotateTo::create( 2, 45);

//参数: 持续时间，x和y轴上旋转的角度
auto actionBy = RotateBy::create(2 ,  360);

//创建一个由actionBy特效逆转的动作
auto actionByBack = actionBy->reverse();
```

延时动作

ScaleTo/ScaleBy：缩放特效。继承自ActionInterval。

```
//参数: 持续时间，缩放到的倍数
auto actionTo = ScaleTo::create(2.0f, 0.5f);

//参数: 持续时间，缩放的倍数
auto actionBy = ScaleBy::create(2.0f, 10.0f);
```

延时动作

SkewTo/SkewBy：倾斜动作。继承自ActionInterval。

```
//参数: 持续时间，x轴上倾斜到的角度，y轴上倾斜到的角度
auto actionTo = SkewTo::create(2, 37.2f, -37.2f);

//参数: 持续时间，x轴上倾斜的角度，y轴上倾斜的角度
auto actionBy = SkewBy::create(2, 0.0f, -90.0f);
```

延时动作

JumpTo/JumpBy：弹跳动作。继承自ActionInterval。

```
//参数: 持续时间，跳跃的目的坐标，跳跃的高度，跳跃的次数
auto actionTo = JumpTo::create(2, Point(300,300), 50, 4);

//参数: 持续时间，跳跃的位移，跳跃的高度，跳跃的次数
auto actionBy = JumpBy::create(2, Point(300,0), 50, 4);
```

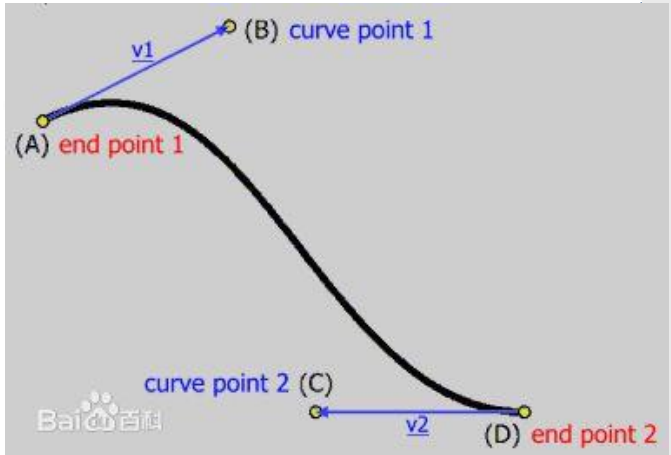
延时动作

BezierTo/BezierBy：贝塞尔曲线动作。继承自ActionInterval。

```
//参数: 获得当前屏幕大小
auto s = Director::getInstance()->getWinSize();

//新建一个贝塞尔曲线，定义它的控制点和结束点
ccBezierConfig bezier;
bezier.controlPoint_1 = Point(0, s.height/2);
bezier.controlPoint_2 = Point(300, -s.height/2);
bezier.endPosition = Point(300,100);

//创建一个贝塞尔曲线特效，参数：持续的时间，贝塞尔曲线
auto bezierForward = BezierBy::create(3, bezier);
```



延时动作

Blink：闪烁特效。继承自ActionInterval。

```
//参数: 持续时间，闪烁次数
auto action1 = Blink::create(2, 10);
```

延时动作

FadeIn/FadeOut：淡入淡出特效。继承自ActionInterval。

```
//参数: 淡入淡出持续时间
auto action1 = FadeIn::create(1.0f);
auto action2 = FadeOut::create(1.0f);
```

延时动作

TintTo/TintBy：染色特效。继承自ActionInterval。

```
//参数: 着色时间, 红色、绿色、蓝色的着色目的值
auto action1 = TintTo::create(2, 255, 0, 255);

//参数: 着色时间, 红色、绿色、蓝色的着色改变值
auto action2 = TintBy::create(2, -127, -255, -127);
```

延时动作

DelayTime：延迟特效。继承自ActionInterval。

```
//参数: 延迟的持续时间
DelayTime::create(2.0f);
```

组合动作

Spawn：同时进行。继承自ActionInterval。

```
//创建一个组合特效，使这些特效同时进行，执行时间以最长的特效为准
//参数: FiniteTimeAction类的动作1， FiniteTimeAction类的动作2.....
FiniteTimeAction类的动作N，NULL
 auto action = Spawn::create(JumpBy::create(2, Point(300,0), 50, 4),
RotateBy::create( 2,  720), NULL);
```

组合动作

Sequence：顺序执行动作序列。继承自ActionInterval。

```
//创建一个组合特效，使这些特效顺序进行
//参数: 动作特效1，动作特效2.....动作特效N，NULL
auto action = Sequence::create(MoveBy::create( 2, Point(240,0)),
RotateBy::create( 2,  540), NULL);
```

组合动作

Repeat/RepeatForever：重复和永久重复特效。继承自ActionInterval。

```
auto act1 = RotateTo::create(1, 90);
auto act2 = RotateTo::create(1, 0);
auto seq = Sequence::create(act1, act2, NULL);

//参数: 需要永久重复执行的ActionInterval动作特效
auto rep1 = RepeatForever::create(seq);

//参数: 需要重复执行的ActionInterval动作特效，重复执行的次数
auto rep2 = Repeat::create( seq->clone(), 10);
```

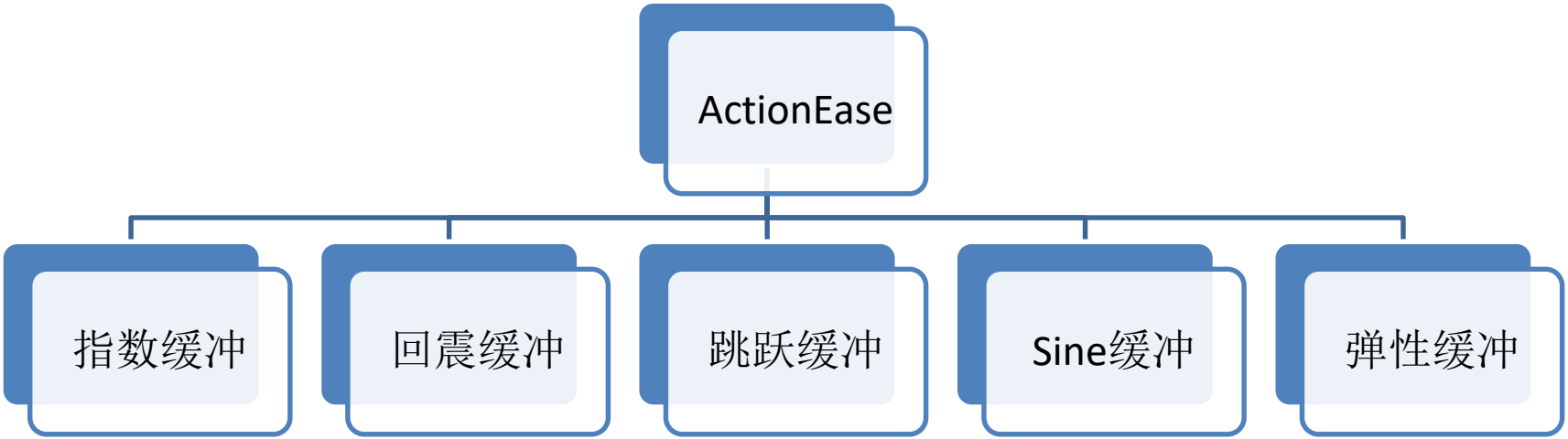
速度指定动作

Speed：线性变速动作。继承自Action。用于线性的改变某个动作的速度。

```
//参数: 要改变速度的ActionInterval类的目标动作， 改变的速度倍率
 auto action = Speed::create(RepeatForever::create(spawn), 0.5f);
```

Cocos2d-x的动作类

ActionEase：曲线变速缓冲动作。继承自ActionInterval。可以实现动作由快到慢、速随时间改变的变速运动。ActionEase共有5类动作，每类动作又有3个不同时期的变换：In、Out和InOut。

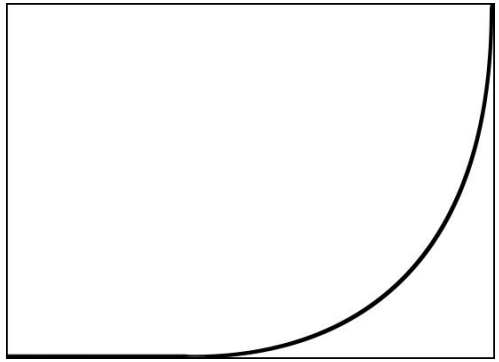


曲线变速缓冲动作

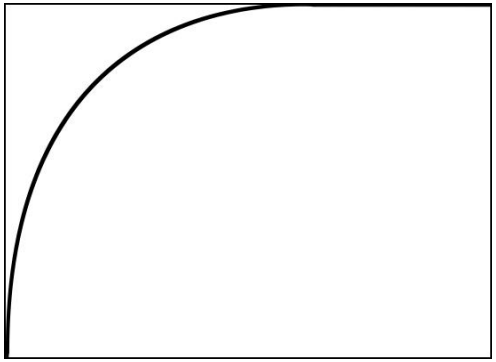
EaseExponentialIn：指数缓冲动作。继承自Action。用于指数曲线（如下图）的变换某个动作的速度。参考ActionEaseTest。

```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseExponentialIn::create(move->clone());
auto move_ease_out = EaseExponentialOut::create(move->clone());
auto move_ease = EaseExponentialInOut::create(move->clone() );
```

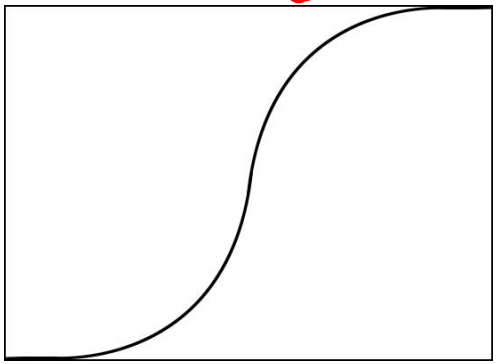
横轴代表实际动画时间，竖轴代表变换后的动画时间。



EaseExponentialIn



EaseExponentialOut

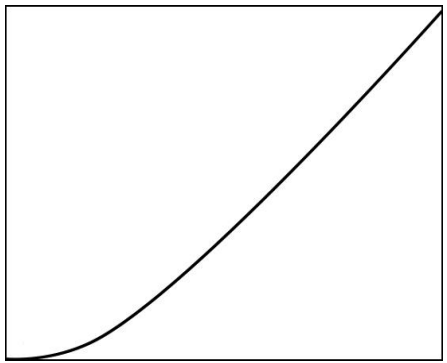


EaseExponentialInOut

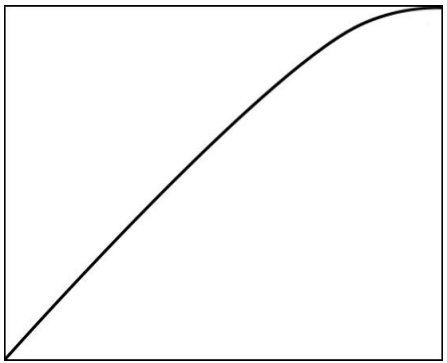
曲线变速缓冲动作

EaseSineInOut：正弦缓冲。继承自Action。用于Sine曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

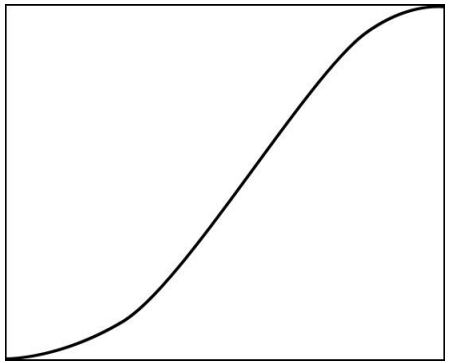
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseSineIn::create(move->clone() );
auto move_ease_out = EaseSineOut::create(move->clone() );
auto move_ease = EaseSineInOut::create(move->clone() );
```



EaseSineIn



EaseSineOut

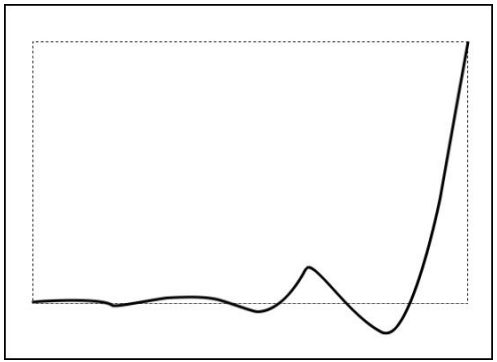


EaseSineInOut

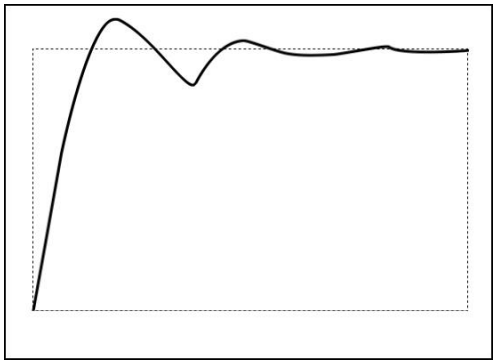
曲线变速缓冲动作

EaseElasticInOut：弹性缓冲。继承自Action。用于弹性曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

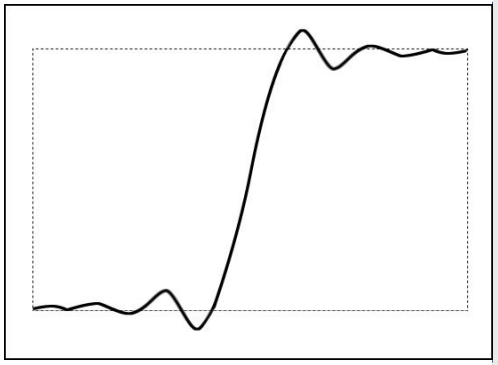
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseElasticIn::create(move->clone() );
auto move_ease_out = EaseElasticOut::create(move->clone() );
auto move_ease_inout1 = EaseElasticInOut::create(move->clone(), 0.3f);
```



EaseElasticIn



EaseElasticOut

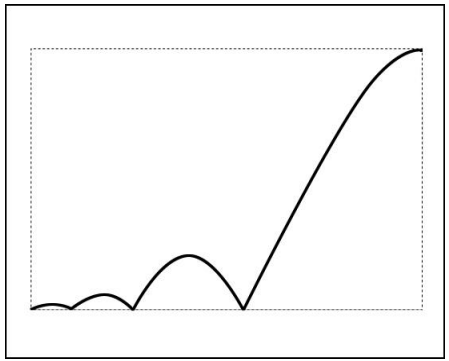


EaseElasticInOut

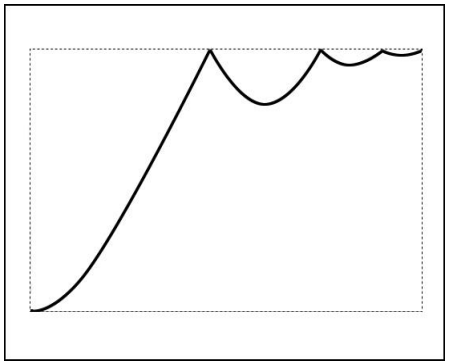
曲线变速缓冲动作

EaseBounceInOut：跳跃缓冲。继承自Action。用于跳跃曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

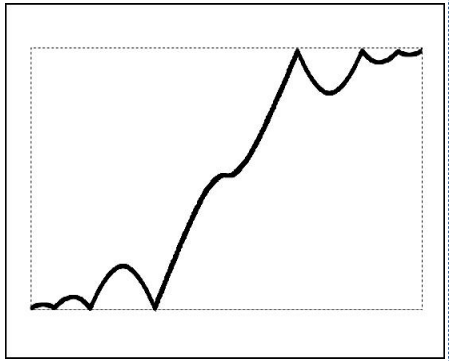
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseBounceIn::create(move->clone() );
auto move_ease_out = EaseBounceOut::create(move->clone() );
auto move_ease = EaseBounceInOut::create(move->clone() );
```



EaseBounceIn



EaseBounceOut

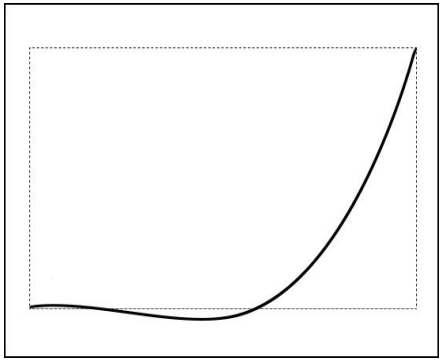


EaseBounceInOut

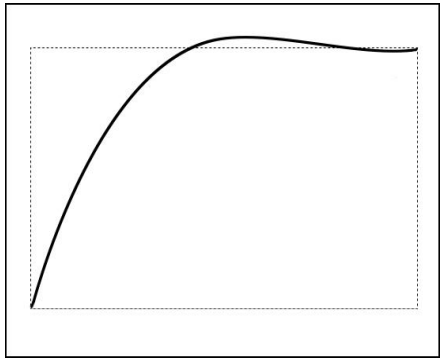
曲线变速缓冲动作

EaseBackInOut：回震缓冲。继承自Action。用于回震曲线（如下图）的改变某个动作的速度。参考ActionEaseTest。

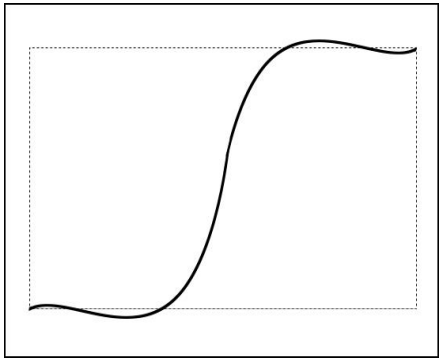
```
//参数: 要改变速度的ActionInterval类的目标动作
auto move_ease_in = EaseBackIn::create(move->clone());
auto move_ease_out = EaseBackOut::create( move->clone());
auto move_ease = EaseBackInOut::create(move->clone() );
```



EaseBackIn



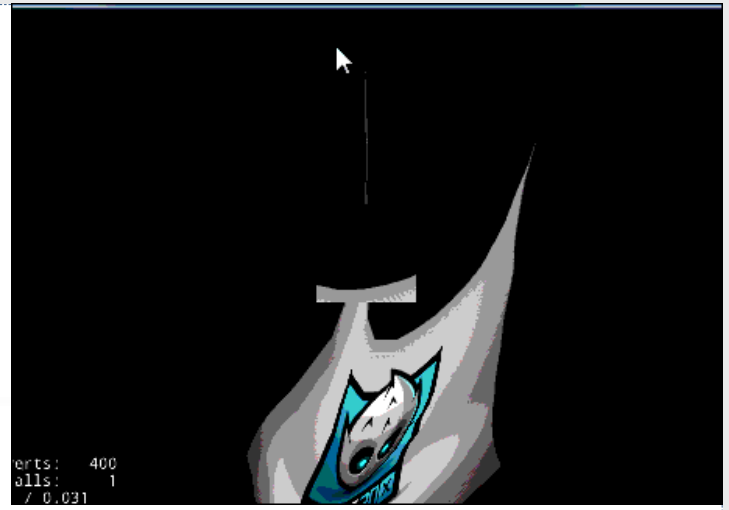
EaseBackOut



EaseBackInOut

Cocos2d-x的动作类

- Show：显示节点对象
  - auto show = Show:create();
  - node->runAction(show);
- Hide：隐藏节点对象



- ToggleVisibility：切换节点对象的visible属性
  - auto visibility = ToggleVisibility:create();
  - node->runAction(visibility);
- Follow：跟随动作（不能用于组合动作中）

本节内容

- Chapter 7
  - 计算机动画概述
  - 常见计算机动画技术
  - Cocos2d-x中的动作类
  - 课后作业

Cocos2d-x的动作类

课后作业：

动作特效代码示例：

```
//创建一个精灵
auto sprite = Sprite::create(“HelloWorld.png”);
 //创建移动的动作特效
?
```

请观看精灵移动视频，在模板中完成示例的? 动作代码，并在BB系统上传。

（Tips：视频中，一套完整的精灵动作特效重复了2遍）

课后作业