

目录

第一篇 游戏中的数学基础

一、离散数学.....1

1、命题逻辑.....1

2、集合.....2

3、函数.....3

4、序列.....4

5、数学证明.....5

6、计数方法.....6

7、鸽巢原理.....7

8、图论.....8

9、树.....9

10、算术基本定理.....10

第二篇 游戏中的物理基础

第三篇 计算机图形学

一、点、线、面.....1

二、向量.....2

第三篇 游戏中的 AI

第四篇 数据结构与算法

一 排序

1、直接插入排序.....1

2、希尔排序.....2

3、直接选择排序.....3

4、归并排序.....4

5、二叉堆排序.....5

6、快速排序.....6

7、基数排序.....7

8、拓扑排序.....	8
二 邻接表	
1、建立邻接表.....	1

第五篇 AS3 语言

1. Embed标签的使用.....	1
2. 创建运行时共享库(RSL)	2
3. Graphics对象的局部擦除方法.....	3
4. BitmapData对象的局部擦除方法.....	3
5. 设计模式.....	4
6. Flex 容器出现滚动条时对子项的影响.....	5
7. 棱形地图坐标转换(一).....	6
8. 棱形地图坐标转换(二).....	7
9. Flash Player垃圾回收机制.....	8
10. 强制调用GC.....	9
11. Flash Player debug版本的安装.....	10
12. Flex Tree增加、删除节点.....	12
13. Flex Tree右键直接选中选项.....	13
14. Flex自定义显示器.....	14
15. AS3 处理Zip档案.....	15
16. E4X删除XML节点.....	16
17. Flash Player降频处理.....	17
18. Alchemy.....	18
19. ASDoc工具的使用.....	19
20. UML类关系.....	20
21. stageWidth与stageHeight始终等于 0 的处理技巧.....	21
22. 满屏切换.....	22
23. 随机生成常用汉字.....	23
24. DES加密解密.....	24
25. AES加密解密.....	25
26. 软件工程.....	26
27. 资源类Assets.....	27
28. 常用工具函数.....	28
29. 正则表达式.....	29
30. 组件元数据.....	30
31. 类的注释规范.....	31
32. FlashBuilder安装ANT.....	32
33. 让TextField的属性值立即更新.....	33
34. 求线段截点坐标.....	34
35. 从ARGB值中取出Alpha值.....	35
36. 显示对象亮度控制.....	36
37. TextFormat设置font.....	37
38. 反射机制.....	38
39. 获取显示对象某点的alpha值.....	39

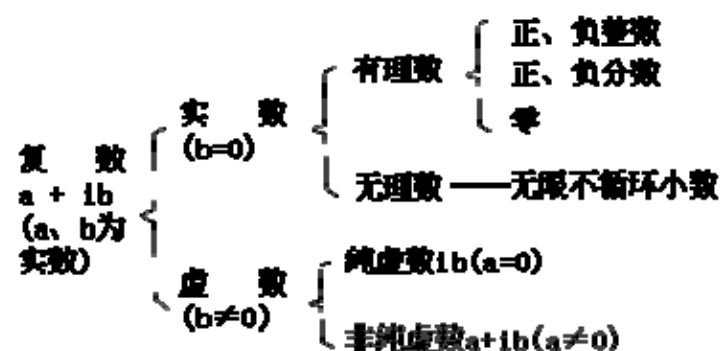
我的新浪博客

<http://blog.sina.com.cn/66flash>

我的和讯博客

<http://t.hexun.com/19022701/default.html>

数系



TextFormat 设置 font

如果要设置的字体为中文名称，则一定要使用对应的英文名称来设置才会生效。

例如：

```
var textFormat:TextFormat = new TextFormat();
textFormat.font = 'Microsoft YaHei'; //微软雅黑
```

字体中英文对照表

新细明体	PMingLiU
细明体	MingLiU
标楷体	DFKai-SB
黑体	SimHei
宋体	SimSun
新宋体	NSimSun
仿宋	FangSong
楷体	KaiTi
仿宋_GB2312	FangSong_GB2312
楷体_GB2312	KaiTi_GB2312
微软正黑体	Microsoft JhengHei
微软雅黑体	Microsoft YaHei
装 office 会多出一些字体	
隶书	LiSu
幼圆	YouYuan
华文细黑	STXihei
华文楷体	STKaiti

华文宋体	STSong
华文中宋	STZhongsong
华文仿宋	STFangsong
方正舒体	FZShuTi
方正姚体	FZYaoti
华文彩云	STCaiyun
华文琥珀	STHupo
华文隶书	STLiti
华文行楷	STXingkai
华文新魏	STXinwei

获取显示对象某点的 alpha 值

```
/**
 * 获取显示对象指定坐标点处的alpha值
 * @param    x            点击点的X坐标(pixel)
 * @param    y            点击点的Y坐标(pixel)
 * @param    display      要判断的显示对象
 * @param    destPoint    显示对象注册点坐标
 * @return   alpha值
 */
function getAlpha( x:int,y:int,display:DisplayObject,destPoint:Point=null):uint
{
    var matrix:Matrix = new Matrix();
    if(null != destPoint){
        matrix.tx = destPoint.x;
        matrix.ty = destPoint.y;
    }
    var clipRect:Rectangle = new Rectangle();
    clipRect.x = x;
    clipRect.y = y;
    var bmpData:BitmapData = new BitmapData(display.width,display.height,true,0);
    bmpData.draw(display, matrix, null, null, clipRect);
    var argb:uint = bmpData.getPixel32(x, y);
    return argb>>>24;
}
```

FlashBuilder 安装 ANT

1. 下载 JDT

<http://www.easyeclipse.org/site/plugins/eclipse-jdt.html>

2. 把 Flash Builder 目录下的 FlashBuilder.exe 拷贝一份，并命名为 eclipse.exe

3. 安装 JDT 到 Flash Builder 的安装目录

4. 设置 JAVA_HOME 系统变量

5. 设置 ANT 环境变量

ANT_HOME=E:\software\apache-ant-1.8.3

PATH=%ANT_HOME%\bin;%ANT_HOME%\lib

类的注释规范

```
/**
 * 类职责说明
 * @Author:          作者
 * @Version:         版本号(ver 1.0)
 * @Create Date:     创建时间(2012-2-29)
 * @Last Modify:     最后一次修改时间(2012-2-29)
 * @Describe        对类的详细描述
 *
 * @Usage           类的使用方式
 */
```

类模板文件

```
${package_declaration}
{
    ${import_declaration}
    /**
     *
     * @Author:          Yanis
     * @Version:         1.0.0
     * @Create Date:     ${date}
     * @Last Modify:     ${date}
     * @Describe:
     *
     * @Usage:
     */
    ${class_declaration}
    {
        ${class_body}
    }
}
```

接口模板文件

```
${package_declaration}
{
    ${import_declaration}
    /**
```

```

*
* @Author:      Yanis
* @Version:     1.0.0
* @Create Date: ${date}
* @Last Modify: ${date}
* @Describe:
*
* @Usage:
*/
${interface_declaration}
{

}
}

```

求线段截点坐标

```

/**
 * 求线段截点坐标
 * @param  x1      起点 X 坐标
 * @param  y1      起点 Y 坐标
 * @param  x2      终点 X 坐标
 * @param  y2      终点 Y 坐标
 * @param  cutLength 截长
 * @return 截点坐标 {x: ,y: }
 */
function interceptLine(x1:int, y1:int, x2:int, y2:int, cutLength:uint):Object
{
    var length12:int = Math.sqrt( Math.pow(x2-x1,2) + Math.pow(y2-y1,2) );
    if(length12 < cutLength){
        return null;
    }else if(length12 == cutLength){
        return {x: x2, y: y2};
    }
    var dx:int = (x2-x1)*cutLength/length12 + x1;
    var dy:int = (y2-y1)*cutLength/length12 + y1;

    return {x: dx, y: dy};
}

```

显示对象亮度控制

```

/**
 * 显示对象亮度控制.
 * value取值范围-1~1, 对应Flash内容制作工具里的-100%-100%

```

```

*/
public static function setBrightness(obj:DisplayObject, value:Number):void {
    var colorTransformer:ColorTransform = obj.transform.colorTransform;
    var backup_filters:* = obj.filters;
    if (value >= 0) {
        colorTransformer.blueMultiplier = 1-value;
        colorTransformer.redMultiplier = 1-value;
        colorTransformer.greenMultiplier = 1-value;
        colorTransformer.redOffset = 255*value;
        colorTransformer.greenOffset = 255*value;
        colorTransformer.blueOffset = 255*value;
    } else {
        colorTransformer.blueMultiplier = 1+value;
        colorTransformer.redMultiplier = 1+value;
        colorTransformer.greenMultiplier = 1+value;
        colorTransformer.redOffset = 0;
        colorTransformer.greenOffset = 0;
        colorTransformer.blueOffset = 0;
    }
    obj.transform.colorTransform = colorTransformer
    obj.filters = backup_filters
}

```

从 ARGB 值中取 Alpha 值

```

var argb:uint = 0xCFAABBCC;
var a:uint = argb >>> 24; //通过位移运算取 alpha 值
trace(a.toString(16));

```

让 TextField 的属性值立即更新

```

//立即更新_textField的属性解决方案, 会触发Event.SCROLL事件
var storeHeight:Number = _textField.height;
_textField.height = _textField.textHeight;
_textField.height = storeHeight;

```

正则表达式

过滤 html 标签 /<[<>]*>/gi

组件元数据(metadata)

AS3 组件元数据从 Flash MX 2004 开始, MacroMedia 为我们定义组件提供了一个新的有力工具--组件元数据. 它让我们可以直接在类中定义组件参数, 大大方便了组件的制作. 到了 Flash CS3, 组件元数据变得更为强大。

元数据标签(目前已知的可以使用的元数据标签.)

Inspectable : 公开“组件检查器”和“属性”中的属性.

InspectableList : 标识可检查属性的哪个子集应在“属性”和“组件检查器”中列出。如果不向组件的类添加 InspectableList 属性,“属性”检查器中会显示所有可检查参数.

Event : 定义组件事件,这个元数据是用来在 ASDoc 中注册事件的.

Collection : 标识在“组件”检查器中公开的 collection 属性.

Style. 定义组件样式,这个元数据是用来在 ASDoc 中注册样式的.

关于组件元数据的详细介绍.

Inspectable 标签:

使用 Inspectable 标签可以指定显示在“组件”检查器和“属性”检查器中的用户可编辑(可检查)的参数。这样,您就可以在同一个位置维护可检查属性和基本的 ActionScript. 代码。

用法:

```
[Inspectable(value_attribute=value,...)]
```

```
property_declaration name:type;
```

或者

```
[Inspectable(value_attribute=value,...)]
```

```
public function get getterFun()
```

```
public function set setterFun()
```

Inspectable 标签的属性:

属性 类型 描述

defaultValue String 或 Number (可选) 可检查属性的默认值。

enumeration String (可选) 指定以逗号分隔的属性合法值列表。

`listOffset Number`（可选）其作用是向后兼容 Flash MX 组件。它用作 `List` 值的默认索引。

`name String`（可选）属性的显示名称。例如，`Font Width`。如果未指定，则使用属性的名称，例如 `_fontWidth`

`type String`（可选）类型指定。如果省略，则使用属性的类型。下面是可接受的值：`Array`, `Boolean`, `Color`, `Font Name`, `List`, `Number`, `Object`, `String`, `Web Service Url`, `Web Service Operation`, `Collection Item`, `Collection`, `Category`, `Class`, `Video Cue Points`, `Video Skin`, `Video Content Path`, `Video Preview`

`variable String`（可选）其作用是向后兼容 Flash MX 组件。指定此参数所绑定的变量。

`verbose Number`（可选）将 `verbose` 属性设置为 1 的可检查属性，它不显示在“属性”检查器中，但显示在“组件”检查器中。通常用于不经常修改的属性。

对于 `type` 参数的一些解释：

Flash CS3 中的 `type` 参数可选项明显多于 Flash 8.

其中,`Array`, `Boolean`, `Color`, `Font Name`, `List`, `Number`, `Object`, `String` 的定义就如同字面解释,`Collection` 与 `Collection` 标签有关, `Video Cue Points` 是加入多个视频提示点, `Video Skin` 是输入一个用于 `FlvPlayBack` 组件的皮肤, `Video Content Path` 是输入一个 `flv` 文件的路径, `Video Preview` 是用于在创作是预览图像,这四个参数是专门用于 `FlvPlayBack` 组件的.

InspectableList 标签

`InspectableList` 标签用于指定“属性”检查器中应显示可检查属性的哪个子集。请将 `InspectableList` 与 `Inspectable` 组合使用，这样可以隐藏子类组件的继承属性。如果不给组件的类添加 `InspectableList` 标签，所有可检查的参数（包括组件父类的可检查参数）都会显示在“属性”检查器中。

用法：

```
[InspectableList("attribute1",[...])]
```

// 类定义

InspectableList 标签必须紧挨着类定义且在它之前,因为它应用于整个类。

Event 标签

Event 标签用于定义组件发出的事件。

用法:

```
[Event(name="event_name", type="event_class")]
```

event_name 是事件的类型, event_class 是对应的事件类. 在 AS2 中, event_class 是不需要的(也没有),但是在 AS3 中, event_class 是必须的,而且需要是类的完全限定名(包括类所在的包)。

在 ActionScript. 文件中将 Event 语句添加到类定义之外,以便将这些事件绑定到类,而不是绑定到类的特定成员。

这个元数据是用来在 ASDoc 中注册事件的,也就是说,这个标签在类的使用中不会起什么作用,但是可以让这个事件显示在 ASDoc 中。

Collection 标签

Collection 标签用于描述在创作时可在“值”对话框中修改为项目集合的对象数组。这些对象的类型由 collectionItem 属性标识。Collection 属性包含一系列在单独类中定义的集合项目。

用法:

```
[Collection(name="name",collectionClass="collection_class",  
collectionItem="collectionItem_class", identifier="string")]
```

标签属性说明:

属性 类型 描述

name String (可选) 显示在“组件”检查器中的集合名称。

`variable String` (可选) 指向基础 `Collection` 对象的 `ActionScript` 变量 (例如, 可以将 `Collection` 参数命名为 `Columns`, 但基础 `variable` 属性可以是 `__columns`)。

`collectionClass String` (必需) 指定要将集合属性实例化的类的类型。

`collectionItem String` (必需) 指定要存储在集合中的一类集合项目。该类包含它自己的通过元数据公开的可检查属性。

`identifier String` (必需) 指定在用户通过“值”对话框添加一个新集合项目时, `Flash` 用作默认标识符的可检查属性 (在集合项目类中) 的名称。每当用户创建一个新集合项目时, `Flash` 都会将该项目的名称设置为 `identifier`, 外加一个唯一索引 (例如, 如果 `identifier=name`, 则“值”对话框显示 `name0`、`name1`、`name2`, 依此类推)。

`collectionClass` 的说明:

`collectionClass` 的一个例子就是 `fl.data.DataProvider` 类, 它是一个数据包装类, 使用 `[{label:"label1", data:"data1"}, {label:"label2", data:"data2"}, {label:"label3", data:"data3"}, ...]` 格式的数据。所以, 只要类似 `DataProvider` 的类都可以用作 `collectionClass`。`collectionClass` 类无需继承自 `DataProvider` 类, 但必须有 `addItem(item:Object)` 方法。

`collectionItem` 的说明:

`collectionItem` 就是 `collectionClass` 中的一项, 它就是一个拥有与 `collectionClass` 类中的项属性相同的类。一个例子是 `fl.data.SimpleCollectionItem` 类, 它就是一个有 `label` 和 `data` 两个属性的类。`collectionItem` 类无需继承自 `SimpleCollectionItem` 类。

当在 `Flash IDE` 中通过“属性”面板或是“组件检查器”更改由 `Collection` 标签标识的属性时, 就会用 `collectionClass` 类的 `addItem` 方法添加多个 `collectionItem` 对象, 这也是为什么 `collectionItem` 类需要与 `collectionClass` 类需要属性相匹配。

`Style` 标签:

`Style` 标签用于标明该组件所用的样式。

用法:

```
[Style(name="style_name", type="style_type", format="type_format")]
```

这个元数据是用来在 `ASDoc` 中注册样式的, 也就是说, 这个标签在类的使用中不会起什么作

用,但是可以让这个样式显示在 ASDoc 中. 因此, type 和 format 属性的值可以随便定义, 因为它们在 ActionScript. 中不具有任何特殊意义, 不过, 以下是一些建议的属性值.

Style 标签的属性:

type 属性

1. Number 指示这是一个数字, 可以使用 Length 和 Time 作为 format 属性, 分别指示这是一个长度数字还是时间数字.

2. Boolean 指示这是一个布尔值.

3. Class 指示这是一个类, 它可以是一个类, 也可以是一个类的实例.

4. flash. text. TextFormat 显然, 这是一个 TextFormat 对象.






一、 反射机制

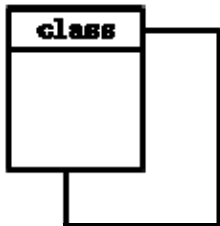
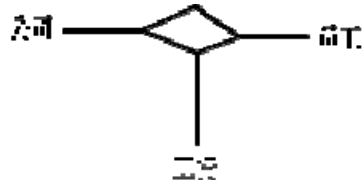

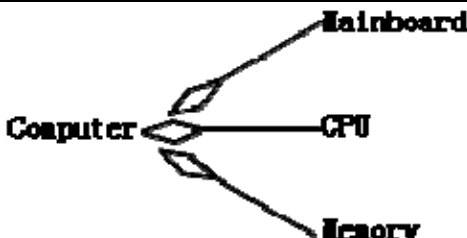

通过类名称的字符串形式来创建类实例。

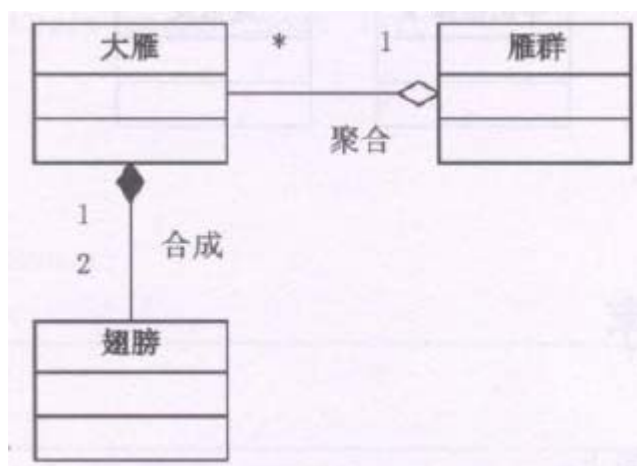
例:

```
import flash.system.ApplicationDomain;
import flash.display.MovieClip;
var cls:Class = ApplicationDomain.currentDomain.getDefinition("flash.display.MovieClip") as Class; //括号里填类名的字符串形式
trace(cls); //输出 [class MovieClip]
```

UML 类关系

名称	符号	举例
泛化		A extends B
实现		A implement IA
依赖		不建议双向依赖 依赖体现在局部变量、方法参数中。 例如： A 对象依赖其他对象提供的服务
单向关联		弱关联：对象之间有导航关系 例如： 商品<-订单
双向关联		弱关联：对象之间有导航关系 一对多的双向关联 例如： 公司—员工

自身关联		弱关联：对象之间有导航关系
多维关联		弱关联：对象之间有导航关系
聚合		 <p>聚合表示一种弱的‘拥有’关系，体现的是 A 对象可以包含 B 对象，但 B 对象不是 A 对象的一部分。 例如：大雁-雁群</p>
组合（合成）		<p>合成则是一种强的‘拥有’关系，体现了严格的部分和整体的关系，部分和整体的生命周期一样。 例如：大雁-翅膀</p>



随机生成常用汉字

```
/**
 * 随机获取 1 个一级汉字的 GB2312 码
 */
function getGB2312Code():uint
```

```

{
    //常用汉字区码 16~55
    var areaCode:uint = Math.random()*40 + 16;
    //常用汉字位码 01~94
    var posCode:uint;
    if(55 == areaCode) {
        posCode = Math.random()*89 + 1;
    }else{
        posCode = Math.random()*94 + 1;
    }
    //位码转 GB2312 码
    areaCode = (160 + areaCode)*256;//256=16*16
    posCode = 160 + posCode;

    var gb2312:uint = areaCode + posCode;

    return gb2312;
}
/**
 * 随机获取 1 个一级汉字
 */
function getZhChar():String
{
    var arr:ByteArray = new ByteArray();
    var gb2312Code:uint = getGB2312Code();
    arr.writeShort(gb2312Code);
    var len:uint = arr.length;
    arr.position = 0;
    var zhChar:String = arr.readMultiByte(len, "gb2312");

    return zhChar;
}

//随机打印出 1 个一级汉字
trace(getZhChar());

```

DES 加密解密

```

package crypto
{
    import com.hurlant.crypto.symmetric.DESKey;
    import com.hurlant.util.Base64;
    import flash.utils.ByteArray;
    /**

```

```

* DES加密、解密
* @author      Yanis
* @version     1.0.0
* @date       2012-2-8 下午07:01:34
*/
public class DESCrypto
{
    /**
     * 对字符串DES加密
     * @param    keyStr        密钥
     * @param    encryptStr    要加密的字符串
     * @return
     *
     */
    public static function encryptString(keyStr:String,
encryptStr:String):String
    {
        var key:ByteArray = new ByteArray();
        key.writeUTFBytes(keyStr);
        var des:DESKey = new DESKey(key);

        var encryptArr:ByteArray = new ByteArray();
        encryptArr.writeUTFBytes(encryptStr);

        des.encrypt(encryptArr, 0);

        //这里采用Base64, 也可用其它编码
        var outStr:String = Base64.encodeByteArray(encryptArr);

        return outStr;
    }
    /**
     * 对字节数组DES加密
     * @param    keyStr        密钥
     * @param    encryptArr    要加密的字节数组
     * @return
     */
    public static function encryptByteArray( keyStr:String,
encryptArr:ByteArray ):ByteArray
    {
        var key:ByteArray = new ByteArray();
        key.writeUTFBytes(keyStr);
        var des:DESKey = new DESKey(key);

```



```

        des.encrypt(encryptArr, 0);

        return encryptArr;
    }

    /**
     * 对字符串DES解密
     * @param    eyStr        密钥
     * @param    decryptStr    要解密的字符串
     * @return
     *
     */
    public static function decryptString(keyStr:String,
decryptStr:String):String
    {
        var key:ByteArray = new ByteArray();
        key.writeUTFBytes(keyStr);
        var des:DESKey = new DESKey(key);

        var decryptArr:ByteArray = Base64.decodeToByteArray(decryptStr);
        des.decrypt(decryptArr, 0);

        var outStr:String = decryptArr.readUTFBytes(decryptArr.length);

        return outStr;
    }

    /**
     * 对字节数组DES解密
     * @param    keyStr        密钥
     * @param    decryptArr    要解密的字节数组
     * @return
     */
    public static function decryptByteArray( keyStr:String,
decryptArr:ByteArray ):ByteArray
    {
        var key:ByteArray = new ByteArray();
        key.writeUTFBytes(keyStr);
        var des:DESKey = new DESKey(key);

        des.decrypt(decryptArr, 0);
        decryptArr.position = 0;

        return decryptArr;
    }

```

```

    }
}
}

```

AES 加密解密

```

package crypto
{
    import com.hurlant.crypto.symmetric.*;
    import flash.utils.*;

    /**
     * AES加密、解密
     * @author Yanis
     * @version 1.0.0
     * @date 2012-2-9 下午03:29:14
     */
    public class AESCrypto
    {
        /**
         * 对字节数组加密
         * @param keyStr 密钥
         * @param encryptArr 要加密的字节数组
         * @param iv 初始向量
         * @return
         */
        public static function encryptByteArray( keyStr:String, encryptArr:ByteArray,
        iv:String ):ByteArray
        {
            //AES密钥
            var key:ByteArray = stringToByteArray(keyStr);
            //密码分组链接模式(CBC)
            var cbc:CBCMode = new CBCMode(new AESKey(key), new NullPad());
            //初始向量
            cbc.IV = stringToByteArray(iv);
            //加密
            cbc.encrypt(encryptArr);

            return encryptArr;
        }
        /**
         * 对字节数组解密
         * @param keyStr 密钥
         * @param decryptArr 要解密的字节数组
         * @param iv 初始向量
         * @return
         */
    }
}

```

```

    */
    public static function decryptByteArray( keyStr:String, decryptArr:ByteArray,
    iv:String ):ByteArray
    {
        //AES密钥
        var key:ByteArray = stringToByteArray(keyStr);
        //密码分组链接模式(CBC)
        var cbc:CBCMode = new CBCMode(new AESKey(key), new NullPad());
        //初始向量
        cbc.IV = stringToByteArray(iv);
        //解密
        cbc.decrypt(decryptArr);

        return decryptArr;
    }
    /**
     * String转ByteArray
     */
    public static function stringToByteArray( str:String ):ByteArray
    {
        var byteArr:ByteArray = new ByteArray();
        byteArr.writeUTF(str);

        return byteArr;
    }
}

```

资源类 Assets

```

package assets
{
    import flash.display.*;
    import flash.media.Sound;
    /**
     * 管理嵌入的资源
     */
    public class Assets
    {
        //加载进度条增减条
        [Embed(source='resources/FacePack.swf')]
        public static var FacePack:Class;

        /**

```

```

    * 获取皮肤位图
    */
    public static function getBitmap( className:String ):Bitmap
    {
        var photo:Class = Assets[className];
        if( null == photo ) return null;

        var skinBitmap:Bitmap = new photo() as Bitmap;
        if( null != skinBitmap ){
            return skinBitmap;
        }
        return null;
    }
    /**
    * 获取皮肤显示对象
    */
    public static function getDisplayObject( className:String ):DisplayObject
    {
        var swf:Class = Assets[className];
        if( null == swf ) return null;

        var skinSwf:DisplayObject = new swf() as DisplayObject;
        if( null != skinSwf ){
            return skinSwf;
        }
        return null;
    }
    /**
    * 获取声音
    */
    public static function getSound( className:String ):Sound
    {
        var snd:Class = Assets[className];
        if( null == snd ) return null;

        var skinSnd:Sound = new snd() as Sound;
        if( null != skinSnd ){
            return skinSnd;
        }
        return null;
    }
}

```

stage.stageWidth 和 stage.stageHeight 始终等于 0 的处理技巧

```
stage.addEventListener(Event.RESIZE, onStageResize);
//在打开浏览器时，这个事件会被多次触发
public function onStageResize( evt:Event ):void
{
    if(stage.stageWidth>0 || stage.stageHeight>0){
        init();//此时就可以进行各种布局操作了
    }
}
```

满屏切换

js

```
<script type="text/javascript" src="swfobject.js"></script>
<script type="text/javascript">
    // For version detection, set to min. required Flash Player version, or 0
    (or 0.0.0), for no version detection.
    var swfVersionStr = "10.2.0";
    // To use express install, set to playerProductInstall.swf, otherwise the
    empty string.
    var xiSwfUrlStr = "playerProductInstall.swf";
    var flashvars = {};
    var params = {};
    params.quality = "high";
    params.bgcolor = "#000000";
    params.allowscriptaccess = "sameDomain";
    params.allowfullscreen = "true";
    var attributes = {};
    attributes.id = "MainApp";//此id为嵌入swf的引用
    attributes.name = "MainApp";
    attributes.align = "middle";
    swfobject.embedSWF(
        "MainApp.swf", "flashContent",
        "1000", "580",
        swfVersionStr, xiSwfUrlStr,
        flashvars, params, attributes);
    // JavaScript enabled so display the flashContent div in case it is not
    replaced with a swf object.
    swfobject.createCSS("#flashContent", "display:block;text-align:left;");

    // 切换视窗模式
    function changeScreenModel( model )
```

```
{
    var swf = swfobject.getObjectById('MainApp');
    switch(model) {
        case 'normal':
            swf.width = 1000;
            swf.height = 580;
            break;
        case 'fullScreen':
            swf.width = '100%';
            swf.height = '100%';
            break;
    }
}
</script>
```

ASDoc 工具的使用

Flash Builder4

运行->外部工具->外部工具配置

位置: **asdoc.exe** 所在文件系统路径

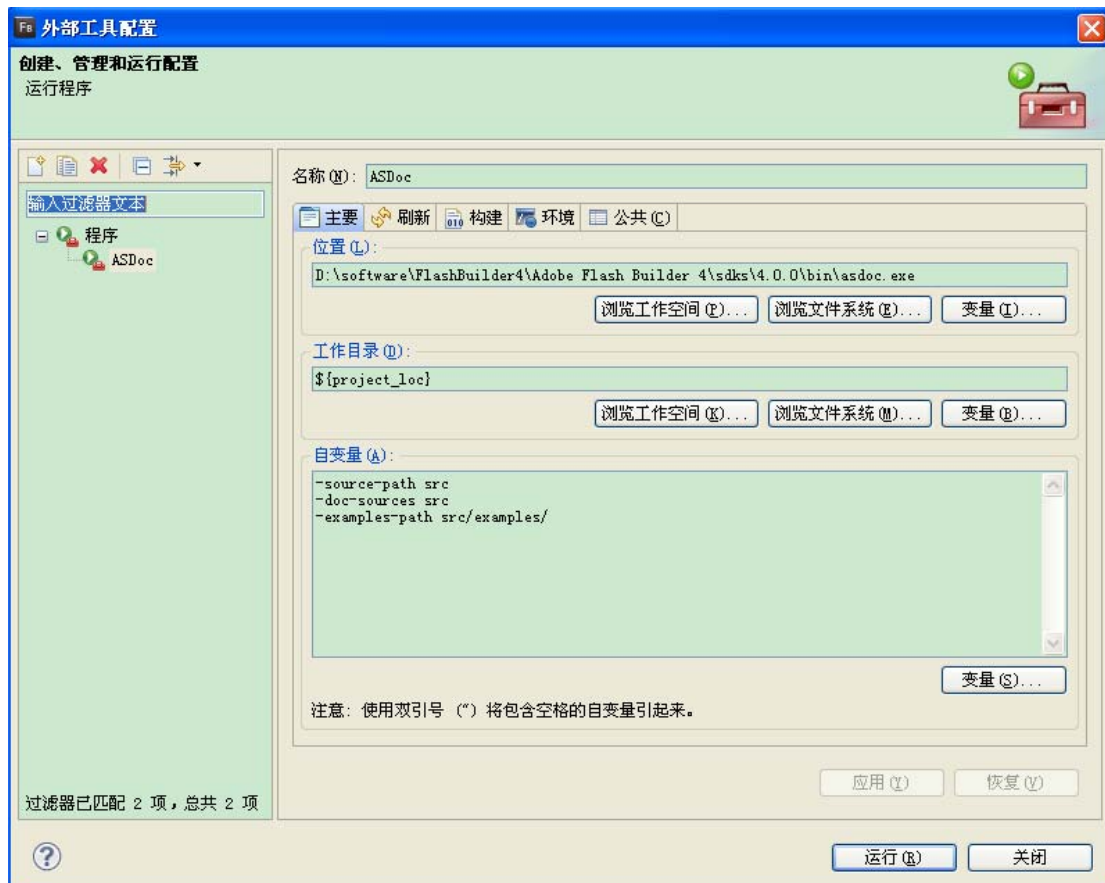
工作目录: **\${project_loc}**

自变量:

-source-path src

-doc-sources src

例如:



/**

* 第一句。

* 第二句。

*/

toString():String

返回一个字符串，其中包含 **Event** 对象的所有属性。

注释中的第一句会被显示在这。

如果 ASDoc 工具是英文，那么第一句应该以 (.) 号结尾

/**

* 第一句.

*/

如果 ASDoc 工具是中文，那么第一句应该以 (。) 号结尾

/**

* 第一句。

*/

导出 ASDoc 文档时给包加注释

-package org.rpg2d.component.core “包注释”

项目路径指向 src 目录,如果在当前目录就用.号表示

-source-path src

递归搜索 src 目录

`-doc-sources src`

递归搜索 component 包

`-doc-sources src/org/rpg2d/component`

指定要导出的类

`-doc-classes org.rpg2d.component.core.UIComponent`

排除某个类

`-exclude-classes org.rpg2d.component.core.UIComponent`

包含第三方 swc

`-library-path src/lib`

通常设置为

`-source-path src`

`-library-path src/lib`

`-doc-sources src`

`-examples-path src/examples/`

以上设置可对 src 下的所有类以及 lib 下的 swc 以及 examples 下的示例一起编译。

@param 标记是为带参数的函数中的参数作注释用的标记。通过此标记可以生成对应的参数的说明。此标记的书写格式如下：

@param 参数名称 参数说明

@example 是可以为某个类、方法或者属性加一个说明性的例子，从而让自己的代码更加容易理解。其书写格式为：

@example 例子的说明性文字

<listing version=" 3.0" >

例子的代码

</listing>

@includeExample 此标记是引入一个外部示例文本到 ASDoc 输出中。ASDoc 将从由 ASDoc 工具的 -examples-path src/examples/ 参数指定的基于类的包名或者目录来搜索示例文件。

假设编译命令为 -examples-path src/examples/

示例文件放在 src/examples/uis/controls/UIButton.txt

例：

```
package uis.controls
```

```
{
```

```
    /**
```

```
     * 游戏常规按钮
```

```
     * @author      Yanis
```

```
     * @version     1.0.0
```

```
     * @date        2012-1-13 下午11:46:16
```

```
     * @internal
```

```
     * @includeExample UIButton.txt
```



```

*/
public class UIButton{

}
}
@includeExample UIButton.txt

```

```

/**
 * @exampleText 这是一个例子文件
 **/
//这是一个例子文件UIButton.txt
var demo:Demo=new Demo();
demo.getString();

```

@return 针对一个方法的返回值进行说明。也就是说此标记是用于方法中的注释的。其中其书写格式如下：

@return 说明文字

@see 标记的作用是生成一个参考引用。在一些情况下某些类、属性或者方法在其他地方有进行说明或者引用,这时候我们可以通过此标记来引用此例子来进行说明。其书写格式如下：

@see 引用 [显示文本]

@private 标记此标记的作用是，当你的部分类、方法或属性不想公开给其他人看到的时候，可以在相应的地方加上此标记。

例如：

```

/**
 * 返回一个字符串
 * @private
 * @return 返回一个字符串
 * @see #print() 具体用法请参考 print 方法
 * @see Demo2#getString()
 **/
public function getString():String{
    return "demo";
}

```

如果 ASDoc 导出的文档存在中文乱码

打开 flash builder 使用的 SDK 的目录, 打开 bin 目录, 打开 jvm.config 文件, 在 java.args= 开始的那行后面加入个参数, 修改后的结果为:

```

java.args=-Xmx384m -Dsun.io.useCanonCaches=false -Dfile.encoding=utf-8
-Dsun.jnu.encoding=utf-8

```

Flash Player 降频补帧处理

```
/**
 * EnterFrame事件处理
 */
private var frameRate:uint;
private var halfRate:uint = frameRate >> 1;
private var intervalTime:uint = 1000/frameRate;
private var deltaTime:Number = 0;
private var lastTime:uint = 0;
private function enterFrame(e:Event):void
{
    /** 计算实际帧频 */
    var nowTime:uint = getTimer();
    deltaTime = nowTime - lastTime;
    lastTime= nowTime;
    if(_loop){
        var realFrame:uint = uint(1000/deltaTime);/** 上一帧的实际帧频 */
        if(realFrame >= halfRate){
            gameLoop();
        }else{
            /** flashplayer最小化后要做的事 */
            /** 补帧 */
            var tweenFrames:uint = frameRate/realFrame;
            for(var i:uint=0; i<tweenFrames; i++){
                gameLoop();
            }
        }
    }
}
```

Flash Player 播放空 Sound 阻止降频

```
var sound:Sound = new Sound(new URLRequest(""));
sound.play();
sound.close();
```

删除 XML 节点

下面的语句无法正常进行:

```
delete _customerList.customer. (@customerID == id);
```

提示: Error #1119: Delete operator is not supported with operand of type XMLList

解决方案为:

```
delete _customerList.customer. (@customerID == id)[0];
```

Flex 容器出现滚动条时对子项的影响

容器第一次出现滚动条时，会触发子项的 Event.REMOVED_FROM_STAGE 和 Event.ADDED_TO_STAGE 事件

如果要避免这种现象，可以在容器初始化时就让它出现一次滚动条

```
var tmpcom:Canvas = new Canvas();
    tmpcom.width=1000;//设大点，足以让_container出现滚动条
_container.addElement(tmpcom);
_container.callLater(function():void{_container.removeChild(tmpcom);});//初始化完
    毕后，移除 tmpcom;
```

Flex Tree 增加、删除节点

```
//添加子节点
private function addNode():void{
    var xml:XML=tree.selectedItem as XML;
    xml.appendChild(<node label='hello' id='0' />);
}
//删除子节点
private function delNode():void{
    tree.dataDescriptor.removeChildAt(tree.selectedItem.parent(), tree.selectedItem
, tree.selectedItem.childIndex(), tree.dataProvider);
}
```

Flex Tree 右键直接选中选项

```
private function rightMouseUpHdl( evt:MouseEvent ):void
{
    if(evt.target!=null&&evt.target is UITextField && UITextFiel(evt.target).owner !=
    null){
        this.selectedItem = TreeItemRenderer(UITextField(evt.target).owner).data;
    }else{
        this.selectedItem = null;
    }
}
```

Flex 自定义呈示器

```
public class UIFrameRenderer extends Canvas implements IListItemRenderer
{
    override public function get data():Object
    {
        return null;
    }
    /**
     * 外部通过设置这个data来达到更新项目呈示器的功能
     */
}
```

```

        override public function set data(value:Object):void
        {

        }
    }
}

```

//使用自定呈示器

```

var ifactory:ClassFactory = new ClassFactory(UIFrameRenderer);
var frameList:HorizontalList = new HorizontalList();
frameList.itemRenderer = ifactory;

```

AS3 处理 Zip 档案

先下载 as3-ziparchive.swc

<http://code.google.com/p/as3-ziparchive/>

在线 API

<http://www.riaidea.com/as3/zip/asdoc/>

```

package {
    import flash.display.*;
    import flash.events.*;
    import flash.utils.*;
    import com.riaidea.utils.zip.*;

    [SWF(width = 500, height = 400, frameRate = 24, backgroundColor =
    0xFFFFFF)]
    ;

    public class Example extends Sprite {
        private var zip:ZipArchive;
        private var swc:ZipArchive;
        public function Example() {
            testZip();
            testSWC();
        }

        private function testZip():void {
            zip = new ZipArchive();
            handleEvents(zip, true);
            zip.load("assets/test.zip");
        }

        private function processZip():void {
            //显示zip文件的详细文件信息
            trace(zip.toComplexString());
            //读取zip中的xml文件
            var xmlFile:ZipFile = zip.getFileByName("sample.xml");
            var xml:XML = new XML(xmlFile.data);
            trace(xml);
        }
    }
}

```

```

//复制zip中的"girl.jpg"为"张曼玉.jpg"
var zmy:ZipFile = zip.getFileByName("girl.jpg");
zip.addFileFromBytes("张曼玉.jpg", zmy.data);
//异步加载并显示zip中的新生成的图片"张曼玉.jpg"
zip.getAsyncDisplayObject("张曼玉.jpg",
    function(img:DisplayObject):void
    {
        addChild(img);
        img.x = 10;
        img.y = 10;
    } );
//删除zip中的文件"girl.jpg"
zip.removeFileByName("girl.jpg");
//异步加载并显示zip中的SWF文件"loading.swf"
zip.getAsyncDisplayObject("loading.swf",
    function(swf:DisplayObject):void
    {
        addChild(swf);
        swf.x = 150;
        swf.y = 10;
    } );
//根据字符串内容创建一个新的txt文件
var txtContent:String = "这是一个测试文本文件";
zip.addFileFromString("empty_dir/test.txt", txtContent);
//显示修改后的zip文件信息
trace(zip.toComplexString());
}

private function testSWC():void {
    swc = new ZipArchive();
    handleEvents(swc, true);
    swc.load("assets/puremvc.swc");
}

private function processSWC():void {
    //显示swc文件的详细文件信息
    trace(swc.toComplexString());
    //读取swc文件中的所有类定义
    var catalog:ZipFile = swc.getFileByName("catalog.xml");
    var catalogXML:XML = XML(catalog.data);
}

trace(catalogXML..(catalogXML.namespace())::def);

}

private function handleEvents(zip:ZipArchive, add:Boolean):void {
    if (add) {
        zip.addEventListener(ZipEvent.PROGRESS, onProgress);
    }
}

```

```

        zip.addEventListener(ZipEvent.LOADED, onLoaded);
        zip.addEventListener(ZipEvent.INIT, onInit);
        zip.addEventListener(ZipEvent.ERROR, onError);
    } else {
        zip.removeEventListener(ZipEvent.PROGRESS, onProgress);
        zip.removeEventListener(ZipEvent.LOADED, onLoaded);
        zip.removeEventListener(ZipEvent.INIT, onInit);
        zip.removeEventListener(ZipEvent.ERROR, onError);
    }
}

private function onInit(evt:ZipEvent):void {
    handleEvents(evt.target as ZipArchive, false);
    switch (evt.target) {
        case zip :
            processZip();
            break;
        case swc :
            processSWC();
            break;
    }
}

private function onProgress(evt:ZipEvent):void {
    //trace(evt.message.bytesLoaded, evt.message.bytesTotal);
}

private function onLoaded(evt:ZipEvent):void {
    //trace(evt);
}

private function onError(evt:ZipEvent):void {
    //trace(evt);
}
}
}

```

Flash Player 垃圾回收机制 (GC)

1. 延缓引用计数
2. 标记-清除

Stack 是存储所有在编译期定义的变量的内存空间。Stack 内存主要是以连续性的方式被使用且重复使用。

对象存储在 Heap 中。GC 只对 Heap 中的对象引用计数。

对于创建的弱引用对象，不会被 GC 标记，所以将在下一次 GC 时被回收。

强制调用 GC

```
/**
```

```

* 调用 GC
*/
public function gc():void
{
    try{
        new LocalConnection().connect("foo");
        new LocalConnection().connect("foo");
    }catch(error:Error){

    }
}

```

Graphics 对象的局部擦除方法

```

drawmc.graphics.beginFill(0xFF0000, 1);
drawmc.graphics.drawRect(0, 0, 100, 100); //绘制
//连续 draw 两次，即可擦除与上句画的重叠区域
drawmc.graphics.drawRect(20, 20, 50, 50);
drawmc.graphics.endFill();

```

以填充方式擦除

```

var pos:Object = {x: 0, y: 0};
drawmc.graphics.beginFill(0xFF0000, 0.5);

drawmc.graphics.moveTo(pos.x, pos.y-50);
drawmc.graphics.lineTo(pos.x+50, pos.y);
drawmc.graphics.lineTo(pos.x, pos.y+50);
drawmc.graphics.lineTo(pos.x-50, pos.y);
drawmc.graphics.lineTo(pos.x, pos.y-50);
/** 连续画两次，即擦除与上次画的重叠区域 */
drawmc.graphics.moveTo(pos.x, pos.y-50);
drawmc.graphics.lineTo(pos.x+50, pos.y);
drawmc.graphics.lineTo(pos.x, pos.y+50);
drawmc.graphics.lineTo(pos.x-50, pos.y);
drawmc.graphics.lineTo(pos.x, pos.y-50);

drawmc.graphics.endFill();

```

注意：一旦调用了 `endFill()` 或重设了 `beginFill()`，就擦不掉了。

BitmapData 对象局部擦除法

利用 `merge()` 方法

```

var bmd1:BitmapData = new BitmapData(100, 80, true, 0xFF00FF00);
var bmd2:BitmapData = new BitmapData(100, 80, true, 0x00000000);
var rect:Rectangle = new Rectangle(0, 0, 20, 20); //要擦除的区域大小

```

```
var pt:Point = new Point(20, 20);
var mult:uint = 0x80;
bmd1.merge(bmd2, rect, pt, mult, mult, mult, mult);
```

利用 **colorTransform()**方法

```
var bmd:BitmapData = new BitmapData(100, 100, true, 0xFFFF0000);
var cTransform:ColorTransform = new ColorTransform();
cTransform.alphaMultiplier = 0;
bmd.colorTransform(new Rectangle(0,0,20,20),cTransform);
this.addChild(new Bitmap(bmd));
```

序列化/反序列化对象

```
package
{
    import flash.geom.Point;
    public class Test {

    }
}
//对于序列化复合类，导入的类也必须注册
registerClassAlias("point",Point);
registerClassAlias("test",Test)

var eg:Test = new Test();
var ba:ByteArray = new ByteArray();
ba.writeObject(eg);
ba.position = 0;
var eg1:* = ba.readObject();
trace(eg1 is Test);//true
```

去除字符串两边空白字符

```
//去掉左边空白字符
function ltrim( s:String ):String
{
    var white:String = "\f\n\r\t\v";
    var slen:uint = s.length;
    for(var i:uint=0; i<slen; i++){
        var f:int = white.indexOf(s.charAt(i));
        if(-1 == f){
            break;
        }
    }
    s = s.substr(i);
    return s;
}
```



```

}
//去掉右边空白字符
function rtrim( s:String ):String
{
    var white:String = "\f\n\r\t\v";
    var slen:uint = s.length;
    for(var i:int=slen-1; i>=0; i--){
        var f:int = white.lastIndexOf(s.charAt(i));
        if(-1 == f){
            break;
        }
    }
    s = s.substring(0, i+1);
    return s;
}
//去掉两边空白字符
function trim( s:String ):String
{
    s = rtrim(s);
    s = rtrim(s);
    return s;
}

```

AS3 程序自身预加载

1. 添加附加编译参数-frames.frame Main Main 把我们的主程序类 Main 编译到第二帧去。
2. 创建自身预加载类 Preloading(设置为文档类)
3. 加载完成后利用反射机制创建主类对象

```

//创建预加载类，并设为文档类
package
{
    [SWF(backgroundColor='0x000000', frameRate='24', width='1000', height='580')]
    public class Preloading extends MovieClip
    {
        private var _loadingText:TextField;

        public function Preloading()
        {
            stop();
            trace("preloading start");
            addEventListener(Event.ADDED_TO_STAGE, eventsHandler);
        }

        /**
         * handle events

```

```

    * 处理ADDED_TO_STAGE和ENTER_FRAME事件
    * @param    evt
    */
private function eventsHandler(evt:Event):void
{
    switch(evt.type)
    {
        case Event.ADDED_TO_STAGE:
            stage.scaleMode = StageScaleMode.NO_SCALE;
            stage.align = StageAlign.TOP_LEFT;
            createLoading();
            removeEventListener(Event.ADDED_TO_STAGE, eventsHandler);
            addEventListener(Event.ENTER_FRAME, eventsHandler);
            break;

        case Event.ENTER_FRAME:
            var percent:int = (loaderInfo.bytesLoaded /
loaderInfo.bytesTotal) * 100 >> 0;
            _loadingText.text = percent + "% Loaded";
            trace("loading:", percent);
            if (loaderInfo.bytesLoaded == loaderInfo.bytesTotal)
            {
                removeEventListener(Event.ENTER_FRAME, eventsHandler);
                removeChild(_loadingText);
                _loadingText = null;
                nextFrame();
                initApp();
            }
            break;
    }
}

/**
 * you can create your own loading content here
 * 在这里你可以定制你自己的loading内容
 */
private function createLoading():void
{
    _loadingText = new TextField();
    _loadingText.x = (this.stage.stageWidth >> 1) - 100;
    _loadingText.y = this.stage.stageHeight >> 1;
    var format:TextFormat = new TextFormat("Arial", 24, 0xFFFFFFFF, true);
    _loadingText.defaultTextFormat = format;
    _loadingText.autoSize = "center";
}

```

```

        addChild(_loadingText);
        trace("create loading");
    }

    /**
     * initialize main application, the application class should be compiled into
     * the swf's second frame by using mxmcl additional compiler option:
frames.frame label classname.
     * for example: -frames.frame Application Application
     * 初始化主应用程序，主应用程序类Application必须用mxmcl的编译参数
frames.frame编译到swf文件的第二帧中去。
     */
    private function initApp():void
    {
        var appClass:Class = getDefinitionByName("Main") as Class;
        if (appClass != null)
        {
            var app:Object = new appClass();
            addChild(app as DisplayObject);
        }
    }
}

```

克隆对象

```

// 此方法并非深度克隆
var copier:ByteArray = new ByteArray();
copier.writeObject(source);
copier.position = 0;
return copier.readObject();

```

另请参见:

```

getQualifiedClassName()
getQualifiedSuperclassName()
describeType()
getClassByAlias()
registerClassAlias()

```

命名空间

```

//声明命名空间
public namespace cat;
//声明方法
cat function eat():void {
    trace("我喜欢鱼");
}

```

```

}
//调用方法
use namespace dog; //使用 use 指令引用
eat();
//或
cat::eat(); //使用限定名称来引用

//对象调用
instance.cat::eat();

```

注：通常 public 的命名空间直接以字母开头，private 的命名空间则以“_”开头，protected 命名空间则以\$开头。

如：

```

public namespace cat;
private namespace _cat;
protected namespace $cat;

```

自定义命名空间用于静态变量，实例变量，静态方法，实例方法。

设置 swf 文件属性

```
[SWF(width="500", height="400", frameRate="60", backgroundColor="#FFFFFF")]
```

```

public 文档类名称 extend Sprite{
}

```

或者在项目上点右键->properties->ActionScript Compiler

在右边

additional compiler arguments

```
-default-size 550 400 -default-frame-rate 12 -default-background-color 0xffffffff
```

取一级汉字首字母原理

1. 参照 GB2312 码表(一级汉字),定义声母区间

2. i, u, v 都不做声母, 跟随前面的字母

```
chartable = { '啊','芭','擦','搭','蛾','发','噶','哈','哈','击','喀','垃','妈','拿','哦','啪','期','然',
'撒','塌','塌','塌','挖','昔','压','匝','座' };

```

```
firstLetter = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V',
'W', 'X', 'Y', 'Z' };

```

3. 把区间汉字转为 GB2312 编码

4. 通过判断汉字编码所在区间来判断首字母。

基础知识

GB 2312 标准共收录 6763 个汉字，其中一级汉字 3755 个，二级汉字 3008 个。

分区表示

GB 2312 中对所收汉字进行了“分区”处理，每区含有 94 个汉字/符号。这种表示方式也称为区位码。

1) 01-09 区为特殊符号。

2) 16-55 区为一级汉字，按拼音排序。

3) 56-87 区为二级汉字，按部首/笔画排序。

4) 10-15 区及 88-94 区则未有编码。

举例来说，“啊”字是 GB2312 之中的第一个汉字，它的区位码就是 1601。

字节结构

在使用 GB2312 的程序中，通常采用 EUC 储存方法，以便兼容于 ASCII。浏览器编码表上的“GB2312”，通常都是指“EUC-CN”表示法。

每个汉字及符号以两个字节来表示。第一个字节称为“高位字节”（也称“区字节”），第二个字节称为“低位字节”（也称“位字节”）。

“高位字节”使用了 0xA1-0xF7(把 01-87 区的区号加上 0xA0)，“低位字节”使用了 0xA1-0xFE(把 01-94 加上 0xA0)。由于一级汉字从 16 区起始，汉字区的“高位字节”的范围是 0xB0-0xF7，“低位字节”的范围是 0xA1-0xFE，占用的码位是 $72 \times 94 = 6768$ 。其中有 5 个空位是 D7FA-D7FE。

例如“啊”字在大多数程序中，会以两个字节，0xB0（第一个字节） 0xA1（第二个字节）储存。区位码=区字节+位字节（与区位码对比：0xB0=0xA0+16,0xA1=0xA0+1）。

显示对象变恢处理

```
var matrix:Array = new Array();  
matrix = matrix.concat([0.3086,0.6094,0.082,0,0]); // red  
matrix = matrix.concat([0.3086,0.6094,0.082,0,0]); // green  
matrix = matrix.concat([0.3086,0.6094,0.082,0,0]); // blue  
matrix = matrix.concat([0,0,0,1,0]); // alpha  
var vast = new ColorMatrixFilter( matrix );  
mc.filters = [vast];  
//去掉滤镜  
mc.filters = null;
```

Matrix 原理

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

旋转

$$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = Y'x + X'x = x\cos(q) + y\sin(q)$$

$$y' = Y'y + X'y = -x\sin(q) + y\cos(q)$$

矩阵与公式对比可知

$$a = \cos(q)$$

$$b = \sin(q)$$

$$c = -\sin(q)$$

$$d = \cos(q)$$

平移(tx、ty)

$$x' = Y'x + X'x = x\cos(q) + y\sin(q) + tx$$

$$y' = Y'y + X'y = -x\sin(q) + y\cos(q) + ty$$

第三行(0, 0)代表齐次坐标系的原点位置

a = 控制 X 的宽度

b = 控制 Y 的倾斜

c = 控制 X 的倾斜

d = 控制 Y 的高度

tx = 控制 X 坐标位置

ty = 控制 Y 坐标位置

Matrix 的计算公式:

$$X' = a * X + c * Y + tx; \quad // X, Y \text{ 为矩形四个角的}(x, y)\text{坐标} \quad X', Y' \text{ 为得到的新坐标}$$

$$Y' = b * X + d * Y + ty;$$

意思是:

新坐标 $X = a * \text{现有 } X \text{ 坐标} + c * \text{现有 } Y \text{ 坐标} + \text{常量 } tx$;

新坐标 $Y = b * \text{现有 } X \text{ 坐标} + d * \text{现有 } Y \text{ 坐标} + \text{常量 } ty$;

注意: Matrix 必须配合 Transform 才能实现所定义的效果

二维坐标 $p(x, y)$ 的齐次坐标表示为 $p(x, y, 1)$; 1 为比例因子(即(8,4,2)和(4,2,1)表示的都是二维点(2,1)), Matrix 的计算原理就是用齐次坐标矩阵乘以点的齐次坐标

矩阵乘法规则:

把前面矩阵的第 i 行与后面矩阵的第 j 列对应元素相乘再相加, 放到结果矩阵的第 (i,j) 这个位置上。

2D 旋转数学公式:

$$x' = Y'x + X'x = y\sin(\alpha) + x\cos(\alpha)$$

$$y' = Y'y + X'y = y\cos(\alpha) - x\sin(\alpha)$$

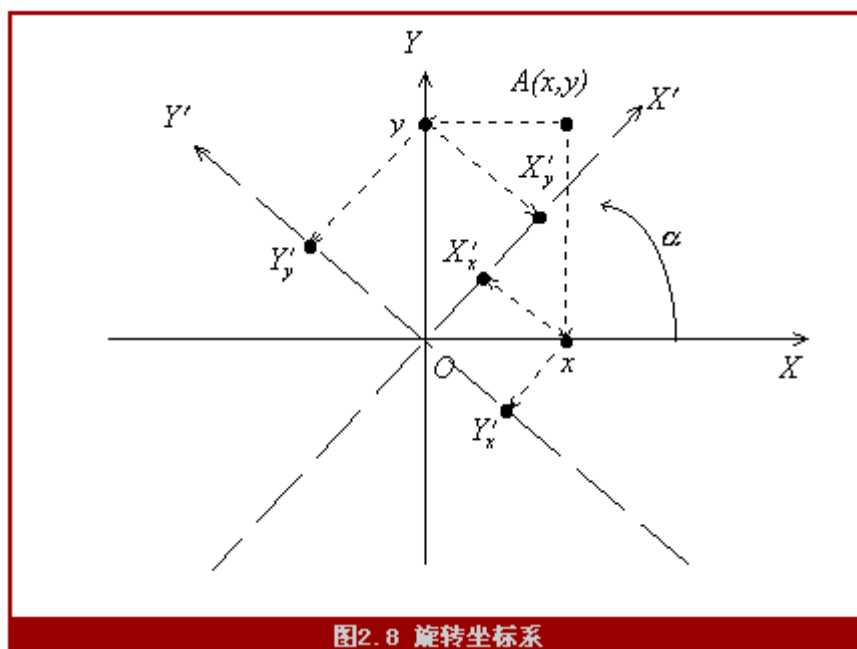


图2.8 旋转坐标系

余弦差公式: $\cos(A-B)=\cos A \cos B + \sin A \sin B$

向量法证明:

取直角坐标系, 作单位圆

取一点 A, 连接 OA, 与 X 轴的夹角为 A

取一点 B, 连接 OB, 与 X 轴的夹角为 B

OA 与 OB 的夹角即为 A-B

A (cosA, sinA), B(cosB, sinB)

OA=(cosA, sinA)

OB=(cosB, sinB)

OA*OB

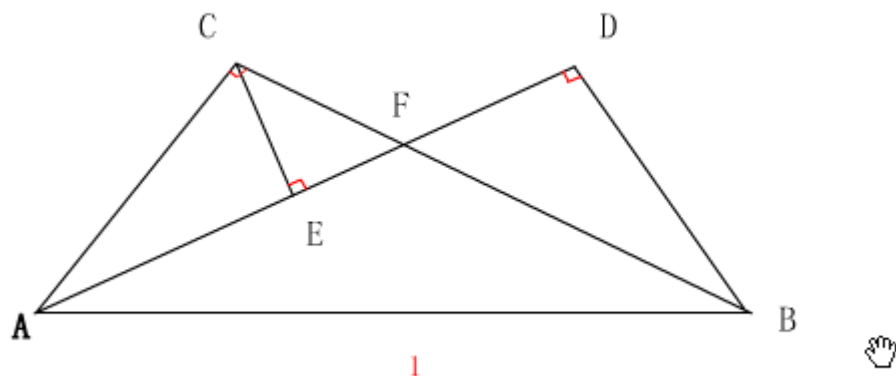
=|OA||OB|cos(A-B)

=cosAcosB+sinAsinB

|OA|=|OB|=1

cos(A-B)=cosAcosB+sinAsinB

证明两角和的余弦公式: $\cos(\alpha - \beta) = \cos\alpha \cos\beta - \sin\alpha \sin\beta$



证明: 构造如图所示图形: 其中 $\angle CAB$ 为直角, $\angle ADB$ 为直角, $CE \perp AD$, 取 $AB = 1$ 。

设 $\angle CAB = \alpha$, $\angle CAD = \beta$, 那 $\angle DAB = \alpha - \beta$

$$\cos(\alpha - \beta) = \cos \angle DAB = \frac{AD}{AB} = AD$$

$$\begin{cases} \cos \alpha = \cos \angle CAB = \frac{AC}{AB} = AC \\ \cos \beta = \cos \angle CAD = \frac{AE}{AC} \end{cases} \Rightarrow \cos \alpha \cos \beta = AC \frac{AE}{AC} = AE$$

$$\begin{cases} \sin \alpha = \sin \angle CAB = \frac{CB}{AB} = CB \\ \sin \beta = \sin \angle CAD = \frac{CE}{AC} \end{cases} \Rightarrow \sin \alpha \sin \beta = CB \frac{CE}{AC}$$

现在只要证明 $AD = AE + CB \frac{CE}{AC}$ 即可, 即证明 $CB \frac{CE}{AC} = ED$ 即可。

又由三角形相似, 可知: $\frac{CE}{AC} = \frac{EF}{CF} = \frac{FD}{FB}$, 故 $\frac{CE}{AC} = \frac{EF+FD}{CF+FB} = \frac{ED}{CB}$, 即 $CB \frac{CE}{AC} = ED$, 证毕。

上面用到了等比公式

$$a/b=c/d=e/f=\dots=m/n=q$$

如果满足

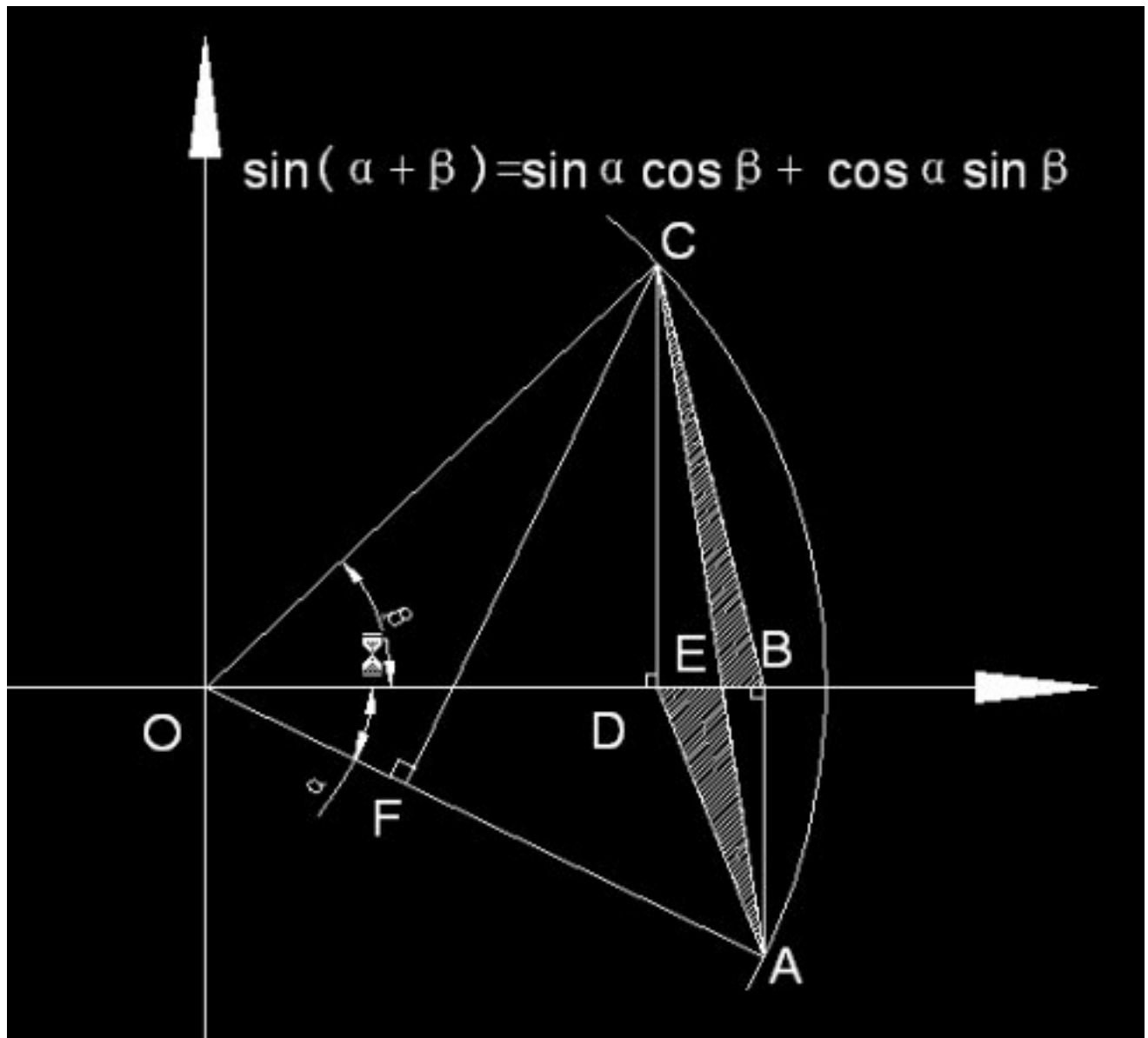
$$b+d+f+\dots+n \neq 0$$

则

$$(a+c+e+\dots+m)/(b+d+f+\dots+n) = q$$

正弦和公式: $\sin(A+B) = \sin A \cos B + \cos A \sin B$

单位圆证明:



$$\sin(\alpha + \beta) = CF; \sin \alpha = AB; \cos \alpha = OB; \sin \beta = CD; \cos \beta = OD$$

$$S_{\triangle OCA} = \frac{1}{2} \times 1 \times CF = \frac{1}{2} \times 1 \times \sin(\alpha + \beta) \quad \leftarrow$$

$$\begin{aligned} S_{\triangle OCE} &= S_{\triangle OCB} - S_{\triangle CEB} \quad \leftarrow \\ &= \frac{1}{2} \times OB \times CD - S_{\triangle CEB} \quad \leftarrow \\ &= \frac{1}{2} \times \cos \alpha \times \sin \beta - S_{\triangle CEB} \quad \leftarrow \end{aligned}$$

$$\begin{aligned} S_{\triangle OAE} &= S_{\triangle OAD} + S_{\triangle AED} \quad \leftarrow \\ &= \frac{1}{2} \times OD \times AB + S_{\triangle AED} \quad \leftarrow \\ &= \frac{1}{2} \times \cos \beta \times \sin \alpha + S_{\triangle AED} \quad \leftarrow \end{aligned}$$

于是：↵

$$\begin{aligned} S_{\triangle OCA} &= \frac{1}{2} \times 1 \times \sin(\alpha + \beta) = S_{\triangle OCE} + S_{\triangle OAE} \quad \leftarrow \\ &= \left(\frac{1}{2} \times \cos \alpha \times \sin \beta - S_{\triangle CEB} \right) + \left(\frac{1}{2} \times \cos \beta \times \sin \alpha + S_{\triangle AED} \right) \quad \dots\dots\dots \textcircled{1} \quad \leftarrow \end{aligned}$$

到了这里，我们再来看看 $S_{\triangle CEB}$ 和 $S_{\triangle AED}$ 有怎样的关系：↵

$$\text{容易看出, } S_{\triangle ABD} = S_{\triangle ABC} = \frac{1}{2} \times AB \times BD \quad \leftarrow$$

而 $S_{\triangle ABE}$ 是公共三角形, $\therefore S_{\triangle AED} = S_{\triangle CEB}$, ↵

于是将①式整理, 便得到: ↵

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta \quad \leftarrow$$

至于↵

$$\sin(\alpha - \beta) = \sin \alpha \cos \beta - \cos \alpha \sin \beta, \quad \leftarrow$$

只要将 $\sin(\alpha + \beta)$ 中的 β 换成 $(-\beta)$ 即可, 或者把 $\sin(\alpha + \beta)$ 的证明过程稍加变化, 也可以用几何方式来证明。↵

getObjectsUnderPoint() 方法 bug

如果你的容器不在 $x=0, y=0$ 的 stage 坐标上时, 就只能用 `stage.getObjectsUnderPoint()` 来获取。

在 AS 工程中遇到个问题, 当元素的 `visible` 设为 `false` 时, 仍然会阻挡鼠标事件。

Flex 开发

自定义事件机制

重要作用: 传递消息

```
package
{
```

```

import flash.events.Event;
// 继承 Event 事件
public class IdManagerEvent extends Event
{
    public var id:String;
    public var user:String;

    public function IdManagerEvent(type:String, user:String, id:String)
    {
        super(type);
        this.id = id;
        this.user = user;
    }

    override public function clone():Event
    {
        return new IdManagerEvent(type, user, id);
    }
}

package
{
    import flash.events.Event;
    import flash.events.EventDispatcher;

    public class DispatcherManager extends EventDispatcher
    {
        // Event 元数据声明事件
        [Event(name="idManagerError", type="Event")]

        public function DispatcherManager (){}

        }
        //TODO::
    }
}

// 注册自定义事件 idManager
var disManager: DispatcherManager = new DispatcherManager();
disManager.addEventListener("idManager", backFun);
function backFun(e:IdManagerEvent):void
{
    trace(e.user); //得到 user
}

```

```

        trace(e.id);    //得到 id
    }
    // 广播自定义事件
    var d: IdManagerEvent = new IdManagerEvent ("idManager", "user", "id");
    disManager.dispatchEvent(d);

```

设置 Tooltip 的样式

```

//Flex Builder 3
StyleManager.getStyleDeclaration("ToolTip").setStyle("fontSize", 12);
StyleManager.getStyleDeclaration("ToolTip").setStyle("backgroundColor", "#33CC99");

//Flash Builder 4
var cssDeclaration:CSSStyleDeclaration =
FlexGlobals.topLevelApplication.styleManager.getStyleDeclaration("mx.controls.Tool
Tip");
cssDeclaration.setStyle('fontSize', 12);
cssDeclaration.setStyle('backgroundColor', "#33CC99");
//FlexGlobals.topLevelApplication.styleManager.setStyleDeclaration("mx.controls.Too
lTip", newStyleDeclaration, true);//运行时安装CSS

```

运行 JS

```
navigateToURL(new URLRequest("javascript:alert(123)"), "_self");
```

解决 mx.controls.TextInput 用了 setFocus() 之后就不能输入中文问题

```

import flash.system.IMC;
textInput.setFocus(); //设置了焦点，如果不加下面两句，TextInput 则不能输入中文
textInput.imeMode = "CHINESE"; //与简体中文 IME 和 繁体中文 IME 配合
IME.enabled = true; //指示系统启动 IME

```

Embed 标签的使用

嵌入图片资源

```
[Embed(source='resource/image/defSeatHead.jpg')]
```

```
public static var DefSeatHead:Class;
```

嵌入 MovieClip

```
[Embed(source='resources/FacePack.swf', symbol='Face00')]
```

```
public static var Face00:Class;
```

嵌入二进制 XML

```
[Embed(source="assets/data/data.xml", mimeType="application/octet-stream")]
```

```
private static const binaryXML:Class;
```

```
/** 还原二进制 XML */
```

```

var xmlData:XML;
var bytes:ByteArray = new binaryXML() as ByteArray
if (bytes)
{
    try
    {
        bytes.uncompress();//如果 data.xml 压缩过，则解压
    }
    catch(e:Error)
    {
    }
    xmlData = XML(bytes);
}

```

隐藏嵌入的资源

```

package
{
    import flash.display.Loader;
    import flash.display.Sprite;

    public class TestHideAsset extends Sprite
    {
        //适合于任何嵌入的资源(swf、图片、声音)
        [Embed(source="images/qign.jpg", mimeType="application/octet-stream")]
        private static const bytes:Class;
        private var loader:Loader;

        public function TestHideAsset()
        {
            loader=new Loader();
            loader.loadBytes(new bytes());
            addChild(loader);
        }
    }
}

```

嵌入 swf 并引用 MovieClip

```

package
{
    import flash.display.MovieClip;
    import flash.display.Sprite;

    public class EmbedSwfTest extends Sprite

```

```

{
    //TestMC是mctest.swf里的一个MovieClip的Linkage Class名
    [Embed(source="assets/mctest.swf", symbol="TestMC")]
    private var TestMC:Class;

    public function EmbedSwfTest()
    {
        var mc:MovieClip = new TestMC();
        mc.gotoAndStop(2);
        addChild(mc);
    }
}

```

对嵌入的图片使用 9 切片放缩技术

```

Embed(source="../assets/msk.gif", scaleGridTop="4", scaleGridLeft="4", scaleGridRight="16",
scaleGridBottom="16");

```

嵌入声音文件

```

[Embed(source='resource/sound/sound_bg.mp3', mimeType='audio/mpeg')]
public static var sound_bg:Class; // as Sound

```

Flash Builder 创建运行时共享库(RSL)

步骤一

新建 Flex 库项目

```

//新建一个类
package org
{
    import flash.display.Sprite;

    /**
     * 共享类
     */
    public class ShareDll extends Sprite
    {
        public function ShareDll()
        {
            init();
        }

        public function init():void
        {
            this.graphics.beginFill(0x000000);

```

```

        this.graphics.drawRect(0,0,200,200);
        this.graphics.endFill();
    }
}
}

```

导出 ShareDll.swc 和 ShareDll.swf

步骤二

建立测试工程 ActionScript 项目

把 ShareDll.swc 添加到库路径, 设置链接类型为外部

步骤三

开始测试

```

package
{
    import flash.display.*;
    import flash.system.*;
    import flash.net.URLRequest;
    import flash.events.Event;
    import org.ShareDll;

    /**
     * 对共享库的测试
     */
    [SWF(backgroundColor='0xFFFFFF', frameRate='12', width='640', height='480')]
    public class RSLTest extends Sprite
    {
        public function RSLTest()
        {
            var rslLoader:Loader = new Loader();
            rslLoader.contentLoaderInfo.addEventListener(Event.INIT, rslInit);
            // 把RSL 中的定义加载到当前应用程序域中
            var context:LoaderContext = new LoaderContext();
            context.applicationDomain = ApplicationDomain.currentDomain;
            var url:String = "ShareDll.swf";
            rslLoader.load(new URLRequest(url), context);
        }

        private function rslInit( e:Event ):void
        {
            // 只有当RSL 中的定义导入到当前应用程序域以后
            // 我们才能用其中的ShareTest 定义通过ShareTest 类的校验
            addChild(new ShareTest());
        }
    }
}

```

AS3 动态添加状态视图(State)

```
private function init():void
{
    var hallChild:AddChild = new AddChild();
    hallChild.target = new TextInput();

    var gamehall:State = new State();
    gamehall.name = "gamehall";
    gamehall.overrides = [hallChild];

    var gamestart:State = new State();
    gamestart.name = "gamestart";
    gamestart.overrides = [];

    var stateArray:Array = new Array();
    stateArray.push(gamehall);
    stateArray.push(gamestart);

    this.states = stateArray;

    this.currentState = "gamehall";
}
```

接收外部 get 参数

```
<object                                classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=9,0,0,0" width="550" height="400" id="Untitled-1" align="middle">
    <param name="allowScriptAccess" value="sameDomain" />
    <param name="allowFullScreen" value="false" />
    <param name="movie" value="Untitled-1.swf?id=xxx&p=kkk" />
    <param name="quality" value="high" />
    <param name="bgcolor" value="#ffffff" />
    <embed src="Untitled-1.swf?id=xxx&p=kkk" quality="high" bgcolor="#ffffff" width="550"
height="400" name="Untitled-1" align="middle" allowScriptAccess="sameDomain"
allowFullScreen="false" type="application/x-shockwave-flash"
pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

AS 代码:

```
var par:Object = this.loaderInfo.parameters;
```



```

txt.text = '接收参数:';
//遍历所有 名称-值 参数
for( var i:String in par )
{
    txt.appendText( i +'=' + par[i] + '&' );
}

```

AS3 中把属性设为 null 即可删除属性， 如果用 delete att 只能删除动态属性。

trace(k in array); 如果 array 中存在索引 k 则返回 true

trace(k in obj);

AVM 将用===来比较 case 分支

函数传入的参数都被保留在 arguments 对象中， 它有一个 arguments.length 属性和一个指向本函数的 arguments.callee() 利用 arguments.callee(参数) 可创建递归

function test(...rest){ //rest 只是推荐命名， 可以换成其它的命名， 一旦用了 rest 就
//不能用 arguments 了

```

    trace(rest[0]);
}

```

```

Function test( a:int, ...rest){

```

```

}

```

AS3 中函数本身是 Function 对象，基本数据类型也是 Object 对象。

给函数添加动态属性

```

var test:Function = function(){
}

```

```

test['t'] = 5;

```

AS3 实例属性和方法的另一种调用方式(数组运算符调用)

instance['属性名']

instance['方法名']

for...in 可以遍历动态类的动态属性和值

动态类中，在运行时加入的方法只能访问 public 成员

包外类：在包中写的其他类，不能与文件名同名，并且这些类对外不可见。

类的访问控制符只有 internal(默认)和 public

命名空间是设计优秀 OOP 架构和管理大型项目的利器。

自定义命名空间

```

package org
{
    import flash.display.Sprite;

    public class SamplePlayer extends Sprite
    {
        use namespace ball; //打开命名空间 ball
        var foo:Player = new Player();
        foo.da(); //输出： 打():我要玩一下球
    }
}

```

```

        //下面两行明确指定使用哪一个命名空间中的方法
        foo.phone::da(); //输出: 打():我要拨个电话    :: 命名限定符
        foo.ball::da(); //输出: 打():我要玩一下球
    }
}
//定义命名空间
internal namespace phone;
internal namespace ball;

class Player
{
    //应用命名空间 phone
    phone function da() {
        trace("打():我要拨个电话");
    }
    //应用命名空间 ball
    ball function da() {
        trace("打():我要玩一下球");
    }
    //自定义空间方法可以重名
    public function da() {

    }
}
}

```

自定义命名空间不能用于类上, 类只有两个访问控制符 `internal` (默认)、`public` 使用 `user namespace xxx` 一次则对当前作用域开放, 意味着当前作用域都只能使用此命名空间, 因为 `user namespace` 只能打开没有关闭。

AS3 的 `Namespace` 和 `Qname` 两个类实现了 XML 的命名空间
命名空间定义在独立的 `as` 文件中有助于其他包中类使用

```

package mx.core{
    public namespace mx_internal = "http://www.adobe.com/2006/flex/mx/internal";
}

```

命名空间可以嵌套使用, 但很少会有这种设计。

如果要应用的命名空间和当前类不在同一个包中, 那么还需要先导入

```
import mx.core.mx_internal
```

出了导入这个命名空间外还要在定义类之前 `user namespace mx_internal;` (初始化命名空间)

命名空间定义在类中

```

class Foo{

    public namespace good;
}

```

```

public namespace bad;

public function getName():Namespace{
    return good;
}
good function hello():void
{

}
bad function hello():void
{

}
}

```

外部访问 `Foo.good` `Foo.bad`; 因为命名空间不依附实例。

自定义命名空间（如 `use namespace good`）会被编译器提前。

如果同时打开多个命名空间，又存在重名，编译器会报“引用模糊”错

```

var a:int = 3;
if(a<5){
    use namespace good;
}else{
    use namespace bak;
}

```

等同于

```

use namespace good;
use namespace bak;
var a:int = 3;
if(a<5){
}else{
}

```

说明 `use namespace` 会被编译器自动提前

命名空间可以用于：

1. 防止命名冲突
2. 更灵活的访问控制
3. 实现不公开的 API
4. 实现程序的多版本控制(比如 `debug` 模式和正常模式切换或不同语言版本的切换)

AS3 中只能重写实例方法，不能重写实例属性。由于静态成员不被继承，可以直接在子类中

定义同名成员达到重写。重写方法不可改变参数类型和数目以及访问控制和返回值类型。

AS3 动态创建 CSS 并安装使用

```
var toggleButtonBar:ToggleBarButton = new ToggleBarButton();
toggleButtonBar.direction = 'vertical';
toggleButtonBar.dataProvider = [{label: '预览设置'}];
toggleButtonBar.setStyle('color', '#333333');
var firstButtonStyle:CSSStyleDeclaration = new
CSSStyleDeclaration('firstButtonStyle');
firstButtonStyle.setStyle('cornerRadius', 0);

StyleManager.setStyleDeclaration('.toggleButtonBarFirstButton',
firstButtonStyle, true);

var lastButtonStyle:CSSStyleDeclaration = new
CSSStyleDeclaration("lastButtonStyle");
lastButtonStyle.setStyle('cornerRadius', 0);

StyleManager.setStyleDeclaration(".toggleButtonBarLastButton",
lastButtonStyle, true);

toggleButtonBar.setStyle('firstButtonStyleName',
'toggleButtonBarFirstButton');
toggleButtonBar.setStyle("lastButtonStyleName",
"toggleButtonBarLastButton");
```

原型继承原理

```
//需要注意的是如果是非动态类那么为类原型添加的属性会成为一个只读属性
Sprite.prototype.copyright="空";

var sp=new Sprite;
var mc=new MovieClip;

trace(sp.copyright);    //输出 "空"
trace(mc.copyright);    //输出 "空", 由于 MovieClip 继承自 Sprite, 所以也能得到属性

sp.copyright="www.iflashigame.com";    //报错了, 无法为 flash.display.Sprite
创建属性 copyright。
mc.copyright="www.iflashigame.com";    //新的值将覆盖继承过来的

trace(sp.copyright);    //无法赋值, 还是输出 "空";
trace(mc.copyright);    //输出"www.iflashigame.com";
trace(Sprite.prototype.copyright);    //没有改变, 还是输出 "空"
```

AIR 创建目录时报安全错误(SecurityError: Error #2060)解决方案

```
var dbDir:File = new
File(File.applicationDirectory.resolvePath("db").nativePath);
if( !dbDir.exists ){
    dbDir.createDirectory();
}
```

Flex Builder 编译错误 An internal build error has occurred 的解释

在 Flex Builder 调试 ActoinScript 项目时,偶尔会出现这种奇怪错误。原因主要是快速编译造成的

查看日志: Help > Product Details. 点击 Configuration Details 按钮, 点击 View Error Log 按钮.

解决方法: 重起 Flex Builder

Flex Builder 建 AS 项目注意:

必需在文档类上加

```
[SWF(backgroundClor='0x222222',frameRate='12',width='1000',height='550')]
```

否则会出现舞台内容对齐错乱, 放缩错乱等。

为 TextField 的 htmlText 的my link</>绑定事件

```
var style:StyleSheet = new StyleSheet();
    style.parseCSS("a:link{color:#00FF00}a:hover{color: #FF0000} a:active{color:
#FF0000} a:visited{color: #00FF00}");

var txt:TextField = new TextField();
txt.styleSheet = style;
txt.htmlText = "<a href='event:myevent'>event</a>";
txt.addEventListener(TextEvent.LINK, linkHandler);
addChild(txt);
function linkHandler(e:TextEvent):void
{
    trace(e.text); //myevent
}
```

ABC 文件

手动把*.as 文件编译成*.abc 文件:

在 Flash sdk 的 lib 目录下, 有个 asc.jar, 该文件就是用来生成 abc 文件的 java 程序, 机器上要安装 java 运行时。

在 asc 源码包下有个 abc 文件, 里面有 builtin.abc, playerglobal.abc, toplevel.abc

三个文件，他们就是 flash 的基本库

```
java -jar asc.jar -import E:\work\sdksrc\modules\asc\abc\builtin.abc -import  
E:\work\sdksrc\modules\asc\abc\playerglobal.abc Hello.as
```

优化篇

1. 合理使用 Shape 与 Sprite, MovieClip, 你可能用 MovieClip 可以完成 Sprite 与 Shape 的功能, 但是他们所需的内存是不一样的 Shape 需要 236 字节, Sprite 需要 412 字节, MovieClip 需要 440 字节, 如果你只想显示图形没有交互那你使用 Shape, 如果是有交互的图形你可以用 Sprite, 如果是动画你才用 MovieClip
2. 使用事件模型通信与使用传统的回调函数相比, 速度更慢且占用的内存更多 AS3 中事件的派发传递参数是采用 Event, 所以在高频率派发事件的地方你可以采用传统的回调函数这样可以大大提高你的效率和内存消耗
3. C 代码用 alchemy 编译成 swc, as3 调用 swc 接口。

[心得] FLASH 内存优化 13 条

转自: <http://bbs.9ria.com/thread-67958-1-1.html>

1. 使用合适的显示对象, 对于非交互的简单形状用 Shape 对象, 对于不需要时间轴的交互式对象用 Sprite, 对于使用时间轴的动画用 MovieClip, 他们的内存使用量分别是 236, 412, 440, 可见 shape 很省内存
2. Number 原始存储内存占 8 个字节, int, uint, Boolean, String 均占 4 个字节, 关于赋值后作占内存, 取决值赋的值
3. 对象的重复利用, 在 FOR 循环内 new 对象要小心, 每次 new 都会增大内存。
4. 通过重复使用 BitmapData 对象可以节省更多的内存, 即不同的 bitmap 公用一个 bitmapdata
5. 重复使用相同对象的时候, 可以考虑下对象池的应用
6. 垃圾回收运行时不断检测出处于非活动状态的对象, 大型项目中此进程会占用大量 CPU, 所以尽量重用对象, 不用的对象设置为 NULL
7. 内存释放方面, 为了确保被垃圾回收, 首先保证该对象没有其他引用, 然后移除监听, 在然后设置为 NULL。关于 bitmapdata, 先用 dispose 在设置为 null
8. 在 Flash Player 10.1 中, 对所有 bitmapdata 实例采用同一版本的 bitmapdata, 大大

节省了内存，当 bitmapdata 数据发生变化的时候，内存中会建立一个新的 bitmapdata 实例

9. 尽量避免使用滤镜，当对显示对象使用滤镜的时候，内存中将创建两个位图，每个位图的大小都与显示对象相同。一个是显示对象的栅格化版本，一个是用于滤镜的位图

10. 当修改滤镜的属性的时候，内存中两个位图都将更新创建新的位图，会消耗一些 CPU，并且会占用大量内存。

11. Flash Player 10.1 在所有平台上引入了一种新的过滤行为。如果滤镜在 30 秒内没有进行修改，或者将其隐藏或置于屏幕之外，将释放未过滤的位图占用的内存。该方式成为动态位图卸载。

12. 使用滤镜时仍要谨慎小心；在对它们进行修改时，仍要求大量 CPU 或 GPU 处理。

13. 如果想要滤镜效果，最好用 PHOTOSHOP 来做一个

发布时忽略 trace

`-compiler.omit-trace-statements`

P2P 开发

Web 开发配置:

- 1) flash player10.1 (fp10.0 不支持组播)
- 2) playerglobal.swc (fp10.1 开发包)

RIA 开发配置:

- 1) Flex SDK 3.2+ or Flex Build 3.0.2
- 2) Runtime 2.0


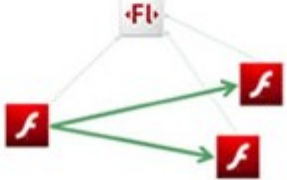

采用的协议:

实时媒体流协议(RTMFP)

开步骤:

- 1) 从 Adobe 申请一个 developer key
如: d5957219e16ad29212214c60-9d7ea825dde5 (此 key 可用)

Stratus 2 新特性

The Evolution of Media and Communication Delivery on the Flash Platform		
Traditional Streaming / Communication with Unicast model	RTMFP in Flash player 10.0 / Stratus 1.0	RTMFP in Flash player 10.1 / Stratus 2.0
		
Traditional streaming requires a client to receive all data from a centralized server cluster. Scale is achieved by adding more servers	First generation of RTMFP in Flash player 10.0 supported rendezvous. Media was always sourced from the publishing peer.	Second generation of RTMFP supporting groups in Flash player 10.1 supports application-level multicast and reduces the load on the source publisher.

- 1) 1.RTMFP 群组(Groups)
- 2) 2. 覆盖网络(Overlay network) — 允许客户不使用网络探测就能轻易的找到对方。
- 3) 3. 支持应用程序级别的实时多播 (Multicast)— 允许发布者发送一个实时媒体流并且按需要尽可能的扩大这个流的范围。应用程序级别上的多播减少了在一个单个发布者上的加载并且通过多个 peers 分发传递。这是专门为在 peers 之间有大量广播信息(即高比特率数据)的小群组设计的功能。
- 4) 4.支持定向路由(Directed Routing) — 允许开发者创建一个应用程序间的连接并且发送数据给群组中特定的 peer。
- 5) 5. 支持群发(Posting) — 允许开发者广播数据给群组里所有的 peer, 但是缺乏可靠性和命令交互。这对 IM、游戏、天气预报站更新等等数据广播应用程序是非常有用的。这是专门为大量的 peers 传输小数据量(即低比特率数据)的信息设计的功能。
- 6) 6.支持对象复制(Object Replication) — 允许开发者在 pees 之间创建一个类似工作空间复制、共享白板、共享文件的应用程序。对象复制允许可靠和有序的数据通过 RTMFP 组来进行传输。
- 7) 注意: 多播和广播(群发)的区别: 广播向每一个目的地投递一个分组的拷贝, 可以通过多个单次分组的投递完成, 也可以通过单独的连接传递分组的拷贝, 直到每个接收方均收到一个拷贝为止。而就多播而言, 当某一个人向一组人发送数据时, 它不必将数据向每一个人都发送数据, 只需将数据发送到一个特定的预约的群组地址, 所有加入该组的人均可以收到这份数据。这样对发送者而言, 数据只需发送一次就可以发送到所有接收者, 大大减轻了网络的负载和发送者的负担。
- 8) 现在我们来仔细看看 RTMFP Groups:
- 9) RTMFP groups 让开发者创建互动的交际体验, 可以用来提升用户在 web 程序的时间或者让用户在公司网络享受更高质量的媒体体验。互动应用程序像视频聊天、语音通话、文字聊天可以内建到一些解决方案里面, 比如在线帮助、交友网站、公司通讯、销售、广告。这些都可以用 RTMFP。
- 10) RTMFP 可以有利于显著地减少部署和带宽的成本, 因为它不象传统的 B/S 解决方案要求非常多的带宽和服务器。Stratus 允许 flash 客户端建立 RTMFP 连接。它允许开发者建立一个很容易在群组里找到其他人但是避免破坏性网络探测的工程。
- 11) RTMFP groups 将实现新的沟通包括应用程序级别的多播如可扩展音频, 视频和数据分发。除此之外, 新的群发(posting)功能和直接路由功能将允许任何客户端参加到一个群组来向群组里其他所有用户或者单个用户发送信息。另一个引人注目的特点就是作为 RTMFP 特征之一的对象复制。对象复制允许一个 RTMFP 群组的所有成员获得一组对象(假定非常巨大)的持续视图。这种模式充分利用了群组自身

的组织结构从具有群组里某对象的节点到另一个需要这个对象的节点的复制的传递连通性。

- 12) RTMFP groups 比原来的集群服务更有效率。RTMFP groups 允许开发者扩大应用程序给全世界成万上亿的人访问因为所有的资源来源不是单一的 peer 或者服务器。
- 13) RTMFP 通信是基于 UDP 的。它总是加密的, 并且可以穿越 NAT 和防火墙。UDP 十分重要因为它支持数据传输损耗——对于低潜在的音频视频通信非常有用。RTMFP 是一个管理和控制协议, 它需要一个服务器(如 stratus) 始终存在。如果没有服务器, 就不会有 p2p 的通信。服务器用来消除其他网络探测, 只提供必要的信息来建立连接到另一个 peer。访问同样被一个共享机密控制而不会被窃听。RTMFP 有独特的移动 IP 功能, 如果客户端改变网络(即无线和移动站), 任然可以保持连接。

Flash Socket

FP socket 与 server 联机验证过程

1. FP 与 server 建立起联接并向 server 发送策略请求
2. FP 断开 socket, 并分析策略文件
3. 如果策略设置允许访问 server, 则再次与 server 建立起 socket 连接,但这次不向 server 发出任何数据.

注意: 防火墙会阻止跨域(如果 swf 与 server 不在同一台电脑上, 一定要把防火墙关了才能成功跨域)

writeUTF()写入限制

如果写入 writeUTF("aaa"); 实际打印出来的字节序列是 0、3、97、97、97 前两个字节(0、3)表示写入的字符数目。也就是说 writeUTF()函数每次只能写入长度不超过 65535 的字符串。否则就会报错(RangeError: Error #2006: 提供的索引超出范围。)

Flex 项目管理(SVN)

FlexBuilder3 项目管理(SVN)

- 1.查看有没安装 Subclipse: Help->software updates->Manage Configuration
- 2.如果没安装 Subclipse,则安装:

Help->software updates->Find and Install...

url: http://subclipse.tigris.org/update_1.6.x/

FlexBuilder4 项目管理(SVN)

帮助->安装新软件

输入 http://subclipse.tigris.org/update_1.6.x

回车

下载完后重启 FB4

窗口->其它视图->SVN

就可以看到 svn 插件功能了



注意千万不要安装 Subversion JavaHL Native Library Adapter (Required) 否则提交代码时会报如下错误:
 org.tigris.subversion.javahl.ClientException: Couldn't perform atomic initialization
 svn: Commit failed (details follow):
 svn: Couldn't perform atomic initialization

解决 FlashBuilder4 注册码失效问题

找到 C:\Windows\System32\Drivers\etc\ 目录下的 hosts 文件，
 用记事本打开，在末尾加上：
 127.0.0.1 activate.adobe.com
 127.0.0.1 practivate.adobe.com
 127.0.0.1 ereg.adobe.com
 127.0.0.1 activate.wip3.adobe.com

127.0.0.1 wip3.adobe.com
127.0.0.1 3dns-3.adobe.com
127.0.0.1 3dns-2.adobe.com
127.0.0.1 adobe-dns.adobe.com
127.0.0.1 adobe-dns-2.adobe.com
127.0.0.1 adobe-dns-3.adobe.com
127.0.0.1 ereg.wip3.adobe.com
127.0.0.1 activate-sea.adobe.com
127.0.0.1 wwis-dubc1-vip60.adobe.com
127.0.0.1 activate-sjc0.adobe.com

AMF

Amfphp1.9(PHP 5.3 以上有问题，建议用 PHP 2.x 系列)

amfphp 修改编码

gateway.php 找到

```
$gateway->setCharsetHandler("mbstring", "gb2312", "gb2312");
```

```
$gateway->setCharsetHandler("mbstring", "utf-8", "utf-8");
```

如果在 **Flash IDE** 中运行报错(**netStatus.BabVision**)

```
if(PRODUCTION_SERVER)
```

```
{
```

```
    //Disable profiling, remote tracing, and service browser
```

```
    //$gateway->disableDebug(); //注释掉
```

```
    // Keep the Flash/Flex IDE player from connecting to the gateway. Used for security to stop
```

remote connections.

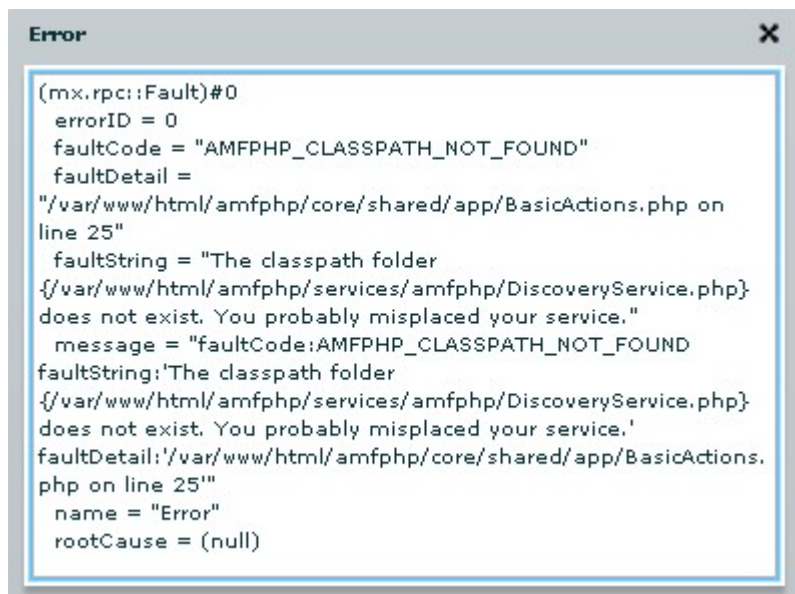
```
    //$gateway->disableStandalonePlayer(); //注释掉
```

```
}
```

在输入以下后台管理时

<http://127.0.0.1/amfphp/browser/>

如果后台报错



原因: `services/`下的 `amfphp` 目录被删了。

注意:

```
var amfNet:NetConnection = new NetConnection();
amfNet.addEventListener(NetStatusEvent.NET_STATUS, onNetStatus);
amfNet.connect(gatewayUrl);//不管连接成功与否, onNetStatus 监听函数是不会触发的。
```

Away3D

Away3D 3.6 在线文档 http://away3d.com/livedocs/3.6.0_lib/index.html

Flash CS3 只支持 Away3D 2.5.0

从 CS4 开始支持在文档类上面加

```
[SWF(width="500", height="4000", frameRate="60", backgroundColor="#000000")]
```

来设置输出文件属性。

要在文档类中使用 `stage.scaleMode = StageScaleMode.NO_SCALE`;必须先导入:

```
import flash.display.Stage;
import flash.display.StageScaleMode;
```

在大多数 3D 软件中, 3D 基本元素是构成 3D 世界的基石。Away3D 自带有 17 种 3D 基本元素以及一些别的帮助物。

三角形是 3D 基本元素中最小单元

3D 基本元素: `Triangle`(三角形)、`Plane`(平面)、`Cube`(立方体)、`Trident`(三叉戟)、`RegularPolygon`(规则多边形)、`Sphere`(球)、`Cone`(圆锥)、`Cylinder`(圆柱体)、`RoundedCube`(圆角立方体)、`GeodesicSphere`、`SeaTurtle`、`GridPlane`(网格面板)、`Torus`(圆环)、`LineSegment`(线段)、`Wire primitives`

创建三角形

```
var tri:Triangle = new Triangle();
tri.a = new Vertex(0,200,0);
tri.b = new Vertex(100,0,0);
tri.c = new Vertex(-100,0,0);
tri.bothsides = true; //不设这个, 则只能看见三角形的一面
view.scene.addChild(tri);
```

创建平面

```
var myPlane:Plane = new Plane({material:"white#black",rotationX:-90});
View.scene.addChild(myPlane);
或
var mat:WireColorMaterial = new WireColorMaterial();
```

```

mat.color = 0xff0000;
mat.wirecolor = 0x000000;
var myPlane:Plane = new Plane();
myPlane.material = mat;
myPlane.rotationX = -90;
View.scene.addChild(myPlane);
增加平面里三角形密度
myPlane.segmentsW = 4;
myPlane.segmentsH = 6;

```

创建立方体

```

cube = new Cube({width:200,height:100,depth:300});
你也可以写成更具可读性:
var cube:Cube = new Cube();
cube.width = 200;
cube.height = 100;
cube.depth = 300;
使用 cubeMaterials 属性设置六个面 (top, bottom, left, right, front and back) 的材质, 就像下面代码那样:
cube.cubeMaterials.left = new ColorMaterial(0xffffff,{alpha:.3});
每个 3D 物体都可以沿 x/y/z 轴进行一些移动、旋转等操作, 如下:
cube.x = 200;cube.rotationY = 45;

```

三叉戟

```

var axis:Trident = new Trident();
view.scene.addChild(axis);
三叉戟有两个可选参数, 轴的长度与是否显示轴代表符号 (x/y/z)。
var axis:Trident = new Trident(200,true);
上面的代码生成一个新的三叉戟, 其三根轴长为 200 单位且每根轴都显示代表符号。注意, 三叉戟在创建出它之后再调整其属性 (length/letters), 这点有别于其他 3D 基本元素。

```

规则多边形

立方体最少可用 12 个三角形就能显示出来。

规则多边形也被称为圆盘。与平面一样它没有厚度。创建时, 你只能指定边数与圆盘半径。

```

polygon = new RegularPolygon({radius:200,sides:3});
view.scene.addChild(polygon);

```

边数可以设为 2 以上的任何整数。看下面例子体验相关设置。

如果多边形的纹理出现失真情况, 你可以增加 subdivision 值:

```

polygon.subdivision = 3;

```

这会将每个三角形细分成 3 个。

球

```

var sphere:Sphere = new Sphere({radius:50,segmentsW:10,segmentsH:10});
当然也可以这样设置:

```

```
var sphere:Sphere = new Sphere();
```

```
sphere.radius = 50;
```

```
sphere.segmentsW = 10;
```

```
sphere.segmentsH = 10;
```

你可以设置 `sphere.yUp` 属性改变球的倾向。

```
sphere.yUp = false;
```

这样设置让球的顶部朝向你。好了，现在你可以创建一个球，那么球的内部是什么呢？在球体内放一个“旋转”摄像机，我们就可以看到球的内部了。你可以以这样的方式创建全景图。为了能看到球的内部我们还要用到一个方法 `invertFaces`。

```
sphere.invertFaces(); //这会反转纹理。
```

记住要要让 3D 物体内外都可以看到，你必须设置 `bothSides` 属性：

```
sphere.bothsides = true;
```

做全景图

Sphere Skybox Skybox6

`Skybox` 和 `skybox6` 都用来做全景图。它们的不同是，`skybox` 需要我们指定 6 张图片，而 `skybox6` 只要一张 3×2 的图片。

```
var mat:BitmapMaterial = new BitmapMaterial( (new texture() as Bitmap).bitmapData );
```

```
largeCube = new Skybox6(mat);
```

如果你用普通的 `skybox`，你要创建 6 个不同材质：

```
largeCube = new
```

```
Skybox(frontMaterial,leftMaterial,backMaterial,rightMaterial,upMaterial,downMaterial);
```

```
largeCube.quarterFaces(); //细分 skybox 各面
```

细分 `skybox` 各面，只有这样做，当旋转纹理里时才不会出现“史前古物”。这方法同样适合其他 3D 元素和导入的模型。x 做全景图用 `skybox` 相对于用球有一个好处，`skybox` 用到的“三角形”更少。当然这是有“代价”的，这会让全景图有一点扭曲变形，但大多数情况下这点扭曲是可以忽略的。

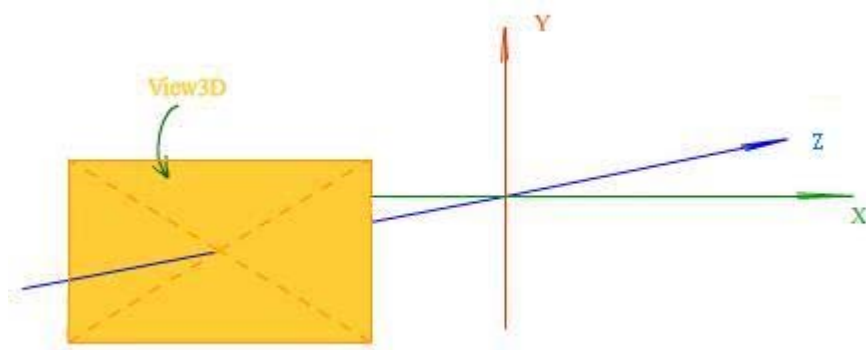
圆锥

圆锥用处不大。做冰淇淋甜筒、锥形交通标和宴会帽等效果时可能会用到。圆锥有 `radius`、`segmentsH`、`segmentsW` 和 `height` 属性,另外一个属性用于指明是否开口，如果设为开口，我们就可以从圆锥底看到圆锥内部。

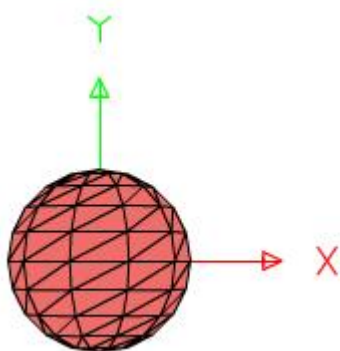
圆柱体

圆柱体与圆锥提供的属性大多相同

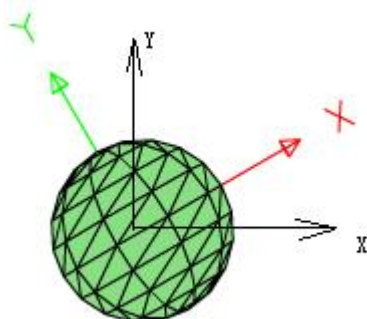
世界坐标



自身坐标



当 3D 元素被移动或转动时，其自身坐标也跟着移动和转动。



```
Sphere.rotationZ=30;
```

照相机

Away3D 提供了三种类型的相机: Camera3D(自由相机), TargetCamera3D(目标相机) 和 HoverCamera3D(旋转相机). 每种照相机都可以进行 zoom(缩放) focus(焦距)、depth of field(景深)、pan (y 轴旋转)、tilt (x 轴旋转)、position (机位) 的设置。三类相机都可以直接在构造函数里对这些特性进行设置, 就象这样:

```
var cam: Camera3D = new Camera3D({zoom:5, focus:200});
```

当然也可以这样设置属性:

```
var cam: Camera3D = new Camera3D({zoom:5, focus:200});
```

```
cam.zoom = 5;
```

Flex Builder3 帮助文档汉化说明

1. 从官方下载中文 html 页面
2. 在 FB3 安装目录下找到 com.adobe.flexbuilder.help_3.0.214193 目录
3. 找到 doc.zip 文件里的 html 文件夹与中文 langref 文件夹一起重新打包成 doc.zip

让 DIV 可以置于 flash 之上

```
<param name="wmode" value="Opaque"> <!-- 让 DIV 可以置于 flash 之上-->
```

强制 IE8 用 IE7 兼容模式解析网页

```
<meta http-equiv="X-UA-Compatible" content="IE=7" />
```

解决跨域访问 swf

*** 安全沙箱冲突 ***

SecurityDomain' http://192.168.1.101/citygame/main/skin.swf' 尝试访问不兼容的内容
' file:///E:/svn102/flashPro/citygame/bin-debug/Preloading.html'

//解决方案

```
var domain:Array = url.match(/http:\/\/\.[^\./]+/);  
if(null != domain) {  
    Security.allowDomain(domain[0]);  
}
```

网页游戏开发须知

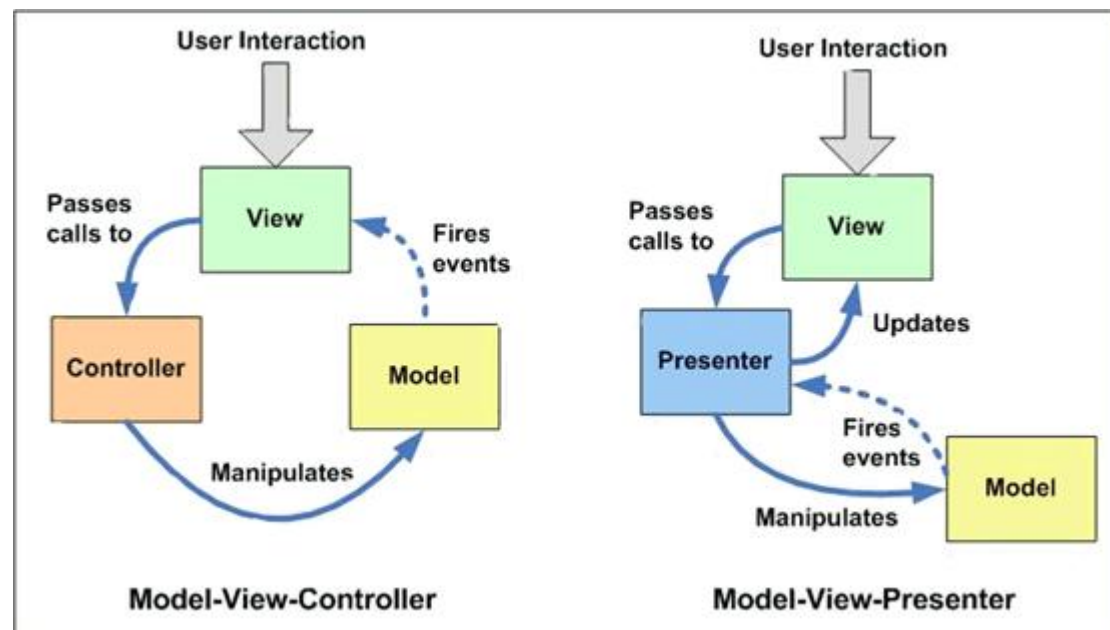
1. 游戏是一种非线性的定向的受限制的体验
2. 好游戏 = 易于上手，难于精通
3. 玩家的游戏消耗，当游戏可消耗的资源被玩家消耗完的时候，游戏就没吸引力了。
4. 游戏钩子，为了钩住玩家的一些小便宜。
5. 网页游戏的玩家忠诚度低
6. 网页游戏更换方便。一个链接就把你的玩家带走了。
7. 网页游戏的速度是关键，在加载你的游戏的时间越久，玩家流失越大。任何加载请在玩家能忍受的时间内完成。
8. 网页游戏容易被复制
9. 进行开发之前，请研究研究现有的游戏。写份游戏研究报告。
10. 登陆的设计界面很重要，不然花做广告招来的用户就被登陆页面给秒杀了。
11. 注册页面更重要，页面尽量小，注册尽量简单。
12. 别以为游戏上线了 就等着赚钱。 其实才开始烧钱。IDC CDN 广告 公关 合作 硬件 设备 人员的扩展 ... 都是大把大把花钱。

游戏开发要留意几个问题

他们是如何使游戏变得有趣而使操作变得简单的？倘若换了自己会怎么做？

他们是如何改进的？什么东西使你在玩了 50 个小时之后依然不对它产生厌烦？

MVC & MVP 设计模式



MVP 通过 Adapter 来分离 View 与 Model 的耦合

Presenter 可以直接操作 Model 与 Adapter

所有业务逻辑在 Presenter 中实现

Adapter:

对象适配器模式

- 在这种适配器模式中，适配器容纳一个它包裹的类的实例。在这种情况下，适配器调用被包裹对象的物理实体。

类适配器模式

- 这种适配器模式下，适配器继承自己实现的类（一般多重继承）。

MVP 的缺点

由于对视图的渲染放在了 Presenter 中，所以视图和 Presenter 的交互会过于频繁。还有一点需要明白，如果 Presenter 过多地渲染了视图，往往会使得它与特定的视图的联系过于紧密。一旦视图需要变更，那么 Presenter 也需要变更了。比如说，原本用来呈现 Html 的 Presenter 现在也需要用于呈现 PDF 了，那么视图很有可能也需要变更。

设计模式学习心得

OOP (Object-Oriented Programming)

软件开发=MVC+设计模式+软件工程

面向对象三大特性: 封装、继承、多态

面向对象的四个好处: 可维护, 可扩展, 可复用 和 灵活性好。

OOP 通过封装、继承、多态把程序的耦合度降低。

面向对象设计其实就是希望做到代码的责任分解

继承和方法重载都是多态的表现。

封装变化点是我们面向对象的一种很重要的思维方式。

从设计角度讲，抽象类是从子类中发现了公共的东西，泛化出父类，然后子类继承父类，而接口是根本不知子类的存在，方法如何实现还不确认，预先定义。

用设计模式使得程序更加的灵活，容易修改，并且易于复用。

有时候可以混用多个设计模式使代码更加优雅。

OO 软件中的很多设计思想来源于电脑硬件系统的设计,比如:

强内聚，松耦合原则，就像电脑各元件遵循易插拔原则,所带来的方便性。

单一职责原则（就一个类而言，应该仅有一个引起它变化的原因），就像内存条坏了，不会成为更换 CPU 的理由。

如果一个类承担的职责过多，就等于把这些职责耦合在一起，一个职责的变化可能会削弱或者抑制这个类完成其他职责的能力。这种耦合会导致脆弱的设计，当变化发生时，设计会遭受到意想不到的破坏。

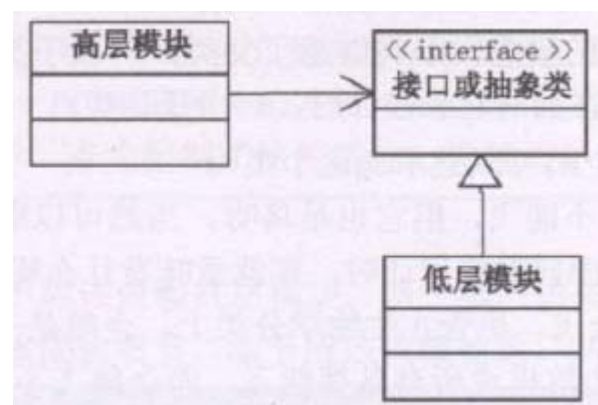
软件设计真正要做的许多内容，就是发现职责并把那些职责相互分离。

如果你能够想到多于一个的动机去改变一个类，那么这个类就具有多于一个的职责,就应该考虑类的职责分离。

开放-封闭原则(对扩展开放，对修改封闭)，就像内存不够我们可以加一根内存条。但不能去修改原来这根内存条大小。

依赖倒转原则(针对接口/抽象编程，不要对实现编程)，无论主板、CPU、内存、硬盘都是在针对接口设计的，如果针对实现来设计，内存就要对应到具体的某个品牌的主板，那就会出现换内存需要把主板也换了的尴尬。依赖倒转其实就是谁也不要依靠谁，除了约定的接口，大家都可以灵活自如。

里氏代换原则(子类型必须能够替换掉它们的父类型)，只有当子类可以替换掉父类，软件单位的功能不受到影响时，父类才能真正被复用，而子类也能能够在父类的基础上增加新的行为，从而使**继承复用成为可能**。遵循了此原则才能谈**对扩展开放，修改关闭**。依赖倒转其实可以说是面向对象设计的标志，用哪种语言来编写程序不重要，如果编写时考虑的都是如何针对抽象编程而不是针对细节编程，即程序中所有的依赖关系都是终止于抽象类或者接口，那就是面向对象的设计，反之那就是过程化的设计了。



由于子类型的可替换性才使得使用父类型的模块在无需修改的情况下就可以扩展。

迪米特法则，如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一个类的某一个方法的话，可以通过第三者转发这个调用。在类的结构设计上，每一个类都应当尽量降低成员的访问权限(信息的隐藏促进了软件的复用)，迪米特法则其根本思想，是强调了类之间的松耦合。

优先考虑合成/聚合来代替继承。

对象的继承关系是在编译时就定义好了，所以无法在运行时改变从父类继承的实现，子类的实现与它的父类有非常紧密的依赖关系，以至于父类实现中的任何变化必然会导致子类发生变化。当你需要复用子类时，如果继承下来的实现不适合解决新的问题，则父类必须重写或被其他更适合的类替换。这种依赖关系限制了灵活性并最终限制了复用性。

合成/聚合复用原则的好处是：优先使用对象的合成/聚合将有助于你保持每个类被封装，并被集中在单个任务上。这样类和类继承层次会保持较小规模，并且不太可能增长为不可控制的庞然大物。

只要真正深入地理解了设计原则，很多设计模式其实就是原则的应用而已，或许在不知不觉中就在使用设计模式了。

各种模式适用范围

简单工厂模式：只是完成对象的创建(new)工作，利用反射机制可消除分支判断。

工厂方法模式：定义一个用于创建对象的接口，让子类决定实例化哪一个类。工厂方法使一个类的实例化延迟到其子类。工厂方法克服了简单工厂违背开放-封闭原则的缺点，又保持了封装对象创建过程的优点。

抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。反射技术+抽象工厂模式可解决数据库访问时的可维护、可扩展问题。

策略模式：它定义了算法家族，分别封装起来，让它们之间可以互相替换，此模式让算法的变化，不会影响到使用算法的客户。

策略模式减少了各种算法类与使用算法类之间的耦合。继承有助于析取出这些算法中的公共功能。策略模式的优点是简化了单元测试，因为每个算法都有自己的类，可以通过自己的接口单独测试。可以用它来封装几乎任何类型的规则，只要在分析过程中听到需要在不同时间应用不同的业务规则，就可以考虑使用策略模式处理这种变化的可能性。

访问者模式：适用于数据结构稳定，算法(访问者)易变的情况。

迭代器模式：就是分离了集合对象的遍历行为，抽象出一个迭代器类来负责，这样既可以做到不暴露集合的内部结构，又可以让外部代码透明地访问集合内部的数据。

备忘录模式：在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态。这样以后就可将该对象恢复到原先保存的状态。

备忘录模式比较适用于功能比较复杂的，但需要维护或记录属性历史的类，或者需要保存的属性只是众多属性中的一小部分时，可以根据保存的信息还原到前一状态。

建造者模式：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。**(适用于构建过程不变的情况)**

建造者模式主要是用于创建一些复杂的对象，这些对象内部构建间的建造顺序通常是稳定的，但对对象内部的构建通常面临着复杂的变化。

建造者模式的好处就是使得建造代码与表示代码分离，由于建造者隐藏了该产品是如何组装的，所以若需要改变一个产品的内部表示，只需要再定义一个具体的建造者就可以了。建造者模式是在当创建复杂对象的算法应该独立于该对象的组成部分以及它们的装配方式时适用的模式。

装饰模式(Decorator)：动态地给一个对象添加一些额外的职责，就增加功能来说，装饰模式比生成子类更为灵活。

装饰模式是为已有功能动态地添加更多功能的一种方式。把类中的装饰功能从类中搬移去除，这样可以简化原有的类。有效地把类的核心职责和装饰功能区分开了，而且可以去除相关类中重复的装饰逻辑。

命令模式: 将一个请求封装为一个对象, 从而使你可用不同的请求对客户进行参数化; 对请求排队或记录请求日志, 以及支持可撤销的操作。

行为请求者与行为实现者要解耦合。

第一, 它能较容易地设计一个命令队列;

第二, 在需要的情况下, 可以较容易地将命令记入日志;

第三, 允许接收请求的一方决定是否要否决请求;

第四, 可以容易地实现对请求的撤销和重做;

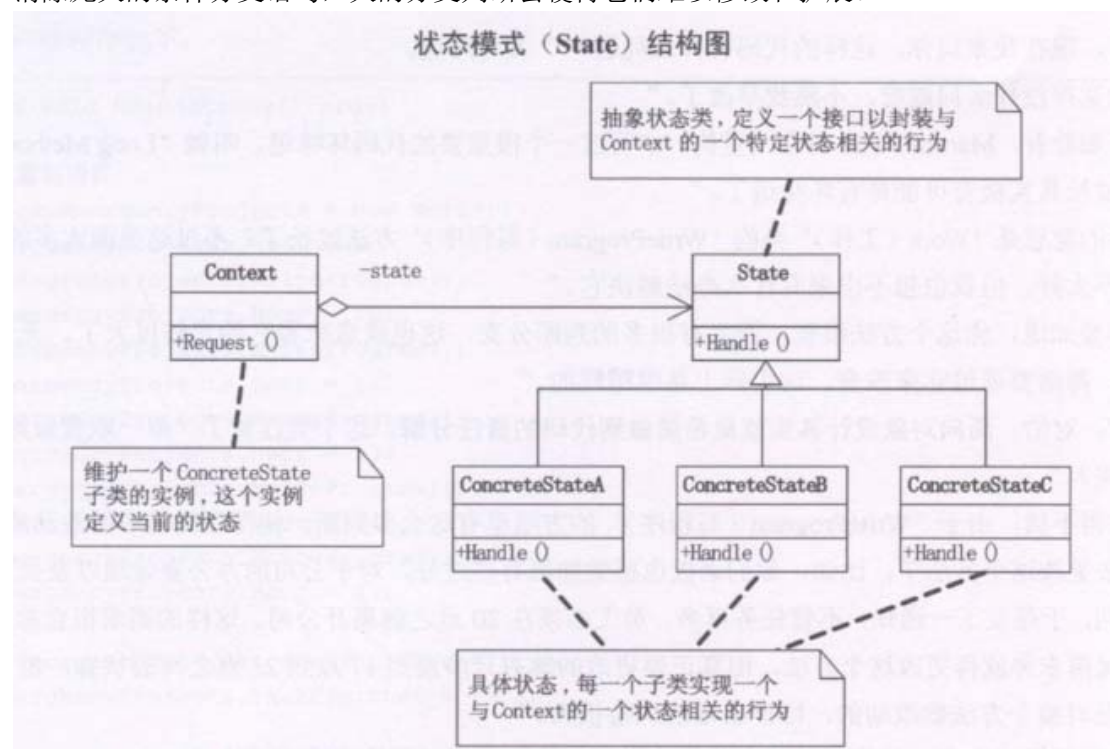
第五, 由于加进新的具体命令类不影响其他的类, 因此增加新的具体命令类很容易。

命令模式把请求一个操作的对象与知道怎么执行一个操作的对象分割开了。

敏捷开发原则告诉我们, 不要为代码添加基于猜测的、实际不需要的功能。如果清楚一个系统是否需要命令模式, 一般就不要着急去实现它, 事实上, 在需要的时候通过重构实现这个模式并不困难, 只有在真正需要如撤销/恢复操作等功能时, 把原来的代码重构为命令模式才有意义。

状态模式(State): 当一个对象的内在状态改变时允许改变其行为, 这个对象看起来像是改变了其类。状态模式主要解决的是当控制一个对象状态转换的条件表达式过于复杂时的情况。把状态的判断逻辑转移到表示不同状态的一系列类当中, 可以把复杂的判断逻辑简化。

状态模式的好处: 将与特定状态相关的行为局部化, 并且将不同状态的行为分割开来。将特定状态相关的行为都放入一个对象中, 由于所有与状态相关的代码都存在于某个 ConcreteState 中, 所以通过定义新的子类可以很容易地增加新的状态和转换。目的就是为了消除庞大的条件分支语句, 大的分支判断会使得它们难以修改和扩展。



外观模式: 为子系统的一组接口提供一个一致的界面, 此模式定义了一个高层接口, 这个接口使得这一子系统更加容易使用。

首先, 在设计初期阶段, 应该要有意识的将不同的两个层分离(数据访问层、业务逻辑层、

表示层),层与层之间建立外观。

其次,在开发阶段,子系统往往因为不断的重构演化而变得越来越复杂,增加外观可以提供简单的接口,减少它们之间的依赖。第三,在维护一个遗留的大型系统时,可能这个系统已经非常难以维护和扩展了.你可以为新系统开发一个外观类,来提供设计粗糙或高度复杂的遗留代码的比较清晰简单的接口,让新系统与外观对象交互。

中介者模式: 用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用,从而使其耦合松散,而且可以独立地改变它们之间的交互。

缺点: 由于中介者控制了集中化,于是就把交互复杂性变为了中介者的复杂性,这就使得中介者会变得比任何一个成员对象都复杂。

适用场合: 中介者模式一般应用于一组对象以定义良好但是复杂的方式进行通信的场合,以及想定制一个分布在多个类中的行为,而又不想生成太多的子类的场合。

尽管将一个系统分割成许多对象通常可以增加其可复用性,但是对象间相互连接的激增又会降低其可复用性。大量的连接使得一个对象不可能在没有其他对象的支持下工作,系统表现为一个不可分割的整体,所以,对系统的行为进行任何较大的改动就十分困难了。

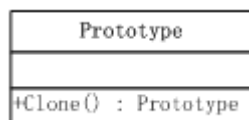
享元模式: 运用共享技术有效地支持大量细粒度的对象。

桥接模式(Bridge): 将抽象部分与它的实现部分分离,使它们都可以独立地变化。

实现系统可能有多角度分类,每一种分类都有可能变化,那么就把这种多角度分离出来让它们独立变化,减少它们之间的耦合。

例如: 硬件平台-虚拟机-应用软件

原型模式(Prototype): 通过 clone()封装创建细节, clone()表示浅复制, copy()表示深复制。



总结: 编程是一门技术,更加是一门艺术,不能只满足于写完代码运行结果正确就完事,时常考虑如何让代码更加简练,更加容易维护,容易扩展和复用,只有这样才可以真正得到提高,最终写出优雅的代码。

要想做到并做好以下 7 条设计原则其实很不容易,但是这些都是我们在设计上努力的方向:

1. 找出应用中可能需要变化之处,把它们独立出来,不要和那些不需要变化的代码混在一起.
2. 针对接口编程,而不是针对实现编程.
3. 多用组合,少用继承.
4. 为了交互对象之间的松耦合设计而努力.
5. 依赖抽象,不要依赖具体类.
6. 最大限度的减少对象之间的交互.
7. 一个类应该只有一个引起变化的原因.

敏捷软件开发宣言

- | | | |
|-----------|----|---------|
| 1.个体和交互 | 胜过 | 过程和工具 |
| 2.可以工作的软件 | 胜过 | 面面俱到的文档 |
| 3.客户合作 | 胜过 | 合同谈判 |

4.响应变化 胜过 遵循计划
虽然右项也有价值，但是我们认为左项具有更大的价值

有限状态机(Finite State Machine)

在描述有限状态机时，状态、事件、转换和动作是经常会碰到的几个基本概念。

状态 (State) 指的是对象在其生命周期中的一种状况，处于某个特定状态中的对象必然会满足某些条件、执行某些动作或者是等待某些事件。"

事件 (Event) 指的是在时间和空间上占有一定位置，并且对状态机来讲是有意义的那些事情。事件通常会引起状态的变迁，促使状态机从一种状态切换到另一种状态。

转换 (Transition) 指的是两个状态之间的一种关系，表明对象将在第一个状态中执行一定的动作，并将在某个事件发生同时某个特定条件满足时进入第二个状态。

动作 (Action) 指的是状态机中可以执行的那些原子操作，所谓原子操作指的是它们在运行的过程中不能被其他消息所中断，必须一直执行下去。

基本框架

```
switch(state){
    case state1:
        // 执行动作
        // 检查是否有某事件
        if(某事件)
        {
            // 当前状态转换为另一个状态
        }
        break;
    case state2:
        break;
}
```

点乘中零向量垂直于任意其他向量

叉乘中零向量平行于任意其他向量

点乘得到一个标量，并满足交换律。叉乘得到一个向量，并不满足交换律。

叉乘满足反交换律 $\mathbf{a} \times \mathbf{b} = -(\mathbf{b} \times \mathbf{a})$

叉乘一般而言也不满足结合律 $(\mathbf{a} \times \mathbf{b}) \times \mathbf{c} \neq \mathbf{a} \times (\mathbf{b} \times \mathbf{c})$

叉乘优先级大于点乘

$\mathbf{a} \cdot \mathbf{b} \times \mathbf{c}$ (三重积)

物体坐标系、惯性坐标系、摄像机坐标系、世界坐标系

用旋转能从物体坐标系转换到惯性坐标系，用平移能从惯性坐标系转换到世界坐标系。

WebGame 文件隐藏方法

1. 通过 UltraEdit 修改文件头信息(图片、SWF 等)
2. 用 URLoader 加载文件，通过 urlLoader.data 取出原始数据(ByteArray)

```
var loader:URLLoader = new URLLoader();  
loader.dataFormat = URLLoaderDataFormat.BINARY; //这里一定要设成二进制
```

3. 还原文件头信息
4. 再利用 Loader 的 loadBytes()方法转换成对应的文件(图片、SWF)

产品开发

1. 你所面对的客户群是谁?
2. 他们为什么要使用你的服务?
3. 他们会为哪些服务付费?
4. 市场上有没有类似产品?
5. 他们的功能怎样?
6. 缺少哪些客户需要的功能?
7. 未来的竞争态势如何?

等等问题,只有当你把这些因素统统考虑一遍之后,在返回头衡量一下自己的出发点,或许那时你将对自己产品有了更加清醒的认识。

图

邻接: 如果两个顶点被同一条边连接,就称这两个顶点是邻接的。

路径: 路径是边的序列,如 BAEJ、BCDJ

连通图: 如果至少有一条路径可以连接起所有的顶点,那么这个图被称作连通的。

连通子图:

无向图: 边没方向

有向图: 边有方向

无权图: 边有权值(代表距离、时间什么的)

图的存储结构: 邻接矩阵、邻接表

树

定义

如果 v 和 w 是 T 中的顶点,从 v 到 w 存在一条唯一的路径

对于 n 个顶点的树 T 来说下列命题等价:

- (a) T 是一棵树。
- (b) T 是连通的非循环图。
- (c) T 是有 $n-1$ 条边的连通图。
- (d) T 是有 $n-1$ 条边的非循环图。

定义

如果树 T 是包含图 G 所有顶点的树,那么树 T 是图 G 的一棵[生成树](#)。

带有 n 个顶点的树含有 $n-1$ 条边

带有 i 个内点的正则 m 元树含有 $n=m*i+1$ 个顶点

顶点数 = 树叶数 + 内点数 (即 $n = l + i$)

树的高度: 根为第 0 层

若一棵高度为 h 的 m 元树的所有树叶都在 h 层或 $h-1$ 层, 则这棵树是平衡的。

在高度为 h 的 m 元树里至多有 m^h 个树叶(归纳法证明)

若树是平衡的, 则 $m^{h-1} < l \leq m^h$, 以 m 为底的对数就得出 $h-1 < \log_m l \leq h$, 因此 $h = \text{Math.ceil}(\log_m l)$

两个基本公式:

$$n = l + i;$$

$$n = m \cdot i + 1; (i \text{ 为内点数 } m \text{ 为元数 } n \text{ 为顶点数 } l \text{ 为叶数})$$

利用二叉树进行位串编码(哈夫曼编码)

树叶存储值, 利用二叉树的唯一路径表示

例如: **sane** 对应 1111 10 1110 0

注意: 把出现频率高的字符编码到前面(比如 e), 出现频率低的字符编码到后面, 这样可节约存储空间。

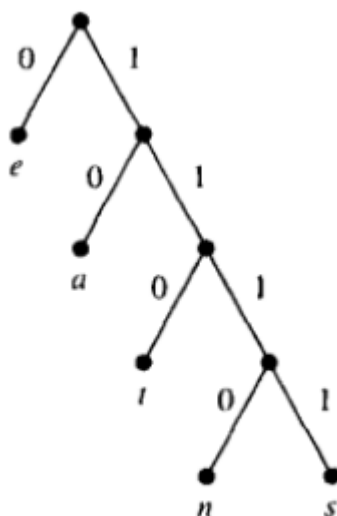
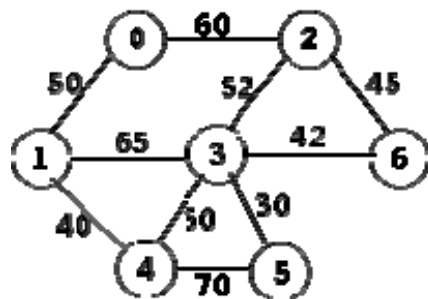


图 9-20 表示前缀码的二叉树

最小生成树

普里姆(Prim)算法



算法描述:

假设 $G=(V, E)$ 是一个具有 n 个顶点的连通图, $T=(U, TE)$ 是 G 的最小生成树, 其中 U 是 T 的顶点集, TE 是 T 的边集, U 和 TE 的初值均为空。算法开始时, 首先从 V 中任取一个顶点(假定取 v_0), 将它并入 U 中, 此时 $U=\{v_0\}$, 然后只要 U 是 V 的真子集(即 $U \subset V$), 就从那些其一个端点已在 T 中, 另一个端点仍在 T 外的所有边中, 找一条最短(即权值最小)边, 假定 (v_i, v_j)

其中 $v_i \in U$, $v_j \in V-U$, 并把该边 (v_i, v_j) 和顶点 v_j 分别并入 T 的边集 TE 和顶点集 U , 如此进行下去, 每次往生成树中并入一个顶点和一条边, 直到 $(n-1)$ 次后把所有 n 个顶点都并入到生成树 T 的顶点集中, 此时 $U=V$, TE 中包含 $n-1$ 条边, T 就是最后得到的最小生成树。

普里姆算法中有两重 for 循环, 所以时间复杂度为 $O(n^2)$ 。它适合于稠密图。

算法演示

```
var INF:int = 1000;
var cost1:Array = [
    [0, 50, 60, INF, INF, INF, INF],
    [50, 0, INF, 65, 40, INF, INF],
    [60, INF, 0, 52, INF, INF, 45],
    [INF, 65, 52, 0, 50, 30, 42],
    [INF, 40, INF, 50, 0, 70, INF],
    [INF, INF, INF, 30, 70, 0, INF],
    [INF, INF, 45, 42, INF, INF, 0]
];

/**
 * 普里姆算法
 * @param cost    带权无向连通图邻接矩阵
 * @param n        顶点数
 * @param v        要作为根的顶点编号
 */
function prim( cost:Array, n:int, v:int ):void
{
    var lowcost:Array = [];
    var closest:Array = [];
    var i:int, j:int, k:int, min:int;
    //给 lowcost[] 和 closest[] 置初值
    for(i=0; i<n; i++){
        lowcost[i] = cost[v][i];
        closest[i] = v;
    }
    //找出 n-1 个顶点
    for(i=1; i<n; i++){
        min = INF;
        //在 V-U 中找出离 U 最近的顶点 k
        for(j=0; j<n; j++){
            if(lowcost[j] != 0 && lowcost[j] < min){
                min = lowcost[j];
                k = j;
            }
        }
        trace('边('+closest[k]+'+'+k+')权为:'+min);
        lowcost[k] = 0; //标记 k 已经加入 U
        //修改数组 lowcost 和 closest
    }
}
```

```

        for(j=0; j<n; j++){
            if(cost[k][j] != 0 && cost[k][j] < lowcost[j]){
                lowcost[j] = cost[k][j];
                closest[j] = k;
            }
        }
    }
}

trace('最小生成树:');
prim(cost1, 7, 0);

```

克鲁斯卡尔(Kruskal)算法

算法描述

假设 $G=(V, E)$ 是一个具有 n 个顶点的连通网, $T=(U, TE)$ 是 G 的最小生成树, U 的初值等于 V , 即包含有 G 中的全部顶点。算法开始时, TE 的初值为空集。将图 G 中的边按权值从小到大的顺序依次选取, 若选取的边使生成树 T 不形成回路, 则把它并入 TE 中, 保留作为 T 的一条边; 若选取的边使生成树 T 形成回路, 则将其舍弃, 如此进行下去, 直到 TE 中包含 $n-1$ 条边为止, 此时的 T 即为最小生成树。

如果给定的带权无向连通图 G 有 e 条边, 那么用克鲁斯卡尔算法构造最小生成树的时间复杂度为 $O(e \log_2 e)$ (是指优化后), 它适合于稀疏图。

算法演示

```

//u:边的起始顶点 v:边的终止顶点 w:边的权值
var edge:Array = [
    {u: 3, v: 5, w: 30}, {u: 1, v: 4, w: 40}, {u: 3, v: 6, w: 42},
    {u: 2, v: 6, w: 45}, {u: 0, v: 1, w: 50}, {u: 3, v: 4, w: 50},
    {u: 2, v: 3, w: 52}, {u: 0, v: 2, w: 60}, {u: 1, v: 3, w: 65},
    {u: 4, v: 5, w: 70}
];

var n:int=7, e:int=10;

/**
 * 克鲁斯卡尔(kruskal)算法
 * @param E 带权无向连通图邻接矩阵
 * @param n 图顶点个数
 * @param e 图的边数
 */
function kruskal( E:Array, n:int, e:int):void
{
    var i:int, j:int, m1:int, m2:int, sn1:int, sn2:int, k:int;
    var vset:Array = [];
    /* 初始化辅助数组 */
    for(i=0; i<n; i++){
        vset[i] = i;
    }
}

```

```

    }
    /* k 表示构造最小生成树的第几条边，初值为 1 */
    k=1;
    /* E 中边的下标，初值为 0 */
    j=0;
    /* 生成的边数小于 n 时循环 */
    while(k<n){
        /* 取一条边的头尾顶点 */
        m1 = E[j].u;
        m2 = E[j].v;
        /* 分别得到两个顶点所属的集合编号 */
        sn1 = vset[m1];
        sn2 = vset[m2];
        /* 两顶点属不同的集合，该边是最小生成树的边 */
        if(sn1 != sn2){
            trace('边('+m1+', '+m2+')权为:'+E[j].w);
            k++; //生成边数增加 1
            /* 两个集合统一编号 */
            for(i=0; i<n; i++){
                /* 集合编号为 sn2 的改为 sn1 */
                if(vset[i] == sn2){
                    vset[i] = sn1;
                }
            }
            j++; //扫描下一条边
        }
    }

    trace('最小生成树:');
    kruskal(edge, n, e);

```

Minimax中Alpha-beta剪枝叶算法的伪代码如下：

```

01 function alphabeta(node, depth, α, β, Player)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node
04     if Player = MaxPlayer
05         for each child of node
06             α := max(α, alphabeta(child, depth-1, α, β, not(Player) ))
07             if β ≤ α
08                 break (* Beta cut-off *)
09         return α
10     else
11         for each child of node
12             β := min(β, alphabeta(child, depth-1, α, β, not(Player) ))
13             if β ≤ α
14                 break (* Alpha cut-off *)
15         return β
16 (* Initial call *)
17 alphabeta(origin, depth, -infinity, +infinity, MaxPlayer)

```

从上面的代码中可以看到如果Player是MaxPlayer，那么就要与上一层传进来的beta进行比较，如果beta小于等于alpha，则进行beta剪枝。如果Player是MinPlayer，那么就要与上一层传进来的alpha进行比较，如果beta小于等于alpha，则进行alpha剪枝。

离散数学

命题逻辑

命题是一个或真或假的陈述语句，但不能既真又假。

$\neg p$ ：非 p

$p \wedge q$ ：p 而且 q（合取）

$p \vee q$ ：p 或 q（析取）

$p \oplus q$ ：p 或 q（异或）

同或 \vee ：p 为真或 q 为真，或 pq 都为真，则结果为真

异或 \oplus ：p 为真或 q 为真，则结果为真

$p \rightarrow q$ ：p 蕴含 q

蕴含：q 由 p 推出(p 是前提或假设，q 是结论)

\rightarrow 为条件操作符

蕴含真值表：

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

$q \rightarrow p$ 称为 $p \rightarrow q$ 的**逆蕴含**

$\neg p \rightarrow \neg q$ 称为 $p \rightarrow q$ 的**反蕴含**

$\neg q \rightarrow \neg p$ 称为 $p \rightarrow q$ 的**倒置蕴含**

当两个复合命题总是具有相同的真值时，我们称它们等价。

一个蕴含与其倒置等价

一个蕴含的逆蕴含与其反蕴含也是等价的

$p \leftrightarrow q$: 双蕴含 (p 当且仅当 q)
 $p \leftrightarrow q$ 与 $(p \rightarrow q) \wedge (q \rightarrow p)$ 有相同的真值
 运算符优先级

\neg	1(最大)
\wedge	2
\vee	3
\rightarrow	4
\leftrightarrow	5(最小)

有命题 p 、 q ，如果 p 推出 q ，则 p 是 q 的 [充分条件](#)， q 是 p 的 [必要条件](#)；如果 p 推出 q 且 q 推出 p ，则 p 是 q 的充分必要条件，简称 [充要条件](#)。

逻辑等价 (\equiv)

$p \rightarrow q \equiv \neg q \rightarrow \neg p$	原命题与逆否命题等价
$p \rightarrow q \equiv (p \wedge \neg q) \rightarrow (r \wedge \neg r)$	
$\neg(p \vee q) \equiv \neg p \wedge \neg q$	De Morgan 逻辑第一定律
$\neg(p \wedge q) \equiv \neg p \vee \neg q$	De Morgan 逻辑第二定律
$\neg(p \rightarrow q) \equiv p \wedge \neg q$	条件命题的否定可以写成“与”的形式
$p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$	p 是 q 的充分必要条件
$\neg p \vee q \equiv p \rightarrow q$	p 是 q 的充分条件
$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	这是析取对合取的分配律

设 $P(x)$ 是包含变元 x 的句子，并且设 D 是一个集合。对于 D 中的每一个 x ， $P(x)$ 是一个命题，称 P 是(相对于 D 的)一个 [命题函数](#) 或 [谓词](#)。 D 称为 P 的 [论域](#)。

设 P 是关于论域 D 的命题函数。句子对每个 x ， $P(x)$ 称为 [全称量词语句](#)。符号 \forall 的意思是“**对每个**”。因此，句子对每个 x ， $P(x)$ 可以写成 $\forall x P(x)$ 符号 A 称为 [全称量词](#)。如果对于 D 中的每一个 x ， $P(x)$ 为真，则句子 $\forall x P(x)$ 为真。如果至少有一个 x 在 D 中使 $P(x)$ 为假，则句子 $\forall x P(x)$ 为假。在论域中使 $P(x)$ 为假的 x 称为语句 $\forall x P(x)$ 的 [反例](#)。我们称命题函数 $P(x)$ 中的变元 x 为 [自由变元](#) (意思是 x 可以在论域中自由选取)。在全称量词语句 $\forall x P(x)$ 中，称变元 x 为 [约束变元](#) (意思是 x 受量词 \forall 的约束)。

$\forall x P(x)$ 也可以表示为 对所有 x ， $P(x)$ 或者 对任意 x ， $P(x)$
 有时，为了表示论域 D ，把全称量词语句写成 对 D 中的每个 x ， $P(x)$

设 P 是关于论域 D 的命题函数。句子 存在 x ， $P(x)$ 称为 [存在量词语句](#)。符号 \exists 的意思是“存在”。因此，句子 存在 x ， $P(x)$ 可以写成 $\exists x P(x)$

符号 \exists 称为 [存在量词](#)。如果至少有一个 x 在 D 中使 $P(x)$ 为真，则句子 $\exists x P(x)$ 为真。如果对 D 中所有的 x ， $P(x)$ 为假，则句子 $\exists x P(x)$ 为假。

要证明 $\forall x \forall y P(x, y)$ 为真，必须证明对于论域中的任意 x ，对于论域中任意的 y ， $P(x, y)$ 为真。
 要证明 $\exists x \forall y P(x, y)$ 为真，必须证明论域中至少存在一个 x ，对于论域中任意的 y ， $P(x, y)$ 为真。

要证明 $\forall x \exists y P(x, y)$ 为真，必须证明论域中的任意 x ，至少存在一个 y ，使 $P(x, y)$ 为真。

要证明 $\exists x \exists y P(x, y)$ 为真，必须证明论域中至少存在一个 x 和一个 y (一组 x 和 y 就足够了)，

$P(x, y)$ 为真。

数学证明

直接证明

利用已知公理、定理、定义、引理、推论进行证明

反证法

首先假定结论为假(即结论的否定为真),再推出矛盾。

逆否证明(反正法的特例)

利用如果原命题为真,则它的逆否命题也为真。

分情况证明

一个带有析取形式前提的命题通常用分情况的证明法证明。

存在性证明

例如证明 $\exists x P(x)$,只需找到一个 x 使 $P(x)$ 为真即可。

归结证明

依据以下逻辑

如果 $p \vee q$ 并且 $\neg p \vee r$ 都为真,则 $q \vee r$ 为真。

因为归结法依赖于单一的简单规则,所以它是许多计算机程序用来推理和证明定理的基础。

数学归纳法证明

原理

假设对于每一个正整数 n ,以正整数为论域的命题函数 $S(n)$ 。假设 $S(1)$ 为真;(基本步)

对任意的 $n \geq 1$,如果 $S(n)$ 为真,则 $S(n+1)$ 为真。(归纳步)

则对每一个正整数 n , $S(n)$ 为真。

强数学归纳法

设有命题函数 $S(n)$,论域为大于等于 n_0 的所有整数。如果 $S(n_0)$ 为真;(基本步)

对任意 $n > n_0$,如果对任意满足 $n_0 \leq k < n$ 的 k , $S(k)$ 为真,那么都有 $S(n)$ 为真。(归纳步)

则对任意的 $n \geq n_0$, $S(n)$ 为真。

几何数求和

$$a + ar^1 + ar^2 + \dots + ar^n = a(r^{n+1} - 1)/(r - 1)$$

等式左边的和称为几何级数的和。在几何级数求和中, $a \neq 0$ 且 $r \neq 0$ 时,相邻的项的比例 r 是一个常数。

演绎推理

从一系列命题推出一个结论的过程称为演绎推理。

论证过程是一系列命题,可以写成 $p_1, p_2, \dots, p_n \therefore q$

p_1, p_2, \dots, p_n 称为假设(或前提),命题 q 称为结论,如果 p_1, p_2, \dots, p_n 都为真,那么 q 必为真,论

证过程是有效的。否则论证过程是无效的(或错误的)。

在有效的论证过程中，有时说结论遵从假设。注意，没有直接说结论为真；只是说如果给定假设，则必然得到结论。一个论证过程是有效的是因为其形式，而不是因为其内容。

例:

如果 $2=3$ ，则我吃掉我的帽子。

我吃掉我的帽子。

$\therefore 2=3$

如果设 $p: 2=3$, q : 我吃掉我的帽子

则论证过程可以写成

$p \rightarrow q$

q

$\therefore p$

如果论证过程是有效的，那么要 $p \rightarrow q$ 和 q 同时为真，则 p 必为真。假设 $p \rightarrow q$ 和 q 都为真。这在 p 为假 q 为真的时候是可能的；在这种情况下， p 为假，所以论证过程是无效的。

下面是一个有效的推理过程

$p \rightarrow q$

p

$\therefore q$

验证推理过程是无有效的二种方法

1. 写出真值表
2. 直接验证只要前提为真，结论就为真

命题推理规则

推理规则	名称	推理规则	名称
$p \rightarrow q$ p <hr/> $\therefore q$	假言推理 (分离推理)	p q <hr/> $\therefore p \wedge q$	合取
$p \rightarrow q$ $\neg q$ <hr/> $\therefore \neg p$	拒取	$p \rightarrow q$ $q \rightarrow r$ <hr/> $\therefore p \rightarrow r$	假设三段论
p <hr/> $\therefore p \vee q$	附加	$p \vee q$ $\neg p$ <hr/> $\therefore q$	析取三段论
$p \wedge q$ <hr/> $\therefore p$	化简		

量词化句子推理规则，论域为 D

推理规则	名称
$\frac{\forall xP(x)}{\therefore P(d) \text{ if } d \in D}$	全称例化
$\frac{\text{对任何 } d \in D \ P(d)}{\therefore \forall xP(x)}$	全称一般化
$\frac{\exists xP(x)}{\therefore \text{有一个 } d \in D \ P(d)}$	存在例化
$\frac{\text{有一个 } d \in D \ P(d)}{\therefore \exists xP(x)}$	存在一般化

计数方法

乘法原理

如果一项工作需要 t 步完成，第一步有 n_1 种不同的选择，第二步有 n_2 种不同的选择,...,第 t 步有 n_t 种不同的选择，那么完成这项工作所有可能的不同的选择总数为 $n_1 \times n_2 \times \dots \times n_t$

加法原理

假定 X_1, \dots, X_t 均为集合，第 i 个集合 X_i 有 n_i 个元素。若 $\{X_1, \dots, X_t\}$ 为两两不交的集合若 $(i \neq j, X_i \cap X_j = \emptyset)$ ，则可以从 X_1, X_2, \dots, X_t 选择出的元素总数为 $n_1 + n_2 + \dots + n_t$ (即集合 $X_1 \cup X_2 \cup \dots \cup X_t$ 含 $n_1 + n_2 + \dots + n_t$ 个元素。)

加法原理可总结为：当要计数的元素可分解为若干个不相交的子集时，可将每个子集元素的个数相加来得到元素的总数。

运用乘法原理可对需要若干步完成的对象计数，当计算不相交子集中对象的总数时，可运用加法原理。

一个含有 n 个元素的集合共有 2^n 个子集。(乘法原理)

排列：一组有序的对象。

定义

n 个不同元素 x_1, \dots, x_n 的一种排列为 x_1, \dots, x_n 的一个排序。

定理

n 个元素的排列共有 $n!$ 种。

定义

n 个(不同)元素 x_1, \dots, x_n 的 r 排列是 $\{x_1, \dots, x_n\}$ 的 r 元素子集上的排列。 n 个不同元素上的 r 排列的个数记做 $P(n, r)$

定理

n 个不同元素上的 r 排列数目为 $P(n, r) = n(n-1)(n-2)\dots(n-r+1)$, $r \leq n$
 写成阶乘形式为 $P(n, r) = n! / (n-r)!$

从一组对象中不计顺序的取出若干个称为**组合**。

定义

给定集合 $X = \{x_1, \dots, x_n\}$ 包含 n 个(不同的)元素,

(a) 从 X 中不计顺序地选择 r 个元素(X 的 r 元素子集)称为 X 上的一个 r 组合。

(b) n 个不同元素上的 r 组合记做 $C(n, r)$

组合数计算公式

$$C(n, r) = P(n, r) / r!$$

定理

n 个不同元素上的 r 组合数为

$$C(n, r) = P(n, r) / r! = [n(n-1)\dots(n-r+1)] / r! = n! / [(n-r)! r!]$$

能生成结果的过程称为**实验**。实验的结果或结果的组合称为**事件**。包含所有可能结果的事件称为**样本空间**。

定义

有限样本空间 S 中事件 E 的概率 $P(E) = |E| / |S|$

离散概率定义

概率函数 P 将样本空间 S 上的每一个结果 x 映射为实数 $P(x)$, 满足

$$0 \leq P(x) \leq 1, x \in S$$

且

$$\sum_{x \in S} P(x) = 1$$

第一个条件保证一个结果的概率为非负数且不超过 1; 第二个条件保证所有结果的概率之和为 1, 即进行实验后, 必出现某个结果。

定义

E 为一个事件, E 的概率 $P(E)$ 定义为

$$P(E) = \sum_{x \in E} P(x)$$

定理

$$P(E) + P(\bar{E}) = 1$$

E_1 和 E_2 为两个事件, 事件 $E_1 \cup E_2$ 表示 E_1 或 E_2 至少有一个发生, 事件 $E_1 \cap E_2$ 表示 E_1 和 E_2 同时发生。

定理

E_1 和 E_2 为两个事件, 则

$$P(E_1 \cup E_2) = P(E_1) + P(E_2) - P(E_1 \cap E_2)$$

若 E_1 和 E_2 不相交, 则

$$P(E_1 \cup E_2) = P(E_1) + P(E_2)$$

若给定一个事件必然发生条件下, 另一事件发生的概率为**条件概率**。

下面一般性地讨论条件概率。给定事件 F 后事件 E 发生的概率记做 $P(E|F)$ 。当事件 F 给定后, F 包含的所有结果成为新的样本空间。事件 F 发生的概率为 $P(F)$, 对于 F 中的所有结果, 应将其在给定事件 F 前发生的概率除以 $P(F)$ 得到新的概率, 否则 F 中所有结果的概率和将不为 1。给定事件 F 后, E 发生的所有结果即为 $E \cap F$ 中的结果。可得 $P(E|F)$ 的值 $P(E \cap F)/P(F)$

定义

设 E 和 F 为两个事件, 且 $P(F) > 0$ 。给定 F 后事件 E 的条件概率定义为

$$P(E|F) = P(E \cap F)/P(F)$$

直观地说, 事件 E 不依赖于事件 F 时, $P(E|F) = P(E)$ 。我们便称 E 和 F 为**独立事件**。

独立事件定义

$$P(E \cap F) = P(E)P(F)$$

模式识别与 Bayes 定理

模式识别根据对象的特性对其进行分类。给定对象特性 F 后, 并认为该对象应属于使条件概率 $P(C|F)$ 最大的那一类。

Bayes 定理可用于计算给定特性 F 后, 属于某类的条件概率。

贝叶斯(Bayes)定理

设有 n 个种类 C_1, \dots, C_n , 任意两个类不相交, 且每个对象必属于其中一类。给定特性 F , 有

$$P(C_i | F) = \frac{P(F | C_i) P(C_i)}{\sum_{j=1}^n P(F | C_j) P(C_j)}$$

证明

$$P(C_j | F) = P(C_j \cap F) / P(F)$$

$$P(F | C_j) = P(F \cap C_j) / P(C_j)$$

由以上两式, 可得

$$P(C_j | F) = P(C_j \cap F) / P(F) = P(F | C_j)P(C_j) / P(F)$$

故只需证明

$$P(F) = \sum_{i=1}^n P(F | C_i) P(C_i)$$

因为每一个对象必属于某一类, 于是

$$F = (F \cap C_1) \cup (F \cap C_2) \cup \dots \cup (F \cap C_n)$$

因为 C_i 两两不相交, 所以 $F \cap C_i$ 也两两不相交。

$$P(F) = P(F \cap C_1) + P(F \cap C_2) + \dots + P(F \cap C_n)$$

由定理 $P(F \cap C_i) = P(F | C_i) P(C_i)$ 得

$$P(F) = \sum_{i=1}^n P(F | G_i) P(G_i)$$

定理得证。

广义的排列和组合

定理

设序列 S 包含 n 个对象，其中第 1 类对象有 n_1 个，第 2 类对象有 n_2 个,...,第 t 类对象有 n_t 个。
则 S 的不同排序个数为 $n! / n_1!n_2!...n_t!$

证明

$$\begin{aligned} & C(n, n_1)C(n-n_1, n_2)C(n-n_1-n_2, n_3)...C(n-n_1-...-n_{t-1}, n_t) \\ &= [n! / n_1!(n-n_1)!] [(n-n_1)! / n_2!(n-n_1-n_2)!] ... [(n-n_1-...-n_{t-1})! / n_t!0!] \\ &= n! / (n_1!n_2!...n_t!) \end{aligned}$$

例如：求以下字母能够组合成多少种字符串(把相同的字母看成是一类,用以上公式求解)

M、I、S、S、I、S、S、I、P、P、I

更一般的结论

定理

X 为包含 t 个元素的集合，在 X 中**允许重复、不计顺序**地选取 k 个元素，共有

$C(k+t-1, t-1) = C(k+t-1, k)$ 种选法。

证明：(竖线分割法)从 m、n、k、w 四个类中允许重复，不计顺序地选 11 个元素出来
mmm|nn|kkkk|w

可见有 3 个竖线

$C(11+3-1, 3-1)$ 种选法。

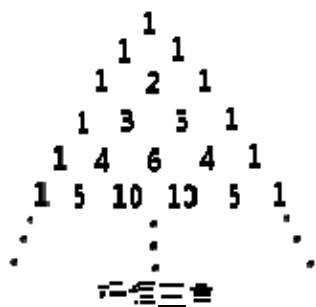
各种情况下的计数公式选择

	无重复	有重复
有序选择	$n!$	$n! / (n_1!...n_t!)$
无序选择	$C(n, r)$	$C(k+t-1, t-1)$

二项式定理

设 a 和 b 为实数，n 为正整数，则

$$(a+b)^n = \sum_{k=0}^n C(n, k) a^{n-k} b^k$$



定理

对任意 $1 \leq k \leq n$, 有 $C(n+1, k) = C(n, k-1) + C(n, k)$

即 $C(n, k) = C(n+1, k+1) - C(n, k+1)$

证明

令 X 为 n 元素集合, $a \notin X$. 则 $C(n+1, k)$ 为 $Y = X \cup \{a\}$ 的 k 元素子集的个数。 Y 的 k 元素子集可分为两类:

1. Y 的不包含 a 的子集。

2. Y 的包含 a 的子集。

第一类子集相当于从 X 中选取 k 个元素, 故共有 $C(n, k)$ 个; 第二类子集相当于选取 a 后再从 X 中选取 $k-1$ 个元素, 故共有 $C(n, k-1)$ 个。

得证。

$$(1+1)^n = \sum_{k=0}^n C(n, k) = 2^n$$

定理

$$\sum_{i=k}^n C(i, k) = C(n+1, k+1)$$

$$C(k, k) + C(k+1, k) + C(k+2, k) + \dots + C(n, k)$$

$$= 1 + C(k+2, k+1) - C(k+1, k+1) + C(k+3, k+1) - C(k+2, k+1) + \dots + C(n+1, k+1) - C(n, k+1)$$

$$= C(n+1, k+1)$$

鸽巢(抽屉)原理

鸽巢原理(第一种形式)

n 只鸽子飞入 k 个鸽巢, $k < n$, 则必存在某个鸽巢包含至少两只鸽子。

鸽巢原理(第二种形式)

设 f 为有限集合 X 到有限集合 Y 的函数, 且 $|X| > |Y|$, 则必存在 $x_1, x_2 \in X$, $x_1 \neq x_2$, 满足 $f(x_1) = f(x_2)$

鸽巢原理(第三种形式)

设 f 为有限集合 X 到有限集合 Y 上的函数, $|X| = n$, $|Y| = m$. 令 $k = \lceil \frac{n}{m} \rceil$ 则至少存在 k 个元素

a_1, \dots, a_k 属于 X , 满足 $f(a_1) = f(a_2) = \dots = f(a_k)$

利用反证法可证明:

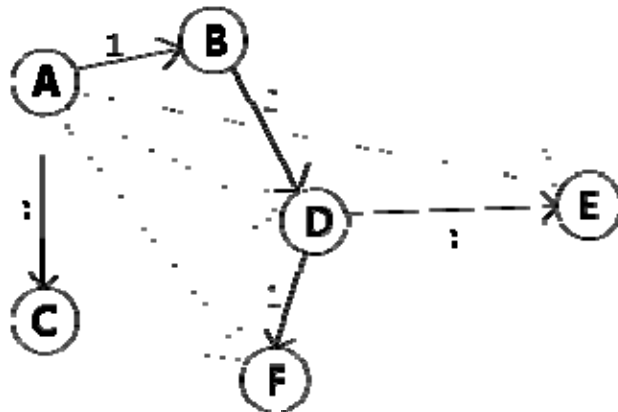
假设结论不成立, 令 $Y = \{y_1, \dots, y_m\}$ 。在 X 中, 满足 $f(x) = y_1$ 的元素 x 不超过 $k-1$ 个; 满足 $f(x) = y_2$ 的元素 x 不超过 $k-1$ 个; ...; 满足 $f(x) = y_m$ 的元素 x 不超过 $k-1$ 个。则 f 的定义域中元素个数不超过 $m(k-1)$ 个。但 $m(k-1) < m(n/m) = n$

矛盾! 故必存在至少 k 个元素 a_1, \dots, a_k 于 X , 使 $f(a_1) = f(a_2) = \dots = f(a_k)$

图论

最短路径算法

狄克斯特拉(Dijkstra)算法 (求单源最短路径)



上图的邻接矩阵为 $\text{cost}[6][6]$

	A	B	C	D	E	F
A	0	1	3	MAX	MAX	MAX
B	MAX	0	MAX	2	MAX	MAX
C	MAX	MAX	0	MAX	MAX	MAX
D	MAX	MAX	MAX	0	3	2
E	MAX	MAX	MAX	MAX	0	MAX
F	MAX	MAX	MAX	MAX	MAX	0

MAX 表示无穷大(无路径)

现在求 A 点到其它各点的最短路径?

原理:

$s[]$ 用来标记已找到的最短路径的顶点。

$s[i] = 0$; // 未找到源点到顶点 v_i 的最短路径

$s[i] = 1$; // 已找到源点到顶点 v_i 的最短路径

$\text{dist}[]$ 记录从源点到其他各顶点当前的最短距离, 初始值为 $\text{dist}[i] = \text{cost}[v][i]$

$\text{path}[]$ 用于保存最短路径长度, 其中, $\text{path}[i]$ 保存从源点 v 到终点 v_i 当前最短路径中的前一个顶点编号, 它的初值为源点 v 的编号(v 到 v_i 有边时)或 -1 (v 到 v_i 无边时)

```
var M:uint = uint.MAX_VALUE; // 此权值代表 无边
```

```

//有向图 G 的邻接矩阵
var cost:Array = [
    [0, 1, 3, M, M, M],
    [M, 0, M, 2, M, M],
    [M, M, 0, M, M, M],
    [M, M, M, 0, 3, 1],
    [M, M, M, M, 0, M],
    [M, M, M, M, M, 0]
];

/**
 *狄克斯特拉算法
 *@param    cost    有向图邻接矩阵
 *@param    v        源点编号
 */
function dijkstra( cost:Array, v:uint ):void
{
    var n:uint = cost.length; //有向图顶点数
    var dist:Array = []; //记录从源点到其他各顶点当前的最短距离
    var s:Array = []; //用于标记已找到最短路径的顶点(0:未找到 1:已找到)
    var path:Array = []; //保存最短路径
    var i:uint, j:uint, u:int, pre:uint, mindis:uint;
    var INF:uint = uint.MAX_VALUE;
    for(i=0; i<n; i++){
        dist[i] = cost[v][i]; //距离初始化
        s[i] = 0; //置空
        if(cost[v][i]<INF){
            path[i] = v; //路径初始化
        }else{
            path[i] = -1;
        }
    }

    s[v] = 1; //源点编号 v 放入 s 中
    path[v] = 0;
    //循环直到所有顶点的最短路径都求出
    for(i=0; i<n; i++){
        mindis = INF;
        u = -1;
        //选取不在 s 中且具有最小距离(相对源点)的顶点 u
        for(j=0; j<n; j++){
            if(s[j] == 0 && dist[j] < mindis){
                u = j;
                mindis = dist[j];
            }
        }
        s[u] = 1;
        for(j=0; j<n; j++){
            if(s[j] == 0 && cost[u][j] < INF){
                dist[j] = dist[u] + cost[u][j];
                path[j] = path[u] + j;
            }
        }
    }
}

```

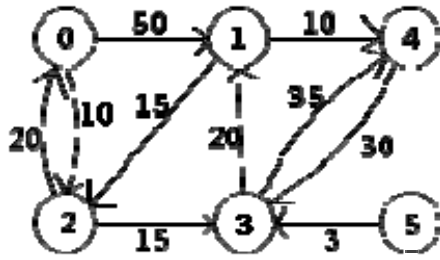
```

    }
}
//找到最小距离的顶点 u
if(-1 != u){
    s[u] = 1; //顶点 u 加入 s 中
    //修改不在 s 中的顶点的距离
    for(j=0; j<n; j++){
        if(0 == s[j]){
            if(cost[u][j] < INF && dist[u] + cost[u][j] < dist[j]){
                dist[j] = dist[u] + cost[u][j];
                path[j] = u;
            }
        }
    }
}
}
}
trace("Dijkstra 算法求解如下:");
//输出最短路径长度，路径逆序输出
var word:Array = ['A', 'B', 'C', 'D', 'E', 'F'];
for(i=0; i<n ; i++){
    if(i != v){
        trace(word[v]+"->" + word[i]);
        if(1 == s[i]){
            trace("路径长度为" + dist[i]);
            pre = i;
            var p:String = "";
            while(pre != v){ //一直求解到初始顶点
                p += word[pre] + ",";
                pre = path[pre];
            }
            trace("路径逆序为" + p + word[pre]);
        } else {
            trace("不存在路径 i=" + word[i]);
        }
    }
}
}
}

dijkstra(cost, 0);

```

弗洛伊德(Floyd)算法（每对顶点之间的最短路径）



算法描述

从图的带权邻接矩阵 $cost$ 出发，其基本思路是：如果从 v_i 到 v_j 存在一条长度为 $cost[i][j]$ 的路径。该路径不一定是最短路径，尚需进行 n 次试探。首先考虑路径 (v_i, v_0, v_j) 是否存在（即判断边 (v_i, v_0) 和 (v_0, v_j) 是否存在）。如果存在，则比较其路径长度。取长度较短者为从 v_i 到 v_j 的中间顶点的序号不大于 0 的最短路径。假如在路径上再增加一个顶点 v_1 ，即如果 (v_i, \dots, v_1) 和 (v_1, \dots, v_j) 分别是当前找到的中间顶点的序号不大于 0 的最短路径，那么， $(v_i, \dots, v_1, \dots, v_j)$ 就有可能是从 v_i 到 v_j 中间顶点序号不大于 1 的最短路径。将它和已经得到的从 v_i 到 v_j 中间顶点的序号不大于 0 的最短路径相比较，从中选出中间顶点的序号不大于 1 的最短路径之后，再增加一个顶点 v_2 ，继续进行试探。依此类推，直至经过 n 次比较，最后求得的必是从 v_i 到 v_j 的最短路径。按此方法，可以同时求得各对顶点间的最短路径。

算法演示:

```

var INF:int = 100;
var cost1:Array = [
    [0 ,50 ,10 ,INF,INF,INF],
    [INF,0 ,15 ,50 ,10 ,INF],
    [20 ,INF,0 ,15 ,INF,INF],
    [INF,20 ,INF,0 ,35 ,INF],
    [INF,INF,INF,30 ,0 ,INF],
    [INF,INF,INF,3 ,INF,0 ]
];

/**
 * 弗洛伊德(Floyd)算法
 * @param cost 带权邻接矩阵有向图
 * @param n 顶点数
 */
function floyd( cost:Array, n:int ):void
{
    var A:Array=[], path:Array=[];
    var i:int, j:int, k:int, pre:int;
    for(i=0; i<n; i++){ //置初值
        A[i] = [];
        path[i] = [];
        for(j=0; j<n; j++){
            A[i][j] = cost[i][j];
            path[i][j] = -1;
        }
    }
}

```



```

    for(k=0; k<n; k++){
        for(i=0; i<n; i++){
            for(j=0; j<n; j++){
                if(A[i][j] > A[i][k] + A[k][j]){
                    A[i][j] = A[i][k] + A[k][j];
                    path[i][j] = k;
                }
            }
        }
    }
}

trace('Floyed 算法求解如下:');
for(i=0; i<n; i++){ //输出最短路径
    for(j=0; j<n; j++){
        if(i != j){
            if(A[i][j] == INF){
                trace(i+'->'+j+'不存在路径');
            }else{
                var pth:String = i+'->'+j+'路径长度为:'+A[i][j]+' 路径为 ' + i + ' ';
                pre = path[i][j];
                while(pre != -1){
                    pth += pre + ' ';
                    pre = path[pre][j];
                }
                pth += j;
                trace(pth);
            }
        }
    }
}
}

floyed(cost1, 6);

```

拓扑排序

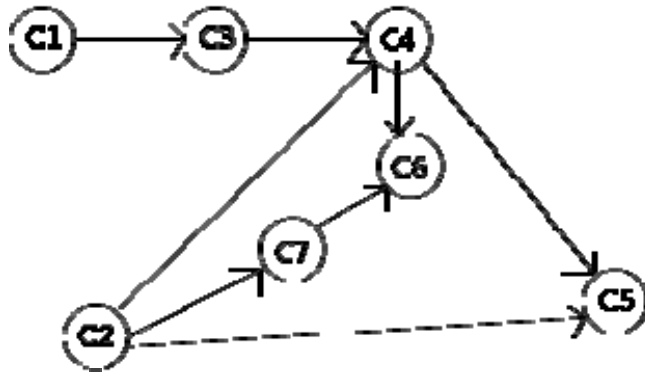
拓扑排序是有向图的一种重要运算。设 $G=(V, E)$ 是一个具有 n 个顶点的有向图，在图中用顶点表示活动，用边表示活动间的优先关系，则这个有向图称为用顶点表示活动的网 (Activity On Vertex Network) 简称为 AOV 网。在 AOV 网中，如果从顶点 v_i 到顶点 v_j 有一条有向路径，则 v_i 是 v_j 的前驱， v_j 是 v_i 的后继；如果 $\langle v_i, v_j \rangle$ 是 AOV 网中的一条边，则 v_i 是 v_j 的直接前驱， v_j 是 v_i 的直接后继。

对于一个 AOV 网，构造其所有顶点的线性序列，建立顶点之间的先后关系，而且使原来没有先后关系的顶点之间也建立起人为的先后关系。这样的线性序列称为拓扑有序序列。构造 AOV 网的拓扑有序序列运算称为拓扑排序。

拓扑排序的方法如下：

1. 从有向图中选择一个没有前驱(即入度为 0)的顶点并且输出它。
2. 从图中删去该顶点, 并且删去从该顶点出发的全部有向边。
3. 重复上述两步, 直到剩余的网中不再存在没有前驱的顶点为止。

如果 AOV 网中存在环路, 顶点之间的先后关系进行了死循环, 则意味着某项活动应该以自身完成为先决条件, 这显然是不合理的, 所以, 求不出该网的拓扑有序序列; 反之, 任何无环路的有向图, 其所有顶点都可以排在一个拓扑有序序列中。显示, 一个 AOV 网的拓扑有序序列并不是唯一的。



算法演示

```

//AOV 网邻接表 count:入度 firstarc:后继结点
var C1:Object = {p: 1,count: 0, firstarc: [2]};
var C6:Object = {p: 6,count: 2, firstarc: null};
var C5:Object = {p: 5,count: 2, firstarc: null};
var C7:Object = {p: 7,count: 1, firstarc: [5]};
var C4:Object = {p: 4,count: 2, firstarc: [4,5]};
var C2:Object = {p: 2,count: 0, firstarc: [3,4,6]};
var C3:Object = {p: 3,count: 1, firstarc: [3]};

var adjl:Array = [C1, C2, C3, C4, C5, C6, C7];

/**
 * 拓扑排序
 * @param adj    AOV 网所有结点
 * @param n      AOV 网结点总数
 */
function topSort( adj:Array, n:int ):void
{
    var i:int, j:int;
    var st:Array=[], top:int=-1; /*栈 st 的指针为 top*/
    var p:Array, o:Object;
    for(i=0; i<n; i++){
        if(adj[i].count == 0){ /*入度为 0 的顶点入栈*/
            top++;
            st[top]=i;
        }
    }

```

```

    }
    while(top>-1){
        i=st[top]; top--; //出栈
        trace(' '+adj[i].p); //输出顶点
        p=adj[i].firstarc; //直接后继顶点集合
        if(p != null){
            for(var m:uint=0; m<p.length; m++){
                j = p[m];
                adj[j].count--; //此顶点的所有直接后继顶点的入度减 1
                if(adj[j].count == 0){ //入度为 0 的相邻顶点入栈
                    top++;
                    st[top]=j;
                }
            }
        }
    }
}

topSort(adj1, 7);

```

偶数可表示为 $2k$ ，奇数可表示为 $2k+1$ (其中 k 为某个整数)

合数(非素数)

求最大公约数和最小公倍数

求正整数 a, b 最小公约数

1 步: 分解因式

2 步: $\gcd(120, 500) = 2^{\min(3,2)} * 3^{\min(1,0)} * 5^{\min(1,3)} = 2^2 * 3^0 * 5^1 = 20$

欧几里得(辗转相除)算法

定理: $\gcd(a, b) = \gcd(b, a \bmod b)$

证明: a 可以表示成 $a = kb + r$, 则 $r = a \bmod b$

假设 d 是 a, b 的一个公约数, 则有

$d \mid a, d \mid b$, 而 $r = a - kb$, 因此 $d \mid r$

因此 d 是 $(b, a \bmod b)$ 的公约数

假设 d 是 $(b, a \bmod b)$ 的公约数, 则

$d \mid b, d \mid r$, 但是 $a = kb + r$

因此 d 也是 (a, b) 的公约数

因此 (a, b) 和 $(b, a \bmod b)$ 的公约数是一样的, 其最大公约数也必然相等, 得证

求正整数 a, b 最小公倍数

1 步: 分解因式

$$2 \text{ 步: } \text{lcm}(120, 500) = 2^{\max(3,2)} * 3^{\max(1,0)} * 5^{\max(1,3)} = 2^3 * 3^1 * 5^3 = 3000$$

最大公约数与最小公倍数之间的关系

$$a*b = \text{gcd}(a, b) * \text{lcm}(a, b)$$

进制转换

二进制转十进制:

$$(1011111)_2 = 1*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0 = 95$$

十六进制转十进制:

$$(2AE0B)_{16} = 2*16^4 + 10*16^3 + 14*16^2 + 0*16 + 11 = (175627)_{10}$$

八进制转十进制同理

十进制展开:

求 $(12345)_{10}$ 的八进制展开

解 首先用 8 除 12345, 得

$$12345 = 8*1543 + 1$$

不断用 8 除商数, 得

$$1543 = 8*192 + 7$$

$$192 = 8*24 + 0$$

$$24 = 8*3 + 0$$

$$3 = 8*0 + 3$$

由于这些余数就是 12345 的八进制展开中的数字, 于是 $(12345)_{10} = (30071)_8$

求 $(177130)_{10}$ 的十六进制展开

解 首先用 16 除 177130, 得

$$177130 = 16*11070 + 10$$

不断用 16 除商数, 得

$$11070 = 16*691 + 14$$

$$691 = 16*43 + 3$$

$$43 = 16*2 + 11$$

$$2 = 16*0 + 2$$

由于这些余数就是 $(177130)_{10}$ 的十六进制展开中的数字, 于是 $(177130)_{10} = (2B3EA)_{16}$

十进制展二进制方法同上

求 $(10)_{10}$ 的二进制展开

解 首先用 2 除 10, 得

$$10 = 2*5 + 0$$

不断用 2 除商数, 得

$$5 = 2*2 + 1$$

$$2 = 2*1 + 0$$

$$1 = 2*0 + 1$$

由于这些余数就是 $(10)_{10}$ 的二进制展开中的数字, 于是 $(10)_{10} = (1010)_2$

注: 可以看出直到进位为 0, 则算法结束。

两个二进制相加

a 的二进制展开为 $(a_{n-1}a_{n-2}\dots a_1a_0)_2$

b 的二进制展开为 $(b_{n-1}b_{n-2}\dots b_1b_0)_2$

要把 a 和 b 相加, 首先把最右边的数字相加。这样可得

$$a_0 + b_0 = c_0 \cdot 2 + s_0$$

其中 s_0 是 $a+b$ 的二进制展开中最右边的一位数字, 而 c_0 是进位, c_0 为 0 或 1。然后把下一对二进制位及进位相加,

$$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$$

其中 s_1 是 $a+b$ 的二进制展开中的下一位(从右算起)数字, c_1 是进位。继续这一过程, 把两个二进制展开中对应的二进制位及进位相加, 给出 $a+b$ 的二进制展开中从右算起的下一位数字。最后把 a_{n-1}, b_{n-1} 和 c_{n-2} 相加得 $c_{n-1} \cdot 2 + s_{n-1}$ 。 $a+b$ 的首位数字是 $s_n = c_{n-1}$ 。这一过程产生 a 与 b 之和的二进制展开, 即 $a+b = (s_n s_{n-1} \dots s_1 s_0)_2$ 。

两二进制相乘

$$a \cdot b = a(b_0 2^0 + b_1 2^1 + \dots + b_{n-1} 2^{n-1}) = a(b_0 2^0) + a(b_1 2^1) + \dots + a(b_{n-1} 2^{n-1})$$

因而可以把 ab_j 的二进制展开向左移位 j 位, 再在尾部加上 j 个 0 来计算 $(ab_j)2^j$ 。最后, 把 n 个整数 $ab_j 2^j, j=0, 1, \dots, n-1$, 相加就得到 $a \cdot b$

线性同余法生成伪随机数

$$x_{n+1} = (ax_n + c) \bmod m \quad (\text{递归定义 } 2 \leq a < m, 0 \leq c < m, 2 \leq x_0 < m)$$

例如选 $m=9, a=7, c=4, x_0=3$

输入的随机数为 3, 7, 8, 6, 1, 2, 0, 4, 5, 3, 7, 8, 6, 1, 2, 0, 4, 5, 3, ...

给定两个正实数, 则算术均值总大于几何均值

$$(a+b)/2 > \sqrt{ab} \quad (a \neq b)$$

证明(后推法):

$$(a+b)^2/4 > ab$$

$$(a+b)^2 > 4ab$$

$$a^2 + 2ab + b^2 > 4ab$$

$$a^2 - 2ab + b^2 > 0$$

$$(a-b)^2 > 0$$

证毕

当不能一下子考虑所有情况时, 可以分情形证明(一定要考虑全部情形)

集合

集合是一些无序对象的全体。

一个函数为集合 X 中的每个元素指定了集合 Y 中惟一的一个元素与之相对应。

序列与集合的区别在于有序

$$A = \{1, 2, 3, 4\} \text{ 等价于 } A = \{2, 4, 3, 1\} \text{ 等价于 } A = \{1, 2, 2, 3, 4\}$$

$B = \{x \mid x \text{ 是正的偶数}\}$ 读作: B 等于所有 x 组成的集合, 条件是 x 为正的偶数。

$$1 \in A, 5 \notin A$$

不含任何元素的集合称为空集(或 NULL), 记为 \emptyset , 于是 $\emptyset = \{\}$

如果 $\forall x ((x \in X \rightarrow x \in Y) \wedge (x \in Y \rightarrow x \in X))$ 为真, 则 $X=Y$

设 X 和 Y 是两个集合, 若 X 的每个元素都是 Y 的元素, 便称 X 是 Y 的子集, 记做 $X \subseteq Y$ 。

可将 X 是 Y 的子集用符号表示为 $\forall x(x \in X \rightarrow x \in Y)$

任何科学理论中有它的研究对象, 这些对象构成一个不空的集合, 称为 **论域**。

任何集合是其自身的子集。

空集是每个集合的子集。

如果 X 是 Y 的子集并且 X 不等于 Y , 称 X 是 Y 的**真子集**, 记做 $X \subset Y$

X 的所有子集(包括真子集和非真子集)组成的集合, 记做 $P(X)$, 称为 X 的**幂集**。

集合元素个数, 记做 $|X|$

含有 n 个元素的集合的幂集含有 2^n 个元素。(数学归纳法证明)

$X \cup Y = \{x \mid x \in X \text{ 或 } x \in Y\}$ 称为 X 与 Y 的**并集**, 并集由属于 X 或属于 Y (或同时属于二者) 的所有元素组成。

$X \cap Y = \{x \mid x \in X \text{ 且 } x \in Y\}$ 称为 X 与 Y 的**交集**, 交集由同时属于 X 和 Y 的所有元素组成。

$X - Y = \{x \mid x \in X \text{ 且 } x \notin Y\}$ 称为 X 与 Y 的**差集**(相对余集), 差集 $X - Y$ 由所有在 X 中但不在 Y 中的元素组成。

如 $X \cap Y = \emptyset$, 则称集合 X 与集合 Y 是**不相交的**

如果 S 中的任意两个不同的集合 X 和 Y 都是不相交的, 则**族** S 称为**两两不相交的**。

$\{1, 4, 5\}$ 和 $\{2, 6\}$ 不相交

$S = \{\{1, 4, 5\}, \{2, 6\}, \{3\}, \{7, 8\}\}$ 两两不相交

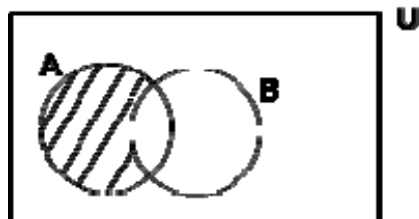
有时所研究的集合都是某个集合 U 的子集。此时 U 称为**全集**或**全域**。全集 U 一定是被明确地给出或者能从上下文中推断出来的。给定全集 U 和 U 的一个子集 X , 集合 $U - X$ 称为 X 的

余(补)集, 记为 \bar{X}

Venn 图

矩形表示全集, 圆表示子集, 特定区域用阴影表示

以下为 $A - B$ 的 Venn 图



De Morgan 律

$$\overline{A \cup B} = \bar{A} \cap \bar{B} \quad , \quad \overline{A \cap B} = \bar{A} \cup \bar{B}$$

(a)结合律

$$(A \cup B) \cup C = A \cup (B \cup C)$$

$$(A \cap B) \cap C = A \cap (B \cap C)$$

(b)交换律

$$A \cup B = B \cup A$$

$$A \cap B = B \cap A$$

(c)分配律

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$$

(d)同一律

$$A \cup \emptyset = A$$

$$A \cap U = A$$

(e)补余律

$$A \cup \bar{A} = U$$

$$A \cap \bar{A} = \emptyset$$

(f)等幂律

$$A \cup A = A$$

$$A \cap A = A$$

(g)零律

$$A \cup U = U$$

$$A \cap \emptyset = \emptyset$$

(h)吸收律

$$A \cup (A \cap B) = A$$

$$A \cap (A \cup B) = A$$

对合律

$$\bar{\bar{A}} = A \quad \text{左边 } A \text{ 顶部是 } \bar{\quad} \quad A \text{ 的补集的补集等于 } A$$

(j)0/1 律

空集的补集是全集

全集的补集是空集

$$X = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

$$S = \{\{1, 4, 5\}, \{2, 6\}, \{3\}, \{7, 8\}\}$$

X 的每一个元素恰属于集族 S 中的一个元素，所以 S 是 X 的一个划分

有序对，记做(a,b)。n 元组，记做(a₁, a₂, ..., a_n)

有序对和 n 元组要考虑元素顺序

用 $X \times Y$ 表示所有有序对(x,y)组成的集合，其中 $x \in X, y \in Y$ 。称 $X \times Y$ 为 X 与 Y 的笛卡儿积

如果 $X = \{1, 2, 3\}$ $Y = \{a, b\}$

$$X \times Y = \{(1, a), (1, b), (2, a), (2, b), (3, a), (3, b)\}$$

$$Y \times X = \{(a, 1), (a, 2), (a, 3), (b, 1), (b, 2), (b, 3)\}$$

在一般情况下 $X \times Y \neq Y \times X$

$|X \times Y| = |X| \cdot |Y|$ (对所有的有限集合都成立)

$|X_1 \times X_2 \times \dots \times X_n| = |X_1| \cdot |X_2| \dots |X_n|$

函数

定义: 函数 (function) 表示每个输入值对应**唯一**输出值的一种对应关系。函数 f 中对输入值的输出值 x 的标准符号为 $f(x)$ 。包含某个函数所有的输入值的集合被称作这个函数的 **定义域**，包含所有的输出值的集合被称作 **值域**。若先定义映射的概念，可以简单定义函数为，定义在非空数集之间的映射称为函数。

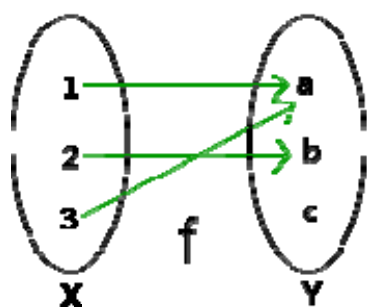
从 X 到 Y 的函数记为 $f: X \rightarrow Y$

集合 X 称为 f 的**定义域** $\{x | (x, y) \in f\}$

Y 的一个子集称为 f 的**值域**

集合 $f = \{(1, a), (2, b), (3, c)\}$ 是从 $X = \{1, 2, 3\}$ 到 $Y = \{a, b, c\}$ 的函数

函数的箭头图



上图定义域 $X = \{1, 2, 3\}$ 值域 $Y = \{a, b\}$

集合表示 $\{(1, a), (2, b), (3, a)\}$

定义域与值域是一对一 或 一对多关系。(即 X 中的每个元素有且只有一个箭头射向 Y)

$f(x) = y$ 是 $(x, y) \in f$ 的另一种写法

商和余数定理 $n = dq + r$ (q 商, r 余数, $0 \leq r < d$, d 和 n 是整数)

$$\begin{bmatrix} \sim \\ \vdots \end{bmatrix} \begin{bmatrix} q + \frac{r}{d} \\ \vdots \end{bmatrix} q$$

如果从 X 到 Y 的函数 f 的值域是 Y ，则称函数 f 为对 Y 映上的(或映上函数、满射函数)

例如:

$X = \{1, 2, 3\}$, $Y = \{a, b, c\}$

$f = \{(1, a), (2, c), (3, b)\}$

是一对一的，并且是对 Y 映上的

如果一个函数即是一对一的又是映上的，则称这个函数为**双射**。

反(逆)函数 f^{-1}

$f = \{(1, a), (2, b), (3, c)\}$

$f^{-1} = \{(a, 1), (b, 2), (c, 3)\}$

或 $\{(x, y) | (y, x) \in f\}$

令 g 为从 X 到 Y 的函数, f 为从 Y 到 Z 的函数。 f 与 g 的复合函数记为 $f \circ g$
 $(f \circ g)(x) = f(g(x))$ 是从 X 到 Z 的函数

一元操作符

二元操作符

定义函数的方式

直接列举: $\{(a, 1), (b, 3), (c, 2), (d, 1)\}$

用公式定义: $\{(n, n+2) \mid n \text{ 是正整数}\}$

函数是一个非常广义的概念。任何一个 $X \times Y$ 的子集 f , 只要满足对每个 $x \in X$, 只有惟一的 $y \in Y$ 使得 $(x, y) \in f$, 这个 f 就是一个函数。

为了证明从 X 到 Y 的函数 f 是一对一的, 必须表明对所有的 $x_1, x_2 \in X$, 如果 $f(x_1) = f(x_2)$, 则 $x_1 = x_2$

为了证明从 X 到 Y 的函数 f 是映上的, 必须表明对所有的 $y \in Y$, 存在 $x \in X$, 使得 $f(x) = y$

序列

定义域无限的序列称为无限序列。定义域有限的序列称为有限序列。

起始下标为 0 的无限序列记做 $\{x_n\}_{n=0}^{\infty}$

下标 n 作为定义域, x_n 为函数值

和符号

$$\sum_{i=m}^n a_i = a_m + a_{m+1} + \dots + a_n$$

乘积符号

$$\prod_{i=m}^n a_i = a_m \cdot a_{m+1} \cdot \dots \cdot a_n$$

i 称为下标, m 称为下限, n 称为上限

不含任何元素的串称为空串(null string), 用 λ 表示。用 X^* 表示 X 上所有的串的集合, 其中包含空串。用 X^+ 表示 X 上所有非空串的集合。

串中重复的字符可以用上标表示。例如, 串 $bbaaac$ 可以写做 b^2a^3c

如果 $\alpha = aabab$, $\beta = a3b4a32$, 则

$|\alpha| = 5$, $|\beta| = 9$

如果 α 和 β 是两个串, 由 α 接在 β 之后构成的串称为 α 和 β 的毗连, 记为 $\alpha \beta$

点、线、面

线段、射线、直线

仿射: 在仿射变换中, 指当物体移动时, 其形状、特点保持不变。

2D 空间中两点间距离公式

$$P_1P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

3D 空间中两点间距离公式

$$P_1P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

2D 空间中点公式

$$M((x_1+x_2)/2, (y_1+y_2)/2)$$

3D 空间中点公式

$$M((x_1+x_2)/2, (y_1+y_2)/2, (z_1+z_2)/2)$$

直线方程

斜率 $k = \tan\alpha = (y_2 - y_1)/(x_2 - x_1)$

一般式 $Ax + By + C = 0$ (其中 A、B 不同时为 0)

点斜式 $y - y_0 = k(x - x_0)$

斜截式 $y = kx + b$ 在 y 轴上的截距为 b, 即过(0, b)

截距式 $bx + ay - ab = 0$ 过(a, 0) (0, b)两点坐标

当 a、b 都不为 0 时, 截距式可写为 $x/a + y/b = 1$

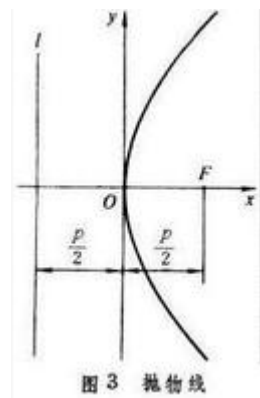
两点式 $(y - y_1)/(y_1 - y_2) = (x - x_1)/(x_1 - x_2)$ (斜率 k 需存在)

法线式 $x\cos\theta + y\sin\theta - p = 0$ (θ 为法线也 x 轴正方向的夹角, p 为原点到垂足的距离)

如果两直线垂直, 则它们的斜率乘积为-1

可能通过解直线方程组来求交点

抛物线



此图抛物线方程 $y^2 = 2px$

焦点 $(p/2, 0)$

准线方程 $x = -p/2$

顶点 $(0, 0)$

$2p$ 越大开口越口

抛物线离心率 $e=1$

离心率定义: 在圆锥曲线中, 动点到焦点的距离和动点到准线的距离之比。

抛物线定义: 平面内, 到一个定点 F 和不过 F 的一条定直线 l 距离相等的点的轨迹(或集合)称

之为抛物线。且定点 F 不在直线上另外 ,F 称为"抛物线的焦点", l 称为"抛物线的准线"。定义焦点到抛物线的准线的距离为"焦准距",用 p 表示 $p>0$ 。

右开口抛物线 $y^2 = 2px$

左开口抛物线 $y^2 = -2px$

上开口抛物线 $x^2 = 2py$

下开口抛物线 $x^2 = -2py$

圆的定义有两个

其一叫集合定义法：在平面内，到定点距离等于定长的点的集合。

其二叫形成定义法：平面上一条线段，绕它的一点旋转 360° ，它的另一端经过的轨迹叫圆。

圆周长：圆的周长与直径的比值。

圆弧和弦：圆上任意两点间的部分叫做圆弧，简称弧(arc)。大于半圆的弧称为**优弧**，小于半圆的弧称为**劣弧**。连接圆上任意两点的线段叫做弦(chord)。

圆心角和圆周角：顶点在圆心上的角叫做圆心角。顶点在圆周上，且它的两边分别与圆有另一个交点的角叫做圆周角。

内心和外心：和三角形三边都相切的圆叫做这个三角形的内切圆，其圆心称为内心。过三角形的三个顶点的圆叫做三角形的外接圆，其圆心叫做三角形的外心。

以 $O(a, b)$ 为**圆心**

标准方程 $(x-a)^2 + (y-b)^2 = r^2$

一般方程 $x^2 + y^2 + Dx + Ey + F = 0$

参数方程 $x = a + r \cdot \cos\theta, \quad y = b + r \cdot \sin\theta$

圆的离心率 $e=0$ (半焦距/长半轴)

周长 $C = 2\pi r$

面积 $S = \pi r^2$

球的方程 $(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$ (球心 $O(a, b, c)$)

球体积公式 $V = (4/3)\pi R^3$

球表面积公式 $S = 4\pi R^2$

椭圆第一定义

椭圆是平面上到两定点的距离之和为常值的点之轨迹。

椭圆第二定义

平面上到定点 F 的距离与到定直线的距离之比为常数 e(即椭圆的离心率, $0 < e = c/a < 1$)的点的集合。

c 为中心点到焦点的距离。

两定点的距离称为**焦距**。表示为 $2c$

定直线称为**准线**。

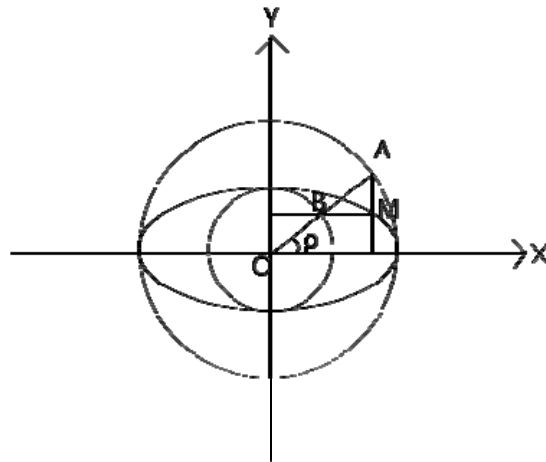
焦点在 x 轴上时, 准线方程为 $x = a^2/c$ 或 $x = -a^2/c$

焦点在 y 轴上时, 准线方程为 $y = a^2/c$ 或 $y = -a^2/c$

离心率为 $e = c/a$ ($0 < e < 1$)

椭圆标准方程

参数方程=>普通方程



几何定义

焦点在 X 轴上

$$x = a \cos \rho$$

$$y = b \sin \rho$$

=>

$$x^2/a^2 + y^2/b^2 = 1$$

焦点在 Y 轴上

$$x = b \cos \rho$$

$$y = a \sin \rho$$

=>

$$y^2/a^2 + x^2/b^2 = 1$$

a 为长半轴长, b 为短半轴长

三角形面积公式

标准公式 $S = 1/2 * \text{底} * \text{高}$

$$\text{三角函数公式 } S = \frac{1}{2} ab \sin C = \frac{1}{2} bc \sin A = \frac{1}{2} ac \sin B$$

外接圆公式 $S = abc/4R$ (其中 R 是三角形外接圆半径)

海伦公式 $S = \sqrt{p(p-a)(p-b)(p-c)}$ 公式里的 p 为半周长:

$$p = (a+b+c)/2$$

海伦公式证明

$$\cos C = (a^2 + b^2 - c^2) / 2ab$$

$$S = 1/2 * ab * \sin C$$

$$= 1/2 * ab * \sqrt{1 - (\cos C)^2}$$

$$= 1/2 * ab * \sqrt{[1 - (a^2 + b^2 - c^2)^2 / 4a^2 b^2]}$$

$$= 1/4 * \sqrt{4a^2 b^2 - (a^2 + b^2 - c^2)^2}$$

$$= 1/4 * \sqrt{(2ab + a^2 + b^2 - c^2)(2ab - a^2 - b^2 + c^2)}$$

$$= 1/4 * \sqrt{(a+b)^2 - c^2)(c^2 - (a-b)^2)}$$

$$= 1/4 * \sqrt{(a+b+c)(a+b-c)(a-b+c)(-a+b+c)}$$

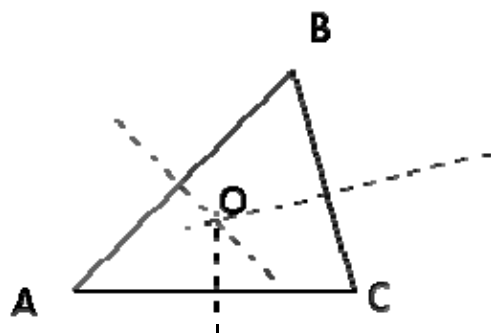
设 $p = (a+b+c)/2$

则 $p = (a+b+c)/2$, $p-a = (-a+b+c)/2$, $p-b = (a-b+c)/2$, $p-c = (a+b-c)/2$

$$\text{上式} = \sqrt{(a+b+c)(a+b-c)(a-b+c)(-a+b+c)/16}$$

$$= \sqrt{p(p-a)(p-b)(p-c)}$$

★以圆或球作为最外层的碰撞检测, 如果产生碰撞再进行内层碰撞检测。



三角形三边的中垂线相交于一点, 且交点离三个顶点的距离都相等。

证明:

∵ O 点在 AB 的中垂线上

∴ OA = OB

同理, OB = OC

∴ OA = OC, 即 O 点也在 AC 的中垂线上。

证毕。

三角函数

从 X 轴正方向开始逆时针旋转的角都是**正角**, 顺时针旋转的角都是**负角**。

弧度定义: $360^\circ = \text{周长}/\text{半径} (\text{弧度})$ ∵ $C = 2\pi r$ ∴ $360^\circ \text{角} = 2\pi \text{弧度}$

$1^\circ (\text{角度}) = \pi/180 (\text{弧度})$

$1 (\text{弧度}) = 180/\pi (\text{角度})$

弧度单位 **rad**

正弦 sin、余弦 cos、正切 tan、余割 csc、正割 sec、余切 cot

正割与余弦互为倒数, 余割与正弦互为倒数, 正切与余切互为倒数。

$\sec\theta = 1/\cos\theta$

$\csc\theta = 1/\sin\theta$

$\cot\theta = 1/\tan\theta$

反函数

$\sin\theta^{-1}$ 或 $\arcsin\theta$

$\cos\theta^{-1}$ 或 $\arccos\theta$

$\tan\theta^{-1}$ 或 $\arctan\theta$

正弦函数 $y = A\sin(\omega x + \varphi)$

φ 为向左平移值

ω 越大, 频率越高, 周期越小, 反之亦然。

A 越大，振幅越大，反之亦然。

正弦函数的周期为 $2\pi/\omega$

$y=\sin x$ 变 $y=A\sin(\omega x+\phi)$

步骤:

1. 沿 X 轴向左平移 ϕ 个单位
2. 纵坐标不变，横坐标乘以 $1/\omega$
3. 横坐标不变，纵坐标乘以 A

单位圆是以原点为圆心，1 为半径的一个圆。方程为 $x^2 + y^2 = 1$

在单位圆中 $x=\cos\alpha$ $y=\sin\alpha$

$$\cos^2\alpha + \sin^2\alpha = 1$$

$$\tan\alpha = \sin\alpha/\cos\alpha$$

$$\cot\alpha = \cos\alpha/\sin\alpha$$

相反数角性质

$$\sin(-\alpha) = -\sin\alpha$$

$$\cos(-\alpha) = \cos\alpha$$

$$\tan(-\alpha) = -\tan\alpha$$

正弦函数和差性质

$$\sin(\alpha_1 + \alpha_2) = \sin\alpha_1\cos\alpha_2 + \cos\alpha_1\sin\alpha_2$$

$$\sin(\alpha_1 - \alpha_2) = \sin\alpha_1\cos\alpha_2 - \cos\alpha_1\sin\alpha_2$$

余弦函数和差性质

$$\cos(\alpha_1 + \alpha_2) = \cos\alpha_1\cos\alpha_2 - \sin\alpha_1\sin\alpha_2$$

$$\cos(\alpha_1 - \alpha_2) = \cos\alpha_1\cos\alpha_2 + \sin\alpha_1\sin\alpha_2$$

三角形正弦定理

$$\text{正弦定理: } \frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$$

$$\text{变型: } a = 2R\sin A, b = 2R\sin B, c = 2R\sin C$$

$$a : b : c = \sin A : \sin B : \sin C$$

正弦定理:

在一个三角形中, 各边和它所对角的正弦的比相等, 即

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}$$

证明: 过A作单位向量 \vec{j} 垂直于 \overrightarrow{AC}

$$\text{由 } \overrightarrow{AC} + \overrightarrow{CB} = \overrightarrow{AB}$$

两边同乘以单位向量 \vec{j} 得 $\vec{j} \cdot (\overrightarrow{AC} + \overrightarrow{CB}) = \vec{j} \cdot \overrightarrow{AB}$.

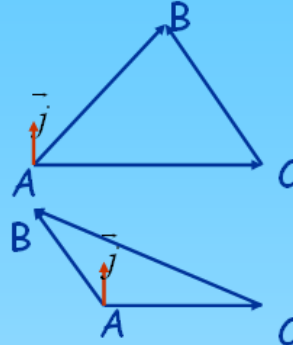
$$\text{则 } \vec{j} \cdot \overrightarrow{AC} + \vec{j} \cdot \overrightarrow{CB} = \vec{j} \cdot \overrightarrow{AB}.$$

$$\therefore |\vec{j}| |\overrightarrow{AC}| \cos 90^\circ + |\vec{j}| |\overrightarrow{CB}| \cos(90^\circ - C) = |\vec{j}| |\overrightarrow{AB}| \cos(90^\circ - A)$$

$$\therefore a \sin C = c \sin A. \quad \therefore \frac{a}{\sin A} = \frac{c}{\sin C}$$

同理, 过点C作与 \overrightarrow{CB} 垂直的单位向量 \vec{j} , 可得 $\frac{c}{\sin C} = \frac{b}{\sin B}$

$$\therefore \frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C}.$$



三角形余弦定理

证明

向量法

若 $\triangle ABC$ 为任意三角形, 已知角C, $BC=a, CA=b$, 求证:

$$c^2 = a^2 + b^2 - 2ab \cos C$$

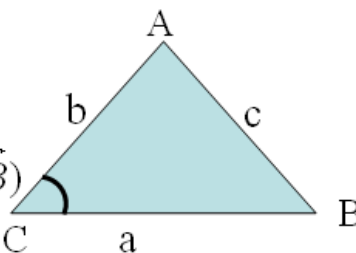
证明: $\therefore \overrightarrow{AB} = \overrightarrow{AC} + \overrightarrow{CB}$

$$\therefore \overrightarrow{AB} \cdot \overrightarrow{AB} = (\overrightarrow{AC} + \overrightarrow{CB}) \cdot (\overrightarrow{AC} + \overrightarrow{CB})$$

$$= \overrightarrow{AC} \cdot \overrightarrow{AC} + 2\overrightarrow{AC} \cdot \overrightarrow{CB} + \overrightarrow{CB} \cdot \overrightarrow{CB}$$

$$\therefore |\overrightarrow{AB}|^2 = |\overrightarrow{AC}|^2 + 2|\overrightarrow{AC}||\overrightarrow{CB}| \cos(180^\circ - C) + |\overrightarrow{CB}|^2$$

$$\therefore c^2 = a^2 + b^2 - 2ab \cos C$$



★我们在写程序时, 经常预先建立三角函数查询表, 这样可以减轻 CPU 计算负担。

★计算机对除法的计算比乘法算起来更麻烦

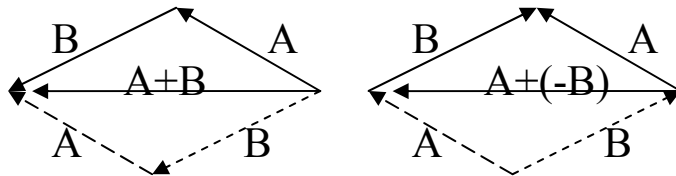
向量

向量在直角坐标系中的坐标表示 $A = xi + yj + zk$ (i, j, k 为单位向量, 分别指向坐标轴正方向)

向量加法几何表示: 三角形法测、平行四边形法测

从图可以看出, 满足[加法交换律](#)

$$A+B = B+A \quad |A+B| \neq |A|+|B|$$



标量乘向量 $3A = 3(xi + yj) = 3xi + 3yj$

[向量单位化](#): 就是指向量除以它的模

单位化后的向量, 记作 A^{\wedge} (顶上加个^)

计算向量的模 $|A| = \sqrt{x^2 + y^2 + z^2}$

向量点乘定义: $A \bullet B = |A||B|\cos \theta$ (θ 为 A, B 两相量夹角), $|B|\cos \theta$ 可看作在 A 上的投影长度。

向量的[点乘](#)是一个标量 $A \bullet B = x_1x_2 + y_1y_2 + z_1z_2$

注意: 维度相同的两向量才能相乘(点乘、叉乘)

如果 $A \bullet B = 0$, 则 $A \perp B$

如果 $A \bullet B < 0$, 那么 $\theta > 90^\circ$; 如果 $A \bullet B > 0$, 那么 $\theta < 90^\circ$, 其中, θ 是 A, B 两向量之间的夹角。

注意: (1) 当且仅当两个非零向量 \vec{a} 与 \vec{b} 同方向时, $\theta = 0$

(2) 当且仅当 \vec{a} 与 \vec{b} 反方向时 $\theta = \pi$

(3) 同时 $\vec{0}$ 与其它任何非零向量之间不谈夹角这一问题。

(4) [结合律不成立](#): $\vec{a} \cdot (\vec{b} \cdot \vec{c}) \neq (\vec{a} \cdot \vec{b}) \cdot \vec{c}$;

(5) [消去律不成立](#) $\vec{a} \cdot \vec{b} = \vec{a} \cdot \vec{c}$ 不能得到 $\vec{b} = \vec{c}$ 。

(6) $\vec{a} \cdot \vec{b} = 0$ 不能得到 $\vec{a} = \vec{0}$ 或 $\vec{b} = \vec{0}$

叉乘

设向量 $A = [a_1, a_2, a_3], B = [b_1, b_2, b_3]$, 则有

$$A \times B = [(a_2b_3 - a_3b_2) \quad (a_3b_1 - a_1b_3) \quad (a_1b_2 - a_2b_1)]$$

$$|A \times B| = |A||B|\sin \theta \quad (\text{此公式可用来求两相量夹角})$$

$A \times B$ 所得的向量同时垂直于 A 和 B

$A \times B$ 所得向量的方向可以用右手定则来判断, 右手放在 A, B 的交点处, 食指指向 A , 曲折其他手指指向 B , 这时候拇指指向 $A \times B$

$A \times B \neq B \times A$ 实际上, $A \times B = -(B \times A)$

[面垂直单位](#)是垂直于这个面, 而且长度为 1 的向量

面垂直单位的计算 $(A \times B) / |A \times B|$

$$(A \hat{\times} B) = \frac{A \times B}{\|A \times B\|},$$

二重叉乘公式

$a \times (b \times c) = b.(a.c) - a.(b.c)$ 利用坐标公式推导

★在编程时利用向量可以计算物体是否在视野中

矩阵

全等矩阵: 两个矩阵的维度相等, 而且所有对应元素也相等。

矩阵相加或相减: 把两矩阵中对应元素相加或相减(注意:两矩阵的维度一定要相同)

矩阵与标量相乘: 用标量乘以矩阵中每个位置的元素。

解矩阵方程

$$2X = 3A - B$$

和解代数式一样去解就行了。

矩阵与矩阵相乘

$$AB = \begin{bmatrix} (a_{00}b_{00} + a_{01}b_{10}) & (a_{00}b_{01} + a_{01}b_{11}) \\ (a_{10}b_{00} + a_{11}b_{10}) & (a_{10}b_{01} + a_{11}b_{11}) \end{bmatrix}$$

$$\text{其中, } A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}, B = \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix}.$$

注意: 第一个矩阵的列数一定要等于第二个矩阵的行数。(这也是矩阵相乘不满足交换律的原因)

转置矩阵: 行列互换

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}, A^T = \begin{bmatrix} a_{00} & a_{10} & a_{20} \\ a_{01} & a_{11} & a_{21} \\ a_{02} & a_{12} & a_{22} \end{bmatrix}.$$

转置后维度的行列互换

利用矩阵加法进行 2D 平移

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

利用矩阵加法进行 3D 平移

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}$$

利用乘法可以进行平移、放缩、旋转(参见“齐次坐标”)

利用齐次坐标进行变换:

正角向逆时针方向旋转, 负角则向顺时针方向旋转。

绕 Z 轴旋转(Z 坐标不变)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

绕 X、Y 轴旋转同理

放缩

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

平移

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

为了使物体绕注册点变换(主要是放缩、旋转), 需要进行以下三步:

1. 把物体注册点平移到原点
2. 进行变换
3. 把物体平移回原位置

串联矩阵: 多种变换一次计算

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_c \\ 0 & 1 & 0 & y_c \\ 0 & 0 & 1 & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_c \\ 0 & 1 & 0 & -y_c \\ 0 & 0 & 1 & -z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

利用串联矩阵可以一次进行 3D 绕 3 个轴旋转, 如

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 90^\circ & 0 & \sin 90^\circ & 0 \\ 0 & 1 & 0 & 0 \\ -\sin 90^\circ & 0 & \cos 90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 180^\circ & -\sin 180^\circ & 0 \\ 0 & \sin 180^\circ & \cos 180^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 30^\circ & -\sin 30^\circ & 0 & 0 \\ \sin 30^\circ & \cos 30^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

2D 串联矩阵

$$\begin{bmatrix} r_{00} & r_{01} & t_x \\ r_{10} & r_{11} & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad r \text{ 存放旋转信息, } t \text{ 存放平移信息}$$

3D 串联矩阵

$$\begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad r \text{ 存放旋转信息, } t \text{ 存放平移信息}$$

DirectX 用单行矩阵表示顶点, OpenGL 用单列矩阵表示顶点。

矩阵与转置的关系 $AB = A^T B^T$ (此关系可以统一 DirectX 与 OpenGL 的计算)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & t_x \\ r_{10} & r_{11} & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad [x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} r_{00} & r_{01} & 0 \\ r_{10} & r_{11} & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

注意: 用单行表示顶点, 计算会比较简单。数组存储顶点也方便。

数列

递推关系和初始条件唯一地确定了一个序列。

迭代法求解递推关系

斐波那契数列

初始条件 $a_1=1, a_2=1$

递推关系 $a_n = a_{n-1} + a_{n-2} \quad (n \geq 3)$

岛上兔子问题:

一对刚出生的兔子(一公一母)被放到岛上。每对兔子出生后两个月才开始繁殖后代。在出生两个月以后,每对兔子在每个月都将繁殖一对新兔子。假定兔子不会死去,找出 n 个月后关于岛上兔子对数的递推关系。

算术基本定理

引理 1

任何大于 1 的正整数 n 可以写成素数之积, 即 $n=p_1p_2\dots p_m$ 其中 $p_i(1 \leq i \leq m)$ 是素数。

证明:

当 $n=2$ 时, 结论显然成立。

假设对于 $2 \leq n \leq k$, 式(1)成立, 我们来证明式(1)对于 $n=k+1$ 也成立, 从而由归纳法推出式(1)对任何大于 1 的整数 n 成立。

如果 $k+1$ 是素数, 式(1)显然成立。

如果 $k+1$ 是合数, 则存在素数 p 与整数 d , 使得 $k+1=pd$ 。

由于 $2 \leq d \leq k$, 由归纳假定知存在素数 q_1, q_2, \dots, q_l 使得 $d=q_1q_2\dots q_l$, 从而 $k+1=pq_1q_2\dots q_l$

证毕。

游戏中的物理学

透视公式

$scale = fl / (fl + z)$ fl : 焦距(通常[200, 300]), z : 远近度

运动公式

$$\overline{v} = \frac{v_t + v_0}{2}$$

$$v_t = v_0 + at$$

$$s = 1/2(v_t + v_0)t$$

$$s = v_0t + 1/2at^2$$

$$v_t^2 = v_0^2 + 2aS$$

静摩擦力 $F = \mu_0 N$ (μ_0 最大静摩擦系数 N 正压力)

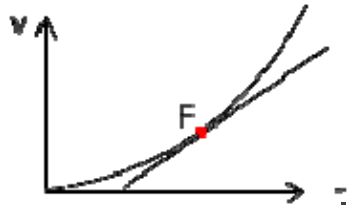
动摩擦力 $F = \mu N$ (μ 动摩擦系数 N 正压力)

最大静摩擦系数总大于动摩擦系数

对位移—时间方程求导, 可计算瞬时速度。

$$f'(2) = \lim_{\Delta t \rightarrow 0} \frac{f(2+\Delta t) - f(2)}{\Delta t}$$

速度—时间图像中



连接两点直线的斜率即为平均速度，某点的切线斜率即为瞬时加速度。

对位移时间函数连续两次求导可得瞬时加速度

牛顿三定律：

惯性定律

$$F=ma$$

作用力与反作用力

功 $W = Fs$ (功 = 合力 \times 合力方向上的位移)

功不能算是物体上的能量，它表示机械能的转移理

动量 $p=mv$ (p 为矢量)

冲量: $Ft = mv' - mv_0$

动量定理 $Ft=mv_2-mv_1$ 反映了力对时间的累积效应，是力在时间上的积分。

动量守恒 $mv_1' + mv_2' = mv_1 + mv_2$ (动量只是从一个物体传递到了另一个物体)

动量守恒适用于无外力参与的碰撞

根据牛顿第三定律

$$F_1=-F_2 \text{ (作用力与反作用力)}$$

$$mv_1' - mv_1 = -(mv_2' - mv_2)$$

$$mv_1' + mv_2' = mv_1 + mv_2$$

弹性碰撞是一种在碰撞时没有动能损失的碰撞，非弹性碰撞则有功能损耗。完全非弹性碰撞，指两个物体碰撞之后，不再分开并以相同的速度运动(动能损失最大)。

动能定理 $Fs=1/2mv^2-1/2mv_0^2$ 反映了力对空间的累积效应，是力在空间上的积分。

重力势能 $E = Gh = mgh$ (h 为离地面的高度)

势能又叫位能，是由于物体特殊位置所具有的能量。**弹性势能**也如此。

能量守恒定律: 能量既不会凭空产生，也不会凭空消失，它只能从一种形式转化为其他形式，或者从一个物体转移到另一个物体，在转化或转移的过程中，能量的总量不变。

机械能守恒定律: 在只有重力或弹力对物体做功的条件下(或者无外力做功，或者外力做功之和为零)，物体的动能和势能（包括重力势能和弹性势能）发生相互转化，但机械能的总量保持不变。

汽车在滑行时，机械能逐渐转化成内能(热能)或声波，而最终停下来。

质量: 抵抗线性运动的物理量

转动惯量: 物体抵抗转动的物理量

$$\text{质量} = \text{密度} \times \text{体积} \quad (m = \rho \times V)$$

总质量 = 各离散质点质量之和 ($m_t = \sum m_i$)

空间重心坐标计算:

设 n 个质点的质量分别是: $M_1, M_2, M_3, M_4, \dots, M_n$

那么, 重心的质量是: $M_1+M_2+M_3+M_4+\dots+M_n$

根据杠杆平衡条件, 得

$$(M_1+M_2+M_3+M_4+\dots+M_n) X_g = M_1 \cdot X_1 + M_2 \cdot X_2 + M_3 \cdot X_3 + \dots + M_n \cdot X_n$$

$$(M_1+M_2+M_3+M_4+\dots+M_n) Y_g = M_1 \cdot Y_1 + M_2 \cdot Y_2 + M_3 \cdot Y_3 + \dots + M_n \cdot Y_n$$

$$(M_1+M_2+M_3+M_4+\dots+M_n) Z_g = M_1 \cdot Z_1 + M_2 \cdot Z_2 + M_3 \cdot Z_3 + \dots + M_n \cdot Z_n$$

从而

$$X_g = (M_1 \cdot X_1 + M_2 \cdot X_2 + M_3 \cdot X_3 + \dots + M_n \cdot X_n) / (M_1 + M_2 + M_3 + M_4 + \dots + M_n)$$

$$Y_g = (M_1 \cdot Y_1 + M_2 \cdot Y_2 + M_3 \cdot Y_3 + \dots + M_n \cdot Y_n) / (M_1 + M_2 + M_3 + M_4 + \dots + M_n)$$

$$Z_g = (M_1 \cdot Z_1 + M_2 \cdot Z_2 + M_3 \cdot Z_3 + \dots + M_n \cdot Z_n) / (M_1 + M_2 + M_3 + M_4 + \dots + M_n)$$

旋转运动

旋转运动是线性运动概念的扩展, 公式的形式都一样, 只不过里面使用的是旋转运动的量。

旋转角度常用弧度表示 θ (圆心角) = 弧长/半径 (弧长对应的圆周角是圆心角的一半, 这里的半径是指重心到质点的距离)

平均角速度(θ 为角位移, 单位 rad)

$$\bar{\omega} = \frac{\Delta \theta}{\Delta t} \quad \bar{\omega} = \frac{\omega_t + \omega_0}{2}$$

平均角加速度(ω 为角速度)

$$\bar{\alpha} = \frac{\Delta \omega}{\Delta t}$$

以下公式中的 α 为角加速度

末角速度 $\omega_t = \omega_0 + \alpha t$

角位移 $\Delta \theta = \Delta s / r$ (Δs 为弧长, r 为半径)

角速度 $\omega = \Delta \theta / \Delta t = (\Delta s / r) / \Delta t = v_t / r$ (v_t 为切向速度)

$v_t = \omega r$

切向角加速度 $a_t = v_t / \Delta t = \omega r / \Delta t = \alpha r$ (α 为瞬时角加速度, r 为半径)

角位移 $\Delta \theta = (1/2)(\omega_t + \omega_0)/t$

角位移 $\Delta \theta = \omega_0 t + (1/2) \alpha t^2$

$$\omega_t^2 = \omega_0^2 + 2\alpha \Delta \theta$$

力矩(torque, 转矩)公式 $M = L \times F$ (L 力臂, F 垂直于 L 的力)

转矩 $t = F \times r = m a r = m r^2 \alpha$

旋转惯量 $I = m r^2$

所以转矩可写成 $t = I \alpha$ (I 是旋转惯量, α 是角加速度)

就像力产生加速度一样, 力矩产生角加速度。如果我们知道了角加速度就可以利用方程求出角速度和角位移。

旋转动能 $E_R = (1/2) I \omega^2$

知道旋转动能之后, 在利用能量守恒时, 应该把势能、线性动能、旋转动能都考虑进去。

旋转动量 $L = I \omega$ (I 是旋转惯量, ω 角速度)

★在编程中，动量是处理碰撞的根本之所在。

★在编程中，经常用还原系数 ϵ 来控制能量损失 ($0 \leq \epsilon \leq 1$)，在台球游戏中，只有球心连线方向上的动量才会传递。(即动量守恒只适用于对心碰撞——线性碰撞)，非对心碰撞下，会导致球改变方向以及产生旋转。

★数学编程的技巧就在于充分理解方程中的每一个量，然后编写一个能使 **cpu** 尽可能快速计算的程序。

数据结构与算法

排序

直接插入排序

最坏时间效率 $O(n^2)$ ，直接排序是稳定的。

例：

```
var list:Array = [75, 87, 68, 92, 88, 61, 77, 96, 80, 72];
function insertSort( arr:Array ):void
{
    var i:int, j:int;
    var len:uint = arr.length;
    for(i=0; i<len; i++){
        var tmp:uint = arr[i];
        j = i-1;
        while(j>=0 && tmp<arr[j])/* 元素后移,以便腾出一个位置插入 tmp */
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = tmp; /* 在 j+1 位置处插入 tmp */
    }
}
insertSort(list);
trace(list);// 61,68,72,75,77,80,87,88,92,96
```

希尔排序

希尔(Shell)排序又称为缩小增量排序方法，其基本思想是：把记录按下标的一定增量 d 分组，对每组记录采用直接插入排序方法进行排序，随着增量逐渐减小，所分成的组包含的记录越来越多，到增量的值减小到 1 时，整个数据合成为一组，构成一组有序记录，则完成排序。时间复杂度约为 $O(n^{1.3})$ ，是一种不稳定的排序方法。

例:

```
var list:Array = [75, 87, 68, 92, 88, 61, 77, 96, 80, 72];
function shellSort( arr:Array ):void
{
    var i:int, j:int, tmp:int, gap:int;
    var n:int = arr.length;
    gap = n/2; //增量置初值
    while(gap > 0){
        for(i=gap; i<n; i++){//对所有相隔 gap 位置的元素组进行排序
            tmp = arr[i];
            j = i-gap;
            while(j>=0 && tmp < arr[j]){
                arr[j+gap] = arr[j];
                j = j-gap; //移到本组中的前一个元素
            }
            arr[j+gap] = tmp;
            j = j-gap;
        }
        gap = gap/2; //减小增量
    }
}
shellSort(list);
trace(list);
```

直接选择排序

时间复杂度 $O(n^2)$, 此排序是不稳定的。

例:

```
var list:Array = [75, 87, 68, 92, 88, 61, 77, 96, 80, 72];
function selectSort( arr:Array ):void
{
    var i:int, j:int, k:int, tmp:int;
    var n:uint = arr.length;
    for(i=0; i<n-1; i++){
        k=i;
        for(j=i+1; j<n; j++){
            if(arr[j] < arr[k]){
                k=j; //用 k 指出每趟在无序区段的最小元素
            }
        }
        //将 arr[k]与 arr[j]交换
        tmp = arr[i];
        arr[i] = arr[k];
        arr[k] = tmp;
    }
}
```



```
}  
selectSort(list);  
trace(list);
```

归并排序

归并排序的基本思想是：首先将 $R[0..n-1]$ 看成是 n 个长度为 1 的有序表，将相邻的有序表成对归并，得到 $n/2$ 个长度为 2 的有序表；然后将这些有序表成对归并，得到 $n/4$ 个长度为 4 的有序表，如此反复进行下去，最后得到一个长度为 n 的有序表。

由于归并是在相邻的两个有序表中进行的，因此，上述排序方法也称为二路归并排序。如果归并操作在相邻的多个有序表中进行，则称为多路归并排序。

归并排序时间复杂度为 $O(n\log_2 n)$ ，是稳定的排序。

例：

```
var list:Array = [75, 87, 68, 92, 88, 61, 77, 96, 80, 72];  
/**  
 * merge 实现了一次归并  
 */  
function merge( R:Array, low:uint, mid:uint, high:uint ):void  
{  
    var R1:Array = [];  
    var i:uint=low, j:uint=mid+1, k:uint=0;//k 是 R1 的下标，i、j 分别为第 1、2 段的下标  
    while(i<=mid && j<=high){//在第 1 段和第 2 段均未扫描完时循环  
        if(R[i] <= R[j]){//将第 1 段中的记录放入 R1 中  
            R1[k] = R[i];  
            i++; k++;  
        }else{//将第 2 段中的记录放入 R1 中  
            R1[k] = R[j];  
            j++; k++;  
        }  
    }  
    //将第 1 段余下的部分复制到 R1  
    while(i<=mid){  
        R1[k] = R[i];  
        i++; k++;  
    }  
    //将第 2 段余下的部分复制到 R1  
    while(j<=high){  
        R1[k] = R[j];  
        j++; k++;  
    }  
    //将 R1 复制回 R 中  
    for(k=0, i=low; i<=high; k++, i++){  
        R[i] = R1[k];  
    }
```

```

    }
}
/**
 * mergePass 解决一趟归并
 */
function mergePass( R:Array, length:uint, n:uint ):void
{
    var i:uint;
    for(i=0; i+2*length-1<n; i=i+2*length){//归并 length 长的两个相邻子表
        merge(R, i, i+length-1, i+2*length-1);
    }
    if(i+length-1 < n){//余下两个子表，后者长度小于 length
        merge(R, i, i+length-1, n-1);//归并这两个子表
    }
}
}
/**
 * 二路归并排序
 */
function mergeSort( R:Array ):void
{
    var n:uint = R.length;
    var length:uint;
    for(length=1; length<n; length=2*length){
        mergePass(R, length, n);
    }
}
mergeSort(list);
trace(list);//输出 61,68,72,75,77,80,87,88,92,96

```

二叉堆排序

堆是一棵顺序存储的完全二叉树，其中每个结点的关键字都不小于其孩子结点的关键字(称为大根堆，如果是进行从大到小的排序，则堆中每个结点的关键字都不大于其孩子结点的关键字，称为小根堆)。

堆排序的基本思想是：首先，按堆的定义将 $R[1..n]$ 调整为堆(这个过程称为初始建堆)，交换 $R[1]$ 和 $R[n]$ ；然后将 $R[1..n-1]$ 调整为堆，交换 $R[1]$ 和 $R[n-1]$ ；如此反复进行，直到交换了 $R[1]$ 和 $R[2]$ 为止。

堆排序在最坏情况下所需的比较次数不超过 $O(n\log_2 n)$ ，显然，元素的移动次数也不超过 $O(n\log_2 n)$ 。堆排序是不稳定的。

例：

```

var list:Array = [null, 75, 87, 68, 92, 88, 61, 77, 96, 80, 72];//有效数据从 list[1]开始
/**
 * 筛选过程
 */

```

```

function sift( R:Array, low:int, high:int ):void
{
    var i:int=low, j=2*i;//R[j]是 R[i]的左孩子
    var tmp:int=R[i];
    while(j<=high){
        if(j<high && R[j]<R[j+1]){//若右孩子较大，把 j 指向右孩子
            j++;//j 变为 2i+1,指向右孩子结点
        }
        if(tmp<R[j]){
            R[i]=R[j];//将 R[j]调整到双亲结点位置上
            i=j;//修改 i 和 j 的值，以便继续向下筛选
            j=2*i;
        }else{
            break;//筛选结束
        }
    }
    R[i] = tmp;//被筛选结点的值放入最终位置
}
/**
 * 堆排序
 */
function heapSort( R:Array ):void
{
    var i:int;
    var n:int = R.length-1;
    var tmp:int;
    for(i=n/2; i>=1; i--){//循环建立初始堆
        sift(R, i, n);
    }
    for(i=n; i>=2; i--){//进行 n-1 次循环，完成堆排序
        tmp=R[1];
        R[1]=R[i];//将第一个元素同当前区间内 R[1]对换
        R[i]=tmp;
        sift(R, 1, i-1);//筛选 R[1]结点，得到 i-1 个结点的堆
    }
}
heapSort(list);
trace(list);//输出,61,68,72,75,77,80,87,88,92,96

```

快速排序

快速排序是由冒泡排序改进而得到的，它的基本思想是：在待排序的 n 个记录中任取一个记录（通常取第 1 个记录），将该记录放入最终位置后，整个数据区间被此记录分割成两个子区间。所有关键字比该记录小的放置在前子区间中，所有比它大的放置在后子区间中，并把该记录排在这两个子区间中间，这个过程称为一趟快速排序。之后所有的两个子区间分别重

复上述过程，直至每个子区间内只有一个记录为止。简而言之，每趟排序使表的第 1 个元素入终位，将数据区间一分为二，对于子区间按递归方式继续这种划分，直到划分的子区间长为 1

快速排序的时间复杂度为 $O(n\log_2n)$ ，是不稳定的。

例：

```
var list:Array = [75, 87, 68, 92, 88, 61, 77, 96, 80, 72];
function quickSort( R:Array, s:int, t:int ):void
{
    var i:int=s, j:int=t;
    var tmp:int;
    if(s<t){//区间内至少存在一个元素的情况
        tmp=R[s];//用区间的第一个记录作为基准
        while(i!=j){//从区间两端交替向中间扫描，直至 i=j 为止
            while(j>i && R[j] > tmp){
                j--;//从右向左扫描，找第 1 个关键字小于 tmp 的 R[j]
            }
            R[i]=R[j];//找到这样的 R[j]，则 R[i]和 R[j]交换
            while(i<j && R[i] < tmp){
                i++;//从左向右扫描，找第 1 个关键字大于 tmp 的 R[i]
            }
            R[j]=R[i];//找到这样的 R[i]，则 R[i]和 R[j]交换
        }
        R[i]=tmp;//区间的第一个元素入终位
        quickSort(R, s, i-1);//对左区间递归排序
        quickSort(R, i+1, t);//对右区间递归排序
    }
}
quickSort(list, 0, list.length-1);
trace(list);// 61,68,72,75,77,80,87,88,92,96
```

基数排序(RadixSort)

基数排序不比较关键字的大小。它是根据关键字中各位的值，通过对待排序的 n 个元素进行若干趟“分配”与“收集”来实现排序的。

设待排序的线性表中每个元素的关键字都是 d 位的十进制正整数。在排序过程中需要对该线性表进行 d 趟的分配与收集处理，每趟处理方法是相同的。在进行第 j(j=1,2,...,d)趟处理时，首先按元素在线性表中的排列顺序，依次将每个元素插入到编号为 0~9 的某个队列(关键字右起第 j 位上的值是几就插入几号队列)，这个过程称为分配,然后按队列编号从小到大、同一队列按插入的先后顺序，从队列中取出所有元素，重新构成一个线性表，这个过程称为收集。在进行了 d 趟分配与收集之后，排序过程结束。

编号	关键字	分配→	10 个队列	收集→	关键字	分配→	10 个队列	收集	关键字
0	75		80		80				61
1	87		61		61				68
2	68		92 72		92				72

3	92				72				75
4	88				75				77
5	61		75		96				80
6	77		96		87		61 68		87
7	96		87 77		77		72 75 77		88
8	80		68 88		68		80 87 88		92
9	72				88		92 96		96

拓扑排序

拓扑排序是有向图的一种重要运算。设 $G=(V, E)$ 是一个具有 n 个顶点的有向图，在图中用顶点表示活动，用边表示活动间的优先关系，则这个有向图称为用顶点表示活动的网 (Activity On Vertex Network), 简称为 AOV 网。在 AOV 网中，如果从顶点 v_i 到顶点 v_j 有一条有向路径，则 v_i 是 v_j 的前驱， v_j 是 v_i 的后继；如果 $\langle v_i, v_j \rangle$ 是 AOV 网中的一条边，则 v_i 是 v_j 的直接前驱， v_j 是 v_i 的直接后继。

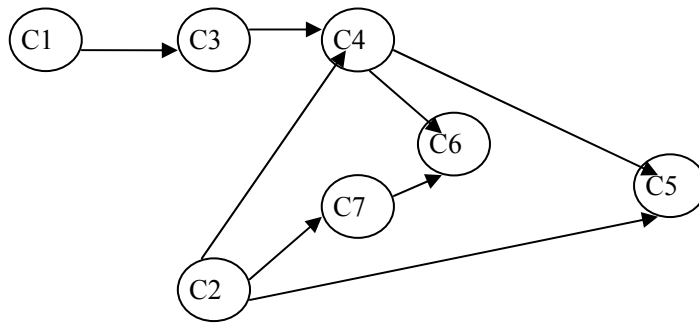
对于一个 AOV 网，构造其所有顶点的线性序列，建立顶点之间的先后关系，而且使原来没有先后关系的顶点之间也建立起人为的先后关系。这样的线性序列称为拓扑有序序列。构造 AOV 网的拓扑有序序列的运算称为[拓扑排序](#)。

AOV 网中顶点序列 v_1, v_2, \dots, v_n 称为一个拓扑有序序列，当且仅当该顶点序列满足下列条件：若 $\langle v_i, v_j \rangle$ 是图中的边(即从顶点 v_i 到 v_j 有一条路径)，则在序列中顶点 v_i 必须排在顶点 v_j 之前。

例：课程名称与相应代号的关系

课程代号	课程名称	选修课程
C1	高等数学	无
C2	程序设计	无
C3	离散数学	C1
C4	数据结构	C2, C3
C5	编译原理	C2, C4
C6	操作系统	C4, C7
C7	计算机组成原理	C2

课程之间先后关系可用有向图表示



对这个有向图进行拓扑排序可得到一个拓扑序列:C1-C3-C2-C4-C7-C6-C5。也可得到另一个拓扑序列: C2-C7-C1-C3-C4-C5-C6, 还可以得到其他的拓扑序列。学生按照任何一个拓扑序列都可以顺序地进行课程学习。

拓扑排序的方法如下:

- (1) 从有向图中选择一个没有前驱(即入度为 0)的顶点并且输出它。
- (2) 从网中删去该顶点, 并且删去从该顶点发出的全部边。
- (3) 重复上述两步, 直到剩余的网中不再存在没有前驱的顶点为止。

如果 AOV 网中存在环路, 顶点之间的先后关系进行了死循环, 则意味着某项活动应该以自身完成为先决条件, 这显然是不合理的, 所以, 求不出该网的拓扑有序序列; 反之, 任何无环路的有向图, 其所有顶点都可以排在一个拓扑有序序列中。

建立邻接表

邻接表是图的一种链式存储结构, 在邻接表中, 对图中每个顶点建立一个带头结点的单链表, 所有头结点构成一个数组, 第 i 个单链表中的结点表示依附于顶点 v_i 的边。单链表中每个结点(边结点)由两个域组成: **顶点域** `adjvex`, 用以指示该邻接点在头结点数组中的序号(下标): **链域** `next`, 用以指向依附于顶点 v_i 的下一条边所对应的结点。如果用邻接表存放网中的信息, 则还需要在结点中增加一个存放**权值**的域 `value`

一般来说, 有 n 个顶点、 e 条边的无向图采用邻接表存储时, 头结点数组元素个数为 n , 边结点个数为 $2e$; 有 n 个顶点、 e 条边的有向图采用邻接表存储时, 头结点数组元素个数为 n , 边结点个数为 e 。从中看到**所占空间与边数有关, 适合存储稀疏图**。

例:

```

package
{
    /**
     * 边结点
     */
    public class EdgeNode
    {
        public var adjvex:int;//邻接点序号
        public var value:int;//边的权值
    }
}
  
```

<pre> public var next:EdgeNode;//每个顶点建立的单链表中结点的类型 } } </pre>
<pre> //////////////////////////////////// package { /** * 头结点 */ public class HeadNode { public var data:String;//顶点信息 public var firstarc:EdgeNode;//指向第一条边结点 } } </pre>
<pre> //////////////////////////////////// package { /** * 头结点集合 */ public class AdjList { public var n:uint,e:uint;//n 为实际顶点数， e 为实际边数 public var adjlist:Array=[];//单链表头结点数组 } } </pre>
<pre> //////////////////////////////////// --Test-- ////////////////////////////////////// var n:uint = 5;//顶点数 var e:uint = 7;//边数 var vertexType:Array = ['a', 'b', 'c', 'd', 'e']; //建立有向图的输入数据 var link:Array = [{sn: 0, en: 1, val: 1}, {sn: 0, en: 2, val: 2}, {sn: 0, en: 3, val: 6}, {sn: 1, en: 3, val: 4}, {sn: 1, en: 4, val: 5}, {sn: 2, en: 4, val: 3}, {sn: 4, en: 3, val: 7}]; /** * 建立有向图邻接表 */ function createAdjList(G:AdjList):void </pre>

```

{
    var i:int, b:int, t:int, w:int;
    var p:EdgeNode;

    for(i=0; i<G.n; i++){
        G.adjlist[i] = new HeadNode();
        G.adjlist[i].data = vertexType[i];
        G.adjlist[i].firstarc = null;
    }
    for(i=0; i<G.e; i++){
        b = link[i].sn;//起点号
        t = link[i].en;//终点号
        w = link[i].val;//权值
        if(b<G.n && t<G.n && w>0){
            p = new EdgeNode();//建立 p 结点
            p.value = w;
            p.adjvex = t;
            p.next = G.adjlist[b].firstarc;//p 插入 adjlist[b]单链表之首
            G.adjlist[b].firstarc = p;
        }else{
            trace("输入错误!");
        }
    }
}
/**
 * 输出邻接表
 */
function dispAdjList( G:AdjList ):void
{
    var i:int;
    var p:EdgeNode;
    trace('图的邻接表表示如下:');
    for(i=0; i<G.n; i++){
        var s:String=[''+i+', '+G.adjlist[i].data+'=>'];
        p = G.adjlist[i].firstarc;
        while(p!=null){
            s += '('+p.adjvex+', '+p.value+'->'
            p = p.next;
        }
        s += '^';
        trace(s);
    }
}
//初始化有向图 G

```



```
var G:AdjList = new AdjList();
G.n = n;
G.e = e;
createAdjList(G);
dispAdjList(G);
//输出信息如下
图的邻接表表示如下:
[0,a]==>(3,6)->(2,2)->(1,1)->^
[1,b]==>(4,5)->(3,4)->^
[2,c]==>(4,3)->^
[3,d]==>^
[4,e]==>(3,7)->^
```

Alchemy

官方: http://blog.comtaste.com/2010/04/alchemy_compiling_cc_code_into_1.html
<http://labs.adobe.com/wiki/index.php/Alchemy>

start Alchemy:

http://labs.adobe.com/wiki/index.php/Alchemy:Documentation:Getting_Started

Alchemy C API:

http://labs.adobe.com/wiki/index.php/Alchemy:Documentation:Developing_with_Alchemy:C_API


Alchemy AS3 API:

http://labs.adobe.com/wiki/index.php/Alchemy:Documentation:Developing_with_Alchemy:AS3_API

Alchemy 项目配置

二、安装 Cygwin

下载地址 <http://labs.adobe.com/downloads/alchemy.html>

 Download the Alchemy Toolkit for Cygwin on Windows (ZIP, 37.6 MB)

选择要安装的包

1 到 Archive 下面去选择安装 zip 包



2 到 Devel 下面去选择安装 gcc-g++ 包

3.4.4-999  3.4.4-3 ☒ ☐ 2,958k gcc-g++: C++ compiler

3 到 Perl 目录,选择完全安装 (Install)

二、下载 Alchemy

下载地址 <http://labs.adobe.com/downloads/alchemy.html>

[Download LLVM-GCC 4.0-2.1 Sources for Alchemy](#)

三、配置 Java 环境变量

四、配置 Cygwin

1、修改 cygwin/etc/profile 文件

在安装目录下找到 etc/profile 文件中的 PATH 并添加路径配置(路径根据实际安装情况而定)

红色部分为新添加的路径配置

PATH="/usr/local/bin:/usr/bin:/cygdrive/e/software/flex_sdk_4/bin: \${PATH}"

注意 /cygdrive/e 这里的 e 代表 E 盘

2、切换到 alchemy 目录下,执行 ./config

```
Administrator@YANIS /cygdrive/e/software/alchemy_sdk_cygwin
$ ./config
Generating alchemy-setup...
Turning execution bit on for Alchemy binaries...

Add "source /cygdrive/e/software/alchemy_sdk_cygwin/alchemy-setup" to your login
script.
"alc-home" takes you to the Alchemy install folder.
"alc-on" puts Alchemy gcc toolchain replacements at the front of your path.
"alc-off" restores original path.
"alc-util" shows you various Alchemy-related environment vars
You need Flash 10 or AIR 1.5 and the Flex 3.2 SDK installed for testing.
```

3、执行 source 命令

source /cygdrive/e/software/alchemy/alchemy-setup

4、关闭 cygwin

5、修改 alchemy/alchemy-setup 文件

将#export ADL=/path/to/your/adl (or adl.exe)改为下面这句
export ADL=/cygdrive/e/software/flex_sdk_4/bin/adl.exe

6、修改 cygwin/etc/bash.bashrc 文件

在安装目录下找到 etc/bash.bashrc 文件,在最后加入以下三行

```
source /cygdrive/e/software/alchemy/alchemy-setup  
PATH=$ALCHEMY_HOME/achacks:/cygdrive/e/software/flex_sdk_4/bin:$PATH  
export PATH
```

7、切换到 alchemy/bin 目录

执行 `ln -s llvm-stub llvm-stub.exe`

8、执行 `alc-on;which gcc`

9、切换到工程目录(比如 Adobe 自带的 `e:/software/alchemy/samples/stringecho`)

10、执行 `gcc -O3 -Wall -swc stringecho.c -o stringecho.swc`

示例测试

HelloWorld.c

```
#include <stdio.h>  
#include "AS3.h"  
static AS3_Val returnString(){  
    char* text="hello world";  
    return AS3_String(text);  
}  
int main()  
{  
    AS3_Val cMethod=AS3_Function(NULL, returnString);  
    AS3_Val result=AS3_Object("returnString:AS3ValType", cMethod);  
    AS3_Release(cMethod);  
    AS3_LibInit(result);  
    return 0;  
}
```

将 HelloWorld.c 编译成 helloWorld.swc

创建 ActionSprite 项目，并添加 helloWorld.swc

```
package  
{  
    import module.helloWorld.CLibInit;  
    import flash.display.Sprite;  
    import flash.text.TextField;  
  
    /**
```

```

* 测试AS3调用Alchemy
*/
[SWF(backgroundColor='0xFFFFFF',frameRate='12', width='640',height='480')]
public class AlchemyTest extends Sprite
{
    public function AlchemyTest()
    {
        var loader:CLibInit = new CLibInit();
        var lo:Object = loader.init();
        var textFild:TextField = new TextField();
        textFild.width=100;
        textFild.height=30;
        addChild(textFild);
        textFild.text = lo.returnString();
    }
}
}

```

网上安装方法原文转载(<http://www.cnblogs.com/jiahuaful/archive/2011/04/07/2008601.html>)

Alchemy3D 是一款基于 Adobe Alchemy 技术的 Flash 3D 引擎，以下是其基本的开发环境：

Cygwin + Adobe Alchemy Toolkit + Flash Builder + Flex SDK

一、安装 Cygwin

<http://www.cygwin.com/setup.exe> 下载Cygwin

之后一路默认到选 package 的地方,注意几个地方:

- 1.是位置,Archive 下的 zip 包,Devel 下的 gcc / g++ ,Perl 全要下.
- 2.是 Perl 那个在文件外面点成 install 就可以了,不用展开来一个个选中了

二、下载 Alchemy

到 <http://labs.adobe.com/downloads/alchemy.html> 下载 Alchemy Toolkit

三、下载 Flex SDK

不解解释，大家都装了的

四、配置环境

到 www.java.com 去下载 JAVA 虚拟机，机器已经安装了 java，可略过此步。

解压 Adobe Flex SDK 到 c 盘根目录，重命名为 flex，注意 flex 目录下面为 bin 等目录。

解压 Alchemy 到 C 盘根目录，重命名为 Alchemy，注意 Alchemy 目录下面直接为 bin 等目录。

修改 C:\cygwin\etc 目录下的 profile 文件，将 flex sdk 的 bin 目录加入到 cygwin 的 path 中。

像这样

```
PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/cygdrive/c/flex/bin:$PATH
```

红色部分为新添加的。

打开 cygwin，切换到 Alchemy 目录，Alchemy 目录路径为 /cygdrive/c/alchemy。

在 cygwin 中 cd 是进入目录，cd .. 是返回到上级目录，关于 bash 的详细命令请用百度搜索一下。

执行 ./config，并根据 echo 出来的提示，输入下面这句代码再回车（不在 cygwin 里输入下面这句的话，后面输入 alc-on;w;会提示 alc-on command not found）

```
source /cygdrive/c/alchemy/alchemy-setup
```

关闭 cygwin。

编辑 C:\alchemy 目录下的 alchemy-setup 文件（这个文件是执行 ./config 命令新生成的）

将 #export ADL=/path/to/fyou/bin/adl (or adl.exe) 一句修改成 export ADL=/cygdrive/c/flex/bin/adl.exe,注意去掉#注释符

编辑 C:\cygwin\etc 目录下 bash.bashrc 文件，在文件最后加入下面三行(一定要拿 ue 编辑器打开，然后把文件中所有文字最后面的那个换行符复制上，粘贴到下面前两行的末尾)

```
source /cygdrive/c/alchemy/alchemy-setup
PATH=$ALCHEMY_HOME/achacks:/cygdrive/c/flex/bin:$PATH
export PATH
```

打开 C:\cygwin，切换到/cygdriver/c/alchemy/bin 目录，执行下面的命令：（可能会提示 创建失败 llvm-stub.exe 已存在，这个不用理他）

```
ln -s llvm-stub llvm-stub.exe
```

执行

```
alc-on;
```

```
which gcc
```

切换到 c 的工程目录然后执行（当然你可以用 Adobe 提供的 sample 进行测试，目录在 /cygdriver/c/alchemy/samples/stringecho）：

执行

```
gcc -O3 -Wall -swc stringecho.c -o stringecho.swc
```

如果成功的话，会看到这两行

```
$ gcc stringecho.c -O3 -Wall -swc -o stringecho.swc
```

```
WARNING: While resolving call to function 'main' arguments were dropped!
```

结果还有这个:

[Compiler] Error #1063: Unable to open file: /cygdrive/c/alchemy/flashlibs/global.abc.

[Compiler] Error #1063: Unable to open file: /cygdrive/c/alchemy/flashlibs/playerglobal.abc.

生成的swc也明显不对才几k,查了下还要改个地方:

<http://forums.adobe.com/thread/201580>

打开\alchemy\achacks下hacks.pl

```
if(`uname` =~ /CYGWIN/)
```

改成

```
if (/bin/uname` =~ /CYGWIN/)
```

再执行刚才的 gcc 那行,就成功了。

其他: 在 window 下直接运行 cygwin 编译的程序需要将 cygwin1.dll, cyg gcc_s-1.dll, cygstc++-6.dll 3 个文件拷到 C:/WINDOWS/system32/下

Alchemy 基本 API 的使用

```
static AS3_Val createBytesMethod () {  
    char* text="hello world";  
    return AS3_String(text);  
}  
static AS3_Val getPositionMethod () {  
    char* text="hello world";  
    return AS3_String(text);  
}  
static AS3_Val bubbleMethod () {  
    char* text="hello world";  
    return AS3_String(text);  
}
```

```

int main(int argc, char** argv) {
    //添加同步函数
    AS3_Val createBytesMethod = AS3_Function( NULL, createBytes );
    AS3_Val getPositionMethod = AS3_Function( NULL, getPosition );
    AS3_Val bubbleMethod = AS3_Function( NULL, bubble );
    //将上面三个函数绑定在 AS3 Object 上
    AS3_Val result = AS3_Object(
        "createBytes: AS3ValType,getPosition: AS3ValType,bubble: AS3ValType",
        createBytesMethod,getPositionMethod,bubbleMethod
    );
    //释放绑定的函数
    AS3_Release( createBytesMethod );
    AS3_Release( getPositionMethod );
    AS3_Release( bubbleMethod );

    AS3_LibInit( result );//通知运行时，该库已初始化
    return 0;
}

```

Alpha 混合技术原理

假设一种不透明东西的颜色是 A，另一种透明的东西的颜色是 B，那么透过 B 去看 A，看上去的颜色 C 就是 B 和 A 的混合颜色，可以用这个式子来近似，设 B 物体的透明度为 alpha(取值为 0-1，0 为完全透明，1 为完全不透明)

$$R(C) = \alpha * R(B) + (1 - \alpha) * R(A)$$

$$G(C) = \alpha * G(B) + (1 - \alpha) * G(A)$$

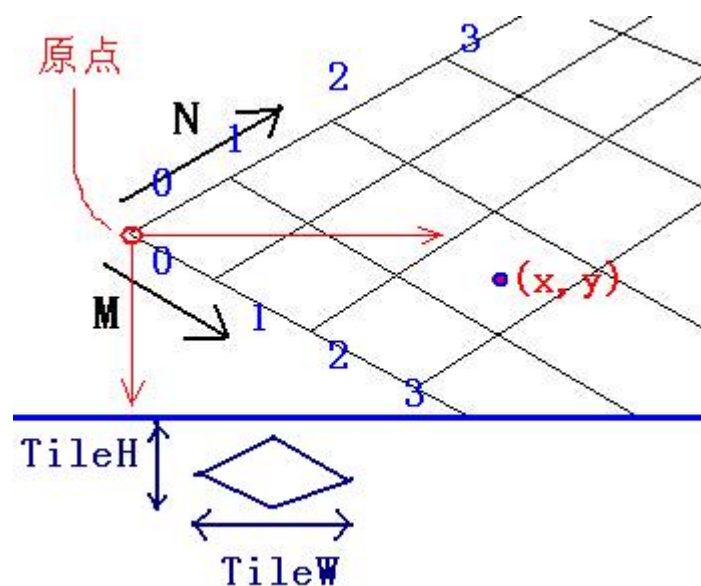
$$B(C) = \alpha * B(B) + (1 - \alpha) * B(A)$$

R(x)、G(x)、B(x)分别指颜色 x 的 RGB 分量。看起来这个东西这么简单，可是用它实现的效果绝对不简单，应用 alpha 混合技术，可以实现出最炫目的火光、烟雾、阴影、动态光源等等一切你可以想象的出来的半透明效果。

棱形地图坐标转换(一)

$$N = \text{int}(x/\text{TileW} - y/\text{TileH})$$

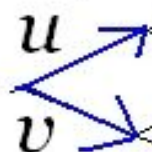
$$M = \text{int}(x/\text{TileW} + y/\text{TileH})$$



新的基向量

新的基集:

$$t = \{u, v\}$$



把新的基向量 u, v 按照标准基集 s 展开:

$$\vec{u}^s = [W/2, -H/2]^T$$

$$\vec{v}^s = [W/2, +H/2]^T$$

标准基向量

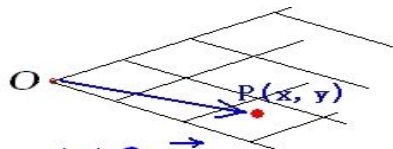
(就是原来的坐标系)



标准基集 $s = \{i, j\}$

i : 屏幕上的 $(1, 0)$ 向量

j : 屏幕上的 $(0, 1)$ 向量



向量 $\vec{a} = \vec{OP}$

\vec{a} 用标准基集S展开的话, 就是:

$$\vec{a}^s = [x, y]^T$$

\vec{a} 用新的基集t展开的话 是什么呢?

这个问题用基变换公式计算:

$$\vec{a}^t = B^{-1} \vec{a}^s$$

上面的矩阵B是:

$$B = [\vec{u}^s, \vec{v}^s]$$

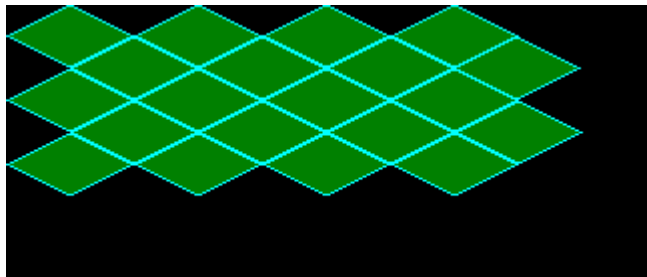
也就是:

$$B = \begin{bmatrix} W/2 & W/2 \\ -H/2 & +H/2 \end{bmatrix}$$

$$B^{-1} = \begin{bmatrix} \frac{1}{W} & -\frac{1}{H} \\ \frac{1}{W} & +\frac{1}{H} \end{bmatrix}$$

$$\begin{aligned} \vec{a}^t \cdot x &= x/W - y/H \\ \vec{a}^t \cdot y &= x/W + y/H \end{aligned}$$

棱形地图坐标转换(二)



坐标转换工具类

```
package util
{
    import flash.geom.Point;

    public class PosTrans
    {
        /**
         * 根据屏幕像素坐标取得网格的坐标
         */
        public static function getCellPoint(tileWidth:int, tileHeight:int, px:int,
py:int):Point
        {
            var xtile:int = 0;    //网格的x坐标
            var ytile:int = 0;    //网格的y坐标

            var cx:int, cy:int, rx:int, ry:int;
            cx = int(px / tileWidth) * tileWidth + tileWidth/2;    //计算出当前X所
            // 在的以tileWidth为宽的矩形的中心的X坐标
            cy = int(py / tileHeight) * tileHeight + tileHeight/2; //计算出当前Y所在
            // 的以tileHeight为高的矩形的中心的Y坐标

            rx = (px - cx) * tileHeight/2;
            ry = (py - cy) * tileWidth/2;
        }
    }
}
```

积

```
        if (Math.abs(rx)+Math.abs(ry) <= tileWidth * tileHeight/4) //<=上三角面
        {
            //xtile = int(pixelPoint.x / tileWidth) * 2;
            xtile = int(px / tileWidth);
            ytile = int(py / tileHeight) * 2;
        }
        else
        { //偶行
            px = px - tileWidth/2;
            //xtile = int(pixelPoint.x / tileWidth) * 2 + 1;
            xtile = int(px / tileWidth) + 1;

            py = py - tileHeight/2;
            ytile = int(py / tileHeight) * 2 + 1;
        }

        return new Point(xtile - (ytile&1), ytile);
    }

    /**
     * 根据网格坐标取得像素坐标
     */
    public static function getPixelPoint(tileWidth:int, tileHeight:int, tx:int,
    ty:int):Point
    {
        //偶数行tile中心
        var tileCenter:int = (tx * tileWidth) + tileWidth/2;
        // x像素 如果为奇数行加半个宽
        var xPixel:int = tileCenter + (ty&1) * tileWidth/2;

        // y像素
        var yPixel:int = (ty + 1) * tileHeight/2;

        return new Point(xPixel, yPixel);
    }

    /**
     * 根据网格索引取得网格坐标
     */
    public static function getICellPoint( gindex:uint, mapwidth:uint,
    mapheight:uint ):Point
    {
        var xtile:uint = gindex % mapwidth;
        var ytile:uint = gindex / mapwidth;
```

```

        return new Point(xtile, ytile);
    }
    /**
     * 根据网格索引取得像素坐标
     */
    public static function getIPixelPoint( gindex:uint, mapwidth:uint,
mapheight:uint, tileWidth:uint, tileHeight:uint ):Point
    {
        var cellPoint:Point = getICellPoint(gindex, mapwidth, mapheight);
        var pixelPoint:Point = getPixelPoint(tileWidth, tileHeight, cellPoint.x,
cellPoint.y);

        return pixelPoint;
    }
    /**
     * 根据网格坐标取得网格索引
     */
    public static function getCellIndex( gx:uint, gy:uint, mapwidth:uint ):uint
    {
        var gindex:uint = gy * mapwidth + gx;

        return gindex;
    }
}

```

常用工具函数

```

/**
 * 格式化金钱显示
 * @param    m    要格式化的钱
 * @return    返回以逗号格式化后的钱
 */
public static function formatMoney( m:Number ):String
{
    var money:String = String(m);
    if(m < 0) money = money.substr(1);
    var mlen:uint = money.length;
    var moneyStr:String = '';
    var end:uint = 3;
    for(var i:int=mlen; i>0; i-=3){
        var start:int = i-3;
        if(start < 0){

```

```

        end += start;
        start = 0;
    }
    var str:String = money.substr(start, end);
    if(start>0){
        str = ',' + str;
    }
    moneyStr = str + moneyStr;
}

if(m < 0) moneyStr = '-' + moneyStr;

return moneyStr;
}

```

```

/**
 * 取掉左边空白字符
 */
public static function ltrim( s:String ):String
{
    var white:String = " \f\n\r\t\v";
    var slen:uint = s.length;
    for(var i:uint=0; i<slen; i++){
        var f:int = white.indexOf(s.charAt(i));
        if(-1 == f){
            break;
        }
    }
    s = s.substr(i);
    return s;
}

/**
 * 取掉右边空白字符
 */
public static function rtrim( s:String ):String
{
    var white:String = " \f\n\r\t\v";
    var slen:uint = s.length;
    for(var i:int=slen-1; i>=0; i--){
        var f:int = white.lastIndexOf(s.charAt(i));
        if(-1 == f){
            break;
        }
    }
    s = s.substring(0, i+1);
}

```

<pre> return s; } /** * 取掉两边空白字符 */ public static function trim(s:String):String { s = ltrim(s); s = rtrim(s); return s; } </pre>
<pre> /** * 对MD5进行二次混淆 */ public static function enKey(key:String):String { var revArr:Array = []; var klen:uint = key.length; //反转 revArr = key.split(''); revArr = revArr.reverse(); //交换 for(var j:uint=1; j<klen; j+=2){ var ts:String = revArr[j-1]; revArr[j-1] = revArr[j]; revArr[j] = ts; } //替换 var n:uint = 1; var m:uint = 1; while(m<klen){ revArr[m] = key.charAt(m); m = Math.pow(2, n++); } _k = revArr.join(''); trace("二次加密key="+_k); return _k; } </pre>

长期记忆分为: 情景记忆、语义记忆

情景记忆: 可以有意识地回忆出来

语义记忆: 如果没有什么特别的契机是不会联想起来的。也就是说, 语义记忆就是不介入自我(意识)的抽象记忆.事实上, “华盛顿是美国第一任总统”的事实和自我经历是没有关联的。像语义记忆那样, 没有自我介入的记忆被称做内隐记忆。与此相对, 联想起来的事情之中伴随着自身的体验, 上升到意识的记忆被称做外显记忆。情景记忆就是外显记忆的典型表现。语义记忆就是知识。

情景记忆和语义记忆根据经历和时间是可以相互转变的。

程序性记忆: 这种记对于行走、使用筷子、打字、打棒球等平时自然而然的行为起着非常重要的作用。

情景记忆和语义记忆可以解释为 **What is**

程序性记忆则可解释为 **How to**

促发记忆: 是一种无意识的记忆,如"菠菜 菠菜 菠菜"。很多人会直接把第三个“菠菜”读成"菠菜", 因为头脑中已经存储了以前曾经出现过的"菠菜"这个词语, 这个记忆比自身的意识优先识别了文字。这种无意识的记忆就是促发记忆。

简单总结:

A.短期记忆: 30 秒至几分钟的记忆, 7 个左右的小容量。

B.长期记忆: 更长时间的记忆。

(1)情景记忆: 个人的回忆

(2)语义记忆: 知识

(3)程序性记忆: 身体所感知的事物的程序。

(4)促发记忆: 误解的根源——潜意识效果。

在这 5 种记忆之中, 短期记忆和情景记忆都是处在个人记忆水准之上的外显记忆, 所以能够有意识地回忆出来。相反, 其余的语义记忆、程序性记忆以及促发记忆这三种记忆都是没有自我意识介入的, 所以称之为内隐记忆。

记忆阶层:

情景记忆 (外显记忆)	海马
短期记忆 (外显记忆)	大脑皮层管理
语义记忆 (内隐记忆)	海马
促发记忆 (内隐记忆)	大脑皮层管理
程序性记忆 (内隐记忆)	由纹状体(位于大脑皮层内侧的基底核)和小脑控制

越下面的记忆越原始, 越上面的记忆越高级。

运动能力(程序性记忆的一种)和小脑有着特别紧密的关系

记忆是以可以提取的完整形态储藏在颞叶里的。

获取的信息会在海马中停留 1 个月左右(所以在将近一个月时, 进行复习是最佳复习时机), 之后海马会将其中应该记忆的信息送还到颞叶。

当脑处于放松状态的时候,会产生 α 波。所以能够促使 α 波产生的古典音乐或安静的环境对人的身心健康都非常有益。

而人们对 θ 波的熟悉程度却不如 α 波。 θ 波主要是海马发出的一种脑波,其特征是以每秒 5 次的频率(5 赫兹)波动,这种频率叫做 θ 节奏。

要想提高记忆力,就要对需要记忆的事物抱有浓厚的兴趣,促使 θ 波产生,从而使得海马的活动更为旺盛。

记忆三原则

应该在反复失败的过程中记忆

应该遵循一定的步骤记忆

应该首先从大局把握

学习上遵循先宏观,后微观的步骤

Flash Player debug 版本安装问题

安装 Flash Player 时,提示:正尝试安装的 Adobe(R) Flash(R) Player 版本不是最新版本。请访问 <http://www.adobe.com/go/getflashplayer> 以获取最新、最全的版本。

解决方法: 修改注册表

运行 regedit

HKEY_LOCAL_MACHINE->SOFTWARE->Macromedia->FlashPlayer->SafeVersions->删除右边的 FP 版本号些

Flash Builder4.0 文件模板

```

${package_declaration}
{
    ${import_declaration}
    /**
     *
     * @author      Yanis
     * @version     1.0.0
     * @date        ${date} ${time}
     */
    ${class_declaration}
    {
        ${class_body}
    }
}

```

MVC 文件及包组织结构

+controllers 控制器目录

+events 事件目录

+managers 管理器目录

+models 数据模型目录

- +models.vos VO 对象目录
- +services 数据服务目录
- +views 视图目录
- +views.uis UI 目录
- +utils 工具类目录
- +Module_{Name}.as 模块类(IFacade)
- +resources 资源目录

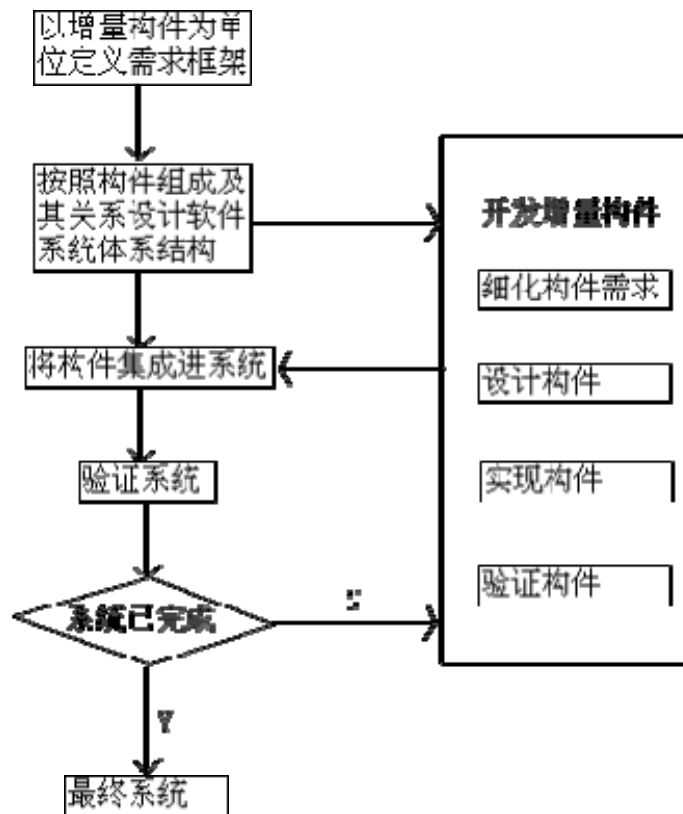
1. 类成员的位置

成员优先级从高到低：常量，静态变量，静态函数，构造器，实例变量，getter/setter 属性存取器，函数

同级别的成员按访问修饰符排序：public, internal, protected, private

软件工程

增量模型



笔记规范

题目：宋体 四号 加粗

要讲解的 API 或函数：Arial 小四 绿色

讲解描述：Arial 小五

例子：Arial 小五 灰底色

电脑技巧

运行里输入 dxdiag 回车，查看系统更详细信息

宇宙：<http://book.qq.com/s/book/0/12/12334/index.shtml>

清除文件夹下所有文件的隐藏属性

`attrib *.* -s -a -h -r /s /d`

+ 设置属性。

- 清除属性。

R 只读文件属性。

A 存档文件属性。

S 系统文件属性。

H 隐藏文件属性。

/S 处理当前文件夹及其子文件夹中的匹配文件。

/D 也处理文件夹。

windows 远程桌面连接 DOS 命令

mstsc

mstsc /v: 192.168.1.102 /console

睡眠：关闭硬盘保持内存通电

休眠：把内存的数据保存到硬盘上,然后再执行关机程序,开机时把硬盘上的数据恢复到内存。

<?xml version="1.0" encoding="utf-8"?>

游戏资源网

<http://www.gameres.com/>

AS3 Lua 脚本

<http://code.google.com/p/lua-alchemy/>

编译原理在线阅读

<http://vip.du8.com/books/sepbjzc/1.shtml>

AS3 Eval Library(动态加载外部 AS3 代码执行)

<http://eval.hurlant.com/>

Zip 档案处理类

<http://code.google.com/p/as3-ziparchive/>

http://help.adobe.com/zh_CN/AIR/1.5/devappsflex/

http://help.adobe.com/zh_CN/AIR/1.5/devappsflash/

http://help.adobe.com/zh_CN/FlashPlatform/reference/actionsript/3/

<http://www.adobe.com/support/flashplayer/downloads.html#fp11>

Java 环境变量配置

JAVA_HOME

D:\software\JDK6

CLASSPATH

.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\toos.jar

PATH

.;%JAVA_HOME%\bin 注意: 当切换 JDK 时, 这句一定要放在最前面, 否则可能无法切换(System32 下也有 java.exe)

用批处理启动 Java 程序

start cmd /c java -jar NioServer.jar

常用符号 ± ≥ ≤ × ⊗ ⊕ ⊃ ⊇ ⊂ ⊆ ∈ ∅ ∉ ∩ ∪ // ∠ ∞ ≈ ≅ ≡ ≠ ≠ ≈ ≡ μ ※ Σ °C % ∞ ★ ☆ △ ▽ 【 】 ¥
α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ ψ ω ∞ ∫ ∞ ∼ ∏ Σ % ∏ Σ
Ξ Ξ Ξ

<?xml version="1.0" encoding="utf-8"?>

ACT 动作类游戏

AVG 冒险类游戏

FTG 格斗游戏

RPG 角色扮演游戏

RTS 即时战略游戏

SLG 策略战棋游戏

SIM 模拟经营

EDU 养成游戏

SPT 体育游戏

STG 射击游戏

RAC 竞速游戏

MUG 音乐游戏

LVG 恋爱游戏

PUZ 益智游戏

CAG 卡片游戏

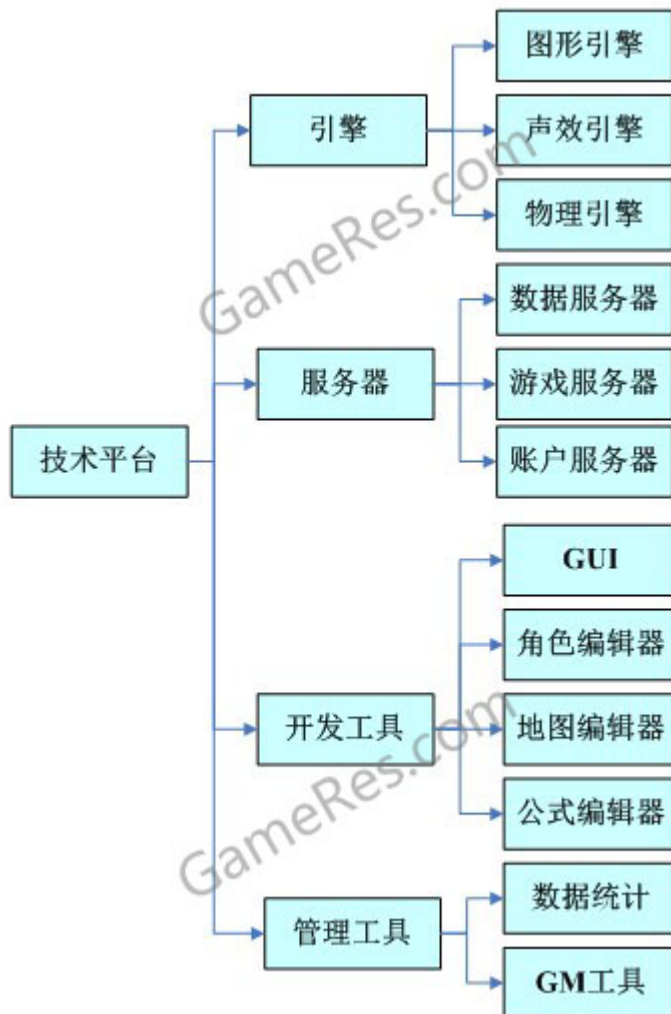
FPS 第一人称射击游戏

PVE(Player VS Environment)

玩家与游戏系统之间的互动游戏特性(单人游戏性)

PVP(Player VS Player, Person VS Person)
玩家之间的互动游戏特性(多人游戏性)

游戏开发元素



软件版本号说明

DEMO、Alpha、Beta、Release

在游戏行业中，欧美的独立开发团队往往是通过 DEMO 落实第一笔投资，才进行后续的开发。在成熟的游戏公司中的后续版本，一般不刻意设置 DEMO 阶段，通常是用设计方案，美术方案以及简单的游戏模型在立项会议阶段就完成了对 DEMO 阶段的预审。

Alpha 版：此版本表示该软件在此阶段主要是以实现软件功能为主，通常只在软件开发内部交流，一般而言，该版本软件的 Bug 较多，需要继续修改。在这个阶段，是核心开发者最为辛苦也最为开心的阶段，这个过程一般会持续 4 至 5 个月，在十余人的共同努力下完成整个游戏的基础工作。当这个阶段完成之后，主要的技术工作都已经宣告结束。游戏也完成

了最初的版本，可以让小范围的玩家进行初步的体验。

Beta 版：该版本相对于 α 版已有了很大的改进，消除了严重的错误，但还是存在着一些缺陷，需要经过多次测试来进一步消除，此版本主要的修改对象是软件的 UI。

RC 版：该版本已经相当成熟了，基本上不存在导致错误的 BUG，与即将发行的正式版相差无几。

Release 版：该版本意味“最终版本”，在前面版本的一系列测试版之后，终归会有一个正式版本，是最终交付用户使用的一个版本。该版本有时也称为标准版。一般情况下，Release 不会以单词形式出现在软件封面上，取而代之的是符号(R)。

版本命名规范

软件版本号由四部分组成



主版本号：当功能模块有较大的变动，比如增加多个模块或者整体架构发生变化。此版本号由项目决定是否修改。

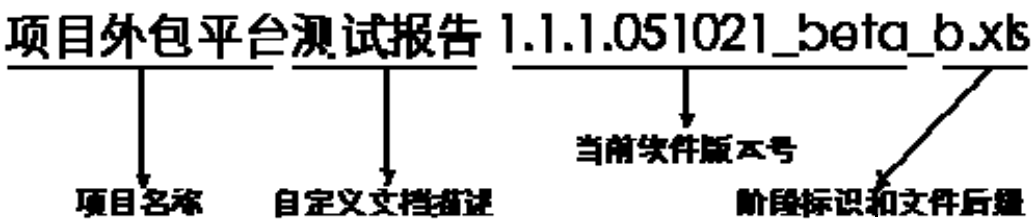
子版本号：当功能有一定的增加或变化，比如增加了对权限控制、增加自定义视图等功能。此版本号由项目决定是否修改。

阶段版本号：一般是 Bug 修复或是一些小的变动，要经常发布修订版，时间间隔不限，修复一个严重的 bug 即可发布一个修订版。此版本号由项目经理决定是否修改。

日期版本号：用于记录修改项目的当前日期，每天对项目的修改都需要更改日期版本号。此版本号由开发人员决定是否修改。

希腊字母版本号：此版本号用于标注当前版本的软件处于哪个开发阶段，当软件进入到另一个阶段时需要修改此版本号。此版本号由项目决定是否修改。

文件命名规范



如果是同一版本同一阶段的文件修改过两次以上，则在阶段标识后面加以数字标识，每次修改数字加 1，项目外包平台测试报告 1.1.1.051021_beta_b1.xls。

当有多人同时提交同一份文件时，可以在阶段标识的后面加入人名或缩写来区别，例如：项目外包平台测试报告 1.1.1.051021_beta_b_LiuQi.xls。当此文件再次提交时也可以在人名或人名缩写的后面加入序号来区别，例如：项目外包平台测试报告 1.1.1.051021_beta_b_LiuQi2.xls。

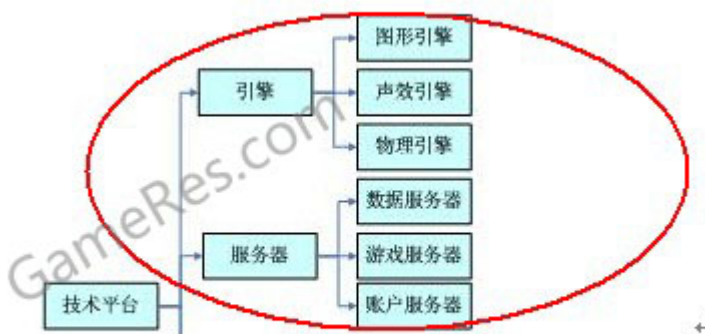
软件的每个版本中包括 11 个阶段，详细阶段描述如下：

阶段名称	阶段标识
需求控制	a
设计阶段	b
编码阶段	c
单元测试	d
单元测试修改	e
集成测试	f
集成测试修改	g
系统测试	h
系统测试修改	i
验收测试	j
验收测试修改	k

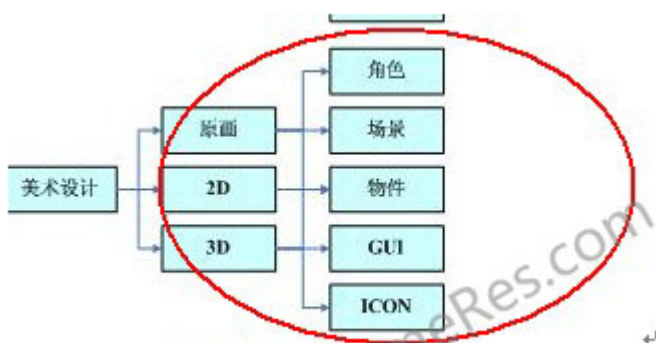
预算分析



DEMO 主要对演示效果负责

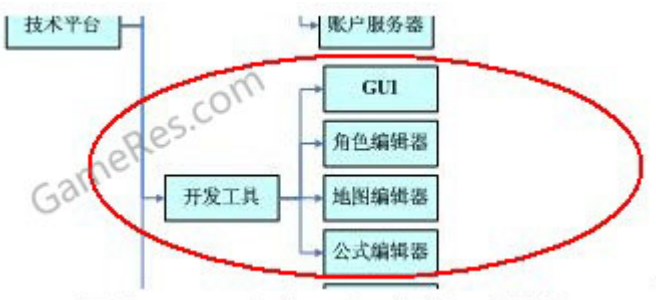
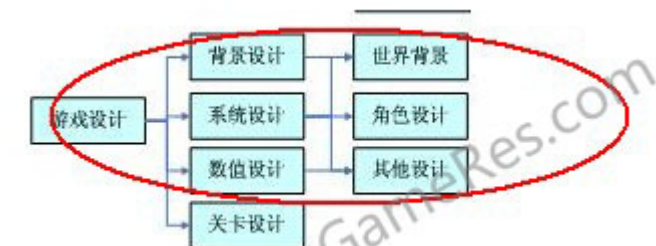


图三：Alpha 版中程序的工作重心



图四：Alpha 版中美术的工作重心

设计部分的核心工作在 DEMO 阶段已经初步完成。但在 Alpha 阶段中，需要配合程序及美术的工作，提供更加细致的设定。



项目管理目录

```

Tiantu -|
        -| Design | 策划小组的文档存放目录
        -| Paint | 美术小组文档存放目录，源文件效果图等等
        -| Dev | 开发小组目录
            -| docs | 开发设计文档 包括需求文档、接口文档、数据库文档
            -| source | 源代码目录
                -| server | 服务器端 程序目录
                    -| lib | 服务器端需要用到的类库目录，保持最新状态
                -| src |
                    -| web | php 源代码目录
                        -| common 共用类库
                        -| api 接口层定义
                        -| {module name} 模块分割的功能实现
                    -| daemon | C 源代码目录
                    -| test | Unit Test 代码目录
                        -| {module name} 模块分割的功能实现
                -| release / 服务器端发布版本目录
            -| client / 客户端 程序目录
                -| lib | 客户端需要的类库目录，保持最新状态
                -| src |
                    -| common 共用类库
                    -| res 资源目录，图片，语言包等
                    -| {module name} 模块分割的功能实现
                    -| test | Unit Test 代码目录
                        -| {module name} 模块分割的功能实现
            -| release | 客户端发布版本目录
        -| release | 版本发布目录
        -| Test | BUG 管理目录
        -| WorkReport | 工作日报及周报目录
            -| dev |
            -| paint |
            -| design |
        -| Template | 项目用到的模板文件目录
    
```

【1】业精于勤，荒于嬉；行成于思，毁于随。—韩愈

【2】学习就是从失败中汲取教训，从成功中汲取经验。

【3】孩儿立志出乡关 学不成名誓不还
埋骨何须桑梓地 人生无处不青山

【4】立奇志 交奇友 读奇书 创奇事 作一个奇男子

【5】世间万物都是遵循某种类似的规律，谁先把握了这些规律，谁就最早成为了强者。

【8】基本游戏规则程序员、AI 程序员、3D 程序员、UI 程序员、物理程序员、网络程序员、工具程序员。现在游戏这么复杂，以一人之聪明和精力，也只能掌握一个领域的最新成果并在这个领域做出成绩。但分得这么细的弊端是对程序员们来说，长期在一个狭小孤立的领域工作，其适应性和创造性肯定会受到

影响。

暴雪八大理念：放眼全球，精益求精，趣味第一，集思广益，诚信为本，王者风范，学无止境，拥抱真我

心理学家马斯洛把人的需求分为 5 个层次，分别是生理需求、安全需求、社交和爱的需求、尊重的需求、自我实现需求。

学相同才能思相近（共识），思相近才能言相合（共鸣），言相合才能行相辅（共振）！
世事洞明皆学问，人情练达即文章。

昨夜西风凋碧树，独上高楼、望断天涯路。—— 人生第一境界

衣带渐宽终不悔，为伊消的人憔悴。—— 人生第二境界

众里寻他千百度，蓦然回首,那人却在灯火阑珊处。—— 人生第三境界

君子爱美，求之以礼，情思深深，心无邪念。

燕雀安知鸿鹄之志？王侯将相，宁有种乎？

学问是智慧的泉源，品德乃事业的根本。

纸上得来终觉浅，绝知此事要躬行。

多情自古空余恨，好梦由来最易醒。

世事洞明皆学问 人情练达即文章。

我不杀伯仁，伯仁却因我而死

失之东隅(yu),收之桑榆

幼无名师，长无良友，壮无实事，老无令名。

多情自古空余恨，好梦由来最易醒。

天长地久有时尽，此恨绵绵无绝期。

莫愁前路无知己，天下谁人不识君。

武以快为尊，唯快不破，爱以舍为尊，谋以忍为尊。

《见与不见》

你见，或者不见我，我就在那里，不悲不喜；

你念，或者不念我，情就在那里，不来不去；

你爱，或者不爱我，爱就在那里，不增不减；

你跟，或者不跟我，我的手就在你手里，不舍不弃；

来我的怀里，或者，让我住进你的心里。

默然，相爱；

寂静，欢喜。

读史使人明智，读诗使人灵秀，数学使人周密，科学使人深刻，伦理学使人庄重，逻辑修辞之学使人善辩；凡有所学，皆成性格。

选自 培根 《论学习》

【加速成功的好习惯】

1 保持激情。只有激情，你才有动力，才能感染自己和其他人。

2 做事专注。抓准一个点，然后像一个钉子一下钻下去，做深、做透。

3 执行力。不仅知道，更要做到！

4 学习的习惯。学习是最便宜的投资！

5 反省的习惯。经常反省自己的得失，会使自己成功得更快一些！

- 【人生智慧】
1. 凡事皆有极困难之时，打得通的 便是勇者。
 2. 凡事皆有极复杂之时，拆得开的 便是智者。
 3. 凡事皆有极关键之时，抓得住的 便是明者。
 4. 凡事皆有极矛盾之时，看得透的 便是悟者。
 5. 凡事皆有极重大之时，沉得住的 便是静者。
 6. 凡事皆有极寂寞之时，耐得住的 便是逸者。

人类要想不断地创新，就需要不断地获取新知识。那么，新知识从哪里来呢？一般来说，新知识主要有两个来源：一个来源是对未知领域的认识，另一个来源是对已知领域的再认识。而对已知领域的再认识又主要有三个方向：一是向更加深入的方向探寻，我们称之为正向思维；二是向四周的方向探寻，我们称之为横向思维；三是向相反的方向探寻，我们称之为逆向思维。

归纳思维
类比思维
逆向思维
发散思维
猜想

拷贝百度文库中的文章

第一步

site:wenku.baidu.com + 题目

第二步

进入百度快照

期权: 经营者可以在指定时间段自由购买公司股份的权力

股权激励的基本特征是“收益共享、风险共担”。

商业智慧: 胆商、财商、情商、舆商

企业家应具备的精神: 创新精神、冒险精神、合作精神、敬业精神、学习精神、执著精神、诚信精神

听从自己内心的声音，做自己想做的事.，不要让别人的意见左右了自己内心的声音。

犹太人智慧:

契约高于逻辑

把合同当作商品出售

绝不许合同出现漏洞

毁约就是亵渎上帝

变通的最高境界是法律再造(既能从形式上遵守，同时又不改变自己原有活动方式)

在商场上的关键问题不在于道德不道德，而在于合法不合法，守约不守约。

管理学

什么叫管理？

通过决策、计划、组织、指挥、协调、控制、激励等职能来协调他人的活动，分配各种资源。

管理元素？

观念、目标、组织、人员、信息、资金、物质、技术

"无为而治"的精髓只是人力本身的"无所作为"，但制度本身则运行不违。严明法纪，制度严明，自然下属的注意力就转移到这些形式上的条文中，而不是管理者身上，管理者隐藏于制度的身后，以制度之“有为”行自身的“无为”，这才是真正聪明的管理之道。

时间	时辰	经络/脏腑
23:00 – 1:00	子时	胆(要上床睡觉)
1:00 – 3:00	丑时	肝
3:00 – 5:00	寅时	肺
5:00 – 7:00	卯时	大肠(起床要喝水)
7:00 – 9:00	辰时	胃(要吃早餐)
9:00 – 11:00	巳时	脾(每小时慢饮一次水)
11:00 – 13:00	午时	心(开心.快乐)
13:00 – 15:00	未时	小肠
15:00 – 17:00	申时	膀胱
17:00 – 19:00	酉时	肾
19:00 – 21:00	戌时	心包(适宜散步)
21:00 – 23:00	亥时	三焦

海兔网(成都优速软件有限公司)

<http://www.hyto.com/>

游戏素材网

<http://www.66rpg.com>