**FLASH抛投类手机游戏的设计与实现**

**摘要：**随着计算机硬件渲染技术的飞速发展，以及网络下载速度的不断提升，FLASH应用从最初的网络边幅广告、小游戏等轻量级应用，发展到了如今如火如荼的RPG网页游戏，客户端终合应用，手持设备APP应用全面开花的时代。在FLASH开发的众多游戏门类中，带有物理引擎的FLASH游戏正在逐渐成为一个闪亮的分支，流行于网络及手持设备上。

由于FLASH游戏的飞速发展，BOX2D作为一款性能优越的C++开源物理引擎，也移植到了FLASH平台。FLASH的手机游戏应用借助于BOX2D物理引擎的力量，开发出很多流行的APP应用，然而直接应用FLASH BOX2D物理引擎开发手机游戏，无论在开发的时间成本，还是在开发的难度上，都遇到了不小的挑战。

本文将通过对FLASH BOX2D物理引擎的深入分析，针对手机游戏的物理引擎应用的主要技术，比如碰撞盒绘制，物理与UI绑定等问题，提出了一套整合优化的开发方法。并运用组件封装的概念，设计与实现了一套快速，灵活，高效的基于FLASH BOX2D的手机游戏开发框架。最后通过实例，进一步验证了基于FLASH BOX2D物理引擎的手机游戏开发框架的实际应用价值。

**关键词：**FLASH，BOX2D，物理碰撞，手机游戏，开发框架

**Design and implementation of mobile game development framework based on FLASH**

\*\*\*

(School of Software, Shanghai Jiao Tong University, Shanghai,China)

**Abstract：**As computer hardware rendering technology rapid development , as well as the continuous improvement of network download speeds , FLASH applications margins from the initial network advertising, games and other lightweight application , developed to the RPG web game is now in full swing , the client eventually combined application handheld APP application full flowering of age. FLASH games in many categories developed in FLASH game with a physics engine is gradually becoming a shining branch networks and popular handheld devices .

As the rapid development of FLASH games , BOX2D superior performance as a C + + open source physics engine, FLASH is also ported to the platform . FLASH mobile gaming applications BOX2D physics engine by means of force, developed many popular APP application , however direct application FLASH BOX2D physics engine developed mobile games, both in development time and cost, or the difficulty in developing , have encountered no small challenge.

This article by FLASH BOX2D -depth analysis of the physics engine for mobile game physics engine applications, the main technologies such as collision boxes drawn with the UI binding and other physical problem, a method of developing an integrated optimization . And the use of component packaging concept , design and implementation of a set of fast, flexible and efficient FLASH BOX2D based mobile game development framework. Finally, an example to further validate FLASH BOX2D physics engine based mobile game development framework of practical value.

**Key words:** FLASH, BOX2D, Physical collision, Cutting games , Development Framework

**目录**

**摘要-------------------------------------------------------------------------------------1**

**ABSTRACT---------------------------------------------------------------------------1**

**1绪论----------------------------------------------------------------------------------3-7**

1.1 Flash手机游戏开发的背景

1.1.1手机游戏平台发展现状

1.1.2 Flash手机游戏发展现状

1.1.3 Flash游戏技术的发展

1.2 项目研究的目的和意义

1.3 本文主要内容

1.4 本文内容结构

**2 FLASH抛投类手机游戏的需求分析**

2.1 Flash手机游戏的概述

2.1.1为Android 创建应用程序

2.1.2为iOS 创建应用程序

2.1.3 FLASH对设备的支持

3.1.4 不受支持的FLASH API对象

2.2 Flash抛投类手机游戏的特点

2.2.1 抛投分析

2.2.2 物理碰撞

2.2.3 物体破碎策略

2.2.4 游戏关卡设计

2.3 Flash Box2D物理引擎的应用优势

2.2.1 Flash物理引擎现状12-13

2.2.2 Flash Box2D应用优势

2.4 Flash Box2D在手机游戏开发中存在的问题

2.4.1 Flash Box2D游戏开发的效率

2.4.2 Flash Box2D游戏开发的难度

2.4.3 Flash Box2D 手机游戏的优化

**3 FLASH抛投类手机游戏的技术基础与关键**

3.1 Flash Box2D技术基础

3.1.1 Flash Box2D核心概念

3.1.2在Flash Box2D中定义世界

3.2在世界中加入刚体

3.2.1创建刚体的定义

3.2.2创建刚体的形状

3.2.3 创建刚体的夹具

3.3 常用物理刚体的创建

3.3.1 简单多边形的创建

3.3.2简单多边形的快速创建

3.3.3凹多边形的创建

3.3.4凹多边形的顶点

3.3.5凹多边形的顶点的优化

3.3.6创建线条刚体

3.4 Box2D特殊效果的实现

3.4.1 浮力效果

3.4.2 流体效果

3.4.3 刚体的缩放效果

3.5 小结

**4 EasyBox2D游戏框架的设计**

4.1 EasyBox2D游戏框架总体结构

4.2 EasyBox2D结构设计

4.2.1 EasyBox2D总体结构

4.2.2 EasyBox2D管理类

4.2.3 FLASH 原生显示列表

4.2.4 Body工厂

4.2.5 Contact

4.2.6 数据模块

4.2.7 工具类模块

4.3游戏编辑器的结构设计

4.4.1场景数据格式的设计

4.4.2场景数据的绘制模式

4.4.3场景数据的搭建模式

4.4.4场景数据的预览

4.4.5场景数据的保存

4.4.6游戏编辑器设计小结

**5．EasyBox2D游戏框架的实现**

5.1 EasyBox2D游戏框架的逻辑实现

5.1.1世界封装类

5.1.2 EasyBox2D工厂类

5.1.3 model包

5.1.4 debug manage包

5.1.5 多边形绘制辅助工具包

5.1.6 event包

5.1.7 utils包

5.2游戏编辑器的逻辑实现

4.5.1Model包

4.5.2Factory包

4.5.3Parser包

5.3 EasyBox2D游戏框架的总结

**6基于EasyBox2D框架的手机物理游戏开发实例**

6.1 游戏架构概述

6.1.1 用户注册模块

6.1.2 城堡建造模块

6.1.3 攻打城堡模块

6.2 游戏中物理城堡的实现

6.2.1 物理城堡概述

6.2.2 物理城堡的构建

6.3攻打城堡模块的实现

6.3.1 游戏中投弹攻打的实现

6.3.2 相应公式介绍

6.4 本章小节

**7 结论和展望**

7.1 结论

7.2 展望

**参考文献**

**个人简介**

**导师简介**

**致谢**

**第一章 绪论**

**1.1 Flash手机游戏开发的背景**

早期的flash应用，因为得不到GPU硬件加速，所有的图片渲染工作全都由CPU来完成。而且当时的网络下载速度也比较慢，对于视觉绚丽的游戏动效，下载速度慢，直接影响了FLASH游戏的体验性。因此当时主要是以矢量图为主，位图动效为辅的游戏。随着Adobe开发的新技术stage3D的诞生，极大的提升了FLASH的位图渲染能力。Stage3D的Flash Player和Adobe AIR的API提供了充分的硬件加速架构，使桌面浏览器和IOS和Android应用程序都能够使用GPU硬件加速。而应用BOX2D物理引擎开发的FLASH游戏，源于其真实的操控体验，受到了越来越多玩家的肯定，《愤怒的小鸟》《[水果忍者](http://www.baidu.com/s?usm=1&wd=%E6%B0%B4%E6%9E%9C%E5%BF%8D%E8%80%85%E4%B8%AD%E6%96%87%E7%89%88&f=3&ie=utf-8&rsv_cq=%E5%88%87%E6%B0%B4%E6%9E%9C&srcid=20108&zx_tpn=0&rsv_dl=0_right_baike_20108)》等都是比较成功的例子。

**1.1.1手机游戏平台发展现状**

目前的手机游戏主要运行在智能手机上，智能手机作为一种运算能力及功能比传统功能手机更强的手机，对游戏的支持有了很大的提升。现在使用最多的操作系统有：Symbian、Windows Phone 7、iOS、Android和 BlackBerry OS。

Symbian作为昔日智能手机的王者，在2005年至2010年曾一度风骚。随着互联网的迅猛发展，由于其对新兴的社交网络和web 2.0内容支持欠佳，塞班占智能手机的市场份额日益萎缩。2010年末，其市场占有量已被Android超过。

Android系统通过开源、联盟，凝聚了几乎遍布全球的力量，而且Android系统的底层操作系统是Linux，Linux作为一款免费、易得、可以任意修改源代码的操作系统，吸收了全球无数程序员的精华，这在技术上确保了它的领先优势。目前Android系统已经成为了市面上主流的智能手机操作系统。

iOS作为苹果移动设备iPhone和iPad的操作系统，在App Store的推动之下，成为了世界上引领潮流的操作系统之一。

Windows Phone，早在2004年时，微软就开始以“Photon”的计划代号开始研发Windows Mobile的一个重要版本更新，但进度缓慢，最后整个计划都被取消了。直到2008年，在iOS和Android的冲击之下，微软才重新组织了Windows Mobile的小组，并继续开发一个新的行动操作系统。Windows Phone起步早，发展慢。到目前来说，Windows Phone7已经应用在NOKIA手机上，不过相比Android 和iOS还有很长的路要走。

Blackberry系统，即黑莓系统，是加拿大Research In Motion(简称RIM)公司推出的一种无线手持邮件解决终端设备的操作系统，由RIM自主开发。它和其他手机终端使用的Symbian、Windows Mobile、ios等操作系统有所不同，Blackberry系统的加密性能更强，更安全。黑莓系统稳定性非常优秀，其独特定位也深得商务人士所青睐。可是也因此在大众市场上得不到优势，国内用户和应用资源也较少。

**1.1.2 FLASH手机游戏发展现状**

目前Flash游戏可以大体上分为三大类：1.单机游戏；2.网页游戏；3.手机游戏类。

Flash很早就有了，开始只是用来做一些操作简单的单机小游戏。它们是由flash软件制作的，主要是放在网站上供大家休闲娱乐的游戏。一般比较小巧，在宽带上网的环境下1分钟就可以开始进行[游戏](http://baike.baidu.com/view/2468.htm)了。

后来随着国内网络速度的提升，Flash网页游戏就开始火起来，其优点是便利性，随时可以玩，而且是以分钟为节点的，不需要进入的很深，而客户端游戏是以1-2两个小时为节点。另外随着网络速度提升了 ，FLASH网页RPG游戏从客户端游戏上借鉴来了很多的玩法，然后在网页上实现。

近年来，在IPhone 和Android 等智能终端推出之后，手机游戏发展迅速。通过移动网络为用户提供单机游戏、手机网游显现越来越热的趋势。2009 年，中国手机游戏总体市场规模18 亿元，同比增长38.5%；手机网游市场规模3.2 亿元，同比增长88.2%。从2007年到2012 年，中国手机游戏市场规模复合增长率为45.4%，预计未来3年复合增长率超过50%，2013 年市场规模超过50 亿元。手机游戏活跃用户在2009 年超过1100 万人，2012 年达到了1亿，预计在2014年将达到4亿。

目前手机游戏中，利用物理引擎开发的游戏种类比较多，而且玩家评价比较好，有较高的粘度，分析一下主要有以下几个大类：

1.切割类游戏

这类游戏的玩法主要是用鼠标拉出线条来切割不同的物体，使得切割后的形成的物体碎片并且获得物理属性，然后根据相应的物理属性来更新物体的状态，以达到游戏所设定的目标。这类游戏代表作品主要有《智慧之刃Splitter》，《冰破维京船Ice Breaker》等。

2.反弹类游戏

反弹类游戏主要是利用物体相互碰撞后的自然折返为基础，通过加入各种趣味性的游戏关卡设计而成的一类手机游戏，比较简单的有台球物理。其他的还有类似《点火Pyroll》，《炮打笑面人Roly-Poly Cannon》，《魔法水滴Enigmo》等。

3.平衡类游戏

平衡类游戏主要是通过判断不同材质的物体所具有的不同密度，根据体积的不同，形成不同的质量，然后通过物体的堆积达到游戏所设定的平衡状态。这类的游戏代表主要有《完美平衡Perfect Balance》，《巨塔Huje Tower》，《图腾破坏神Totom Destroyer》等。

4.绘制类游戏

绘制类游戏通常会让玩家控制画笔，根据玩家选择的材质，以及绘制的不同形状，形成不同的具有物理属性的物体，去达成游戏所设定的不同的目标。比如，画一个圆，就形成一个球，球会在重力作用下下落，形成一定冲力。这类游戏主要有《神奇画笔Magic Pen》，《魔法画笔Draw Play》等。

5.道具摆放类游戏

道具摆放类游戏类似于绘制类游戏，所不同的是游戏会预先给予玩家各种不同的道具，而不是由玩家去绘制来形成物体。玩家可以将不同的道具摆放在各种可能的位置上，点击完成后赋予物理属性，来完成游戏所设定的目标。这类游戏主要有《多米诺传动系统Dynamic Systems》，《疯狂机器》，《粒子对撞Collider》等。

6.抛投类游戏

通过炮塔、弹弓一类的抛投设备，将炮弹等抛投物体按相应的角度与力量投射出去，给目标物体以撞击等损害，这是其基本原理。在此基础上通常可以发展出具有不同风格，不同轨迹的抛投对象。这类游戏的典型代表就是《愤怒的小鸟》。

**1.1.3 FLASH游戏技术现状**

Flash使用矢量的技术制作成动画，使文件大大减小。矢量图优点是拉大的时候还是那么清晰，但占的容量不会很大，Flash可以把矢量图在制作动画的时候减小其容量，再进行下载，从而使动画在网络上更快速的播放，实现动画交互。

　　很多玩家都希望能省去客户端游戏下载的漫长过程，如果能边下载边玩，就会有更多的人去玩，而这样的低起点正是Flash技术可以做到的。在进行一款Flash网游的过程中，Flash技术后端已经在传送数据了，是即时传送的。客户端网游是一次性把所有的内容下载完毕，Flash页游则是在处理某个内容的时候，他的后端才开始下载。比如玩某一款Flash页游，经常可以看到这样情况，在打开某页面的时候，会先loading，然后画面再跳出来，这是Flash在后端下载内容，然后才做到即时打开页面，这样实现了边下载边玩，这是Flash最大的技术特点所在。但也是因为即时下载，Flash页游的玩法很多都是回合制，很少即时类，是因为在玩游戏的过程中，前端的表现、后端的数据不断在传输，如果是即时战斗玩法会导致延迟，但是这种玩法在回合制里便会削弱，这是回合制的优势，但随着网速的提升也都是可以改善的。

3D是 Flash将来的发展方向。虽然做3D游戏的成本会比较高，但对于游戏公司来说，3D产品不容易被同质化。3D游戏的下载量大，下载时间长可能会是个问题，但将来随着国内网络速度的进一步提升都会得到解决。

Starling 是一个基于Stage3D（这是Flash Player11及Adobe AIR 3中新增的为3D加速功能所提供的API）所开发的一个能够使用GPU来加速的2D Flash应用程序的ActionScript3框架。Starling主要是为游戏开发而设计的, 但是它的用途不仅限于此。Starling最大的好处在于你可以很快地写出使用GPU加速的应用程序而不必接触那些复杂的底层Stage3D API。大多数 Flash 开发人员希望利用这种能力提高 GPU 的加速功能。

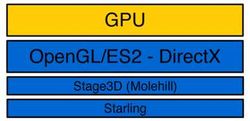
[](http://baike.baidu.com/picview/8717067/8691134/0/0b55b319ebc4b745f0dac8becffc1e178b8215cf.html)

图1. Stage3D (Molehill) 分层位于 Starling 之上

**1.2 研究的目的和意义**

截止到2013年2月，作为真正跨平台、跨浏览器并在各种浏览器版本中提供一致网络体验的Web互动媒体标准，Flash Player全球电脑装机量为13亿台，是PS、Wii和XBOX系列游戏主机销量总和的5倍，Flash Player新版本发布一周内浏览器更新量为4亿+；

Flash游戏应用通过Adobe AIR打包，发布成iOS和Android等移动平台原生App并可开启自身硬件加速的移动设备数为5亿台，目前已经发布超过2.8万个移动游戏App。Facebook上96%的游戏是用Flash开发，每月四千万人玩的“开心农场2”采用的也是Flash新技术Stage3D，同时在中国，排名前九位的Flash游戏（页游）带来了超过7千万美元的月收入

众所周知，移动游戏对于独立开发商们来说是最热门的平台。Flash游戏服务公司Mochi Media的一个调查显示，86%的Flash游戏开发商都考虑在转向移动平台。

　　不过63%的开发商表示，仍将开发页游来测试市场反映并为开发移动平台的游戏做准备。　　该调查目前已经进行了5年，是对游戏行业里页游开发商观察最细致的。Flash工具是个很好的东西，因为它降低了开发商入行的门槛。

　　说到要转向的平台，开发商们大多数认为iOS和Android平台将是首选(81%iOS，78%Android)。76%的开发商认为需要为尽可能多的平台开发游戏，包括在线和移动平台。　　开发商们认为Adobe Air是把他们的游戏转向移动市场最好的工具。

因此，我们选择FLASH 手机游戏开发作为本课题的内容。选取这个课题进行深入研究具有以下意义：

1.对从flash网页游戏开发转向FLASH手机游戏开发的人员来说，全面了解FLASH手机游戏开发的现状，并在此基础上提高对FLASH BOX2D开发的认识

2.通过对FLASH 手机游戏的各个功能模块进行细致的分析，为实际开发提高开发的效率，保证开发的质量。

3.通过实例的分析与实现，巩固对FLASH BOX2D手机游戏开发的认识，并找个不足之处，为将来的进一步提升打下坚实的基础。

**1.3 本文的主要内容**

在开发带物理引擎的FLASH游戏中，FLASH常用的引擎主要是BOX2D。应用FLASH BOX2D构建物理世界的过程是比较繁琐，相当枯燥的，需要程序员与设计师的长时间沟通与协作，在不断磨合中，才有可能达到精确、合理的实现游戏物理世界的目标。如何简化物理游戏开发，优化碰撞盒数据的自动采集，实现物理与可见UI之间的精确绑定，是物理游戏快速成功构建的关键。

本文将通过对Flash Box2D核心概念的阐述，进而分析了物理基本形状的构建，并且逐步延伸到对不规则物体的模拟，最终实现复杂物体物理碰撞的搭建。在此基础上，分析设计、实现物理碰撞盒的图形化绘制。最后应用面向对象的设计方法与相关设计模式，对FLASH BOX2D物理引擎进行二次封装，实现物理碰撞盒与UI的快速绑定，最终整合成一套快速开发BOX2D物理游戏的框架。

**1.4本文内容结构**

论文的重点是通过对FLASH BOX2D的技术模块应用的分析与实现，实现在FLASH 手机游戏开发中的实例应用，因些论文将以此为主线，分析FALSH手机游戏的发展现状，开发技术，以及FLASH BOX2D的各种场景的应用，最终形成比较完整的FLASH手机游戏开发框架。论方共分6个章节，各章节安排如下：

第一章绪论，介绍了FLASH 手机游戏发展的现状，以及FLASH BOX2D手机游戏的发展，并论述的项目的目的与意义，以及论文主要的内容。

第二章介绍了FLASH BOX2D的基础技术，对基本的FLASH物理游戏进行了深入浅出的分析，以使读者对FLASH BOX2D有了全面基本的认识。

第三章主要是结合手机游戏的关键技术，按FLASH 物理游戏开发常用模块进行了分析与实现。

第四章在之前FALSH BOX2D手机游戏技术的基础上，整合了FLASH手机游戏各功能模块，形成总体框架设计。

第五章FLASH 手机游戏siege-duell的设计与实现。

最后一章是对论文的总结各展望，对论文特色进行阐述，同时分析设计方法存在的不足，并对各模块的应用在将来的工作中的进一步探索进行了展望。

**第二章FLASH抛投类手机游戏的需求分析**

**2.1 Flash抛投类手机游戏的特点**

随着2007 年AIR 的发布，Flash开发人员第一次可以为Windows、Mac OS X 和Linux 平台创建独立的跨平台富因特网应用程序(RIA)。这些AIR 桌面应用程序不仅具有原生应用程序的外观和体验，而且可以利用原生操作系统的功能，例如本地文件访问、原生菜单和用户界面元素以及操作系统特定事件。

**2.1.1为Android 创建应用程序**

在开发基于 Flash 的移动应用程序之前，首先深入内部解释一下Adobe 如何处理一个

Flash 文件(.fla)，并将它作为适用于Android 的.apk 文件或适用于iOS 的.ipa 文件发布。

在 Android 上创建应用程序与在桌面系统中创建AIR 应用程序没有太大差别。AIR for

Android 运行时提供一个环境，在其中开发人员可以使用Flash 技术创建应用程序，并将它

作为独立于任何浏览器之外的应用程序进行交付。用户需要在自己的Android 设备上安装

AIR for Android 运行时环境，然后在其上运行基于Flash 的Android 应用程序。

AIR for Android 嵌入跨平台虚拟机Flash Player，用于运行使用Adobe Flash 或Flash

Builder 创建的多媒体文件和应用程序。在AIR 应用程序内部，可以编程访问已有的Flash

Player API 调用以及一些增强功能，如实现基于矢量的绘图、多媒体支持以及完整的网络

连接栈。

AIR for Android 还嵌入了SQLite，这是一个支持本地数据库访问的数据库引擎。它是

一个非常轻量级的、开放源代码的、跨平台SQL 数据库引擎，嵌入在许多桌面系统和移动

产品中。与大多数SQL 数据库不同，它不需要单独的服务器进程，并且使用标准文件来存

储整个数据库(表、索引等)。

当为 Android 发布一个Flash 文件时，.fla 源代码文件和其他源文件都将由ActionScript

编译器转换为一种称为“ActionScript 字节码”(ABC)的二进制格式。这种ABC 封装在一

个.swf 文件内部(参见图1-2)，然后这个.swf 文件和相关支持资源文件打包成一个Android

包(.apk)，准备安装到设备上。

在运行时，Android 应用程序内部的.swf 文件由ActionScript 虚拟机(AVM2)进行处理，

这个虚拟机是AIR for Android 运行时的一部分。AVM2 加载ABC 文件到内存中，并进行

解码。然后通过一个解释器来运行字节码，并作为原生机器代码执行(参见图1-3)。这个由

AVM2 执行的字节码编译过程是Android 平台所特有的。

**2.1.2为iOS 创建应用程序**

Android 应用程序在独立安装的AIR 运行时环境之上运行，而iOS 应用程序却将运行

时代码封装在应用程序内部，使得它可以独立运行。因而在其中创建iOS 应用程序的进程

也毫无疑问地有很大区别，下面将详细解释。

在为 iPhone 编译Flash 项目时，ABC 代码由“低级虚拟机”(LLVM，Low Level Virtual

Machine)编译器编译，它是一个开放源代码的编译器架构，用于为iOS 生成机器码。(Apple

公司自己也使用LLVM)。但是，如图1-4 中所示，在其Packager for iPhone 中Apple 公司

提供了一个LLVM 的ActionScript 前端，用于处理Flash 文件。

虽然 AVM2 支持JIT for Web 和AIR，但是LLVM 使用预编译生成原生的ARM 汇编代码，并被封装到iPhone 可执行文件内。这种.ipa 文件还包含.swf 文件，其中包含资产和配置文件。

在发布过程中使用 AIR Developer Tool(ADT)将.p12 证书添加到.ipa 应用程序文件中

以便于开发者认证。结果得到的.ipa 文件是一个原生的iPhone 应用程序，可以安装到

iPhone 设备上。

**2.1.3 FLASH对设备的支持**

FLASH对于 Android 和iOS API 的支持总体来讲是比较类似的，对于摄像头和麦克风等输入设备的支持有一些区别，具体情况如下表。

|  |  |  |
| --- | --- | --- |
| 功能模块 | Android | IOS |
| 触摸/手势 | YES | YES |
| 加速度计 | YES | YES |
| 地址位置传感器 | YES | YES |
| 摄像头 | YES |  |
| 麦克风 | YES |  |
| StageWebView | YES |  |
| 摄像头媒体库/照片库 | YES | 仅支持添加到库 |
| 文件 I/O | YES | YES |
| SQLite 数据库 | YES | YES |
| 通过 URL 协议加载移动设备的能力，例如电话(tel:)、电子邮件(mailto:)和短信(sms:) | YES | YES |

另外还有一些不支持的核心移动服务，比如原生UI 控制，音乐播放器和音乐库，对蓝牙的控制，联系人列表的操作，日历，参数设置等。

支持最多的方面集中在多点触摸和手势、加速度计、持久性文件和数据库存储。AIR for

Android 和Packager for iPhone 在与各自设备的其他部分(包括硬件和系统服务)集成方面都

要弱于其对应的原生SDK，因而在创建应用程序时需要充分考虑到这些局限。

**2.1.4不受支持的FLASH API对象**

在创建移动应用程序时，需要访问核心AS3 库和AIR API 扩展的许多部分。但是，在

Android 和iOS 上并不支持所有核心和AIR 功能。下面是不受支持的AS3 API 对象或成员

列表：

|  |
| --- |
| Accessibility |
| DNSResolver |
| DockIcon |
| DRMManager |
| EncryptedLocalStore |
| HTMLLoader |
| LocalConnection |
| NativeApplication 包括exit(), isSetAsDefaultApplication(), menu,和startAtLogin |
| NativeMenu |
| NativeProcess |
| NativeWindow 和NativeWindow.notifyUser() |
| NetworkInfo |
| PDF 支持 |
| PrintJob |
| Socket 支持(DatagramSocket, SecureSocket, ServerSocket) |
| Shader |
| ShaderFilter |
| Socket.bind() |
| StorageVolumeInfo |
| XMLSignatureValidator |

除此之外，虽然能够使用Flash Builder 来创建AIR for Android 应用程序，但是Flex

MXML 架构在Android 上并没有得到官方支持，并且与iOS 不兼容。

**2.2 Flash抛投类手机游戏的特点**

**2.2.1 抛投分析**

游戏中任何将物体扔，抛，撞，弹到空中的运动，都可以理解为抛投运动。抛投的概念就是我们选择一个对象，并对其进行拖拽，沿某个方向移动一段距离，然后忪开鼠标，物体沿着拖拽的方向继续移动。

通常从游戏的分析可以看出，我们有多种方法来计算物体运行的速度，下面我们将主要的几种计算方法进行分析

第一种，二维空间中计算抛物线最简单的方法就是将向量拆分成竖直分量和水平分量，然后分别计算。由于分量之间是完全独立的，所以可以分别对其进行计算。

在Flash游戏中可以，我们可以更直接的理解为显示对像每帧向左移到5个像素，也就是说该物体在X轴上的速度向量为Vx=5。

抛投的关健是如何确定速度向量的大小，而我们在模拟物理运动过程时，所应用的公式是：旧的位置+速度向量=新的位置，那么我们计算速度向量就可以由前面的公式变化而来：速度向量=新的位置-旧的位置，在拖拽的过程中，会在每一帧上产生新的位置，用当前帧的位置减去前一帧的位置，就可以行道这一帧所移动的距离，也就是物体的速度向量。当然在实际开发中我们还要考虑重力作用和摩擦力，这样就能使物体运动更加真实可靠。

第二种方法是先根据条件计算出抛物线公式，我们在抛投类游戏中通常还会考虑到的几何学知识就是抛物线，抛物线是轴对称的曲线，其开口可以是任意方向。

抛物线表达式通常有标准式和顶点式

标准式：y＝ax^2+bx+c，a,b,c为常数

顶点式y=a(x-h)2+k,顶点为（h,k）,对称轴为x=h

在理想状态下，我们可以认为抛物线的形状是由三个因素决定的，第一个是顶点，也就是抛物线的顶端，第二个是对称轴，它经过项点，并将抛物线对称分割，第三个是a常数，常数a代表了抛物线的开口方向和开口大小，如果A是正数，则抛物线开口向上，如果A是负数，则抛物线的开口向下。A的绝对值越大，抛物线的开口越小。

第三种方法是应用成熟的游戏引擎，这也是比较高效的选择，我们可以选择Box2D物理引擎来模拟物体的抛投

**2.2.2 物理碰撞**

Flash关于物体碰撞的检测方法有很多，而这也是抛投类游戏必不可少的，必须要思考解决的问题。通常来讲，Flash原生的碰撞检测有hitTestObjet()和hitTestPoint()，但是有很多抛投类游戏都不止于使用这些原生的方法，为了达到完美的游戏效果，通常还有几种更精确的方法，下面我们一一进行分析。

物体经过抛投之后，就要进入碰撞检测阶段了，这也是Flash抛投类游戏必不可少的阶段。

碰撞检测的思想非常简单，我们只要知道两个物体是否有在同一时间内某个部分处在了同一位置上。当然，也许物体不只两个，

这就需要知疲乏其中一个是否和共它的物体发生了碰撞【】。

Flash碰撞检测的方法有很多：可以是判断两个显示对像是否重叠，也可以对物体实际像素进行检测，还可以根据距离来判断，当然

如果使用Flash Box2D物理引擎的话，则可以使用Box2D提供的碰撞检测。各种碰撞检测各有特点，下面我们一一分析。

hitTestObjet()主要用于判断两个显示对象间是否发生碰撞，这个最简单，也是最高效的碰撞检测方法。但是这种方法不够精确

因为它是判断两个显示对象的矩形边界是否相交。下图中只有正方形是发了了碰撞，但由于实际上Flash检测的是矩形边界，因此这三组

对象都被检测发生了碰撞，因此其局限性就在于只能进行初略的碰撞检测，或者是矩形的碰撞检测。

hitTestPoint()方法用于判断某个点与显示对象间是否发生了碰撞。默认情况下，这种方法也会遇到点与显示对象矩形边界发生

碰撞的问题，但是它有一个参数shapeFlag可以控制在碰撞中是检测显示对象的可见图形还是矩形边界。这种方法的本生限制也制

约了它的适用范围，它比较适用于非常小的显示对象与其他显示对象之间的碰撞。

基于距离的碰撞检测，它的主要思想是先确寂静两个物体分开的最小距离，再检测当前它们之间的距离，然后比较这两个距离的大小。

如果当前距离小于最小距离，那么就断定两个物体发生了碰撞。从上面的描述不难发现，这种碰撞检测比较适合于圆形物体之间。

不规则形状对象的碰撞检测主要是基于像素的检测，通过比较两个BitmapData对象，检测它们是否有像素的重叠来判断是不发生了碰撞。

这种方法虽然可以适用于任何形状，比较精确，但是对于每个显示对象，尤其是矢量对象，都要准备一份BitmapData来用于碰撞检测，

这对内存的消耗以及CPU的消耗比较大，对于Flash抛投类手机游戏的开发，必须时间注意其对性能的影响。

Box2D物理引擎来碰撞检测，Box2D是一个非常强大的2D物理引擎，可以帮我们实现精确的碰撞检测

Box2D中获取碰撞对象的方法有两种。一个是通过world.GetContactList().bodyA和bodyB来获取碰撞双方；

另外一个是自定义Box2D.Dynamics下的b2ContactListener类，侦听碰撞后的事件，然后做进一步的处理。

下面我们来看看具体的实现方法。

1.用world.GetContactList()方法获取碰撞对象

world.GetContactList()会返回一个b2Contact对象。b2Contact用来管理碰撞的shape，任何有超过两个及以上接触点的刚体，

Box2D都认为发生了碰撞，并用b2Contact来管理。

通过b2Contact的GetFixtureA()和GetFixtureB()方法，我们可以获取碰撞对象的b2Fixture属性引用，进而获取碰撞对象。

二、用b2ContactListener获取碰撞对象

b2ContactListener是Box2D对象碰撞检测侦听器，当刚体之间发生碰撞时，Box2D引擎会自动调用这个类的相关方法，

然后做进一步的处理。这些方法包括：

BeginContact：当碰撞发生时触发该方法

EndContact：当碰撞结束时触发该方法

使用b2ContactListener的方法有两种。

一、修改b2ContactListener的BeginContact和EndContact方法，添加碰撞处理代码；

二、继承b2ContactListener类，如自定义一个myContactListener子类，然后重写子类的BeginContact和EndContact方法，

然后将子类传递给world.world.SetContactListener()方法，自定义碰撞侦听器。

在Flash中编写碰撞检测代码时，通常我们会用A对象(如游戏主角)的hitTest或hitTestObject方法检查它与另外一个对象的碰撞，

当碰撞发生时，分别对A和B进行碰撞处理。这里我们很清楚哪个是A对象(游戏主角)，那个是B对象(敌人)。

但是在b2Contact或b2ContactListener中，我们获取的bodyA和bodyB无法知道哪个是游戏主角，哪个是敌人，

比如我现在想找到碰撞的对象是否是游戏主角，那么就得分别确认一下bodyA和bodyB了，游戏碰撞对象种类越多，判断越复杂。

需要检测大量碰撞的时候，所有需要在碰撞时触发的事件都要写到那一个继承自b2ContactListener的类中未免显得杂乱，不易管理，

如果能把刚体的碰撞处理分开写到各自的UserData类中看起来就好多了。

包围盒方法的基本思想是通过建立对象的包围盒层次来逐渐逼近对象的几何模型,从而用体积略大而形状简单的包围盒代替复杂的几何对象参加碰撞检测,通过包围盒间的相交测试快速地排陈不相交的基本几何元素对,以减少相交测试的次数。

一般的碰撞检测分为两个检测阶段：粗略检测阶段主要是利用物体的包围盒来进行粗略的层次碰撞检测，目的是快速过滤掉没有发生碰撞的物体。如果物体的包围盒发生了碰撞，那么就进入到精细检测阶段进行更精细的判断。

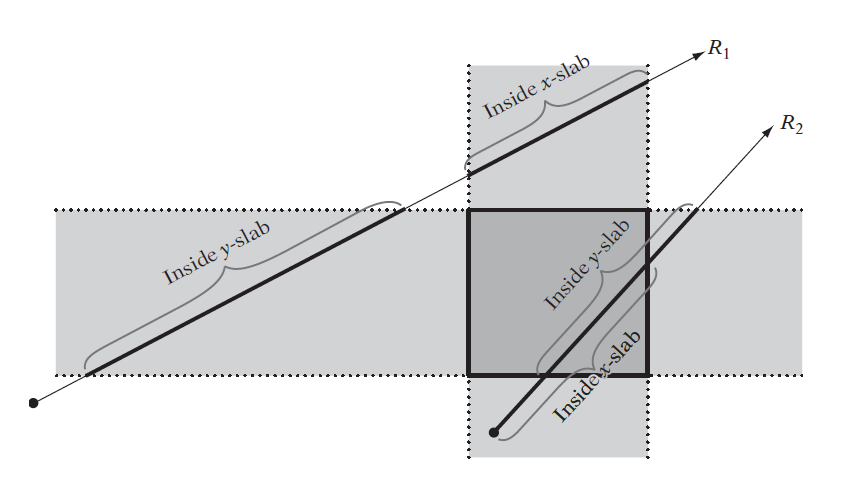
精细检测阶段也可以继续包围盒的思路，把物体细分，对于物体的每个部件确定包围盒，然后再继续进行碰撞检测，直到系统规定的，可以容忍的误差范围。

box2d使用了一种叫做slab的碰撞检测算法。所谓slab是指两个平行平面之间的空间，由此我们可以把3D空间中的AABB盒子看做是由AABB的3组平行面形成的3个方向的slab的交集。根据这个定义，我们可以得到以下两个结论：

1.如果一个点在AABB中，那么这个点必定同时在这3个slab中。

2.如果一条射线和AABB相交，那么这条射线和3个slab的相交部分必定有重合部分。

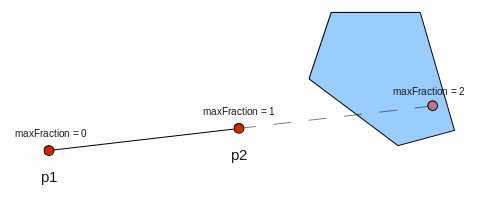
这很容易理解，如果射线和3个slab的相交线段没有重合，那么这些线段就不可能同时存在于3个slab中，也就不可能在AABB盒子中。下图展示了2D空间中射线R1和R2与AABB相交的情形。R1在x-slab和y-slab中的线段没有重合部分，因此R1和AABB不相交。R2在x-slab和y-slab中的线段有重合部分，因此R2和AABB相交。



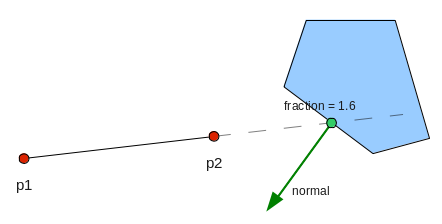
根据上述原理，检查2D中射线和AABB的碰撞，只需要检查射线和x-slab，y-slab的交线是否有重合。

首先我们需要得到射线和slab边界平面的交点。射线可以用参数方程表示为R(t) = P0 + t d , (其中P0为射线起点，d为射线的方向向量)，平面由隐式定义方程X·n=D， (其中X为平面上的点，n为平面法向量，D为原点到平面的距离)给出。将平面方程中的X用P0 + t d替换解得交点的参数t=(D−P0·n)/(d·n).由于AABB的slab平面都分别和两个坐标轴平行，公式可以进一步简化：设P0=(px,py,pz), d=(dx,dy,dz), t和x-slab面的交点的参数计算公式可化简为t=(D-px)/dx，而此处的D就是AABB的边界面x坐标。

b2RayCastInput是射线输入，P1是射线起点，P2是射线上的另一个点，maxFraction表示需要检测的射线范围的参数值。如果需要检测的是一个线段，则把P1，P2设成线段的两个端点，maxFraction设为1即可。



b2RayCastOutput是碰撞检测的输出，fraction是碰撞点位置的参数值。



算法很简单，就是利用上面的方程分别求出射线和AABB四个面的交点参数t1和t2，然后检查射线和x-slab，y-slab的交线是否有重合部分。

**2.2.3 物体破碎策略**

前面我们分析了很多碰撞的方案，那么碰撞之后我们又要做什么呢？对象爆炸，改变颜色或者只是简单的消失，这些多是通常Flash游戏可以考虑的方式，但是对于抛投类游戏而言，应用最多的还是对象的破碎。

如果我们应用Flash Box2D

**2.2.4 游戏关卡设计**

游戏关卡主要通过设计不同难度的游戏场景，让游戏操作者在某些条件未满足时，不能任意前往下一关[1]。而不同关卡具有不同的游戏道具，这些道具具有不同的作用与生命值，如何有机搭配组合出符合游戏总体进程的关卡，往往是决定游戏能否成功的关键。

游戏关卡设计通常会经过策划，场景搭建，测试关卡几个部分，如果完全手工的进行Flash抛投类游戏的场景搭建，关卡测试，则既费时费力，又容易出错。而除了手工进行关卡设计之外，通常还有以下几种方式

第一种，借助Flash CS设计工具，先将游戏道具整理到资源库中，然后根据需要拖放不同的道具到舞台上，最后使用JSFL对整个场景进行遍历，记录下各种道具的类型，位置信息，最近通过XML等文本形式保存关卡的信息。

关卡测试则需要另外打开游戏，读取场景搭建数据后进行测试，整个流程虽然比完全手工运作来的简便，但还是需要在不同工具间进行切换，在关卡搭建的过程中不能实时预览真实的道具搭建效果。

**2.3 Flash Box2D物理引擎的应用优势**

**2.3.1 FLASH物理引擎现状**

Flash物理引擎是一组计算机应用程序，用来模拟牛顿物理模型，通过对物体赋予质量，速度，作用力，摩擦力，弹性等物理变量，来模拟计算在不同情况下的物体应力效果。它主要应用于模拟物理状态的互动游戏中。

目前的FLASH物理引擎主要有FLASH BOX2D, APE,FLADE,FOAM,FISIX等，用的比较多的主要是FLASH BOX2D和APE。

Box2D最初是一个用于模拟2D刚体的[C++](http://baike.baidu.com/view/824.htm)引擎，开发人员可以应用Box2D引擎，使物体在运动时拥有物理世界的效果，像惯性、自由落体、碰撞等等。因为其出色的性能和丰富的功能，后来被开发者移植到了FLASH平台，便有了FLASH Box2D物理引擎。

Flash Box2D是模拟刚体的物理引擎，物理世界的刚体具有不同的形状，比如圆形，矩形，三角形，凸多边形，复杂多边形等等。这些刚体在物理世界中创建后受到重力，摩擦力等物理作用力的影响。同时在运行过程中，还可以根据游戏设计的要求，施加各种作用力在刚体上，从而实现各种动作需求。

APE(Actionscript Physics Engine)是一个ActionScript3写成的物理引擎，用于模拟现实中物体发生的运动和碰撞。它是免费、开源的，遵循MIT协议。但其成熟度不够好，相对于FLASH Box2D来讲功能更为单一

FLASH BOX2D与APE的功能比较如下表

|  |  |  |
| --- | --- | --- |
|  | Flash Box2D | APE |
| 多场景 | 支持 | 不支持 |
| 模型 | 模型多，功能强大 | 有基本模型 |
| 渲染 | 无渲染模型 | 有渲染模型 |
| 碰撞精确度 | 比较精确 | 有时存在视觉bug |
| 单位 | MKS(Meter-Kilogram-Second() | Pixel |

**2.3.2 Flash Box2D应用优势**

       1、首先，对于需要进行各种物体碰撞检测的，建议不要自己去写一堆方法，而用Box2D去进行模拟，这样便于后续的应用扩充，真的，应用扩充起来的方便程度远远大于你自己写函数进行判断。  
        2、对物理世界中的物体进行添加和删除，一定要在模拟结束后，可能有人问什么时候算模拟结束，这个点也就是在step执行后。如果不是这么删除 body，那么很可能引起程序崩溃，你可以想像，一个物体正在运动，某位大仙突然让它穿越了，物理世界很茫然，当数到这个物体时，应该进行它的动作，但发 现他不在了，于是想疯了，然后崩溃了。碰撞检测中需要物体消失也一样。  
        3、物理世界的step模拟时间为0时，意味着世界禁止，也就是停止了，所以暂停和开始的功能很容易。

它已被用于[蜡笔物理学](http://baike.baidu.com/view/2138718.htm)、[愤怒的小鸟](http://baike.baidu.com/view/4142841.htm)、Rolando、Fantastic Contraption、Incredibots、Tiny Wings、Transformice、Happy Wheels等游戏的开发

他们都是一款物理游戏，而且，他们使用的都是box2d物理引擎。简单地说，“物理游戏”就是严格按照物理规律移动，体现重力、引力、反作用力、加速度等 物理特性的游戏。《愤怒的小鸟》需要控制好小鸟弹出的力度以及方向来击中目标，而《Tiny Wings》则需要掌握好滑坡物理加速度以及风的力量以让小鸟尽可能飞得高飞的远。而box2d 是一个功能丰富的2D物理引擎。使用 Box2D，开发人员可以使对象运动时具有物理效果，如惯性、自由落体、碰撞等等。在开发游戏过程中，利用此物理引擎开发会比较简单，学会了创建物理世界 后，加上相应的物体就可以实现物理模仿。

利用物理引擎制作的游戏，因为手感真实，通关方法多变而越来越受玩家欢迎。而box2d引擎由于上手简单、性能良好、效果逼真等优点，成为越来越多物理游戏制作者的首选。在国内第一个推出客户端游戏开发平台和SDK的网易iTownSDK游戏开发者计划，在上周更新的SDK版本中，也趁热打铁地整合了 box2d引擎接口，开放2D物理引擎。2D物理引擎的开放使得开发者使用iTownSDK所能制作的游戏有了更多的选择。物理游戏因为其来自于人们对现 实的认识，操作简单，玩法容易理解，一直受到玩家的喜爱。而物理游戏的题材、玩法设计还有很多可挖掘之处。

**2.4 Flash Box2D在手机游戏开发中存在的问题**

**2.4.1 Flash Box2D游戏开发的效率**

应用Flash Box2D来进行手机游戏开发，尤其是抛投类手机游戏的开发是个不错的选择，但是开发的效率还是存在不少问题。

总体上来讲，对Flash开发人员来说，主要是前端开发，习惯了对显示对象的控制，然而Flash Box2D主要是对物理世界数据的模拟，并没有直接的显示对象，虽然可以通过调试模式看到模拟对象的线框表示，但对于注重数据模拟的编程习惯来说，还是有一定的学习成本曲线。

对于物理形状的构建，Flash Box2D只支持矩形，圆形，及凸多边形的构建。然而在游戏中往往还会有很多特殊形状物体的创建，比如任意多形边，这些物理刚体的创建如果都用硬编码的形式为做的话，效率非常低。我们举个例子，如果一个游戏场景有20个凸多边形的话，如果一个多边形从顶点的计算，位置的摆放，角度的旋转一系列功能的调整来计算，差不多会要10分钟的工作量，那么一个游戏场景就要花3小时20分钟，这还只是按Flash Box2D支持的凸多边形来计算，如果要涉及到凹多边形的话，工作量会更高，单单一个场景数据的调配有时差不多要到一天的工作量。在Flash Box2D 抛投类手机游戏开发过程中，一个游戏几十个，上百个游戏场景的情况是很普遍的，如果直接用Flash Box2D来开发，效率是比较低的。

另外，为了游戏的效果，通常我们还会有一些特殊效果的考虑，比如浮力效果，流体效果等等，这些都会比较花时间。

**2.4.2 Flash Box2D游戏开发的难度**

Flash Box2D 游戏开发除了效率方面的问题，还存在着不少难点，以下是我们在实际开发中碰到的一些难点的归纳：

**抛投**

开发Flash抛投游戏，我们第一个想到的应该是如何进行抛投。抛投按视角来分可以分为第一视角和第三视角的抛投。

所谓第一视角的抛投，就是游戏操控者从屏幕的底部向顶部抛投，也可以说是向前抛投。如果应用3D游戏引擎或者算法，做成

3D效果的第一视角抛投游戏是最具有代入感的，但通常Flash开发的第一视角抛投游戏还是以2D效果来开发的居多，其算法主要

是利用物体的缩放来达到前后抛投的效果。在抛投的过程中游戏操控者通常看不到游戏的主角。

根据我们对Flash抛投类游戏的统计分析，受市场欢迎的还主要是以第三视角开发的。第三视角的抛投算法主要依据是抛物线，

游戏操控者一般从屏幕的左边向右边抛投物体。在抛投的过程中游戏操控者可以看到游戏的主角。

下面我们主要探讨一个第三视角抛投的难点

如果我们只是抛投圆形的物体，比如炮弹，球，我们只要计算出抛物线的路径基本上就可以解决问题了，不论我们是自己开发

抛物线公式，还是应用向量分量的计算方式，也或者直接使用Flash Box2D物理引擎。但是如果我们要抛投的是矩形的物体，比如

弓箭，那么问题就复杂的多了。

我们如果只计算抛物线，使用Flash Box2D来实现抛投的一般步骤是这样的。

1.获取游戏操控者操作弓箭的发射力度以及角度。

2.把角度值赋给弓箭的angle属性。

3.把力度值赋给弓箭的linearVelocity属性。

4.设置线性推力，以使弓箭在重力作用下按抛物线飞出

效果就会像下面这样

这显然不是我们要的效果。也不是正确的物理抛投现象，真实的情景应该是让发射出去的弓箭，能随着”弹道“发生相应的角度变化。

**破碎**

在Flash抛投游戏中，物体由于碰撞而产生破碎是我们要探讨的另一个难点

位图序列破碎动画是比较常用的破碎策略。因为位图序列动画一般是由动效师在先期就渲染制作好的，程序员只要将渲染好的

位图动画加载到游戏中，在适当的时候控制它的播放就好了，无论从效果还是实现难度都是比较好的，但是因为其是预先渲染好的位图序列，

这限制了它的可变化性，所有的碰撞所产生的破碎，不论力度，角度，破碎的效果都一样，而且破碎物不论大小，通常都不再具备物理性。

如何解决物体碰撞所产生的破碎的多样性与物理性就成了是Flash抛投类游戏的难点

破碎的多样性，我们可以考虑预先渲染多套位图序列动画，然后根据具体的碰撞力度与角度来分类播放不同的位图序列。这在一

定程序上能解决这个问题，但是这样一样则势必会增加内存的使用，对手Flash抛投类手机游戏，内存的合理使用也是一个必须的考量。

破碎的物理性，显然，在碰撞之前，我们的物体是有质量，弹性，形状等一系列物理属性的。在碰撞之后，我们如果应用位图

序列动画来表现物体的破碎的话，为了在视觉上完美，我们必须让位图序列动画代替原来的物体，也就是说我们必须把原来的物

体从世界中删除，而只显示破碎动画。破碎动画只是位图动画，其本实不属于物理世界，所以也就不可能与生俱来物理属性。

如果我们要让每个破碎的独立物体具有物理属性，则必须重新构建所有的破碎物体的刚体，基本的步骤如下：

1.根据物体的破碎程度，筛选出比较大的破碎物，也就是说剔除掉相对较小的颗粒物。

2.扫描破碎物的位图形状，如果是类似凸多边形的，计算出各个破碎物的多边形顶点。

3.通过Flash Box2D的一般流程创建出各个破碎物刚体。

4.如果是类似凹多边形的，切割成多个凸多边形。并计算出各个凸多边形顶点。

5.通过Flash Box2D的一般流程分别创建出各个凸多边形

6.最后通过组合法将各个凸多边形组合成原凹多边形，以形成破碎物的物理属性的赋予

这些步骤每一步多困难重重，对位图图形的分析涉及到了渲染显示的底层算法，其复杂程序可想而知。

**2.4.3 FLASH BOX2D 手机游戏的优化**

在FLASH开发WEB应用和桌面应用时，我们通常不需要太多关注CPU和Memory的使用限制，因为现在的电脑通常性能都是比较优越的，而对于移动设备来说则要注意这些资源的使用，虽然最新的Android/IPhone的处理器性能和内存容量有了很大的提高，但跟电脑应用相比，还是小了很多，因此，我们在创建FLASH手机应用时要始终关注性能，并对发布的版本进行必要的优化

如何使FLASH手机应用所使用的CPU和内存最小，是我们优化的方向，但是这两者之间时常是相互影响的，当我们采取措施减少内存的使用时，往往会使用更多的矢量图，以致CPU资源的使用上升了，反之，如果我们为了减少CPU的使用，而用位图来代替矢量效量的绘制时，就会增加内存的使用。

因此，我们在优化的同时要考虑一个问题，当前更重要的是内存还是CPU资源，并尽量能使他们的使用达到一个平衡。

**第三章FLASH抛投类手机游戏的技术基础与关键**

**3.1 Flash Box2D技术基础**

**3.1.1** **Flash Box2D核心概念**

世界(world)：物理世界就是模拟物理刚体的一个集合，物理刚体由世界创建，在这个集合中的物理刚体受到共同的物理特性的作用，比如重力。理论上来讲，在游戏中可以创建多个世界，以便模拟多个物理作用集合，但通常是不需要的。

刚体(rigid body)：刚体可以理解为一个十分坚硬的物体，其中的任何两点之间的相对距离都是完全不变的，刚体的对立面就是柔体，像水。柔体的任意两点间的相对距离会随着流动而改变。Flash Box2D世界中任何对象都是刚体，也就是说Flash Box2D只支持对刚体的模拟。

形状(shape)：形状是各种二维几何体，例如圆形和多边形。在几何学中，多边形通常可以分为简单多边形和复杂多边形。简单多边形是边不相交的多边。简单多边形按[凸性](http://zh.wikipedia.org/w/index.php?title=%E5%87%B8%E6%80%A7&action=edit&redlink=1)区分，可以分为凸多边形和凹多边形，凸多边形是指它的内角都不大于180°。Flash Box2D只支持凸多边形。

夹具(fixture)：其概念源于[机械工程](http://baike.baidu.com/view/21354.htm)，在[机械制造](http://baike.baidu.com/view/996492.htm)过程中用来固定加工对象，使之占有正确的位置，以接受施工或检测的装置。Flash Box2D中的夹具是用来将形状绑定到物体上，并添加密度(density)、摩擦(friction)和弹力(restitution)等物理属性的对象。

关节（joint）：Flash Box2D允许我们通过关节（joint）来创建刚体之间的约束。我们常用的关节有鼠标关节，距离关切，旋转关节等。通过鼠标关节，在游戏中可以实现对物品的拾取、拖拽以及抛掷；通过距离关节可以设置两个刚体之间的固定距离；使用旋转关节可以使刚体围绕中心点旋转。

以上是Flash Box2D的一些核心概念，下面我们将通过具体的代码，来详细说明Flash Box2D在游戏中是如何时作用的。

**3.1.2在Box2DFlash中定义世界**

基于Flash Box2D的游戏开发，第一步通常就是对物理世界的创建。

与现实世界一样，Flash Box2D的世界（World）也是有重力（gravity）的，所以我们先定义物理世界重力（world gravity）。

var gravity:b2Vec2=new b2Vec2(0,9.8);

b2Vec2是一个2D的向量数据类型，它将储存x和y矢量分量。b2Vec2的构造函数有两个参数，都是数值，代表了x和y分量。通过这种方法定义gravity变量作为一个矢量。 x=0说明水平方法上没有重力。如果要模拟一个恒定的风力作用，可以在这里定义一个数值来模拟。 y=9.8说明这个Flash Box2D模拟了一个近似于地球重力的重力。

在物理世界中创建的所有刚体都会受到这个重力的影响，这意味着如果一个刚体被创建在空中，则它将在重力的作用下自由下落。

对物理世界中的所有刚体，Box2D物理引擎在Flash播放的每一帧多需要去计算其物理的位置，这在刚体较多的情况下是比较耗费内存的。那么怎样来优化这种物理模拟的计算呢？Flash Box2D为我们提供了一个对外接口。当在物理世界中的刚体静止时，Box2D可以允许他们进入睡眠状态，这样它们将不受作用力的影响。

一个睡眠的刚体就不需要进行物理模拟计算，睡眠的刚体只是表示自己的存在，并静止在它的位置上，不会对世界中的任何事物产生影响，Flash Box2D物理引擎将会忽略它，从而极大的提升处理速度。

为了让Flash Box2D知道是否可以让刚体睡眠，我们可以定义一个变量来指示。

var canSleep:Boolean=true;

最后，我们要定义的世界通过下面这句话来创建：

var world:b2World = new b2World(gravity, canSleep);

这样我们的物理世界就创建好了。为了便于在游戏中的任何模块都能访问到这个世界，我们通常会将其保存在一个全局的变量中，这点我们在后面的章节中将会细述。

**3.2在世界中加入刚体**

Flash Box2D的世界已经创建了，那世界中的刚体怎么创建呢？接下来我们将深入的阐述。

我们通常使用以下步骤将刚体添加到世界中：

* 创建一个刚体定义，它将持有刚体的信息，例如刚体的位置信息等。
* 创建一个形状，它将决定刚体的显示形状。
* 创建一个夹具，将形状附加到刚体定义上。
* 创建刚体在世界中的实体，使用世界来具体创建刚体。

**3.2.1创建刚体的定义**

无论我们是创建圆形还是多边形，第一步都是创建一个刚体的定义：

var bodyDef:b2BodyDef=new b2BodyDef();

b2BodyDef类是一个刚体的定义类，它将持有创建我们刚体所需要的所有数据。比如刚体的位置、角度、是否活动状态等。

下面是具体定义刚体位置的语句

bodyDef.position.Set(x,y);

通过position属性显示的设置了刚体在世界中的位置。有一个需要注意的地方是，这里的单位是米。虽然Flash是以像素（pixels）为度量单位，但是在Box2D中尝试模拟真实的世界并采用米（meters）作为度量单位。对于米（meters）和像素（pixels）之间的转换没有通用的标准，但是我们可以采用下面的转换标准，这是根据经验得出的，通常可以有很好的运行效果：

1meter= 30Pixels

在实际的编程过程中，我们可以定义一个变量来帮助我们将米（meters）转换成像素（pixels），比如var pixelToMeter:int = 30; ，这样我们便可以在Box2D世界（world）中进行操作时使用像素（pixels）而不用使用米（meters）来作为度量单位，只需要在数值后面除上这个比例值。这将帮助我们在制作Flash游戏时，使用像素来思考，从而变得更加直观。

**3.2.2创建刚体的形状**

形状（shape）是一个2D几何对象，例如一个圆形或者多边形，在这里我们先只讨论凸多边形（每一个内角小于180度）。因为Flash Box2D原生的API只能处理凸多边形。

接下来我们以创建一个圆形为例，来详细说明刚体形状的创建：

var circleShape:b2CircleShape;

circleShape=new b2CircleShape(25/ pixelToMeter);

b2CircleShape是用来创建圆形形状的API，它的构造函数只需要一个半径（radius）作为参数。上面的代码中，我们创建了一个圆形，它的半径为25像素（pixels）。这里我们可以使用像素来思考是因为我们在像素数值后面除以了用刚刚创建的米/像素比例值。

我们也可以设置一个全局常量worldScale来保存这个比例，这样便可以在其他任何需要的地方方便的访问这个比例值了。以后每次想要使用像素进行操作时，只要将它们除以worldScale即可。当然也可以定义一个方法名为pixelsToMeters的方法，在每次你需要将像素（pixels）转换成米（meters）时调用。

当我们有了刚体定义和形状时，我们就可以使用夹具（fixture）来将它们粘合起来。接下来我们具体来看看刚体的夹具。

**3.2.3 创建刚体的夹具**

夹具（fixture）用于将形状绑定到刚体上，同时也可以定义刚体的材质，比如设置密度（density），摩擦系数（friction）以及恢复系数（restitution）等等一系列物理属性。

首先，我们来创建夹具（fixture）：

var fixtureDef:b2FixtureDef = new b2FixtureDef();

b2FixtureDef是创建夹具的定义类，很简单，可以不传任何参数。定了了夹具，我们就可以把刚体的形状赋值给夹具了：

fixtureDef.shape=circleShape;

最后，我们通过世界来创建具体的刚体：

var theBall:b2Body=world.CreateBody(bodyDef);

这样我们创建的刚体球就算加入到Flash Box2D的世界中了。然后我们通过刚体实例来创建刚才定义好的夹具：

theBall.CreateFixture(fixtureDef);

这样可以说我们就创建完了有血有肉的刚体了。当然还有很多其他的具体属性，我们在这里不一一列举了，详细的应用可以参考Flash Box2D的API文档。

**3.3 常用物理刚体的创建**

通常我们创建物理游戏都是要创建一个可见对象，然后为这个可见对象创建一个与之相匹配的物理对象。而创建物理对象又需要经历一套复杂的流程：

* 创建Body definition,
* 创建形状
* 创建Fixture definition
* 通过世界创建Body
* 通过Body创建 Fixture

通过普通的流程来创建物理游戏会遇到以下几类问题，下面我们一一细述

**3.3.1 简单多边形的创建**

简单多边形是指矩形，圆形，凸多边形等Box2d原生支持的形状。这类多边形我们通常可以通过create body by type 和 create body by vectories来创建。

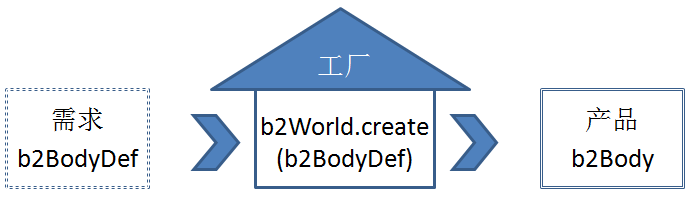
在FLASH开发物理游戏的时，我们要明确二个基本的概念，刚体与显示对象DisplayObject。你可以理解刚体就是BOX2D里的b2Body，而显示对象则可以先简单的理解为Sprite，他们是有着本质的区别的。

刚体是Box2D中的一个物理模拟对象，不是可显示对象，但是可以通过b2DebugDraw来进行调试模拟，这时刚体会显示成线框。刚体在运动过程中是以米为计量单位的，它通过b2World.createBody()来创建。

显示对象Sprite很从字面上理解它就是显示对象，通过addChild添加到舞台就可以被显示，它在运动过程中是以像素为计量单位的，它可以通过new来创建，然后addChild添加后直接显示在舞台上。

Box2D主要是对刚体的物理性模拟，我们为了让刚体的运行可视化，会根据刚体的状态改变来逐帧更新显示对象的状态，这样就产生了显示对象具有物理属性的感觉。

刚体创建的过程，与AS中new一个对象，addChild到舞台中显示比起来要复杂很多，刚体的创建可以看做是一个工厂模式，根据你的需求(b2BodyDef)生产指定的刚体



刚体的创建过程可以分为以下几个步骤：

1. 创建刚体需求b2BodyDef。

var bodyRequest:b2BodyDef = new b2BodyDef();

|  |
| --- |
|  |
|  |

在这个需求中，我们可以包括下面的内容：

a) position：刚体的坐标位置，单位是米m。

b) angle：刚体的角度，单位是弧度。

c) userdata：用来存储应用中所特需的信息。

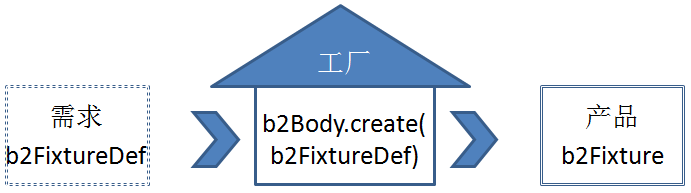
d)......

2. b2World工厂用createBody方法创建刚体产品，并自动将其添加到Box2D世界中，然后返回该刚体。

body=world.CreateBody(bodyRequest);

3. 定义刚体物理特性。

虽然我们已经成功添加了一个刚体。但是这个刚体密度有多大，具体是什么形状还没有定义，刚体的物理属性不是与生俱来的，而刚体的形状则决定了刚体碰撞的边界，因些我们需要用一个b2FixtureDef的对象来指定刚体的物理属性。b2Fixture的创建过程同样也是一个工厂模式。



这个创建过程与b2Body的创建过程是一样的。我们的形状需求b2FixtureDef可以定义以下内容：

a) density：质量。

b) friction：表面摩擦力。

c) restitution：表面张力，这个值越大，刚体越硬。

d) shape：设置刚体为形状。

e ) ......

4. 定义刚体的形态。

前面我们已经用到了shape来指定刚体的形状，但是这个shap也是需要我们来定义和创建 的。我们用的一般是b2shape的子类，主要有b2PolygonShape、b2CircleShape，下面我们以b2PolygonShape为例说明

var boxShape:b2PolygonShape = new b2PolygonShape();

boxShape.SetAsBox(width/2, height/2);

我们常用的除了SetAsBox，还有SetAsVector()，定义好之后传递给b2FixtureDef就可以了。

从上面的创建过程来看还是比较麻烦的，不见得每次创建一个刚体都要手动去设置各值，我们可以考虑比较方便的方案。

**3.3.2简单多边形的快速创建**

开发物理游戏需要创建的物理对象通常比较多，如果通过普通流程逐一进行物体创建，比较的费时费力，而且维护更新也比较困难。

如果我们可以先实现静态物体的创建，然后在需要物理属性的物体上注入Box2D的物理特性，就可以达到点石成金的快速效果。

Box2D在模拟现在世界的计算过程其实是跟显示层是没有关系的，它只是运行在显示层背后的一种完全数字逻辑的运算，物体的显示层就好比一个人的外形躯体，而Box2D则是外形躯体的内在精神，是支配外形去实现物理运动内在力量，因此，我们完全可以考虑将物理特性注入表现层的UI，来实现整个物理对象的创建。

除了简单多边形的创建，我们通常还会碰到很多不规则物体的创建。

**3.3.3凹多边形的创建**

物理游戏不仅是要创建矩形，圆形，凸多边形等Box2d原生支持的形状，还会有很多不规则物体，比如凹多边形。但是Flash Box2D是不支持原生的凹多边形创建的，这主要是受限于Box 2D的自身结构设计。

那我们怎么创建凹多边形呢？这在现在项目中是不可避免的问题。受到其他一些第三方类库的启发，我们的解决思路基本是这样的：先将复杂的凹多边形切割成一系列的凸多边形，然后各个创建，最后将这一系列凸多边形组合成原来的凹多边形。

对于凹多边形的切割，我们引用了Antoan Angelov 发布的一个工具类,叫 b2Separator. 当我们在要创建一个复杂多边形的时候，我们需要手工创建很多，这不仅增加了开发的复杂度，而且还很费时间去结合这一系列凸多边形，而B2Separator 可以帮我们将一个凹多边形分割成一系列小的凸多边形，并将它们作为B2Fixture加入到B2Body上，而你所要做的就是把凹多边形的顶点传入B2Separator

**3.3.4凹多边形的顶点**

我们通过[最小二乘算法](http://en.wikipedia.org/wiki/Marching_squares)跟踪的图像的轮廓，从而获得我们所需要的凹多边形的顶点。具体的思路如下：

1）找到一个放置在图像边缘的像素（这无关做与行军广场，假设你已经发现了这样的像素）。这将是当前象素分析。

2）考虑2×2像素的正方形，其中包括当前像素一般位于在左上角或右下角的平方。

3）此时你有4个像素，和他们每个人可以是透明或不透明。因此，我们将有16个可能的2×2的正方形全透明像素或所有不透明的像素的情况下，虽然不会发生，因为我们正在走动的图像的边缘。

4）根据在2×2平方不透明像素的数量和位置，我们可以猜测轮廓的方向，当前像素在这样的方向移动，并继续从第2步，直到你再次达到在步骤1中发现的像素。

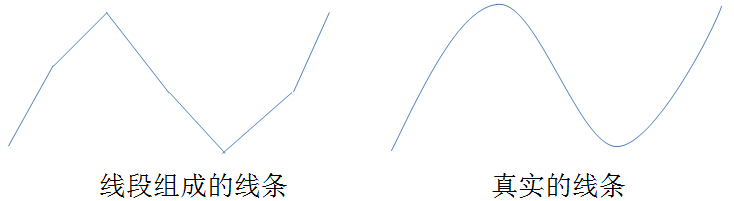
现在，我们怎样才能找到在步骤1中所需要的起始像素？这仅仅是一个蛮力的问题，我们需要扫描图像像素的像素，直到我们找到一个不透明像素。

**3.3.5凹多边形的顶点的优化**

**3.3.6 创建线条刚体**

BOX2D的作为刚体的物理引擎，通常都是用来模拟块状物体，但是在实际的游戏开发中又经常会碰到线条类的物体，比如绳子，树枝等等。对于这些物体，我们怎么用BOX2D来创建 呢？思路其实就是用多个线段组合起来模拟具有一定柔性的线条。

从理论上讲，线条是由无数个点组成的，我们可以创建圆来代替点，进而形成线，但是从性能效率方面来考虑，实际开发中其实只要用多个线段组合成线就可以了。



如上图，我们假设每个线断长度为segmentLength，线段的长度越短，线段的数量越多，那么线条模拟的就越逼真。

创建线条的算法如下：

1）先定义线段的长度segmentLength

2）在鼠标点下后，开始绘制线条，并记录之前的鼠标坐标位置为prePoint

3）在线条绘制过程中，持续检测鼠标坐标curPoint与前一个点prePoint之间的距离为distance，当distance大于线段长度时，创建一个新的线段，将线段添加到segmentList中，并将curPoint赋值给prePoint，这样循环往复，直到线条绘制完成。

4）鼠标弹起后，遍历所有的线段，并利用多边形组合法，创建对于的线段刚体，然后组合成一个完整的线条。

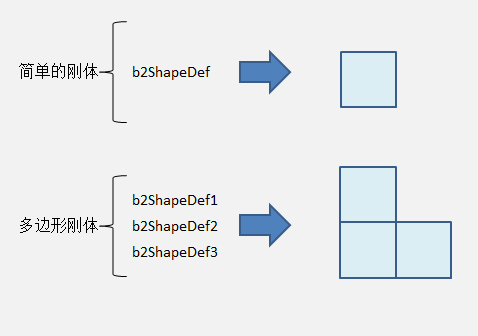
具体代码参见附件

**多边形刚体绘制**

Box2D创建多边形刚体通常有两种方法，一种是组合法，另一种是顶点构造法。下面我来详细看看这两种方法的具体实现过程。

一、组合法

顾名思义，组合法就是用多个基本刚体（矩形刚体和圆形刚体）组合成的一个目标刚体，其实确切的讲，是用多个图形shape，组成一个整体的图形；而刚体只有一个。下图可以更清楚的说明这一点。



从上图看，多边形刚体的创建过程就简单多了，首先创建多个b2ShapeDef需求，然后根据这些需求创建一个复杂的刚体。

具体代码参见附件

二、顶点构造法

与组合法不同，顶点构造法是通过创建一个b2ShapeDef需求，然后在这个b2ShapeDef中详细描述每个顶点的座标。需求的原则是先指定多边形的顶点个数，然后把每个顶点的坐标描述清楚。在Box2D中这些顶点都保存在b2ShapeDef.vertices数组中。在设置的时候也要遍历shapeRequest.vertices数组里的每个顶点，然后调用Set方法设置顶点的坐标，具体代码参见附件

三、两种方法的区别

如前面所述，组合法创建的刚体中有多个图形，所以在Box2D模拟碰撞检测时，就会对这些图形逐个进行检测计算；而顶点构造法的刚体只有一个图形，所以也只需要一次碰撞检测计算，计算效率相对要高一些。另外通过指定顶点的坐标，可以创建出任意的形状，而不是简单的矩形和圆形刚体的组合。

b2Separator类可以根据b2Shape顶点，把它分成一个个凸多边形，然后用组合法组成复杂的多边形刚体。b2Separator主要有下面两个函数：

Separator方法用来将verticeVec中的顶点分成多个小的凸多边形，并赋值给fixtrueDef，创建body多边形刚体。

每个参数说明如下：

body：多边形刚体对象

fixtureDef：fixture需求对象

verticeVec：存储多边形所顶点的Vector数组

scale：Box2D模拟时的缩放比例，这个值与b2DebugDraw中的SetDrawScale的参数值是一样的。

Validate的功能是检测多边形的刚体是否符合创建的标准，在不符合标准时，返回不符合标准的原因。

参数说明如下：

verticeVec：存储多边形所顶点的Vector数组

返回值类型有下面几种：

0：顶点符合绘制多边形的标准

1：顶点连接的线段之间有交叉

2：顶点的顺序非顺时针绘制

3：出现1和2两种错误

只有返回值为0时，才可以用Separator方法创建多边形。

了解了b2Separator之后，我们用组合法完成了多边形的绘制

值得一提的是，因为非顺时针顺序的顶点是无法创建多边形的，所以在validate检测顶点返回2之后，调用verticesList.reverse()方法，反转顶点的顺序。这样用户不管顺时针还是逆时针都可以绘制出刚体来。

**3.4 Box2D特殊效果的实现**

通过上面对Flash Box2D的基础技术分析，我们可以看到，在每次开发一款游戏的时候，都会需要对物理世界进行初始化，而这一过程在很大程序上是相似的，为了避免不必要的重复劳动，我们可以将物理世界的初始化抽离出来，设计在我们游戏框架中，这样就便于每次开发时应用了。

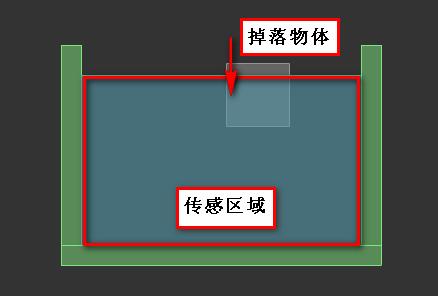
另外我们也可以根据不同的物理环境，比如，水体，风洞等特殊场景，设置不同的物理参数，然后通过继承，合成的编程模式加以扩展。

下面我们具体分析一下特殊场景的液体浮力效果。

**3.4.1 浮力效果**

通常我们不会是整个场景都是在液体里，而是一部分在自然空间，一部分需要模拟液体浮力。虽然Flash Box2D提供了浮力参数，但它是对整个世界起作用的，这显然不能满足我们的需求，因此如果我们能够定义一个区域来实现浮力效果，那么我们就可以实现类似池塘的场景了。

如何实现呢？通过假设验证，我们发现可以通过创建一个跟设想的池塘一样大小的传感器，然后将将浮力控制附加在这个传感物体上，只要其他物体掉落到传感区域上，就模拟浮力效果，设计原理图如下：



具体实现代码如下

  buoyancyController.normal.Set(0,-1);

            buoyancyController.offset=-180/worldScale;

            buoyancyController.useDensity=true;

            buoyancyController.density=2.0;

            buoyancyController.linearDrag=5;

            buoyancyController.angularDrag=2;

            world.AddController(buoyancyController);

 private function update(e:Event):void {

            world.Step(1/30,10,10);

            for (var currentBody:b2Body=world.GetBodyList(); currentBody; currentBody=currentBody.GetNext()) {

                if (currentBody.GetType()==b2Body.b2\_dynamicBody) {

                    var currentBodyControllers:b2ControllerEdge=currentBody.GetControllerList();

                    if (currentBodyControllers!=null) {

                        buoyancyController.RemoveBody(currentBody);

                    }

                    for (var c:b2ContactEdge=currentBody.GetContactList(); c; c=c.next) {

                        var contact:b2Contact=c.contact;

                        var fixtureA:b2Fixture=contact.GetFixtureA();

                        var fixtureB:b2Fixture=contact.GetFixtureB();

                        if (fixtureA.IsSensor()) {

                            var bodyB:b2Body=fixtureB.GetBody();

                            var bodyBControllers:b2ControllerEdge=bodyB.GetControllerList();

                            if (bodyBControllers==null) {

                                buoyancyController.AddBody(bodyB);

                            }

                        }

                        if (fixtureB.IsSensor()) {

                            var bodyA:b2Body=fixtureA.GetBody();

                            var bodyAControllers:b2ControllerEdge=bodyA.GetControllerList();

                            if (bodyAControllers==null) {

                                buoyancyController.AddBody(bodyA);

                            }

                        }

                    }

                }

            }

            world.ClearForces();

            world.DrawDebugData();

        }

**3.4.2 流体效果**

流体是气体和液体的总称。在人们的生活和生产活动中随时随地都可遇到流体，所以流体力学是与人类日常生活和生产事业密切相关的。大气和水是最常见的两种流体，大气包围着整个地球，地球表面的70%是水面。大气运动、海水运动乃至地球深处熔浆的流动都是流体力学的研究内容。

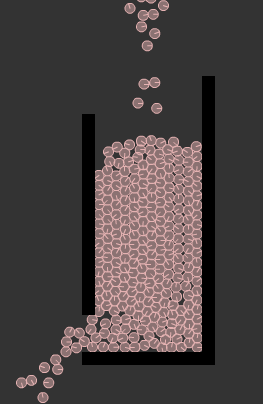
为模拟现实中的水、火、雾、气等效果由各种三维软件开发的制作模块，原理是将无数的单个粒子组合使其呈现出固定形态，借由控制器，脚本来控制其整体或单个的运动，模拟出现真实的效果。

粒子系统几乎是每个引擎中比不可少的部分，它主要用来解决由大量按一定规则运动（变化）的微小物质组成的大物质，在计算机上的生成与显示的问题。经常使用粒子系统模拟的现象有火、爆炸、烟、水流、火花、落叶、云、雾、雪、尘、流星尾迹或者象发光轨迹这样的抽象视觉效果等等

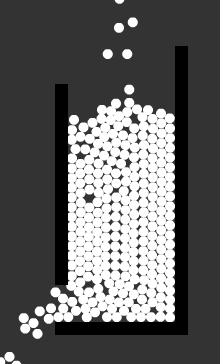
其实粒子系统是一种特效发生器，它可以制造大量的小粒子来达到某种特殊的效果，比如烟雾、火焰或者爆炸。好的粒子效果甚至比渲染好的动画都出色，正因为如此，当前几乎所有的游戏中都使用了粒子系统。为了制造真实的粒子效果，粒子发生器必须在不影响帧率的情况下控制成千上万的粒子运动。简单的粒子系统只允许设置粒子的一些属性，比如生存时间、重力或者颜色；复杂的粒子系统允许为每个粒子的运动函数编写代码。一些粒子系统也包含了光线生成器来创建光线或者轨迹。

实现粒子系统，我们首先需要对粒子系统进行一个简单的划分，它可以简单的氛围两种：点粒子和四边形粒子。点粒子就是说粒子系统中的所有粒子都是由一个点构成的，因此这个点通常就需要不同的颜色来展示，四边形粒子实际上就是将一个粒子纹理给映射到一个四边形上，作为一个粒子对象，当然我们可以看出点粒子的实现肯定比四边形粒子的实现简单很多，下面我们分析分析这两种类型的粒子的实现。

虽然Flash box2d只能模拟刚体，但是我们可以用一系列小的刚体的流动来实现流体的效果。



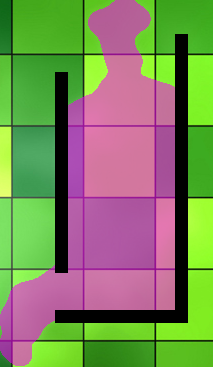
为了让它具有真实的质感，我们可以更改贴图



为了让流体更真实，可以加入一些模糊



现在有点像雾的感觉，如果再加入一个阀值来改变颜色，并且更新一个更好的背景

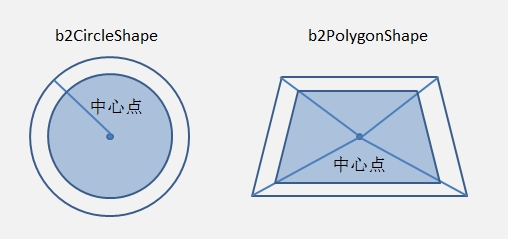


**3.4.3刚体的缩放效果**

因为Flash Box2D模拟的是物理场景及运动的数据，它们不是显示对象，因此也就无法使用width、height或scaleX、scaleY来缩放刚体的大小了。而且Flash Box2D物理引擎也没有提供相应的API调用，因此给相应需求的开发带来了不小的麻烦。

我们研究Flash Box2D API发现，b2Body类并没有提供直接调整刚体尺寸的属性或方法，而是要通过b2Shape类的一些方法来实现，所以首先要用b2Body.GetFixtureList().GetShape()获取刚体的图形，然后根据图形的类型进行不同的设置。

我们知道b2Shape的子类有两种，一个是b2CircleShape类，这个图形的尺寸调整起来比较简单，直接调用b2CircleShape.setRadius()就可以重新设置新的半径。参数可以设置为b2CircleShape.getRadius()\*scale来进行比例缩放。另一个是b2PolygonShape类，因为多边形的顶点数量和位置不同，形成的图形也不同，所以没有类似于SetRadius()的SetWidth()或SetHeight()方法。实际上，我们可以把多边形分解成多个顶点，多边形等比例缩放时，顶点到中心点的距离也是等比例缩放的(如下图所示)，所以我们只需要用顶点向量的Multiply()方法乘以指定的缩放比例即可。

[](http://www.ladeng6666.com/blog/wp-content/uploads/2013/01/b2shapeSample.jpg)

**3.5 小结**

通过对Flash Box2D的核心概念的了解，以及物理世界的初始化，物理刚体的创建，可以说我们大体上能够创建基本的基于Box2D引擎的游戏了。但是我们也不难发现，基本上每次开发一个物理游戏，都要重复物理世界的初始化。另外对于游戏中的每一个物理刚体，如果我们都按照上面的步骤去创建的话，也是相当繁琐的过程。因此，下面我们就需要去具体分析一下，看看我们的游戏框架怎样设计，才能够使我们开发物理游戏既方便实用，又性能卓越。

破碎策略

1.b2world.RayCast(callback:Function, point1:b2Vec2, point2:b2Vec2)切割

利用光线投射物理世界，找出所有在光线路径中的夹具，你可以在回调函数中控制是否获得了最接近光线发射源的刚体。光线投射会忽略包含出发点的形状。

图示

回调函数会有四个参数function Callback(fixture:b2Fixture, point:b2Vec2, normal:b2Vec2, fraction:Number):Number

fixture：被投射线射中的刚体的夹具

point：初始交点，也就是射线进入刚体的入点

normal：初始交点处的法向量

fraction：从射线的起点到初始交点的长度占整个射线的比例

两点决定一个直线，在数学上知道了两点，再定义直线上的其它点，常使用参数方程。

也就是定义 P(fraction) = p1 + fraction \* (p2 - p1)。当fraction为0时，就代表p1, 当fraction=1时，就代表p2。

这样的定义下，两点之间的线段就是参数从0到1之间变化。参数小于0，表示反向的点，大于1就表示正向超出线段的点。

maxFraction就表示要测试的点对应的参数是在[0, maxFraction]内。

数学上很喜欢将一些变量归结成0到1之间变化，这叫做规范化。

处理问题的常用手段是用某个变换(这里说的变换是广义的)将变量归结成0到1之间，再在规范化之下计算, 之后再用个反变换得到原问题的答案。

上面说的直线参数化，可以看成规范化的一种。那为什么要规范化呢？因为这样计算起来方便。

因为每次射线切割对象时都会调用laserFired方法，而且切割到的对象不止一个，所以如果你不希望射线继续切割，返回0，那么射线会停止继续检测。返回1，射线会继续切割其他的对象。

point1:射线的起点

point2:射线的终点

2.切割完之后会形成二个点，一个是射线的入点，一个是出点

我们将位于射线上方的多边形顶点和射线的入点，出点共同组成一个新刚体的顶点，位于射线下方的多边形项点和射线的入点，出点组成另一个新刚体的顶点

3.因为Flash Box2D在创建多边形时只支持顺时针排序的项点序列，所以我们要对新的刚体的顶点做顺时针排序，具体算法如下

首先，根据每个项点的x坐标，按升序排列的所有给定的点。

其次，找出最左边和最右边的点（让我们称之为L和R），并创建tempVec，将LR点按顺时针顺序保存到Vec中。

然后，遍历其它顶点，并使用determinant函数，把LR线以上的项点按顺序序列插入LR点之间，并将位于LR线以下的点按逆序排列。

4.determinant函数作用就是用来判断目标项点(我们称之为O点)是在LR线的上向，下方还是线上,

这计算之前我们确定了3个点L,R,O，根据计算3×3矩阵的行列式，我们可以判定，如果返回一个正数，那么三个点是顺时针顺序，如果是负数，那么他们是逆时针顺序，如果为零，那么他们在同一行。

5.根据多边形面积筛选多边形。先将多边形按顶点顺序切割成多个三角形，然后根据3×3矩阵的行列式与三角形面积的关系式，依次计算出三角形的面积并累加，计算出多边形

的面积，根据多边形面积的大小，剔除掉面积较小的多边形，以提高游戏运行效率。

6.最后根据筛选出来的多边形的顶点创建出新的多边形刚体。

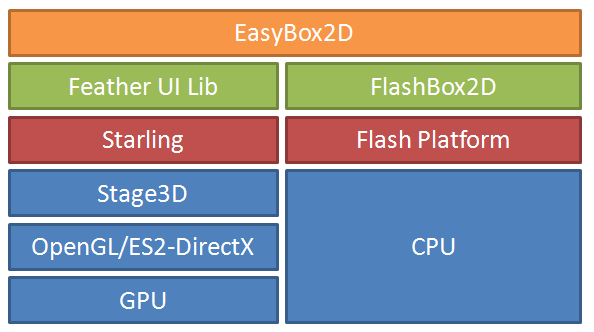
**第四章EasyBox2D游戏框架的设计**

我们的游戏框架设计目标是设计一个基于Starling框架，支持Feather扩展UI组件类库，对FLASH BOX2D进行进一步封装的轻量级的，易于扩展的，简单易用的FLASH手机游戏框架。我们将这个游戏框架命名为EasyBox2D，以下描述中EasyBox2D就是我们框架的名字。EasyBox2D能够显著简化物理刚体的初始化，并且提供了极为简单，具有多种选择性的刚体皮肤绑定方案。

**4.1 EasyBox2D游戏框架总体结构**

在FLASH推出硬件加速渲染的Starling框架之前，FLASH都是应用CPU来运算的，当时的FLASH平台主要有FLASH PLAYER 和FLASH AIR，而BOX2D作为时下最流行的物理引擎，也推出了FLASH AS3.0版本，随着硬件技术水平的发展，越来越多的应用，游戏都提出了硬件加速位图显示的解决方案，ADOBE FLASH也与时俱进的推出了基于Stage3D的渲染引擎Starling，并且集成到了随后更新发布的FLASH PLAYER 和 FLASH AIR这二个运行平台中。而Feather UI Lib 作为Starling的扩展UI库，通过Starling框架，利用GPU的强大能力渲染组件，实现了更加平滑和友好的用户界面体验。

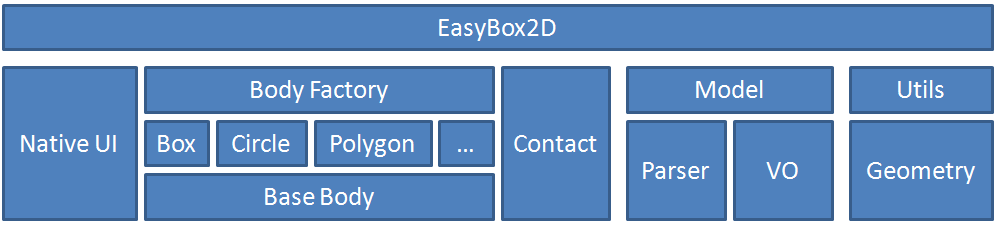
EasyBox2D以组合的形式将Starling,Feather包含其中，并在FlashBox2D的基础上进一步封装了BOX2D的功能，以实现对物理刚体的初始化简化，提高游戏开发的效率，降低了开发的难度。下图就是EasyBox2D游戏框架总体结构

****

**4.2 EasyBox2D结构设计**

**4.2.1 EasyBox2D总体结构**

EasyBox2D总体结构主要分二大层，界面层主要提供相应的调用接口，以方便游戏业务逻辑获得物理引擎的支持，在界面层下面的则是具体提供各种功能的功能层。功能层又分为5个大的模块，由FLASH原生的显示模块，刚体创建 工厂模块，Contact模块，数据模块以及提供数学几何运算的工具类模块。基本的结构设计可以从下面的图中得到初步的了解



**4.2.2 EasyBox2D管理类**

为了便于游戏开发中应用FLASH BOX2D来进行物理物件的创建 与使用，我们设计的EasyBox2D将提供唯一的入口来完成基本上所有的物理特性的创建与管理。对BOX2D的初始化，Debug模式的切换，边界的设定等通常的功能都会在统一的入口文件中提供，并且用户如果觉的预定义的方法不能完全满足游戏开发需求的话，也可以通过重写来实现特定的功能。

结构图中最上面的模块就是我们提供的唯一入口，它相当于一个管理器，负责BOX2D的初始化，刚体的创建，管理等等。

**4.2.3 FLASH 原生显示列表**

FLASH 原生显示的支持主要是为了在FLASH BOX2D调试模式中提供各种调试数据的绘制与显示，这是由FLASH BOX2D物理引擎所定义的调试模式所决定的。

当然除此之外，因为我们是基于Starling开发的手机游戏框架，而Starling所有的显示内容都不属于FLASH原生显示的范畴，而是位于在FLASH原生显示列的下面一层，如果有些功能Starling不支持，或者通过GPU运算效率比较低的显示内容，我们也可以考虑使用FLASH原生显示列表。这一模块就是为了方便对FLASH原生显示列表的调用，使得Starling GPU显示与原生FLASH CPU显示能够融合在一起。

**4.2.4 Body工厂**

通过FLASH BOX2D技术基础部分的描述，相信大家对刚体的生成有了较新入的了解，那么在游戏开发中如果每次都重复构建b2BodyDef，b2FixtureDef，b2Shap，然后再由b2World创建的话就效率比较低了，所以我们将基本的刚体创建封装在框架中，以提高游戏开发的速度。

刚体工厂会提供包括方形，圆形，凸多边形等基本形状的创建，如果这些不能满足开发需求的，刚可以通过继承BaseBody来扩展。

所有刚体的创建都会按照统一的格式来，用户只要提供刚体的类型和相应参数就可以完成创建，比如easyB2D.addBox(property)。

**4.2.5 Contact**

Contact模块主要是提供对碰撞的个性化处理。

因为碰撞的处理具有很多的不确定性，对hp的伤害规则通常也比较复杂，所以我们的框架只是提供了碰撞的基本信息的提供，并以事件的形式向外抛出，在界面层去做个性化的处理。

**4.2.6 数据模块**

数据模块主要用来处理与物理引擎相关的数据，其中B2UserData是对各种类形物体的物理特性的数据描述，当然为了方便相关数据的存取，也加入了表示物体生命值的变量hp， 以及可见对象的Reference引用。

**4.2.7 工具类模块**

工具类模块包主要提供数学几何相关的计算支持。

因为在Box2d只支持矩形，圆形和凸多边形的创建，而且对于凸多边形的顶点数据也有顺序的要求，必须是顺时针的顶点数组，这给物理形状的创建带来了很大的不便。

**4.4游戏编辑器的结构设计**

游戏编辑器主要为用户提供一个能通过图形化的界面，对新建的或者已有的关卡进行编辑，设计并保存关卡数据的平台。游戏编辑器不仅能让玩家能玩自己设计的关卡，更可以分享给其他玩家，当然游戏开发者也能通过编辑器快速搭建游戏场景，以便在开发中不断测试各种环境下的游戏性能。具体来说，EasyBox2D游戏编辑器多包括以下几个方面：

1）场景数据格式的设计与存储。

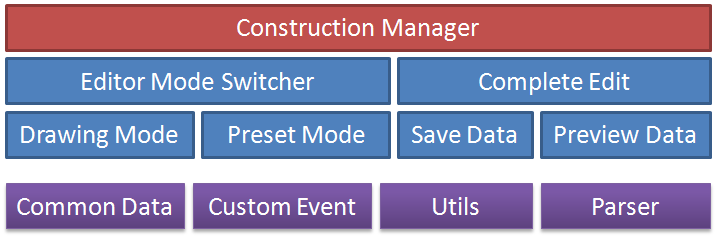
2）导入设计好的场景数据，在此基础上对场景数据进行编辑。

3）根据预先设计好的元素来新建场景数据，用户可以通过拖拽的方式进行搭建。

4）根据编辑器提供的画图功能来进行场景的绘制。

5）编辑完成后对场景数据进行预览与保存。

下图可以看作是EasyBox2D编辑器的总体结构



Construction Manager作为编辑器的操控中心，负责各模块间的互动合作。下面具体说明各功能模块的功能设计。

**4.4.1场景数据格式的设计**

为了便于场景数据在FLASH EasyBox2D环境下的使用，我们考虑使用JSON(JavaScript Object Notation)数据格式。

JSON 是一种轻量级的数据交换格式。 易于人阅读和编写。同时也易于机器解析和生成。 它基于[JavaScript Programming Language](http://www.crockford.com/javascript), [Standard ECMA-262 3rd Edition - December 1999](http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf)的一个子集。 JSON采用完全独立于语言的文本格式，可建构为两种结构：

1）“名称/值”对的集合（A collection of name/value pairs）。不同的语言中，它被理解为对象（object），纪录（record），结构（struct），字典（dictionary），哈希表（hash table），有键列表（keyed list），或者关联数组（associative array）。

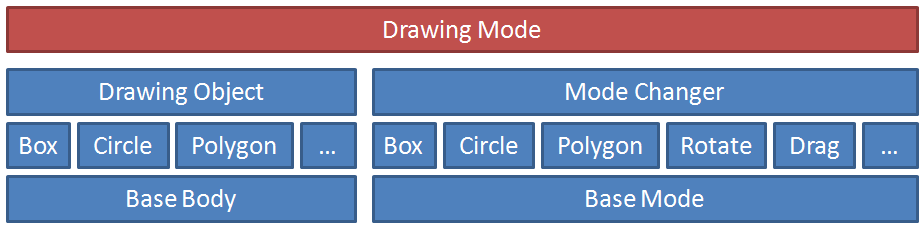
2）值的有序列表（An ordered list of values）。在大部分语言中，它被理解为数组（array）。

这些都是常见的数据结构。事实上大部分现代计算机语言都以某种形式支持它们。这使得一种数据格式在同样基于这些结构的编程语言之间交换成为可能。EasyBox2D采用“名称/值”对的集合结构来组织数据

**4.4.2场景数据的绘制模式**

如果游戏场景只需要Box2D原生形状来进行搭建，只是对各形状位置，旋转等有要求的，我们可以绘制模式来搭建场景。

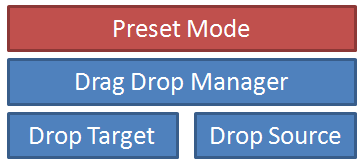
绘制模式是通过选取适合的形状，通过在屏幕上拖拽来实现形状，大小，位置控制的场景搭建模式，主要提供方形，圆形，凸多边形等Box2D支持的原生形状。在绘制过程中会有绘制模式，拖拽移动模式，旋转模式，强制正方形模式，强制贴合整数位置模式等的切换功能，总体结构如下图：



不论是绘制形状，还是绘制模式，都是允许在具体游戏开发中根据实际需求进行扩展的。

**4.4.3场景数据的搭建模式**

场景数据的搭建模式主要是提供预先设计好的构建部件，通后选取相应部件，然后以拖拽的方式在场景是进行搭建，总体结构如下图：



**4.4.4场景数据的预览**

场景数据的预览主要是为了便于在搭建过程中对最终生成游戏场景的预览，预览过程是调用EasyBox2D进场景数据进行解析，生成的过程。

**4.4.5场景数据的保存**

场景数据的保存可以选择数据库存储，也可以选择本地文件存储，具体可以根据游戏开发实际需求选择。

**4.4.6游戏编辑器设计小结**

EasyBox2D游戏编辑器实现基本的游戏场景编辑需求，可以在绘制模式和搭建模式间进行切换，并且支持预览与保存。

对于绘制模式和搭建模式的混合编辑，则是后续扩展的方向。

**4.5游戏编辑器的逻辑实现**

**4.5.1Model包**

**4.5.2Factory包**

**4.5.3Parser包**

**4.6 EasyBox2D游戏框架框架的总结**

* GameWorld实现了对物理世界的封装，减少了每次开发游戏的重复对物理世界的初始化操作。
* 用Factory实现所有Body的创建，只需要准备好相应物理属性，一步实现物理的创建。
* 在处理质心的问题时，默认显示对象的注册点在左上角，质心在整体矩形的1/2宽高处。
* 对凸凹多边形进行分类处理，找到了相应的解决方案。
* GameWorld 还提供了通过显示对象快速查找物理对象的公开方法，方便了对物理对象的控制。

**第五章 EasyBox2D游戏框架的实现**

**5.1 EasyBox2D游戏框架的逻辑实现**

**5.1.1世界封装类**

世界封装类是进入物理世界的入口， 主要是为了简化游戏开展中每次都要手动去初始化物理世界的过程，我们设计了GameWorld类来具体实现。

GameWorld类是继承b2World的，并且是单例类，这样能保证了游戏中始终只有一个世界。因此，我们不需要去手动初始化世界了，只要在需要使用b2world的地方，通过GameWorld.getInstance() 方法。

下面是GameWorld的类图



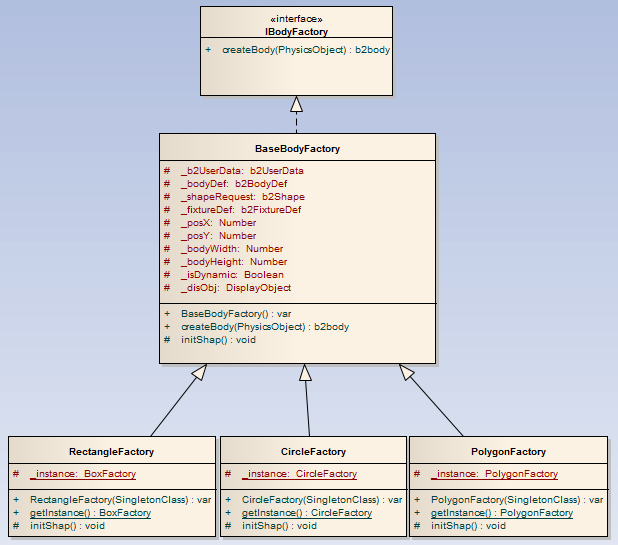
我们在这个类中初始化了世界，提供了相关属性的设置，比如Gravity等。并且提供了通过显示对象来快速获取物理对象的公开方法getBodyByDO(displayObject:\*):b2body，这极大的方便了对物理对像的调用。

**5.1.2 EasyBox2D工厂类**

我们通过GameWorld类来实现对物理世界的封装，这只是我们框架的开始，下面我们要来实现对物理刚体创建的工厂类。

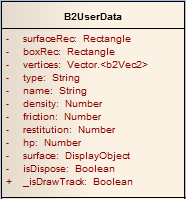
我们主要是应用工厂模式来实现对物理刚体的创建，model包主要是物理数据的格式，manager包提供了相对独立功能的管理类，math包将独立处理一些几何图形相关的数据，contact和event 包为以后实现自定义的碰撞方案预留了接口，utils包主要是提供一些静态的工具类方法，以方便我们开发中使用。

Factory是方便创建物理对象的工厂，主要包括基本形的创建和任意多边形的创建，为了在将来的开发中能更好的扩展， 这里采用了工厂模式的设计方法，如果以后有一些特殊类型的物件，也能很方便的通过继承BaseBodyFactory类来扩展，只要复写initShap()方法来提供特殊的形状。



**5.1.3 model包**

Model包内主要是实现了数据的预处理以及对数据的保存，具体实现如下



**5.1.4 debug** **manage包**

debug manage包主要提供对物理对象调试的管理

因为Box2D 的刚体是不可见的，这对于调试是个问题，为了能够使刚体可视化，我可需要设置成Debug模式，Box2D在Debug模式下会绘制线框图来表示刚体的形状和位置，具体有6种显示选择：

e\_shapeBit 绘制形状

e\_jointBit 绘制关节连接

e\_aabbBit 绘制刚体的边界框线

e\_pairBit

e\_centerOfMassBit

e\_controllerBit

为了便于对调试模式的管理，我们设计了DebugDrawManager类，用来控制调试模式的开启与关闭，能及对各种调试数据的输出选择，这有助于我们方便的管理Box2D刚体的调试。

在物理游戏中鼠标拖拽功能也是非常基本，常用的功能，因些对这个功能进行了封装。DragManager类提供了对物理对象的鼠标响应， 当鼠标点击的时候，自动获取在鼠标下的物体，并提供了拖拽的功能。

**5.1.5**  **多边形绘制辅助工具包**

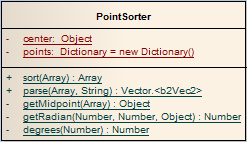
多边形绘制辅助工具包主要提供数学几何相关的计算支持

我们在项目中如果是凸多边形的创建，也可以使用Flash CS工具

* + 先将可见图形放在一个图层中作为参考对象
  + 使用钢笔等绘画工具绘制多边形。使用钢笔工具的好处是可以方便的调整各个顶点的位置。
  + 使用JSFL(Flash 提供的角本语言) 对图形顶点进行扫描并记录X与Y坐标，并输出成数据格式

这时我们可以使用凸多边形的顶点数据来绘制物理对象，但有个问题是使用JSFL输出的顶点数组顺序是乱的，这就带来了一个问题：直接使用Box2d原生方法不能正常绘制图形。

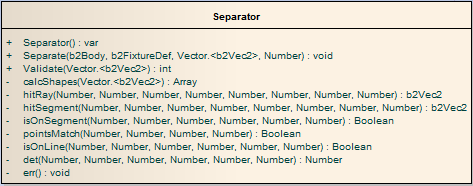
当然我们可以手动的来记录这些凸多边形的顶点数据，从而形成相应顺序的顶点数据，但这将是一项非常耗时的工作，基于此，我们设计了了PointSorter类来处理顶点排序的问题，这将大大提供我们的工作效率。



那么对于凹多边形呢，因为凹多边形的顶点排序有着不唯一性，因些，我们不能使用上面的方法来处理，经过多方搜寻，我们得到了一个变相的解决方案，原理如下：将凹多边形切割成多个三角形或四边形，然后组合成凹多边形。

创建顶点我们还是跟上面一样使用Flash CS工具，首先导入参考图， 然后实现鼠标移动记录顶点功能，进行多边形顶点采集。这个过程是根据相对时间来读到鼠标所在位子的坐标信息的。最后结束绘制时按先后顺序保存顶点数据到一个数组，以便程序使用。

从开源社区获得一个工具类Separator，我们这里采用这个工具类来帮助我们处理。

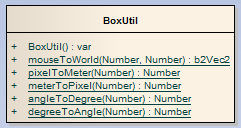


**5.1.6 event包**

event包目前只是对碰撞事件的支持，以后将根据实际情况来调整

**5.1.7 utils包**

utils包提供了简化数据转换的一些工具类



**5.2游戏编辑器的逻辑实现**

4.5.1Model包

4.5.2Factory包

4.5.3Parser包

**5.3 EasyBox2D游戏框架的总结**

**第六章 基于EasyBox2D框架的物理游戏开发实例**

**6.1 游戏架构概述**

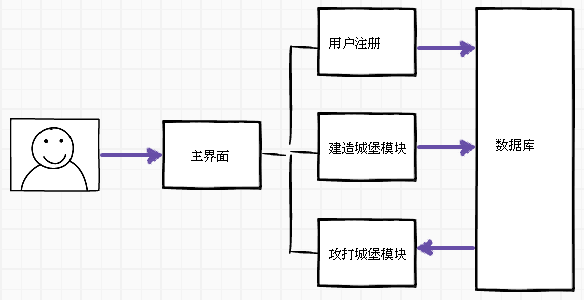
我们开发的是一款基于EasyBox2D的跨平台（IOS/Android）的手机游戏，命名为“SiegeDuel(攻城对决)”。 “攻城对决”主要讲述的是玩家对自我城堡的构建，以及对他人城堡的进攻，以此来积累自身的荣誉，实现游戏世界中的价值。

玩家通过游戏编辑器进行自我城堡的构建，构建完成后可以通过模拟攻城来对城堡的坚固程序进行测试，最后发布自已的城堡地图。

玩家可以在城堡列表中选择自已想要攻打的城保，在有限的攻力能力范围内对敌对城堡进行攻打，试图摧毁对方的城堡，以此获得不同的荣誉。

游戏主要由三个模块组成：用户注册，建造城堡，攻打城堡。

游戏架构见下图：



**6.1.1 用户注册模块**

用户注册主要是对用户数据进行生成与存储的模块，用于辨识不同的用户。这是一个游戏通常都有的模块。

对新用户的引导，不应在用户注册之后便结束。将新用户丢弃在一个陌生的页面，并且没有任何的提示，会让他们觉得不知所措。相反的，比较合适的做法是用一个欢迎信息和隐晦或明确的引导来告知用户他们接下来能做的事情。

SiegeDuel在通过用户注册后，会显示友好的欢迎信息，并且提供二个可选择的选项：建造城堡和攻打城堡。在选项的设计上不需太多，并且会描述一下每一个选项的作用，而不只是给出链接。当然描述信息可以以悬停提示等方法给出。

**6.1.2 城堡建造模块**

建造城堡模块是一个类似编辑器的游戏界面，这个界面根据玩家的不同等级，会提供二种建造模式：绘制模式和搭建模式。不同等级的玩家所能使用的城堡构建材料也可以适当的配置。

绘制模式是玩家通过选择绘制形状，通过拉拽，平移，旋转等动作来搭建游戏城堡。

搭建模式是城堡构建材料是预先设计好的，是从数据库中读取城堡材料的数据，玩家可以通过拖拽的方法来完成城堡的搭建。

**6.1.3 攻打城堡模块**

攻打城堡模块是基于EasyBox2D来实现的。

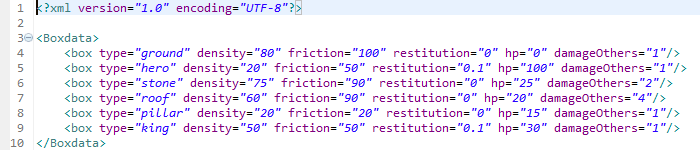
**6.2 游戏中物理城堡的实现**

6.2.1 物理城堡概述

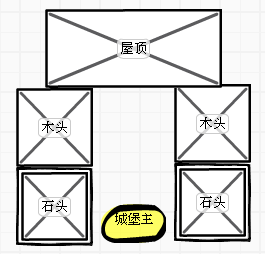
物理城堡是用不同矩形块组成的， 我们对矩形块进行分类为石块，木块，屋顶以及城堡主。我们要搭建的城堡是为了保护城堡主的安全，城堡主的灭亡则视为城堡攻破。

6.2.2 物理城堡的构建

针对不同的建筑块，会有不同的物理属性，为了便于测试以及扩展，我们将这些属性保存在相应的配置文件中



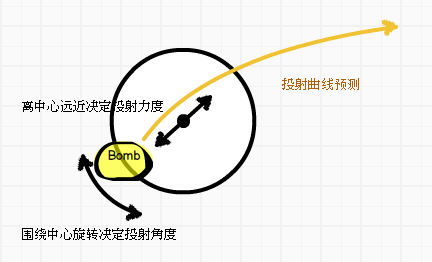
我们在进入攻城模块之后，会读取相应的存储于数据库中的城堡信息，然后根据每一个城堡建筑块的类型，去匹配相关的物理属性，最后根据建筑块的类型，调用框架中相应的工厂（Factory）去创建具有物理属性的建筑块



6.3 游戏中投弹攻打的实现

6.3.1 投弹攻打概述

投弹机制类似愤努的小鸟，投射物可以围绕圆中心内旋转，最后释放时的角度将决定投射的角度，而距离中心点的远近则决定投射的力度，在拖拽投射物的过程中，将实理绘制投射预测曲线。



6.3.2 相应公式介绍

角度公式

\_distanceX=\_heroClip.x-\_originPoint.x;

\_distanceY=\_heroClip.y-\_originPoint.y;

\_heroAngle=Math.atan2(\_distanceY,\_distanceX);

速度公式

**var** velocityX:Number=-\_distanceX/\_physicsData.\_velocityFactor;

**var** velocityY:Number=-\_distanceY/\_physicsData.\_velocityFactor;

\_arrowVelocity = **new** b2Vec2(velocityX,velocityY);

投射代码

\_hero.SetLinearVelocity(\_arrowVelocity);

6.4 本章小节

* 本章应用我们的框架，实现了城堡的搭建。
* 通过对投射机制的分析，计算出相应的作用力，并通过Box2D物理引擎将相应的作用力加到投射物上，使之产生相应的动能，实现抛物投射的效果。
* 当投射物与城堡碰撞时，进行相应的碰撞检测，并实行不同的减血策略来控制城堡的损坏。

**第七章 总结与展望**

**7.1 总结**

本文通过对Flash Box2D的细致分析，在此基础上设计实现了一套轻量级的物理开发框架

通过GameWorld实现了对物理世界的初始化封装，简化了开发的流程，并重点对物理对象的创建进行了工厂模式的设计，极大的简化了物理对象的创建。

**7.2 展望**

* + 对于任意多过形的顶点的采集，在将来的开发中将作进一步的整合，真正实现界面化操作。
  + 目前只是对Body的创建实现了封装，对相对简单的应用已经足够。将来将会把关切的创建也整合进框架中

因为能力时间有限，还有很多不足之处，希望批评指正，共同进步。

# 参考文献

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

[1]《Box2D For Flash Game》ISBN 978-1-84951-962-5

[2] Flash Box2D API Doc

[3]《ActionScript 3.0 Design Patterns》ISBN-10: 0-596-52846-9 ISBN-13: 978-0-59652846-1

[4]《Flash移动开发高级教程》ISBN 9787302282099

[5]《AIR Android应用开发实战》ISBN 9787111391777

[6]《Android移动应用开发——基于Adobe AIR》ISBN 787302285205

[7] Flash ActionScript 3.0 API Doc

[8]《ActionScript 3.0游戏编程(第2版)》ISBN 9787115272898

1. 卢胤，卜军义．手机游戏策划设计[J]．电子工业出版社，2010．
2. Keith Peters. Flash ActionScript 3.0动画高级教程. 人民邮电出版社，2010

[Box2d类图结构](http://www.cnblogs.com/Memo/archive/2012/10/05/2712400.html)

<http://www.cnblogs.com/Memo/archive/2012/10/05/2712400.html>

[Create non-convex, complex shapes with Box2D](http://www.emanueleferonato.com/2011/09/12/create-non-convex-complex-shapes-with-box2d/)

<http://www.emanueleferonato.com/2011/09/12/create-non-convex-complex-shapes-with-box2d/>

[运行时绘制多边形刚体](http://bbs.9ria.com/thread-141365-1-1.html)

<http://bbs.9ria.com/forum.php?mod=viewthread&tid=141365&highlight=box2d>

Box2D封装

E:\BOX2D\PhysInjector-master\physinjector\docs\index.html

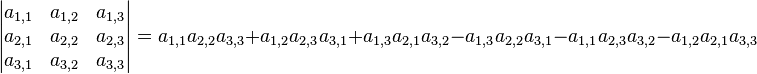
JSFL

http://wenku.baidu.com/view/e313d21fc5da50e2524d7f42.html

粒子系统原理

<http://wenku.baidu.com/view/67f05280b9d528ea81c779c5.html>

= +



*y*  *ax*2  *bx*  *c*

