**GaGaFrameForClient框架**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 创建日期 | 内容 | 修订人 |
| v2.0.0 | 2012-10-08 | 创建文档 | 王加静 |
|  |  |  |  |
|  |  |  |  |

文档目录

[1 介绍 2](#_Toc426983328)

[2 日志管理 2](#_Toc426983329)

[2.1 公共属性 2](#_Toc426983330)

[2.2 公共方法 3](#_Toc426983331)

[3 对象桥 3](#_Toc426983332)

[3.1 公共属性 3](#_Toc426983333)

[3.2 公共方法 3](#_Toc426983334)

[4 事件控制 4](#_Toc426983335)

[4.1 公共属性 5](#_Toc426983336)

[4.1 公共方法 5](#_Toc426983337)

[5 下载模块 6](#_Toc426983338)

[5.1 公共属性 8](#_Toc426983339)

[5.1 公共方法 8](#_Toc426983340)

[6 国际化(多语言) 8](#_Toc426983341)

[6.1 公共方法 9](#_Toc426983342)

[7 时间模块 9](#_Toc426983343)

[7.1 公共方法 10](#_Toc426983344)

[8 常用工具 10](#_Toc426983345)

[8.1 Obj对象相关 10](#_Toc426983346)

[8.1.1 设置查询对象 10](#_Toc426983347)

[8.1.2 复制对象 11](#_Toc426983348)

[8.1.3 零散工具 11](#_Toc426983349)

[8.2 display显示对象 11](#_Toc426983350)

[8.2.1 查找对象 11](#_Toc426983351)

[8.2.2 MovieClip辅助功能 12](#_Toc426983352)

[8.2.3 零散工具 12](#_Toc426983353)

[8.3 时间相关操作 12](#_Toc426983354)

[9 Worker多线程 12](#_Toc426983355)

[9.1 主线程启动 12](#_Toc426983356)

[9.2 次线程启动 13](#_Toc426983357)

[9.3 线程添加任务 13](#_Toc426983358)

[9.3.1 添加普通无返回任务 13](#_Toc426983359)

[9.3.2 添加预设任务 14](#_Toc426983360)

[9.4 线程返回结果 14](#_Toc426983361)

[9.5 线程回收 15](#_Toc426983362)

[10 JSON快速处理 15](#_Toc426983363)

# 介绍

为程序提供公用方法，唯一，快捷的入口。通过它你可以连接引入的各个模块.

引入方式

import cn.wjj.g;

g加“.”可联想出方法，如果对框架源码引入程序，编辑器可以获得详细的注释说明。

用单字母是为了更加快捷的编程。

# 日志管理

提供日志记录

* 提供最基础的日志记录功能,替换默认的trace()方法
* 可以统一整个程序对输出日志进行控制
* 可以记录触发的准确时间,以及距离上一次日志的间隔时间
* 可以记录触发的时候的内存变化.
* 可以返回所发生的对象的引用.
* 如果是显示对象,将输出显示对象距离场景Stage的位置.
* 可以记录触发的日志的类型.判断危害度.
* 为以后日志客户端监控工具做铺垫.

g.log : 日志系统

例:

//添加一条提示内容

(g.log.isLog) g.log.pushLog(this, g.logType. \_UserAction, "是提示内容", “多条提示”);

## 公共属性

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **默认值** |
| **isTrace** | Boolean | 是否将日志 trace 出来,true:输出 trace 内容,false:隐藏 trace 内容 | true |
| **isTraceMemory** | Boolean | 是否输出内存数据 | true |
| **isAddLogMemory** | Boolean | 是否同时记录内存的消耗情况 | true |
| **config\_ShowTimeType** | Boolean | 如果这个参数为true,Trace [2011年12月31日 下午 04:16:23 星期六 毫秒:040] 这样的时间,如果是false就是[2011/11/31 AM12:32:30] | false |
| **maxItem** | int | 日志的最大长度,如果是-1就不限制,注意:设置后日志会被截取 | 10 |

## 公共方法

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **参数** | **说明** | **返回值** |
| **pushLog** | target:发出日志的对象  type:记录的日志类型  ...args:内容 | 添加一条日志,可以包含发出日志的对象,日志的类型,以及要输出的一系列参数 | void |
| **pushMinLog** | ...args:内容 | 添加一条简单的日志,只包含日志的内容 | void |
| **toString** |  | 输出全部的日志 | String |

# 对象桥

对象记录的桥梁.可以快速获取各种对象

解决面向对象如果不引用类就不方便交互数据,用了对象桥,可以使用一个协定好的String类型的名称来获取到需要的对象.

最好有文档记录都使用了那些对象桥,否则会搞混.

例:

//从框架中查找是” mainFrame”对象,并且不输出日志,如果找到返回,没有返回null

g.bridge.getObjByName("mainFrame" , false)

//设置文档类为整个SWF的root,类似于AS2.0的方法.

g.bridge.swfRoot = this;

//取出根目录

var stage:\* = g.bridge.swfRoot ;

//设置一个新的Object为”整体数据”,并以强引用驻扎到内存中,如果已经存在,不执行

g.bridge.setObjByName("整体数据", new Object(), false, true);

//删除定义的” 整体数据”

g.bridge.delObjByName("整体数据");

## 公共属性

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **默认值** |
| **sharedObjectName** | String | 设置SharedObject的名称,先入为主 | “” |
| **swfRoot** | \* | 获取整个Flash运行时的根目录 | null |

## 公共方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **reSetAndGetSWFRoot** | 重置获取\_swfRoot,注意:这个方法是备用方法,使用的时候请谨慎 |
| **setObjByName** | 设置一个对象的全局名称 |
| **getObjByName** | 通过一个对象的名称获取这个对象 |
| **hasObjName** | 检查对象列表中是否有名称为theName的对象了 |
| **changeObjByName** | 修改一个全局名称的对象连接 |
| **delObjByName** | 删除一个全局名称,并取消这个全局名称下的对象关联连接 |

# 事件控制

方便管理全局的所有的事件,并提供一些其他方便的事件

函数捆绑对象:当设置捆绑对象,那么当捆绑对象被内存回收的时候,这个添加的事件会自动被删除,可以理解为弱引用.

监听处理:替换标准的事件添加方法(同样是弱引用)

g.event.addListener("添加的对象", MouseEvent.CLICK, mouseDo);

g.event.removeListener("添加的对象", MouseEvent.CLICK, mouseDo);

EnterFrame:替换MovieClip的Event.ENTER\_FRAME效率更高,不用通过MoiveClip

g.event.addEnterFrame(enterFrame, "函数捆绑对象");

g.event.removeEnterFrame(enterFrame);

FPSEnterFrame:固定FPS触发类似EnterFrame事件,例:FPS20的频率触发函数

g.event.addFPSEnterFrame(20, method, "true:执行后刷新场景UI", "函数捆绑对象");

g.event.removeFPSEnterFrame(20, method);

//不限制FPS的大小的情况下,删除所有FPS下的监听,一次删除多个

g.event.removeFPSMethod(method);

SuperEnterFrame这个SWF所能执行的最快EnterFrame频率,按此频率执行方法

g.event.addSuperEnterFrame(method, "函数捆绑对象");

g.event.removeSuperEnterFrame(method);

事件桥:可以全局触发快速事件,定义String类型事件,触发事件自动触发定义函数,可以参数

//添加一个事件桥,可以在你程序的任何一个地方,可以添加多个

g.event.addEventBridge(this, "事件桥名称", method);

//移除一个事件桥,当然第一个参数的对象被内存回收,事件桥自动会被删除

g.event.removeEventBridge(this, "事件桥名称", method);

//可以在程序任何地方触发你的事件桥,后面可以带参数,所有添加的事件都会被执行.

g.event.runEventBridge("事件桥名称");

数据桥:检测数据对象,例new Object()的数据,当发生改变的时候自动触发函数

//添加一个数据改变要触发的函数

g.event.addEventData("检测数据引用(弱引用)", "位置", method, "函数捆绑对象");

g.event.removeEventData("检测数据引用(弱引用)", "位置", method);

//当你改变这个数据后,如果要触发所有的监听,必须运行一下这个引用对象,否则添加的监听不会自动被触发

g.event.runEventData("检测数据引用(弱引用)");

自动设置:检测数据对象,发生变化,自动把值赋值到设置位置

//容器设置为”设置对象”,设置位置为”info.txName.text”,然后检测Object对象里的”user.name”,当Object的”user.name”改变,并执行runEventData(Object),就会自动将”user.name”赋值给”设置对象”的”info.txName.text”的属性里

g.event.addAotoLinkVar("设置对象", "检测数据引用(弱引用)", "设置对象位置", "检测数据位置");

g.event.removeAotoLinkVar("设置对象", "检测数据引用(弱引用)", "设置对象位置", "检测数据位置");

//当你改变这个数据后,如果要触发所有的监听,必须运行一下这个引用对象,否则添加的监听不会自动被触发

g.event.runEventData("检测数据引用(弱引用)");

注:你可以设置一个函数public function set text(vars:String):void{},来接受这个自动赋值,但是这里的作用域必须是public

## 公共属性

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **默认值** |
| **config\_UserLog** | Boolean | 是否对事件启用日志提示功能 | false |
| **autoGloableFPSUpdateUI** | Boolean | 设置全局的FPS对象是否可以自动刷新UI层 | true |

## 公共方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **addListener** | (弱引用)添加对象上的监听事件,拥有自动卸载,删除自己对象的功能 |
| **removeListener** | 删除对象上的监听事件 |
| **removeListenerForObj** | 清除一个对象上的全部监听 |
| **removeListenerForObjAddChild** | 清除一个显示对象上的全部监听,包括这个显示对象的子集 |
| **removeListenerForListener** | 清除一个监听函数的全部引用 |
| **clearAllListener** | 清除全部的Event监听,(这里有问题) |
| **addEnterFrame** | (弱引用)添加一个ENTER\_FRAME事件 |
| **removeEnterFrame** | 删除一个ENTER\_FRAME事件 |
| **addSuperEnterFrame** | (弱引用)添加一个超快的ENTER\_FRAME事件. |
| **removeSuperEnterFrame** | 删除一个超快的ENTER\_FRAME事件. |
| **addFPSEnterFrame** | (弱引用)添加一个FPS的监听. |
| **removeFPSEnterFrame** | 删除一个FPS的监听 |
| **removeFPSMethod** | 删除所有的FPS对象里的这个固定的方法 |
| **superEventFrameUpdateUI** | 是否在这个Timer结束的时候自动调用刷新场景UI,TimerEvent.updateAfterEvent |
| **addEventBridge** | (弱引用)添加事件桥 |
| **removeEventBridge** | 删除事件桥连接 |
| **runEventBridge** | 运行事件桥事件 |
| **addEventData** | (弱引用)监听一个数据源,当里面的冒值发生变化的时候就触发,这个特定函数 |
| **addEventDataForFrame** | (弱引用)监听一个数据源,当里面的冒值发生变化的时候就触发,这个特定函数 |
| **removeEventData** | 移除特定数据源的监听 |
| **removeEventDataForFrame** | 移除特定数据源的监听 |
| **runEventData** | 当数据源变化的时候,调用这个函数可以触发数据源里有变化的值 |
| **addAotoLinkVar** | (弱引用)自动检测linkData.linkGropName的值是否变化,并且自动设置到setObj.setGroupName属性 |
| **removeAotoLinkVar** |  |

# 下载模块

可以新建一个队列，然后向这个队列里加入数据。

|  |
| --- |
|  |

下载模块设计图

|  |
| --- |
|  |

添加下载对象的流程

XML下载XML列表使用方法

//新建一个队列的属性设置

var teamVars:LoaderVarsTeam = new LoaderVarsTeam();

//设置队列

teamVars.name(“队列名称”).onComplete(configLoadOver);

//通过对象设置对象,来获取一个队列对象

var team:Team = g.loader.addTeam(teamVars);

//将XML的URL添加到这个队列

g.loader.loadXMLConfig(“队列名称”, “队列URL”);

下载图片文件使用方法

import cn.wjj.g;

public function addImgLoader():void{

var imgVars:LoaderVarsImage = new LoaderVarsImage();

imgVars.url("url.jpg").onComplete(imgLoaderComplete);

g.loader.addItem(imgVars);

//让下载的Loader开始运行起来

g.loader.start();

}

//下载xxx.jpg完毕后的回调方法

public function imgLoaderComplete():void{

var bmp:Bitmap = g.loader.asset.asset.getAssetImage("url.jpg", false);;

addChild(bmp);

}

## 公共属性

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **类型** | **说明** | **默认值** |
| **asset** | AssetManage | 全新的资源管理模块 |  |
| **isLog** | Boolean | 是否使用Log输出日志 | true |
| **config\_checkSOSize** | Boolean | 是否检查输出的OS的尺寸 | true |
| **config\_ShowTimeType** | Boolean | 如果这个参数为true,Trace [2011年12月31日 下午 04:16:23 星期六 毫秒:040] 这样的时间,如果是false就是[2011/11/31 AM12:32:30] | false |
| **config\_runSpeed** | Boolean | 网速模块 : 是否计算Loader的下载速度 | true |
| **config\_autoDelProgressTime** | Number | 网速模块 : 自动删除多少时间之前的数据,毫秒 | 10000 |
| **config\_useSpeedTime** | Number | 网速模块 : 通过几秒内的时间去计算网速,毫秒 | 10000 |

## 公共方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **addItem** | 添加单个下载内容 |
| **getItem** | 通过字符串URL,或者Vars实例来获取队列 |
| **setAssetFinishRun** | 当一个资源载入完毕的时候运行 |
| **addTeam** | 添加一个队列 |
| **getTeam** | 通过字符串队列名称,或者Vars实例来获取队列 |
| **start** | 开始下载任务,队列和单独都需要执行这个 |
| **isCompleteItem** | 查询一个Item是否下载完毕 |
| **loadXMLConfig** | 添加一个下载的队列 |
| **getInfo** | 获取一个下载队列的内容,根据格式,输出对应的格式 |
| **getInfoForName** | 参数参考 getInfo ,通过一个资源的名称Name获取这个资源的引用 |
| **getSpeed** | 获取下载速度 |

# 国际化(多语言)

提供了多种语言即时切换，后台驻扎下载，文本内容替换，样式替换，坐标修正，图片

的替换,动画的替换。

暂时没有处理的是声音的自动替换.

添加渐变色功能,会生成多余的Shape在Text的场景中

载入一个XML的语言包配置文件

private function loaderLanguage():void {

g.language.loadConfig(languageName, languageXMLUrl, languageComplete);

}

//语言包及内容,图片,字体等下载处理完毕

private function languageComplete():void { }

## 公共方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **setField** | 设置一个文本,让他自动和某一个language的ID对象绑定 |
| **setFieldSkin** | 设置一个文本框的皮肤,字体 |
| **setBitmap** | 设置一个Bitmap,让他自动和某一个language的ID对象绑定 |
| **getObject** | 获取记录某一个ID的数据全部对象 |
| **getString** | 获取一个language的ID对象的文本引用 |
| **getBitmapData** | 获取一个language的ID对象的图像内容引用 |
| **allDefaultInfo** | 获取现在正在执行的语言包的全部语言对象的引用 |
| **getFont** | 获取一个language的ID对象的Font引用 |
| **getMovie** | 获取一个LanguageImgMovie动画对象 |
| **loadConfig** | 获取一个字体配置文件,并且保存下来 |
| **setConfig** | 设置一个默认的语言包 |
| **delConfig** | 删除一个语言包配置 |

# 时间模块

这个模块是一些时间相关的模块.提供获取时间,倒计时,固定时间触发等.

例:

快速获取UNIX时间戳

//获取一个UNIX时间戳

g.time.getTime()

添加一个循环计时器

//添加一个循环计时器,名叫:测试,添加任务后自动启动,以1000毫秒触发一次go函数的频率,共触发5次,触发完毕自动删除

import cn.wjj.g;

import cn.wjj.gagaframe.client.time.TimerItemInterval;

var timerGo:TimerItemInterval = g.time.addIntervalTimer("测试");

timerGo.setIntervalTime(go,1000,5,true,true);

function go():void{

trace(g.time.getTime());

}

## 公共方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **getTime** | 获取一个时间,可以获取系统时间,模拟时间,和Server同步的时间 |
| **addRunTimer** | 添加一个定时触发的时间 |
| **getRunTimerForName** | 通过名称添加一个定时器 |
| **getRunTimer** | 添加一个定时炸弹,如果没有就新建一个 |
| **delRunTimer** | 查询是否有特定的计时器,如果有就删除掉 |
| **addIntervalTimer** | 添加一个循环计时的时间 |
| **getIntervalTimerForName** | 通过名称添加一个循环计时 |
| **getIntervalTimer** | 添加一个循环计时,如果没有就新建一个 |
| **delIntervalTimer** | 查询是否有特定的计时器,如果有就删除掉 |
| **getTimerForName** | 通过名称获取计时器对象 |
| **addAutoForPoint** | 添加自动倒计时的时间文本 |
| **addAutoForRemaining** | 添加还剩余多少时间到00:00:00 |

# 常用工具

一些常用的静态方法的类调用.包含了框架里已经使用的一些方法.

## Obj对象相关

这里放了一些常用的操作Object对象的方法,比如查询对象内内容,深度复制

### 设置查询对象

//为obj对象设置propertyName名称的属性,值为vars

g.tool.obj.action.setVar(Object, name, vars);

//查询obj中的名称为name的属性,并输出,如果没有返回null,有返回这个属性值

g.tool.obj.action. getVar(Object, name);

//在obj对象中找groupName次序内容并设置它,例:"display.menu.glod",在Object里找display对象,在display里找menu对象,在menu里在找glod对象,然后设置他的值为vars

g.tool.obj.action.setGroupVar(Object, groupName, vars, isTrace);

//在一个容器里通过名称检索对象,例:"display.menu.glod",在容器里找display对象,在display里找menu对象,在menu里在找glod对象

g.tool.obj.action.getGroupVar(Object, groupName, isTrace);

### 复制对象

//对象深度复制 : 将实例及子实例的所有成员(属性和方法, 静态的除外)都复制一遍, (引用要重新分配空间!)

g.tool.obj.clone.deepClone(Object);

//浅复制一个对象

g.tool.obj.clone. clone(Object);

### 零散工具

//是否是原始类型 , Booelan, Number, String

g.tool.obj.util.isPrimitiveType()

//判断两个对象是否相等

g.tool.obj.util.equals(a,b):Boolean

//赋值对象的引用

g.tool.obj.util.getDictValue (vars:\*): DictValue

//设置一个弱引用对象,如果这个对象没有被回收,就会辅助记录一个连接对象

//如果这个对象被回收,就会把辅助记录的一个连接对象也释放.

g.tool.obj.util.getDictLink(link:\*,vars:\* = null): DictLink

## display显示对象

### 查找对象

//搜索容器所有的显示对象,并且以数组返回

g.tool.display.search.allChild(container:DisplayObjectContainer,searchInContainer:Boolean = true):Array

//搜索容器中指定类型的对象并以数组方式返回

g.tool.display.search.child(container:DisplayObjectContainer, searchType:Class):Array

//搜索容器中含有超过一幀的MovieClip的对象并以数组方式返回,递归所有的子对象

g.tool.display.search.mcFrame(container:DisplayObjectContainer):Array

//在容器display获取名称为name对象,没有返回null,有就返回对象

g.tool.display.search.nameIn(display:DisplayObjectContainer, name:String):\*

//在一个容器里通过名称检索对象,例:"display.menu.glod",在容器里找display对象,在display里找menu对象,在menu里在找glod对象

g.tool.display.search.groupNameIn(display:DisplayObjectContainer,groupName:String, isTrace:Boolean = true):\*

//在g.bridge.swfRoot中查询,例:"display.menu.glod"

g.tool.display.search.groupNameInRoot(groupName:String, isTrace:Boolean = true):\*

//获取一个显示对象到根的全面路径,display.menu.glod

g.tool.display.search.getGroupName(display:DisplayObject):String

//检测一个对象中是否含有显示对象.

g.tool.display.search.objHasDisplay(obj:\*, maxDoing:int = 0, maxDoingConfig:int = 1000):Boolean

### MovieClip辅助功能

//替换AS3的gotoAndStop.,解决有时候不能goto到指定位置

g.tool.display.mc.gotoStop(mc:MovieClip, str:\*, maxTimes:int = 50):void

### 零散工具

//remove清除display容器内的全部内容

g.tool.display.util.removeAllChild(display:\*):void

## 时间相关操作

将一个毫秒时间转换为特定格式输出

2000年01月01日 下午 11:59 星期一 毫秒:999

2000/01/01 PM 11:59:59

2000/01/01 23:59:59

2000-01-01 23:59:59

1Day 23:59:59.999 (这段时间所消耗的时间)

99:59:59 (这段时间所消耗的时间)

g.tool.time. getString(time:Number, strType:String = ""):String

# Worker多线程

只实现了双线程，提供异步线程和阻塞式线程（没写完），提供线程切换处理，兼容线程失败自动切回主线程处理.（使用中可以无需关心线程是否启动，当不支持多线程自动切换单线程处理）

全部线程使用一个框架

## 主线程启动

需要在主swf中启动线程。当然可以额外处理你想处理的其他线程内容。

var worker:Worker = WorkerDomain.current.createWorker(workerBytes, true);

g.worker.init(worker, true, workerComplete);

worker.start();

## 次线程启动

需要在次线程中启动次线程，建立起线程框架连接

//设置主引导场景

g.bridge.swfRoot = this;

//开启BUG监控

g.status.error.start();

//初始化线程

var worker:Worker = Worker.current;

g.worker.init(worker, false, workerComplete);

## 线程添加任务

添加任务需要建立WorkerTask线程任务对象,。

WorkerTask的参数和方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **list** | 任务中的子工作集合，全部的任务完成才会回调 |
| **item** | 如果任务只有一个子工作，只用设置此参数 |
| **track** | 任务在执行过程中是否需要返回一些状态值 |
| **complete** | 整个任务完成的时候的回调 |

每个任务可以添加一个或多个子工作WorkerInfo

WorkerInfo的参数和方法

|  |  |
| --- | --- |
| **名称** | **说明** |
| **init** | 工作发起者为了方便程序书写，在工作中添加的信息，方便回调的时候程序获取 |
| **out** | 工作完成的时候，返回回来的结果 |
| **send** | 工作参数，当执行这项工作所需的参数 |
| **track** | 工作中的一些状态值，例如：下载百分比，处理条数 |

### 添加普通无返回任务

本演示提供了在次线程中向主线程发送一个任务类型为888888的任务，并且传送了一个空对象作为数据，你也可以不用传递

var worker:WorkerTask = new WorkerTask();

worker.item = new WorkerInfo();

worker.item.type = 888888;

worker.item.send.info = new Object();

g.worker.taskSend(worker);

### 添加预设任务

这个演示了，预先制作了工作任务类型W\_HTTPBase，初始化并完成移动返回的全过程

var workerInfo:WorkerInfo = W\_HTTPBase.init(url, o, true, 0, false, false);

var task:WorkerTask = new WorkerTask();

task.complete = complete;

task. track = track;

task.item = workerInfo;

g.worker.taskSend(task);

**/\*\* 发送成功 \*\*/**

private function complete(e:WorkerTask):void

{

if (e.item.out.isError != false && e.item.out.hasInfo && e.item.out.info.state == 0)

{

//已经返回成功

var data:String = e.item.out.info.data

}

}

private function track(e:WorkerTask):void

{

**//这里获取下载量数据**

}

## 线程返回结果

线程的返回结果，会返回在WorkerInfo中。

//在处理过程中产生了错误

WorkerInfo.out. isError

//是否有返回info的内容

WorkerInfo.out. hasInfo

//如果有返回,这里记录了info的内容

WorkerInfo.out. info

## 线程回收

执行类WorkerTask对象中的dispose方法

# JSON快速处理

// JSON将一个Object对象转换为String

g. jsonGetStr(o:Object):String

// JSON将一个String对象转换为Object

g. jsonGetObj(str:String):Object