

PainterEngine API manual

Author:DBinary

Matrixcascade@gmail.com

## Core

### 预编译

#### 计算数组个数

#define px\_countof(x) (sizeof(x)/sizeof(x[0]))

### 内存

#### PainterEngine内存池实现与细节

###### 引言

内存池作为一种的内存管理机制被广泛地运用于各种领域当中，内存池拥有快速的内存分配与更加健壮的管理机制，同时在不同的平台与环境当中也拥有不同的实现方式，本文提出一种轻量级的内存池实现，可以非常方便的移植到内存空间较小的平台当中，可运用在不同的嵌入式平台，服务端及小范围的内存管理当中.

###### 从内存开始

巧妇难为无米之炊，既然我们要说是内存池，显然你首先至少需要找到一个能存储数据的东西，否者接下来都没有讨论的意义，存储的意义可以非常的广泛，它可以是名正言顺的RAM，也可以是你的硬盘，甚至是你的显存或者SD卡当中，为了给PainterEngine的内存池指定内存地址，你需要预先给出一块可用内存的大小与及它的起始地址.例如下面的代码将会给你做一个示范

//为内存池分配1M的内存空间，MemoryPool指向首地址

void \*MemoryPool=malloc(1024)

//MemoryPool指向地址0x08600000地址，这块地址可能指向一个片外RAM，一块NANDFlash地址或者其它的任意存储设备

void \*MemoryPool=(unsigned char\*)0x08600000;

//MemoryPool可存在于堆中也可存在于栈当中，但是你需要确保它的地址在内存池的使用阶段都是有效的。

char MemoryPool[1024];

可以看到，PainterEngine的内存池可以存在于任何的地址与区域当中，你需要确保的只有三点，1.内存地址是线性的，2.内存是可读的 3.内存是可写的.

###### 一块内存

取得一块内存区域并不是什么非常困难的事情，在windows或者是linux当中，你可以使用现有的malloc或者C++的new来申请一块可用的内存地址，在嵌入式设备当中，你也可以使用片上RAM或者是类似于FSMC或外部地址总线等方式来指定一块内存，具体如何实现由您决定。

那么现在我们假设你已经拥有了这一块可用的内存并且做好了其它的一些别的准备工作，为了方便，我们假设您的内存地址是从0x00000000开始的，并且我们假设它的大小是

Size，为了说明方便我们使用图1来表示。

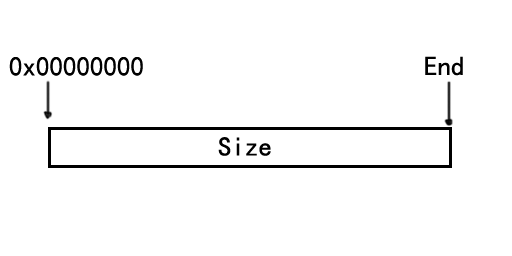


图1

从上面来看我们已经初始化一些基本的数据并且可以推导一些有用的数据，首先我们有一块内存，而且它的大小是Size，当中的所有区域我们都是可用的。我们还知道它的结束地址，是0x00000000+size-1;

那么既然是内存池，最基本的一个功能当然是可以从内存池当中分割出一块内存来供我们使用，这有点儿像分饼干，假如你读过一些内存池相关的文章，也许现在我们普遍的做法就是把这一块内存区域均等的分为多个block，比如假如这里的size是512字节，假如每个block占64字节，那么这个内存区域就会被分为8个block，当然还有更进一步，4个block组成一个chunk，那么512字节的内存区域，就包含有2个chunk或者说包含有8个block，当然，这里仅仅只是用512字节作为一个比方，也许你觉得这样分是在是多此一举，但假如这个size大到64M或者是128M，那么，用一个chunk为单位，对内存池进行寻址，显然会比你用字节为单位进行寻址要快得多，并且寻址范围也会大的多，并且以chunk（或者别的更大的单位）进行内存管理，对内存碎片的整理合并也会方便且快得多而且很容易使用map进行映射，这样就能够非常快的寻址处理，这种思想被广泛的运用在了文件系统与内存管理方面，例如windows操作系统就会建立一个页目录及页表来管理其内存空间，采用这种映射方式，它允许其内存在物理上不是连续的，甚至映射到硬盘或其他存储设备当中（虚拟内存）。

###### 分割一块可用的内存

但遗憾的是，PainterEngine并不将内存进行这种分割处理，我们可以假如我们将一个Block分割为64字节，假设我需要一个86字节的内存空间，那么我们需要2个block也就是128字节，因为block作为一个整体是不可分的，那么我们将有42字节被白白的浪费了，在内存以G为单位的PC机上，也许这些损失实在是无足轻重，然而对于仅仅只有几百kb甚至是几kb的片上系统来说，这些损失就显得实在是太过于昂贵了。你也许会考虑到进一步缩小block提高细分度来减少这种损失，但是，即便是对block进行的寻址，也是需要内存空间的，并且更小的细粒度除了浪费内存之外，也显得毫无意义

技术不是扯淡，更不是拿起一本书就开始照本宣科，脱离实际地区域争论某种高深架构或者方法如何优秀，只是为了掩饰自己能力的贫庸.我们再来看看PainterEngine面向的环境.

1. 轻量级内存池，首先决定了它不能过于的臃肿与庞大，在保证功能的前提之下，能多简单就多简单，没必要把简单的问题复杂化
2. 内存分配较小。这就意味着使用映射表的方法意义实在不大，也许10000\*10000的复杂度会很大的优于100lg100，但是10lg10和10\*10的差别确实不怎么大，即使我们建立的内存表都用链表来进行管理，因为内存较小，表也不大，即使是遍历也未必会比映射慢上多少
3. 假如我们要记录一块分配内存，至少需要哪些信息呢，一.这块内存的起始地址

二．这块内存的结束地址或者它的大小。

那么，问题就开始变得简单了，假如内存池是一条等待切割的绳子，现在一个人想要3米长的绳子，那么非常简单，0-3米我们规定就是他的了，又一个人想要2米的绳子，那么我们再规定3m-5m这一段距离的绳子就属于它的了。为了记录下一次我们将从哪里开始切割绳子并且我们也需要知道剩下绳子的长度，以免我们只剩下1m的绳子了仍然答应一个想要2m长绳子的家伙，因此我们需要2个变量来进行记录1.下一次分割绳子的开始点，2.剩余绳子的长度。也就是下一次分配内存的起始地址和剩余内存的大小。那么，最基本的内存分配就这样实现了

我们先定义内存池的基本结构

typedef struct \_MemoryPool

{

void \*AllocAddr;//下一次内存的分配地址

void \*EndAddr;//内存池的尾地址

unsigned int Size;//内存池的总大小

unsigned int FreeSize;//剩余内存池空间大小

}MemoryPool;

我们定义一个函数,用来初始化这个内存池

void CreateMemoryPool(void \*MemoryAddr,unsigned int MemorySize);

它的实现是

void CreateMemoryPool( void \*MemoryAddr,unsigned int MemorySize )

{

G\_MemoryPool.AllocAddr=MemoryAddr;//最开始的分配地址就是起始地址

G\_MemoryPool.EndAddr=((unsigned char\*)MemoryAddr)+MemorySize-1;//计算出内存池终止地址

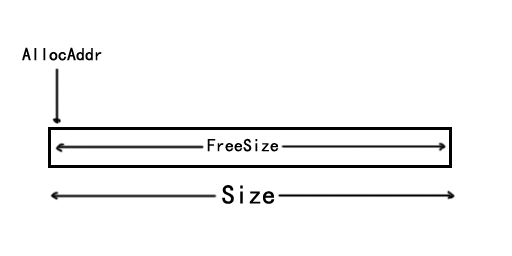
G\_MemoryPool.Size=MemorySize;//内存池大小

G\_MemoryPool.FreeSize=MemorySize;//一开始剩余内存池空间大小就是内存池大小

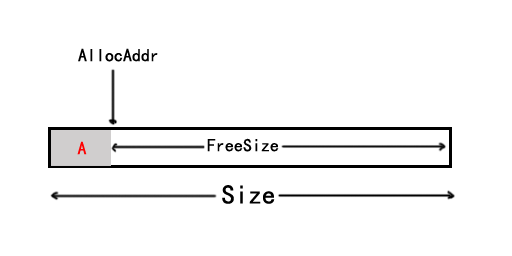
}

下面几张图表示内存的分配情况

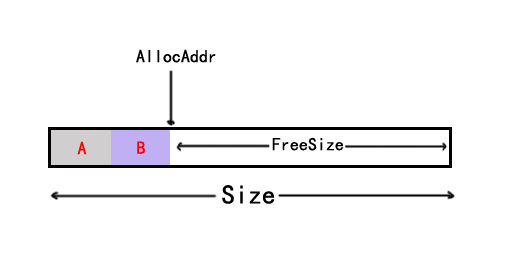
1. 内存初始化时



1. 分配内存A后



1. 分配内存B后



###### 有借有还:内存碎片与回收

记录分配节点

计算机是一个伟大的发明，毫无意味的在某些数学角度方面，它的计算速度要比人快得多，但是遗憾的是，计算机未必比人聪明，而且是一个奇怪的家伙，他记忆力很很差劲，只能靠一些存储器件来帮助他进行记忆，比如寄存器和硬盘。

上面的方案看上去似乎工作的非常好，而且也不会出上面问题，当然，假如你的内存是无限大的话，你完全可以这么干，然而在某些平台上，你却不得不抠门地去节省那几个字节甚至是几个位，上面的方案显然碰到了一些问题，我们分配了AB两个内存块，也许A当中只是存储了一些临时的数据，或者说这些数据我们用不到了，我们也许会希望释放掉这个内存，并且把空闲的内存归还到内存池。

然而上面的方案我们只是把内存简简单单的划分出去了，而没有做任何记录，显然的，我们应该给它做个记录，为了节省内存，我们尽量将记录变得简略，这并不复杂，我们只要记录分配的起始地址和结束地址就可以了.就像出租房房东一样，在账本下记下几号房到几号房出租给了某某某，当那人来退房的时候，那几个房间就可以回收利用了，在这里，我们将这个记录称为内存节点，用c语言表示如下：

typedef struct \_MemoryNode

{

void \*StartAddr;

void \*EndAddr;

}MemoryNode;

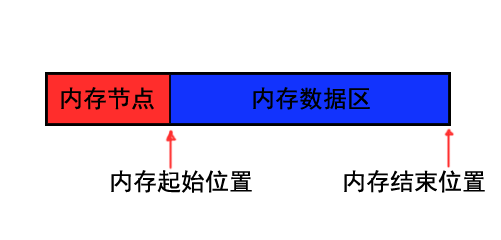
如何放置内存节点

上面的思路也许看上去非常简单，不需要太多的思考也能想出来，但是如何放置内存节点和如何回收回来的内存节点确值得思考，在网上流行的很多内存池中，基本都采用了链表的方式去连接和管理节点，假如新增加一个节点，就使用malloc分配一个，然而在一个要求高度可移植的内存池方面，这个想法却是不完善的，首先，你不知道堆到底有多少空间供你分配，因为非常多的小型系统的堆大小仅仅只有可怜的几kb，而且用户很可能已经将那些空间使用的所剩无几了，所以这个办法显然是被否决的，另一种办法是采用类似于windows的内存管理机制，使用页表，但之前也已经提及了，这种办法是在是太奢侈了，另外，放置节点的位置也需要讲究，我们看看释放内存的函数

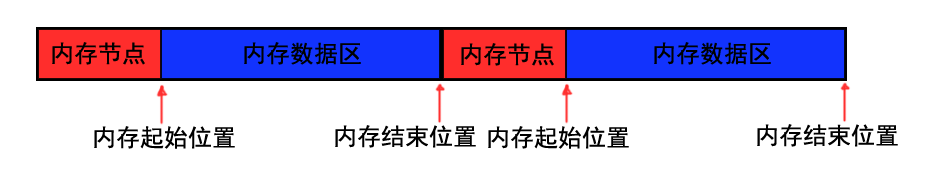
Free(void \*address)

Address是需要释放内存的首地址，这也意味着，你需要依靠首地址去寻找这个节点，假如你将内存节点都存储在一个区域当中，你很可能要去遍历来寻找这个节点，倘若你调用分配函数1000次，每次free都有可能遍历节点数百次，这显然对于cpu而言，实在是太过的奢侈繁琐了。

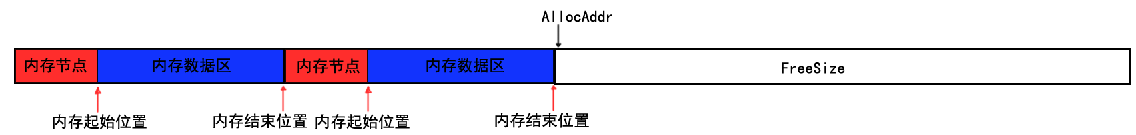
因此最合适的位置，应该是分配到address福建，将节点放在开始地址的前面，应该是一个不错的方案。



使用sizeof(MemoryNode)我们很容易确定内存节点的大小，所以在free的时候，我们仅仅需要将address减去sizeof(MemoryNode)，我们就可以非常容易获得这个内存节点，并且根据这个节点来取得内存数据区的起始位置和结束位置。

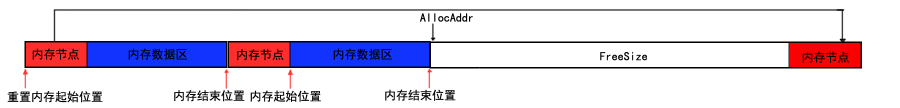


释放的内存节点与管理

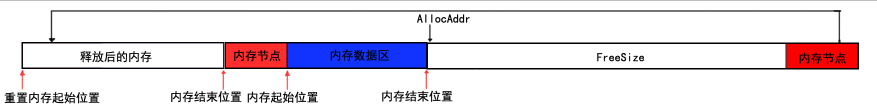
现在，我们可以将那些释放掉的内存节点进行管理了，我们仍然需要将释放的内存节点存储在内存池当中，但我们将不采用链表的形式存储，相对于链表，线性存储显然颇具优势。而我们需要做的，仅仅只是为其挑一个存储的位置与空间，不言而喻的，将这些释放的节点存储在内存池的尾部，是一个不错的选择，因为受到池内存大小的限制，释放的内存节点经过整理后也不会显得过于的庞大，因此，即使是遍历搜索也不会慢到哪里去，释放内存节点非常简单，我们仅仅只需要将节点的起始地址设置为内存节点的开始地址，然后复制节点节点到尾部就可以了，就像下图所表示的那样：

释放第一块内存：

1.拷贝节点到内存池尾部



2.实际上无需其它的操作，这块内存意义上已经释放了



我们用下面的代码，简单为内存池增加一个释放的内存节点：

MemoryNode \*PX\_AllocFreeMemoryNode()

{

//增加释放的节点数目

G\_MemoryPool.FreeTableCount++;

//以MemoryNode指针的形式返回该内存节点的首地址

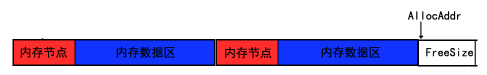
return PX\_MemoryPool\_GetFreeTable(G\_MemoryPool.FreeTableCount-1);

}

不能失败的内存释放

但是上面的办法真的没有问题吗，我们看看Free的原型

Void Free(void \*Address)它的返回值是null也就是必须成功，那么什么时候上面的方案会失败呢，非常明显的，每当释放一个内存节点，我们将从尾部去记录释放的内存节点，尾部的空间实际上是输入FreeSize的，当FreeSize小于一个内存节点的大小的时候，分配就会失败



（FreeSize已经不足以保存一个节点，释放失败）

释放失败的后宫是严重的，它将导致那块内存被永久的占用，比如你内存池大小是1M，已经分配了1023字节，那么，这个内存池除非你恰好释放的是最后一个节点（稍后讨论），否者这个内存池将直接陷入瘫痪，但是办法总是有的，其中最有效的是，我们在内存分配期间，就需要为内存的释放预留出空间，这个虽然稍微有点儿浪费空间，但是这对内存池的健康与维护大有好处，并且MemoryNode结构本身并不大，释放空间操作相比于分配操作往往要少的多，绝大部分的空间都是随着内存分配的开始到程序运行的结束。



那么这也就意味着，在我们分配内存的时候，至少需要分配两个MemoryNode节点大小的内存空间才能够满足要求，同时我们也需要记录下当前有多少个已经释放的内存节点与在释放的节点中最大的节点大小是多少,这样做方便后期的查找与遍历与在释放的内存节点中分配新的内存，我们修改MemoryPool结构体：

typedef struct \_MemoryPool

{

void \*AllocAddr; //FreeSize内存区域起始地址

void \*EndAddr; //内存池终止地址

unsigned int Size;//内存池大小

unsigned int FreeSize;//FreeSize生于大小

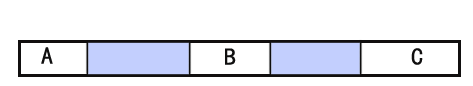
unsigned int FreeTableCount;//释放节点的数目

unsigned int MaxMemoryfragSize;//释放节点中最大的碎片大小

}MemoryPool;

###### 内存的碎片与减少碎片

也许最合理的内存回收是能够回收多少是多少，回收完后仍然是线性的，例如现在我释放 了一个大小为A（8字节） B(12字节) C（4字节）的内存，下次我需要分配24字节，可以直接从之前释放的空间中申请，但是，实际情况上却常常难以做到，因为我们申请的内存，必须是线性连续分配的,但释放后的内存空间,往往不是连续的.



（非连续内存导致碎片）

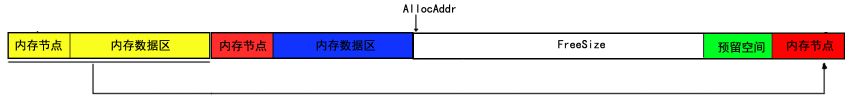
为了最大程度地减少碎片,我们需要将那些原本连续的碎片进行拼接合并,例如内存节点A的结束地址和内存节点B的开始地址恰好是重合的,那么我们就可以考虑将节点A与节点B进行合并,合并内存不论是在效率上还是内存的再分配上都有好处.

我们对以下几种情况的内存碎片分别讨论

1. 不能合并的内存节点



A是一个释放的内存节点,很遗憾的是,不论是A的前方或者后方都是一家被内存池分配出去的内存，A将不能进行合并，对于这种情况，调用之前写好的PX\_AllocFreeMemoryNode函数分配一个节点，并将A的内存节点信息直接保存在内存池的尾部



1. 可以向前合并的节点

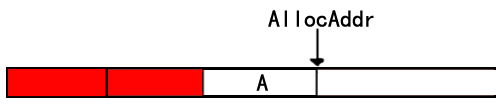


A是一个待释放的内存节点，B是一个已释放的的内存节点，可以看到，BA在内存空间上是相连的，所以我们需要将BA两个内存空间进行合并，在这个过程当中，我们不需要再分配一个释放的内存节点，原先为A释放节点的预留空间也可以删除回归到Freesize当中，之后仅需要修改原来就存在的B的释放内存节点，将他的结束地址修改为节点A的结束地址，就可以完成释放了。

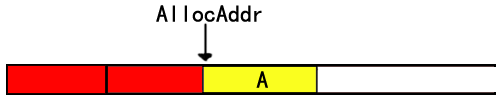
1. 向后合并的节点



A是一个待释放的内存节点，B是一个已释放的的内存节点， AB在内存空间上是相连的，所以我们需要将AB两个内存空间进行合并，其过程与向前合并时类似的。但是向后合并我们还可以更进一步地进行优化，来看看下面这种情况



在节点A的后面，就是AllocAddr，在这种情况下，我们可以尽量地扩大FreeSize而不是将A的节点放入释放内存节点中，我们直接将AllocAddr放到A的首地址，并且删除原先为A的预留空间，这样，A就被释放掉了



1. 前后合并节点



前后合并的其实就是向前合并和向后合并的组合，但我们需要的是注意一下合并的顺序，因为向后合并存在一种特殊的情况，我们可以将其选择出来并进行专门的处理，就如下图所示的情况



第一种情况的前后合并时，需要删除一个预留空间（看顺序）并且删除释放节点，最后只留下一个节点，第二种情况首先向前合并，同时删除B的释放内存节点和A的预留空间，将AllocAddr移动到B的起始地址完成内存释放.

###### 分配内存

从空闲空间中分配内存

在上一个章节的” 分割一块可用的内存”,”不能失败的内存释放”中,给出了这个空闲空间分配内存的方法与分配代码,在这里就不再复述了,它的流程非常简单

1. 分割出一块合适的空间大小
2. 初始化内存节点
3. 预留空间
4. 返回起始地址

短短几步,就能够完成空闲空间的内存分配,当内存空间不够时,返回0

从已释放内存中分配内存

回收内存再分配,是一个内存池能否能够持续良好持续运行的一个必要功能,同时,这也常常成为碎片产生的原因.但是内存池是否能够完全避免产生碎片呢,当然这并不是不可能的,但是付出的代价往往不值得我们这样去做.所以,内存碎片,尽可能的减少却不能够完全避免.

那么应该如何减少碎片呢,其实这和环境与用法有很大的不同,一个老练的码农往往喜欢把内存分配到2的整数幂,这也对减少碎片有很大的帮助,比如你分配了一个64字节大小的内存释放之后又分配一个32字节和4个8字节的内存,刚好就填充了之前的空间,越小的内存分配越容易填充之前的碎片,当然,假如你申请的内存和释放的内存一直保持一致的大小,这个碎片也就能够最大程度的消除了.

过多的碎片不仅占用了不必要的内存空间,也拖慢了再分配和释放,因此,内存分配并不仅仅只是算法上的,很多时候也是使用和习惯上的.

那么如果需要从已分配内存当中新分配内存应该怎么办呢,我们先来讨论一下不产生碎片的情况,最直观的是,你释放了一个内存的大小,刚好就是你想再分配内存的大小,这是无缝的,自然不会产生碎片.但是我们不能就仅仅止步于此,我们再来看看内存的结构,

是的,在PainterEngine内存池结构中,分配一个size大小的内存,需要size+2\*sizeof(MemoryNode)大小的内存空间,分配内存分配的大小,是我们主观上的大小,我们为其多分配一些内存空间,并不会出什么大不了的问题,比如你申请 16字节大小,而我分配了32字节给你,多余的字节并不会对其它功能造成影响,顶多我不知道不用就是了,而且这样比我们分割这个已经释放的内存空间更有好处,毕竟,为新划分的空间需要预留额外的sizeof(MemoryNode)大小的空间而且也会照成碎片,那么,我们可以这样规定

假如已经释放的内存大小大于等于size并且小于sizeof(MemoryNode)+size,我们就将这个已经释放的内存全部分配给需求的内存.

当然,我们不可能期待每个已经释放内存都像上面一样,很多时候,我们释放了大小为128字节的空间,下一次却只需要32字节的内存空间,把这个128字节的释放空间都分配给这个只需要32字节的内存节点,显然是太过的浪费了,这个时候,我们必须要对内存进行分割

分割并不复杂,只需要把原先的释放内存首地址作为分配内存的首地址,填充MemoryNode节点,然后将结束地址+1替换到原来的释放内存节点的首地址就可以了,同时别忘了预留一个内存节点的空间.

MemoryNode \*PX\_AllocFromFreq(unsigned int Size)

{

unsigned int i,fSize;

MemoryNode \*itNode,\*allocNode;

//大小加上头部内存节点需要空间

Size+=sizeof(MemoryNode);

//比对最大已释放内存大小,若小于,分配肯定是失败的

if (G\_MemoryPool.MaxMemoryfragSize>=Size)

{

//开始搜索满足条件的节点

for(i=0;i<G\_MemoryPool.FreeTableCount;i++)

{

itNode=PX\_MemoryPool\_GetFreeTable(i);

//计算节点大小

fSize=(char \*)itNode->EndAddr-(char \*)itNode->StartAddr+1;

//满足内存节点完全分配的情况

if (Size<=fSize&&(Size+sizeof(MemoryNode)>fSize))

{

//初始化新的内存节点

allocNode=(MemoryNode \*)itNode->StartAddr;

allocNode->StartAddr=(char \*)itNode->StartAddr+sizeof(MemoryNode);

allocNode->EndAddr=itNode->EndAddr;

//因为是完全分配,直接删除原来的节点,因为空间已经预留过了,所以不需要预留空间

PX\_MemoryPoolRemoveFreeNode(i);

//重新跟新最大释放内存节点大小

PX\_UpdateMaxFreqSize();

return allocNode;

}

else

{

//不满足完全分配的情况,开始进行内存分割

if(Size<fSize)

{

//确认FreeSize足够分配一个内存节点

if (G\_MemoryPool.FreeSize<sizeof(MemoryNode))

{

return 0;

}

//初始化内存节点

allocNode=(MemoryNode \*)itNode->StartAddr;

allocNode->StartAddr=(char\*)itNode->StartAddr+sizeof(MemoryNode);

allocNode->EndAddr=(char \*)itNode->StartAddr+Size-1;

//重新计算释放内存节点的起始位置,完成分割

itNode->StartAddr=(char \*)allocNode->EndAddr+1;

//预留空间

G\_MemoryPool.FreeSize-=sizeof(MemoryNode);

//再次重新计算最大释放内存节点大小

PX\_UpdateMaxFreqSize();

return allocNode;

}

}

}

return 0;

}

else

{

return 0;

}

}

###### 内存操作

//Copy memories to dest

//dst:Pointer to dest address

//src:Pointer to source address

//size:copy size

内存拷贝函数

dst:指向目标内存

src:指向源内存

size:需要拷贝的内存大小

void px\_memcpy(void \*dst,void \*src,px\_uint size);

|  |  |
| --- | --- |
| 函数名 | void px\_memset(void \*dst,px\_byte byte,px\_uint size) |
| 说明 | **设置内存值** |
| 参数 | Dst 目标内存指针  Byte 值  Size 设置内存大小 |

|  |  |
| --- | --- |
| 函数名 | void px\_memdwordset(void \*dst,px\_dword dw,px\_uint count) |
| 说明 | **按4字节对齐设置内存值** |
| 参数 | Dst 目标内存指针  dw 值  count 需要设置的四字节个数 |

|  |  |
| --- | --- |
| 函数名 | px\_bool px\_memequ(void \*dst,void \*src,px\_uint size) |
| 说明 | **比较目标内存是否相同** |
| 参数 | Dst 目标内存指针  src 目标内存指针  Size 比较大小 |
| 返回值 | 如果相同返回PX\_TRUE,否者返回PX\_FALSE |

#### 内存池

##### 创建内存池

//Create a memory pool & return a MemoryPool structure

//MemoryAddr :Start address of memory

//MemorySize :Size of memory pool

|  |  |
| --- | --- |
| 函数名 | MemoryPool MP\_Create (void \*MemoryAddr,unsigned int MemorySize); |
| 说明 | **创建一个内存池** |
| 参数 | MemoryAddr :内存池的开始地址  MemorySize :需要创建的内存池的大小  返回一个MemoryPool结构体 |
| 返回值 | 内存池结构 |

##### 返回一块内存指针所指向内存的内存大小

//Get memory size of Ptr

//Pool: Pool MemoryPool structure pointer

//Ptr: memory pointer

//Return - if succeeded return the size of Ptr,else return zero

|  |  |
| --- | --- |
| 函数名 | px\_uint MP\_Size(px\_memorypool \*Pool,px\_void \*Ptr); |
| 说明 | **返回一块内存指针所指向内存的内存大小** |
| 参数 | Pool 内存池  Ptr 指针 |
| 返回值 | 该指针所指向内存块大小 |

##### 从内存池分配一块内存

//Alloc a memory from memory pool

//Pool: Pool MemoryPool structure pointer

//Size: Size of alloc

//Return - if succeeded return the begin address of memories

// if faith return null

|  |  |
| --- | --- |
| 函数名 | **void \*MP\_Malloc (MemoryPool \*Pool,unsigned int Size);** |
| 说明 | **从内存池分配一块内存**  **该内存将会优先从已回收内存节点中分配,如果回收节点无法分配再从空余空间分配** |
| 参数 | Pool：指向MemoryPool的结构体，这个结构体使用MP\_Create进行创建  Size：需要分配的大小  如果成功返回这个内存的指针，如果失败返回null/0 |
| 返回值 | 分配的内存指针,如果分配失败将会返回PX\_NULL |

##### 从内存池释放一块内存

//Free the memory from memory pool

//Pool: Pool MemoryPool structure pointer

//pAddress: Pointer memory need to be free

Pool：指向MemoryPool的结构体，这个结构体使用MP\_Create进行创建

pAddress:内存首地址指针

**void MP\_Free (MemoryPool \*Pool,void \*pAddress);**

void MP\_Release (MemoryPool \*Pool);

|  |  |
| --- | --- |
| 函数名 | **void MP\_Free (MemoryPool \*Pool,void \*pAddress);** |
| 说明 | **释放一个内存节点** |
| 参数 | Pool：指向MemoryPool的结构体，这个结构体使用MP\_Create进行创建  pAddress:内存首地址指针 |
| 返回值 | - |

##### 重置内存池

void MP\_Reset (MemoryPool \*Pool);

|  |  |
| --- | --- |
| 函数名 | **void MP\_Reset (MemoryPool \*Pool);** |
| 说明 | **重置内存池,** **重置内存池将该内存池所有内存分配节点快速释放,对于某类归类或计算算法的加速尤为有用** |
| 参数 | Pool：指向MemoryPool的结构体，这个结构体使用MP\_Create进行创建 |
| 返回值 | - |

##### 设置内存池异常回调

**px\_void MP\_ErrorCatch(px\_memorypool \*Pool,PX\_MP\_ErrorCall ErrorCall);**

**typedef px\_void (\*PX\_MP\_ErrorCall)(PX\_MEMORYPOOL\_ERROR);**

**当内存池触发异常时将会执行该异常回调函数,如果没有设置异常回调将会以PX\_ERROR进行处理**

**异常回调消息包括**

**PX\_MEMORYPOOL\_ERROR\_OUTOFMEMORY 内存池内存不足**

**PX\_MEMORYPOOL\_ERROR\_INVALID\_ACCESS 内存池指向内存无法读/写访问**

**PX\_MEMORYPOOL\_ERROR\_INVALID\_ADDRESS 释放无效的内存地址**

### 字符串操作

##### 字符串拷贝函数

dst:指向目标内存

src:指向源字符串

size:目标内存的大小

void px\_strcpy(px\_char \*dst,px\_char \*src,px\_uint size);

##### 宽字符串拷贝函数

dst:指向目标内存

src:指向源字符串

size:目标内存的大小

void px\_wstrcpy(px\_word \*dst,px\_ word \*src,px\_uint size);

##### 字符串长度

|  |  |
| --- | --- |
| 函数名 | px\_int px\_strlen(px\_char \*dst) |
| 说明 | **字符串长度** |
| 参数 | 字符串指针 |
| 返回值 | 返回长度 |

##### 宽字符字符串长度

|  |  |
| --- | --- |
| 函数名 | px\_int px\_wstrlen(px\_word \*dst) |
| 说明 | **宽字符字符串长度** |
| 参数 | 字符串指针 |
| 返回值 | 返回长度 |

##### 字符串设置

|  |  |
| --- | --- |
| 函数名 | px\_void px\_strset(px\_char \*dst, const px\_char \*src); |
| 说明 | **将目标字符串拷贝为源字符串** |
| 参数 | Dst 目标字符串原指针  src 源字符串 |
| 返回值 | - |

##### 字符串拼接

|  |  |
| --- | --- |
| 函数名 | px\_void px\_strcat(px\_char \*src, const px\_char \*cat); |
| 说明 | **字符串拼接** |
| 参数 | src字符串原指针  cat 需要拼接到src的字符串 |
| 返回值 | - |

##### 宽字符串拼接

|  |  |
| --- | --- |
| 函数名 | px\_void px\_wstrcat(px\_word \*src,const px\_ word \*cat); |
| 说明 | **宽字符串拼接** |
| 参数 | src字符串原指针  cat 需要拼接到src的字符串 |
| 返回值 | - |

##### 字符串转大写

|  |  |
| --- | --- |
| 函数名 | px\_void px\_strupr(px\_char \*src); |
| 说明 | **将src指向字符串转换为大写** |
| 参数 | src字符串原指针 |
| 返回值 | - |

##### 字符串转小写

|  |  |
| --- | --- |
| 函数名 | px\_void px\_strlwr(px\_char \*src); |
| 说明 | **将src指向字符串转换为小写** |
| 参数 | src字符串原指针 |
| 返回值 | - |

##### 字符串全等

|  |  |
| --- | --- |
| 函数名 | px\_bool px\_strequ(px\_char \*src,px\_char \*dst); |
| 说明 | **字符串全等** |
| 参数 | src字符串指针  dst 字符串指针 |
| 返回值 | 若两个字符串全等,返回PX\_TRUE,否者返回PX\_FALSE |

##### 字符串是否是数字

|  |  |
| --- | --- |
| 函数名 | px\_bool px\_strIsNumeric(px\_char \*src); |
| 说明 | **字符串是否是数字** |
| 参数 | src字符串指针 |
| 返回值 | 若字符串是数字,返回PX\_TRUE,否者返回PX\_FALSE |

##### 字符串是否是小数

|  |  |
| --- | --- |
| 函数名 | px\_bool px\_strIsFloat(px\_char \*src); |
| 说明 | **字符串是否是小数** |
| 参数 | src字符串指针 |
| 返回值 | 若字符串是小数,返回PX\_TRUE,否者返回PX\_FALSE |

##### 字符串是否是整数

|  |  |
| --- | --- |
| 函数名 | px\_bool px\_strIsInt(px\_char \*src); |
| 说明 | **字符串是否是整数** |
| 参数 | src字符串指针 |
| 返回值 | 若字符串是整数,返回PX\_TRUE,否者返回PX\_FALSE |

##### 字符串查找字符

|  |  |
| --- | --- |
| 函数名 | px\_char \*PX\_strchr(const char \*s,int ch); |
| 说明 | **字符串查找字符** |
| 参数 | s查找字符串,ch 字符 |
| 返回值 | 如果找到返回首指针,否者返回PX\_NULL |

##### 字符串查找字符串

|  |  |
| --- | --- |
| 函数名 | char\* PX\_strstr(const char\* dest, const char\* src); |
| 说明 | **字符串查找字符串** |
| 参数 | dest 待比较字符串  src 比对字符串 |
| 返回值 | 如果找到返回首指针,否者返回PX\_NULL |

##### 字符串格式化

|  |  |
| --- | --- |
| 函数名 | px\_int px\_sprintf1(px\_char \*\_out\_str,px\_int str\_size,px\_char fmt[], px\_stringformat \_1)  px\_int px\_sprintf2(px\_char \*\_out\_str,px\_int str\_size,px\_char fmt[], px\_stringformat \_1, px\_stringformat \_2)  px\_int px\_sprintf2(px\_char \*\_out\_str,px\_int str\_size,px\_char fmt[], px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3)  ….. |
| 说明 | **字符串格式化函数** |
| 参数 | \_out\_str输出字符串指针  Fmt 格式化输出字符串  使用%1 %2 %3….来表示需要格式化的字符串  当需要指定浮点精度时,可以使用%1.x的格式来表述,其中x为拓展精度  \_1,\_2,\_3…..参数,数量和px\_sprintf后的数字对应,用于对应%1 %2…对应格式化字符串  px\_stringformat可以用  PX\_STRING\_FORMAT\_INT  PX\_STRING\_FORMAT\_FLOAT  PX\_STRING\_FORMAT\_STRING  函数,进行构造分别用于表述一个整数,浮点数,字符串 |
| 返回值 | - |

### 转换

#### 字节类型转十六进制字符串

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_BufferToHexString(px\_byte data[],px\_int size,px\_char hex\_str[]); |
| 说明 | **数据类型转十六进制字符串类型** |
| 参数 | data数据指针,size为数据长度,hex\_str字符串指针长度为size\*2+1, |
| 返回值 |  |

#### 十六进制字符串转字节类型

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_HexStringToBuffer(const px\_char hex\_str[],px\_byte data[]) |
| 说明 | **十六进制字符串转数据类型** |
| 参数 | hex\_str字符串指针,data数据指针,长度是hex\_str字符串长度的一半 |
| 返回值 | 成功返回PX\_TRUE,失败返回PX\_FALSE; |

#### 十六进制字符转数字

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_htoi(char\* ) |
| 说明 | **十六进制字符串转px\_uint** |
| 参数 | 字符串指针 |
| 返回值 | 返回转换值 |

#### 字符转数字

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_atoi(char\* ) |
| 说明 | **字符串转px\_int** |
| 参数 | 字符串指针 |
| 返回值 | 返回转换值 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_atof(char\* ) |
| 说明 | **字符串转px\_float** |
| 参数 | 字符串指针 |
| 返回值 | 返回转换值 |

#### 字符串转IPv4地址

|  |  |
| --- | --- |
| 函数名 | px\_dword PX\_inet\_addr(px\_char i[]); |
| 说明 | **字符串转IPv4地址** |
| 参数 | I字符串ip地址 |
| 返回值 | 转换后的双字 |

#### IPv4地址转字符串

|  |  |
| --- | --- |
| 函数名 | px\_char\* PX\_inet\_ntoa(px\_dword ipv4); |
| 说明 | **IPv4转字符串地址** |
| 参数 | Ipv4双字ip地址 |
| 返回值 | 转换后的字符串 |

#### 数字转字符串

整数

|  |  |
| --- | --- |
| 函数名 | px\_int \*PX\_itoa(int num,char \*str,px\_int size,int radix) |
| 说明 | **数字转字符串** |
| 参数 | num 数字  str 结果字符串指针  size 缓存大小  radix 进制 |
| 返回值 | 返回转换长度 |

|  |  |
| --- | --- |
| 函数名 | PX\_RETRUN\_STRING PX\_itos(px\_int num,px\_int radix); |
| 说明 | **数字转字符串** |
| 参数 | num 数字  radix 进制 |
| 返回值 | 返回一个字符串结构体,访问其data成员可直接取得转换字符串 |

小数

|  |  |
| --- | --- |
| 函数名 | px\_int \*PX\_ftoa(float num,char \*str,px\_int size,px\_int precision) |
| 说明 | **数字转字符串** |
| 参数 | num 数字  str 结果字符串指针  size 缓存大小  precision精度长度 |
| 返回值 | 返回转换长度 |

|  |  |
| --- | --- |
| 函数名 | PX\_RETRUN\_STRING PX\_ftos(float f, int precision); |
| 说明 | **数字转字符串** |
| 参数 | num 数字  precision精度长度 |
| 返回值 | 返回一个字符串结构体,访问其data成员可直接取得转换字符串 |

#### 弧度角度转换

|  |  |
| --- | --- |
| 函数名 | PX\_RadianToAngle(angle) |
| 说明 | **弧度转换为角度** |
| 参数 | angle 角度 |
| 返回值 | 角度 |

|  |  |
| --- | --- |
| 函数名 | PX\_AngleToRadian(radian) |
| 说明 | **角度转弧度** |
| 参数 | radian 弧度 |
| 返回值 | 弧度 |

### 字节序

#### 主机字节序转网络字节序

|  |  |
| --- | --- |
| 函数名 | px\_dword PX\_htonl(px\_dword h);  px\_word PX\_htons(px\_word h); |
| 说明 | **主机字节序转网络字节序** |
| 参数 | h:dword或word类型值 |
| 返回值 | 转换后的字节序 |

#### 网络字节序转主机字节序

|  |  |
| --- | --- |
| 函数名 | px\_dword PX\_ntohl(px\_dword n);  px\_word PX\_ntohs(px\_word n); |
| 说明 | **网络字节序转主机字节序** |
| 参数 | h:dword或word类型值 |
| 返回值 | 转换后的字节序 |

### 数学

#### 素数模乘同余发生器

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_srand(px\_uint64 seed); |
| 说明 | **初始化一个素数模乘同余发生器的随机数种子** |
| 参数 | seed 种子,如果未调用该函数,将会以默认值对其初始化 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_rand(); |
| 说明 | **返回一个0~PX\_RAND\_MAX-1之间的随机数** |
| 参数 |  |
| 返回值 | 返回随机数,范围为0~PX\_RAND\_MAX-1 |

|  |  |
| --- | --- |
| 函数名 | px\_uint32 PX\_randEx(px\_uint64 seed) |
| 说明 | **返回一个0~PX\_RAND\_MAX-1之间的随机数以seed作为运算种子** |
| 参数 |  |
| 返回值 | 返回随机数,范围为0~PX\_RAND\_MAX-1 |

#### 高斯随机数发生器



|  |  |
| --- | --- |
| 函数名 | px\_int PX\_GaussRand(); |
| 说明 | **返回一个符合高斯分布的随机数** |
| 参数 |  |
| 返回值 | 返回高斯随机数,见上图分布 |

#### 对数函数

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_ln(px\_double \_\_x)  px\_double PX\_log(px\_double \_\_x) |
| 说明 | **求一个数e为底对数函数** |
| 参数 | \_\_x这个数的值 |
| 返回值 |  |

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_lg(px\_double \_\_x)  px\_double PX\_log10(px\_double \_\_x) |
| 说明 | **求一个数10为底对数函数** |
| 参数 | \_\_x这个数的值 |
| 返回值 |  |

#### 三角函数

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_sind(px\_ double radius);  px\_double PX\_cosd(px\_ double radius); |
| 说明 | **px\_double PX\_sind(px\_ double radius); 弧度制求正弦**  **px\_double PX\_cosd(px\_ double radius); 弧度制求余弦**  **该函数使用泰勒展开,精度高于查表的实现,效率低于查表实现** |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_sin\_radian(px\_float radius);  px\_float PX\_cos\_radian(px\_float radius);  px\_float PX\_tan\_radian(px\_float radius);  px\_float PX\_sin\_angle(px\_float angle);  px\_float PX\_cos\_angle(px\_float angle);  px\_float PX\_tan\_angle(px\_float angle);  px\_double PX\_atan(px\_double x);  px\_double PX\_atan2(px\_double y, px\_double x); |
| 说明 | **px\_float PX\_sin\_radian(px\_float radius); 弧度制求正弦**  **px\_float PX\_cos\_radian(px\_float radius); 弧度制求余弦**  **px\_float PX\_tan\_radian(px\_float radius); 弧度制求正切**  **px\_float PX\_sin\_angle(px\_float angle);角度制求正弦**  **px\_float PX\_cos\_angle(px\_float angle); 角度制求余弦**  **px\_float PX\_tan\_angle(px\_float angle); 角度制求正切**  **以上函数使用查表实现,速度高于泰勒展开,精度低于泰勒展开,精度为0.001级,主要运用于动画和2D物理计算中**  **px\_double PX\_atan(px\_double x); 弧度制求反正切**  **px\_double PX\_atan2(px\_double y, px\_double x); 弧度制atan2** |

#### 平方根

|  |  |
| --- | --- |
| 函数名 | float PX\_Sqrt( float number ) |
| 说明 | **求一个数的平方根** |
| 参数 | Number 需要求平方根的数 |
| 返回值 | 返回这个数的平方根（这个数必须是正数） |

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_Sqrtd(px\_double number ) |
| 说明 | **求一个数更高精度的平方根** |
| 参数 | Number 需要求平方根的数 |
| 返回值 | 返回这个数的平方根（这个数必须是正数） |

|  |  |
| --- | --- |
| 函数名 | float PX\_SqrtRec( float number ) |
| 说明 | **求一个数平方根的倒数** |
| 参数 | Number 需要求平方根的倒数 |
| 返回值 | 返回这个数的平方根的倒数（这个数必须是正数） |

#### 方差

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_Variance(px\_double x[],px\_int n) |
| 说明 | **计算一个序列的方差** |
| 参数 | x 序列数据  n 元素个数 |
| 返回值 | 返回这个矩阵计算方差 |

#### 协方差

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_Covariance(px\_double x[],px\_double y[],px\_int n); |
| 说明 | **计算一个矩阵的协方差** |
| 参数 | x, y两数据  n 数据长度 |
| 返回值 | 返回这两个数据协计算方差 |

#### 求绝对值

|  |  |
| --- | --- |
| 函数名 | PX\_ABS(x) |
| 说明 | **宏函数求绝对值** |
| 参数 | 值 |
| 返回值 | 返回绝对值 |

#### tanh

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_tanh(px\_double x) |
| 说明 | **tanh函数** |
| 参数 | x |
| 返回值 | tanh计算结果 |

#### sigmoid

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_ sigmoid(px\_double x) |
| 说明 | **sigmoid函数** |
| 参数 | x |
| 返回值 | **sigmoid**计算结果 |

#### ReLU

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_ sigmoid(px\_double x) |
| 说明 | **ReLU函数** |
| 参数 | x |
| 返回值 | ReLU计算结果 |

#### 四舍五入

|  |  |
| --- | --- |
| 函数名 | PX\_APO(x) |
| 说明 | **宏函数小数点后四舍五入求整数** |
| 参数 | 小数 |
| 返回值 | 返回四舍五入的整数 |

#### 幂

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_pow\_ii(px\_int i,px\_int n); |
| 说明 | **整数整幂** |
| 参数 | i底数 n次幂 |
| 返回值 | i的n幂 |

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_pow\_ff(double num,double m); |
| 说明 | **幂** |
| 参数 | num底数 m次幂 |
| 返回值 | num的m幂 |

#### 复数

|  |  |
| --- | --- |
| 函数名 | px\_complex PX\_complexBuild(px\_float re,px\_float im); |
| 说明 | **构造一个复数** |
| 参数 | re为实部,im为虚部 |
| 返回值 | 构造的复数结构 |

|  |  |
| --- | --- |
| 函数名 | px\_complex PX\_complexAdd(px\_complex a,px\_complex b); |
| 说明 | **复数相加** |
| 参数 | a,b 需要相加的两复数 |
| 返回值 | 结果 |

|  |  |
| --- | --- |
| 函数名 | px\_complex PX\_complexMult(px\_complex a,px\_complex b); |
| 说明 | **复数相乘** |
| 参数 | a,b 需要相乘的两复数 |
| 返回值 | 结果 |

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_complexMod(px\_complex a) |
| 说明 | **复数的模** |
| 参数 | 需要求解的复数 |
| 返回值 | 结果 |

|  |  |
| --- | --- |
| 函数名 | px\_complex PX\_complexLog(px\_complex a); |
| 说明 | **复数的e的对数** |
| 参数 | 需要求解的复数 |
| 返回值 | 结果 |

|  |  |
| --- | --- |
| 函数名 | px\_complex PX\_complexExp(px\_complex a); |
| 说明 | **复数的e的次数** |
| 参数 | 需要求解的复数 |
| 返回值 | 结果 |
| 函数名 | px\_complex PX\_complexSin(px\_complex a); |
| 说明 | 复数的sin值 |
| 参数 | 需要求解的复数 |
| 返回值 | 结果 |

#### 矩阵

Px\_matrix 结构体，声明为

union {

struct {

px\_float \_11, \_12, \_13, \_14;

px\_float \_21, \_22, \_23, \_24;

px\_float \_31, \_32, \_33, \_34;

px\_float \_41, \_42, \_43, \_44;

};

px\_float m[4][4];

}

是一个4x4的矩阵

|  |  |
| --- | --- |
| 函数名 | void PX\_MatrixZero(px\_matrix \*Mat); |
| 说明 | **构造一个0矩阵** |
| 参数 | Mat指向该矩阵的指针 |

|  |  |
| --- | --- |
| 函数名 | void PX\_MatrixIdentity(px\_matrix \*Mat); |
| 说明 | **构造一个单位矩阵** |
| 参数 | Mat指向该矩阵的指针 |

|  |  |
| --- | --- |
| 函数名 | px\_matrix PX\_MatrixMultiply(px\_matrix Mat1,px\_matrix Mat2); |
| 说明 | **矩阵相乘** |
| 参数 | Mat1，Mat2 需要相乘的两个矩阵 |
| 返回值 | 返回矩阵相乘结果矩阵 |

|  |  |
| --- | --- |
| 函数名 | px\_matrix PX\_MatrixAdd(px\_matrix Mat1,px\_matrix Mat2); |
| 说明 | **矩阵相加** |
| 参数 | Mat1，Mat2 需要相加的两个矩阵 |
| 返回值 | 返回矩阵相加结果矩阵 |

|  |  |
| --- | --- |
| 函数名 | px\_matrix PX\_MatrixSub(px\_matrix Mat1,px\_matrix Mat2); |
| 说明 | **矩阵相减** |
| 参数 | Mat1，Mat2 需要相减的两个矩阵 |
| 返回值 | 返回矩阵相减结果矩阵 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_MatrixEqual(px\_matrix Mat1,px\_matrix Mat2); |
| 说明 | **判断两个矩阵是否相等** |
| 参数 | Mat1，Mat2 需要判断的两个矩阵 |
| 返回值 | 如果相等返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixTranslation(px\_matrix \*mat,float x,float y,float z); |
| 说明 | **构造平移矩阵** |
| 参数 | Mat指向该矩阵的指针  X,Y,Z平移向量 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateX(px\_matrix \*mat,float Angle); |
| 说明 | **构造X轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  Angle 旋转角度（角度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateY(px\_matrix \*mat,float Angle); |
| 说明 | **构造Y轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  Angle 旋转角度（角度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateZ(px\_matrix \*mat,float Angle); |
| 说明 | **构造Z轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  Angle 旋转角度（角度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateXRadian (px\_matrix \*mat,float rad); |
| 说明 | **构造X轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  rad 旋转角度（弧度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateYRadian (px\_matrix \*mat,float rad); |
| 说明 | **构造Y轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  rad 旋转角度（弧度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixRotateZRadian(px\_matrix \*mat,float rad); |
| 说明 | **构造Z轴旋转矩阵** |
| 参数 | Mat指向该矩阵的指针  rad 旋转角度（弧度制） |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixScale(px\_matrix \*mat,float x,float y,float z); |
| 说明 | **构造缩放矩阵** |
| 参数 | Mat指向该矩阵的指针  X,Y,Z缩放方向 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_MatrixInverse(px\_matrix \*mat); |
| 说明 | **矩阵求逆** |
| 参数 | Mat指向该矩阵的指针  如果求逆成功，则mat为逆矩阵 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MatrixTranspose(px\_matrix \*matrix); |
| 说明 | **矩阵转置** |
| 参数 | Marixt为需要转置的矩阵指针 |

|  |  |
| --- | --- |
| 函数名 |  |
| 说明 |  |
| 参数 |  |

#### 点/向量

px\_point 点描述结构体

typedef struct \_px\_point

{

px\_float x;

px\_float y;

px\_float z;

}px\_point;

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointRotate(px\_point p,px\_float angle); |
| 说明 | **返回一个点绕原点顺时针旋转角度后的点** |
| 参数 | P绕原点旋转的参考点  angle 旋转角度 |
| 返回值 | 返回结果点 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointAdd(px\_point p1,px\_point p2); |
| 说明 | **向量相加** |
| 参数 | P1,p2需要相加的两个向量 |
| 返回值 | 返回结果向量 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointSub(px\_point p1,px\_point p2); |
| 说明 | **向量相减** |
| 参数 | P1,p2需要相减的两个向量 |
| 返回值 | 返回结果向量 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointMul(px\_point p1,px\_float m); |
| 说明 | **向量缩放** |
| 参数 | P1 缩放向量  M 倍数 |
| 返回值 | 返回结果向量 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointDiv(px\_point p1,px\_float m); |
| 说明 | **向量缩放** |
| 参数 | P1 缩放向量  M 除以倍数 |
| 返回值 | 返回结果向量 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_PointDot(px\_point p1,px\_point p2); |
| 说明 | **向量点乘** |
| 参数 | P1,p2两点乘向量 |
| 返回值 | 返回结果 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointCross(px\_point p1,px\_point p2); |
| 说明 | **向量叉乘** |
| 参数 | P1,P2 两叉乘向量 |
| 返回值 | 返回叉乘结果 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_PointMod(px\_point p); |
| 说明 | **向量的模** |
| 参数 | P 该向量 |
| 返回值 | 返回该向量的模 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_PointSquare(px\_point p); |
| 说明 | **向量模的平方** |
| 参数 | P 该向量 |
| 返回值 | 返回该向量模的平方 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointDistance(px\_point p1,px\_point p2); |
| 说明 | **求两点间距离** |
| 参数 | P1 p2 两点 |
| 返回值 | 返回两点间距离 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointUnit(px\_point p); |
| 说明 | **求单位向量** |
| 参数 | P 该向量 |
| 返回值 | 返回该向量模的单位向量 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointMulMatrix(px\_point p,px\_matrix mat) |
| 说明 | **点乘矩阵** |
| 参数 | P 该向量，mat变换矩阵 |
| 返回值 | 返回变换后的点 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_Point\_sin (px\_point p) |
| 说明 | **求该向量与x轴夹角sin值** |
| 参数 | P 该向量 |
| 返回值 | Sin值 |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_Point\_cos (px\_point p) |
| 说明 | **求该向量与x轴夹角cos值** |
| 参数 | P 该向量 |
| 返回值 | cos值 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointReflectX(px\_point vector\_refer,px\_point respoint) |
| 说明 | **参照向量关于x轴夹角对一个点做相应旋转** |
| 参数 | vector\_refer 向量  respoint 原点 |
| 返回值 | 旋转后的点 |

|  |  |
| --- | --- |
| 函数名 | px\_point PX\_PointInverse(px\_point p1) |
| 说明 | **向量方向取反** |
| 参数 | P1 需要取反的向量 |
| 返回值 | 取反的向量 |

#### 交叉碰撞

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isLineCrossRect(px\_point p1,px\_point p2,px\_rect rect,px\_point \*cp1,px\_point \*cp2); |
| 说明 | **检测一条线与矩形是否相交** |
| 参数 | P1,p2 这条线的两个点  Rect 矩形范围  Cp1 如果有相交,输出交点1  Cp2 如果有相交,输出交点2 |
| 返回值 | 如果有交叉返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isRectCrossRect(px\_rect rect1,px\_rect rect2); |
| 说明 | **检测两个矩形是否有交叉碰撞** |
| 参数 | Rect1  Rect2  两矩形范围 |
| 返回值 | 如果有交叉返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isRectCrossCircle(px\_rect rect1,px\_point center,px\_float radius); |
| 说明 | **检测一个矩形和一个圆是否有交叉碰撞** |
| 参数 | Rect1 矩形  Center 圆心  Radius 圆半径 |
| 返回值 | 如果有交叉返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isCircleCrossCircle(px\_point center1,px\_float radius1,px\_point center2,px\_float radius2); |
| 说明 | **检测两个圆是否有交叉碰撞** |
| 参数 | Center1 Center2圆心1,2  Radius1 Radius2圆1,2半径 |
| 返回值 | 如果有交叉返回PX\_TRUE,否者返回PX\_FALSE |

#### 快速排序(由小到大)

|  |  |
| --- | --- |
| 函数名 | void PX\_Quicksort\_MinToMax(PX\_QuickSortAtom array[], px\_int left, px\_int right); |
| 说明 | **对特定结构体进行由小到大排序** |
| 参数 | typedef struct  {  px\_int power;//决定排序的权值  px\_void \*pData;  }PX\_QuickSortAtom;  array 排序序列  left 开始索引  right 结束索引 |

#### 快速排序(由大到小)

|  |  |
| --- | --- |
| 函数名 | void PX\_Quicksort\_MaxToMin(PX\_QuickSortAtom array[], px\_int left, px\_int right); |
| 说明 | **对特定结构体进行由大到小排序** |
| 参数 | typedef struct  {  px\_int power;//决定排序的权值  px\_void \*pData;  }PX\_QuickSortAtom;  array 排序序列  left 开始索引  right 结束索引 |

#### 矩形

px\_rect

typedef struct \_px\_rect

{

px\_float x,y,width,height;

}px\_rect;

x,y为矩形左上角坐标，width与height为宽高

|  |  |
| --- | --- |
| 函数名 | px\_rect PX\_RECT(px\_float x,px\_float y,px\_float width,px\_float height); |
| 说明 | **构造一个矩形** |
| 参数 | x,y 左上角坐标  width,height 矩形的宽度和高度 |
| 返回值 | 返回构造的矩形结构体 |

|  |  |
| --- | --- |
| 函数名 | px\_rect PX\_RECTPOINT2(px\_point p1,px\_point p2); |
| 说明 | **用两个点构造一个矩形** |
| 参数 | p1,p2 构造的两点 |
| 返回值 | 返回构造的矩形结构体 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isPointInRect(px\_point p,px\_rect rect) |
| 说明 | **判断点是否在矩形内** |
| 参数 | p该点，rect为该矩形 |
| 返回值 | 若在矩形内返回PX\_TRUE 否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_isRectInRect(px\_rect rect1,px\_rect rect2) |
| 说明 | **判断是两个矩形是否相交** |
| 参数 | Rect1 矩形1,rect2 矩形2 |
| 返回值 | 若在矩形内返回PX\_TRUE 否者返回PX\_FALSE |

#### CRC校验（CRC32）

|  |  |
| --- | --- |
| 函数名 | px\_uint32 PX\_crc32( px\_byte \*buffer, px\_uint size); |
| 说明 | **对一段缓存区进行CRC32校验** |
| 参数 | Buffer 缓存区  Size 缓存区大小 |
| 返回值 | CRC32校验值 |

#### 累加和校验（Sum32）

|  |  |
| --- | --- |
| 函数名 | px\_uint32 PX\_sum32( px\_byte \*buffer, px\_uint size); |
| 说明 | **对一段缓存区进行累加和校验** |
| 参数 | Buffer 缓存区  Size 缓存区大小 |
| 返回值 | 累加和校验值 |

#### 第一类修正贝塞尔函数

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_Bessel(int n,double x); |
| 说明 | **第一类修正贝塞尔函数** |
| 参数 | n,x |
| 返回值 | 对应PX\_Bessel(0,x), PX\_Bessel(1,x), PX\_Bessel(2,x)….. |

### 数据结构

#### 取得结构体偏移量

|  |  |
| --- | --- |
| 函数名 | PX\_STRUCT\_OFFSET(t,m) |
| 说明 | **取得结构体成员m对于结构体的偏移量** |
| 参数 | t 结构体类型  m 成员名 |
| 返回值 | 一个px\_uint类型,为m在该结构体的偏移量 |

#### 字母

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_CharIsNumeric(px\_char ch) |
| 说明 | **判断字母是否是数字** |
| 参数 | ch字母 |
| 返回值 | 若是返回PX\_TRUE,若不是返回PX\_FALSE |

#### 链表

PainterEngine提供链表模板函数px\_list，对该链表使用前都要改由PX\_ListInit进行初始化并在使用完毕后使用PX\_ListFree释放内存

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ListInit(px\_list \*list,px\_memorypool \*mp); |
| 说明 | **对链表结构进行初始化** |
| 参数 | list 需要初始化的结构指针  mp 链表部署内存池指针 |

##### 插入数据

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ListPush(px\_list \*list,px\_void \*data,px\_int size); |
| 说明 | **插入一个数据到链表** |
| 参数 | list 结构指针  data 数据指针  size 数据大小 |

##### 删除数据

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ListPop(px\_list \*list,px\_list\_node \*node); |
| 说明 | **将一个节点为node的数据从链表中删除** |
| 参数 | list 结构指针  node 链表节点 |
| 返回值 | 若成功删除返回PX\_TRUE，否者为PX\_FALSE |

##### 取索引节点

|  |  |
| --- | --- |
| 函数名 | px\_list\_node\* PX\_ListAt(px\_list \*list,px\_int index); |
| 说明 | **返回对应索引节点** |
| 参数 | list结构指针  index 链表节点索引 |
| 返回值 | 若成功返回该索引指针，否者返回PX\_NULL |

##### 取得链表当前节点数

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_ListSize(px\_list \*list); |
| 说明 | **取得链表当前节点数** |
| 参数 | list结构指针 |
| 返回值 | 当前节点数 |

##### 清空链表（释放占用内存）

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ListClear(px\_list \*list); |
| 说明 | **清空链表** |
| 参数 | list结构指针 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ListFree(px\_list \*list); |
| 说明 | **清空链表** |
| 参数 | list结构指针 |
| 返回值 | - |

##### 取节点数据

|  |  |
| --- | --- |
| 函数名 | PX\_LIST\_NODEDATA(x) |
| 说明 | **取节点数据** |
| 参数 | list结构指针 |
| 返回值 | - |

##### DEMO

px\_list list;

PX\_ListInit(&list,&mempool2);

PX\_ListPush(&list,"Hello",6);

PX\_ListPush(&list,"World",6);

PX\_ListPush(&list,"binar",6);

PX\_ListPop(&list,PX\_ListAt(&list,1));

PX\_ListPop(&list,PX\_ListAt(&list,1));

PX\_ListPop(&list,PX\_ListAt(&list,1));

for (int i=0;i<PX\_ListSize(&list);i++)

{

printf((px\_char \*)PX\_LIST\_NODEDATA(PX\_ListAt(&list,i)));

}

#### 容器

PainterEngine提供容器模板函数px\_vector，对该链表使用前都要改由PX\_VectorInit进行初始化并在使用完毕后使用PX\_VectorFree释放内存

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_VectorInit(px\_vector \*vec,px\_memorypool \*mp,px\_int nodeSize,px\_int init\_size); |
| 说明 | **对容器结构进行初始化** |
| 参数 | vec 容器指针  mp 容器部署内存池指针  nodesize 每个元素的大小  init\_size 初始容器大小(初始化分配的元素个数) |

##### 添加元素

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_VectorPushback(px\_vector \*vec,px\_void \*data); |
| 说明 | **在容器中添加一个元素** |
| 参数 | vec容器指针  data 压入数据指针 |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 设定元素

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_VectorSet(px\_vector \*vec,px\_uint index,px\_void \*data); |
| 说明 | **在容器中设定对应索引的元素,注意,如果索引小于分配容量,改容器将会重新分配内存空间以设定元素,同时索引小于使用大小,容器也会设定对应使用大小以适应设定索引** |
| 参数 | vec容器指针  index 索引  data 压入数据指针 |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 删除元素

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_VectorErase(px\_vector \*vec,px\_int index); |
| 说明 | **在容器中删除一个元素** |
| 参数 | vec容器指针  index 需要删除的索引 |
| 返回值 | 若成功返回PX\_TRUE，否者PX\_FALSE |

##### 清空容器(不释放预留内存)

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_VectorClear(px\_vector \*vec); |
| 说明 | **清空容器** |
| 参数 | vec容器指针 |
| 返回值 | - |

##### 取容器数据

|  |  |
| --- | --- |
| 函数名 | PX\_VectorAt(vec,i) |
| 说明 | **取容器数据** |
| 参数 | vec容器指针  I 索引 |
| 返回值 | - |

##### 释放容器(释放预留内存)

|  |  |
| --- | --- |
| 函数名 | PX\_VectorFree(x) |
| 说明 | **释放容器,同时释放内存,若需要重新使用该容器必须对其重新初始化** |
| 参数 | x容器指针 |
| 返回值 | - |

##### DEMO

px\_vector pxvector;

PX\_VectorInit(&pxvector,&mempool2,sizeof(int),4);

for(int i=0;i<4;i++)

PX\_VectorPushback(&pxvector,&i);

int i=9;

PX\_VectorPushback(&pxvector,&i);

for (int i=0;i<PX\_VectorSize(&pxvector);i++)

{

printf("%d",\*PX\_VectorAt(&pxvector,i));

}

#### 字符串映射表

##### 初始化映射表

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_MapInit(px\_memorypool \*mp,px\_map \*hashmap); |
| 说明 | **初始化一个**映射**表** |
| 参数 | mp,内存池  hashmap 映射表 |
| 返回值 | 如果成功返回PX\_TRUE,否者PX\_FALSE |

##### 插入元素

|  |  |
| --- | --- |
| 函数名 | PX\_HASHMAP\_RETURN PX\_MapPut (px\_map \* m, px\_char\* key, px\_void \* value); |
| 说明 | **插入一个节点到表中** |
| 参数 | m 哈希表  key 映射字符串  value 值 |
| 返回值 | 如果成功返回PX\_HASHMAP\_RETURN\_OK |

|  |  |
| --- | --- |
| 函数名 | PX\_HASHMAP\_RETURN PX\_MapPutInt (px\_map \* m, px\_char\* key, px\_int value); |
| 说明 | **插入一个int节点到表中** |
| 参数 | m 哈希表  key 映射字符串  value 值 |
| 返回值 | 如果成功返回PX\_HASHMAP\_RETURN\_OK |

|  |  |
| --- | --- |
| 函数名 | PX\_HASHMAP\_RETURN PX\_MapPutFloat (px\_map \* m, px\_char\* key, px\_int value); |
| 说明 | **插入一个float节点到表中** |
| 参数 | m 哈希表  key 映射字符串  value 值 |
| 返回值 | 如果成功返回PX\_HASHMAP\_RETURN\_OK |

##### 查找元素

|  |  |
| --- | --- |
| 函数名 | px\_void \*PX\_MapGet(px\_map \* m, px\_char\* stringkey); |
| 说明 | **查找哈希表对应key的元素** |
| 参数 | m 哈希表  key 映射字符串 |
| 返回值 | 对于返回类型 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_MapGetInt(px\_map \* m, px\_char\* stringkey,px\_int \*v); |
| 说明 | **查找哈希表对应key的int元素** |
| 参数 | m 哈希表  key 映射字符串  v返回赋值指针 |
| 返回值 | 如果成功返回PX\_TRUE,否则返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_MapGetFloat(px\_map \* m, px\_char\* stringkey); |
| 说明 | **查找哈希表对应key的float元素** |
| 参数 | m 哈希表  key 映射字符串  v返回赋值指针 |
| 返回值 | 如果成功返回PX\_TRUE,否则返回PX\_FALSE |

##### 删除元素

|  |  |
| --- | --- |
| 函数名 | PX\_HASHMAP\_RETURN PX\_MapErase (px\_map \* m, px\_char\* key); |
| 说明 | **删除对应key的元素** |
| 参数 | m 哈希表  key 映射字符串 |
| 返回值 | 如果成功返回PX\_HASHMAP\_RETURN\_OK |

##### 释放map

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MapFree(px\_map \* m) |
| 说明 | **删除对应map所有元素并释放内存资源** |
| 参数 | m 哈希表 |
| 返回值 | - |

#### 自适应内存

PainterEngine Core提供自动内存类型px\_memory,注意,px\_ memory使用前,都应该使用PX\_MemoryInit进行初始化,并在使用完毕后使用PX\_MemoryFree释放内存

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MemoryInit(px\_memory \*mp,px\_memory \*memory); |
| 说明 | **对自适应内存结构进行初始化** |
| 参数 | mp 容器部署内存池指针  memory 内存指针 |

##### 拼接内存

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_MemonyCat(px\_memory \*memory,px\_void \*buffer,px\_int size); |
| 说明 | **对自适应内存进行数据拼接** |
| 参数 | memory内存指针  buffer 需要拼接的内存指针  size 拼接大小 |
| 返回值 | 若成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 重置内存分配大小

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MemonyResize(px\_memory \*memory, px\_int size); |
| 说明 | **重置内存分配大小** |
| 参数 | memory内存指针  size 新的分配大小 |
| 返回值 | 若成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 释放内存

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MemoryFree(px\_memory \*memory); |
| 说明 | **释放内存** |
| 参数 | memory 内存指针 |

##### 取内存数据

|  |  |
| --- | --- |
| 函数名 | px\_byte \* PX\_MemoryData(px\_memory \*memory); |
| 说明 | **取内存数据指针** |
| 参数 | memory 内存指针 |
| 返回值 | 内存数据指针 |

##### 内存查找

|  |  |
| --- | --- |
| 函数名 | px\_byte \* PX\_MemoryFind(px\_memory \*memory,px\_void \*buffer,px\_int size); |
| 说明 | **在内存中查找匹配内存** |
| 参数 | memory 内存指针  buffer 匹配内存缓存  size 匹配内存缓存大小 |
| 返回值 | 如果找到返回该内存数据指针,否者返回PX\_NULL |

##### 内存拷贝

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MemoryCopy(px\_memory \*memory,px\_void \*buffer,px\_uint startoffset,px\_int size); |
| 说明 | **拷贝内存到目标内存结构中,注意,这个函数会依据拷贝内存大小自动调整内存部署** |
| 参数 | memory 内存指针  buffer 拷贝内存缓存  startoffset 拷贝到的起始地址  size 拷贝内存缓存大小 |
| 返回值 |  |

##### 内存清理

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_MemoryClear(px\_memory \*memory); |
| 说明 | **清空内存,这个函数并不会释放占用内存空间** |
| 参数 | memory 内存指针 |
| 返回值 |  |

#### 内存自适应字符串

PainterEngine Core提供自动内存调节的字符串类型px\_string,注意,px\_string的需要更多的性能开销,字符串使用前,都应该使用PX\_StringInit进行初始化,并在使用完毕后使用PX\_StringFree释放内存

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringInit(px\_memorypool \*mp,px\_string \*str); |
| 说明 | **对字符串结构进行初始化** |
| 参数 | mp 容器部署内存池指针  str 字符串指针 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringInit(px\_memorypool \*mp,px\_string \*str,px\_int allocsize); |
| 说明 | **对字符串结构进行初始化,使用最小2基数的allocsize分配内存** |
| 参数 | mp 容器部署内存池指针  str 字符串指针 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringInitFromConst(px\_string \*str,px\_char \*conststr); |
| 说明 | **使用一个字符串常量对字符串结构初始化,这个字符串额外占据的内存为0** |
| 参数 | str 字符串指针  conststr 字符串常量指针 |

##### 转换

###### 转换为整数

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_StringToInteger(px\_string \*str); |
| 说明 | **转换字符串为整数,支持十六进制转换(以0x或0X开头)** |
| 参数 | str 字符串指针 |
| 返回值 | 转换整数结果, |

###### 转换为整数

|  |  |
| --- | --- |
| 函数名 | px\_float PX\_StringToFloat(px\_string \*str); |
| 说明 | **转换字符串为浮点数** |
| 参数 | str 字符串指针 |
| 返回值 | 转换浮点结果, |

##### 设置字符串文本

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringSet(px\_string \*str,px\_char fmt[]); |
| 说明 | **设置字符串文本** |
| 参数 | str 字符串指针  fmt 需要设置的文本 |
| 返回值 | 成功返回PX\_TRUE,否者PX\_FALSE(内存不足) |

##### 去除首位空格

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringTrim(px\_string \*str); |
| 说明 | **去除首位空格,** |
| 参数 | str 字符串指针 |
| 返回值 | - |

##### 取字符串长度

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_StringLen(px\_string \*str); |
| 说明 | **取得字符串长度** |
| 参数 | str 字符串指针 |
| 返回值 | 字符长度 |

##### 字符串拼接

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringCatChar(px\_string \*str,px\_char ch); |
| 说明 | 将ch拼接到字符串str的尾部 |
| 参数 | str 字符串指针  ch 拼接字母 |
| 返回值 | - |
| 函数名 | px\_void PX\_StringCat(px\_string \*str,px\_char \*str2); |
| 说明 | **将str2拼接到字符串str的尾部** |
| 参数 | str 字符串指针  str2 拼接字符串指针 |
| 返回值 | - |

##### 释放字符串

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringFree(px\_string \*str); |
| 说明 | **释放字符串内存** |
| 参数 | str 字符串指针 |

##### 清理字符串

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringClear(px\_string \*str); |
| 说明 | **清理字符串文本(不释放内存)** |
| 参数 | str 字符串指针 |

##### 插入字符

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringInsertChar(px\_string \*str,px\_int index,px\_char ch); |
| 说明 | **将一个字符插入到字符串中** |
| 参数 | str 字符串指针  index 插入的位置索引  ch 要插入的字符 |

##### 删除字符

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringRemoveChar(px\_string \*str,px\_int index); |
| 说明 | **将一个字符到从字符串中删除** |
| 参数 | str 字符串指针  index 插入的位置索引  ch 要插入的字符 |

##### 裁剪字符串

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringTrimLeft(px\_string \*str,px\_int leftCount); |
| 说明 | **裁剪字符串左边的字符** |
| 参数 | str 字符串指针  leftCount裁去左边的字符数 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringTrimRight(px\_string \*str,px\_int RightCount); |
| 说明 | **裁剪字符串右边的字符** |
| 参数 | str 字符串指针  RightCount裁去左边的字符数 |

##### 格式化字符串

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringFormat8(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3, px\_stringformat \_4,px\_stringformat \_5, px\_stringformat \_6, px\_stringformat \_7, px\_stringformat \_8);  px\_bool PX\_StringFormat7(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3, px\_stringformat \_4,px\_stringformat \_5, px\_stringformat \_6, px\_stringformat \_7);  px\_bool PX\_StringFormat6(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3, px\_stringformat \_4,px\_stringformat \_5, px\_stringformat \_6);  px\_bool PX\_StringFormat5(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3, px\_stringformat \_4,px\_stringformat \_5);  px\_bool PX\_StringFormat4(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3, px\_stringformat \_4);  px\_bool PX\_StringFormat3(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2, px\_stringformat \_3);  px\_bool PX\_StringFormat2(px\_string \*str,px\_char fmt[],px\_stringformat \_1, px\_stringformat \_2);  px\_bool PX\_StringFormat1(px\_string \*str,px\_char fmt[],px\_stringformat \_1); |
| 说明 | **格式化字符串** |
| 参数 | 参考px\_sprintf1…的实现 |
|  |  |

##### 筛选

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringIsNumeric(px\_string \*str); |
| 说明 | **判断字符串是否为合法数字** |
| 参数 | str 字符串指针 |
| 返回值 | 如果是返回PX\_TRUE,否者PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_StringIsFloat(px\_string \*str); |
| 说明 | **判断字符串是否为合法浮点数** |
| 参数 | str 字符串指针 |
| 返回值 | 如果是返回PX\_TRUE,否者PX\_FALSE |

##### 字符串替换1

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringReplaceRange(px\_string \*str,px\_int startindex,px\_int endindex,px\_char \*replaceto) |
| 说明 | **字符串替换,将目标范围的字符串替换为replaceto里的字符串** |
| 参数 | str 字符串指针  startindex 开始索引  endindex 结束索引  replaceto 替换成的字符串 |
| 返回值 | - |

##### 字符串替换2

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_StringReplace(px\_string \*str,px\_char \*source,px\_char \*replaceto); |
| 说明 | **字符串替换,将str中包含source的文本都替换为replaceto** |
| 参数 | str 字符串指针  source 需要搜索的原始文本  replaceto 替换成的文本 |
| 返回值 | - |

### 几何绘制

#### 线

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawLine(px\_surface \*psurface, px\_int x0, px\_int y0, px\_int x1, px\_int y1 ,pt\_int lineWidth, px\_color color); |
| 说明 | **绘制一个反走样线段** |
| 参数 | psurface 渲染表面  x0 y0起始点坐标  x1 y1 终点坐标  lineWidth 线宽  color 颜色 |

#### 边框

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawBorder(px\_surface \*psurface, px\_int left, px\_int top, px\_int rignt, px\_int bottom ,px\_int lineWidth,px\_color color); |
| 说明 | **绘制一个边框** |
| 参数 | psurface 渲染表面  left top right bottom 位置描述  lineWidth 边框宽度像素  color 颜色 |

#### 矩形

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_color color); |
| 说明 | **绘制一个实心矩形** |
| 参数 | psurface 渲染表面  left top right bottom 位置描述  color 颜色 |

#### 实心圆

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawSolidCircle(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius,px\_color color ); |
| 说明 | **绘制一个反走样实心圆** |
| 参数 | psurface 渲染表面  x，y 圆心  radius 半径  color 颜色 |

#### 圆

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawCircle(px\_surface \*psurface,px\_int x,px\_int y,px\_int Radius ,pt\_int lineWidth,px\_color color); |
| 说明 | **绘制一个反走样圆** |
| 参数 | psurface 渲染表面  x，y 圆心  radius 半径  lineWidth 线宽  color 颜色 |

#### 圆笔

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawPenCircle(px\_surface \*psurface, px\_float x,px\_float y,px\_float Radius,px\_color color) |
| 说明 | **绘制一个反走样圆,这个算法用于高质量圆的绘制,可用这个算法组合为线段路径** |
| 参数 | psurface 渲染表面  x，y 圆心  radius 半径  lineWidth 线宽  color 颜色 |

#### 环

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawRing(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius,px\_int lineWidth,px\_color color,px\_uint start\_angle,px\_uint end\_angle) |
| 说明 | **绘制一个反走样环** |
| 参数 | psurface 渲染表面  x，y 环心  radius 半径  lineWidth 线宽  color 颜色  start\_angle 起始角度  end\_angle 终止角度  这个环遵循顺时针方向 |

#### 扇形

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawSector(px\_surface \*psurface, px\_int x,px\_int y,px\_int Radius\_outside,px\_int Radius\_inside,px\_color color,px\_uint start\_angle,px\_uint end\_angle); |
| 说明 | **绘制一个反走样扇形** |
| 参数 | psurface 渲染表面  x，y 环心  radius 半径  outside 外径  inside 内径  color 颜色  start\_angle 起始角度  end\_angle 终止角度  这个扇形遵循顺时针方向 |

#### 圆角矩形

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawRoundRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_float roundRaduis,px\_float linewidth,px\_color color); |
| 说明 | **绘制一个反走样圆角矩形** |
| 参数 | psurface 渲染表面  left top right bottom 位置描述  roundRadius 圆角半径  lineWidth 线宽  color 颜色 |

#### 实心圆角矩形

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawSolidRoundRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_float roundRaduis,px\_color color); |
| 说明 | **绘制一个反走样实心圆角矩形** |
| 参数 | psurface 渲染表面  left top right bottom 位置描述  roundRadius 圆角半径  color 颜色 |

#### 描边路径

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_GeoDrawPath(px\_surface \*psurface, px\_point path[],px\_int pathCount,px\_float linewidth,px\_color color) |
| 说明 | **描边路径**  **\*注意:渲染算法使用圆点插值算法进行绘制以获得高质量的描线,alpha值因为step也会有所放大(约4倍),这个算法耗费性能资源较大,这个渲染算法应该谨慎用于实时渲染表面** |
| 参数 | psurface 渲染表面  path 路径上所有的点  pathCount 点的数量  linewidth 线宽  color 颜色 |
|  |  |

### 图形图像

#### 渲染表面

##### 颜色

Px\_Color结构体

union

{

struct

{

px\_uchar b;

px\_uchar g;

px\_uchar r;

px\_uchar a;

};

px\_ulong ucolor;

}\_argb;

用于描述颜色的结构体，其中a代表alpha通道，r代表red通道，g代表green通道，b代表blue通道。

|  |  |
| --- | --- |
| 函数名 | px\_color PX\_COLOR(px\_uchar a,px\_uchar r,px\_uchar g,px\_uchar b); |
| 说明 | **构造颜色** |
| 参数 | A,r,g,b颜色分量 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ColorIncrease(px\_color \*color,px\_uchar inc); |
| 说明 | **增量颜色值** |
| 参数 | Color 指向需要增量的颜色结构  Inc 增量值 |

|  |  |
| --- | --- |
| 函数名 | px\_color PX\_ColorAdd(px\_color color1,px\_color color2); |
| 说明 | **颜色相加** |
| 参数 | Color1，color2 需要相加的两个颜色 |
| 返回值 | 返回相加的颜色结果 |

|  |  |
| --- | --- |
| 函数名 | px\_color PX\_ColorSub(px\_color color1,px\_color color2); |
| 说明 | **颜色相减** |
| 参数 | Color1，color2 需要相减的两个颜色 |
| 返回值 | 返回相减的颜色结果 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ColorEqual(px\_color color1,px\_color color2); |
| 说明 | **判断两个颜色是否相等** |
| 参数 | Color1，color2 需要判断的两个颜色 |
| 返回值 | 如果相等返回PX\_TRUE,否者返回PX\_FALSE |

##### 表面操作

px\_surface结构体

typedef struct \_PX\_Surface

{

px\_color \*surfaceBuffer;

px\_uint height;

px\_uint width;

px\_memorypool MP;

}px\_surface;

Note：用于描述渲染内存的结构体，仅支持32位ARGB色彩格式。

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SurfaceCreate(px\_memorypool \*mp,px\_uint height,px\_uint width,px\_surface \*surface); |
| 说明 | **创建一个渲染表面** |
| 参数 | Mp 内存池指针  Height 高  Width 宽  Surface 渲染表面指针 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SurfaceFree(px\_surface \*psurface); |
| 说明 | **释放一个渲染表面** |
| 参数 | psurface 指向需要释放的渲染表面 |

##### 像素操作

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SurfaceDrawPixel(px\_surface \*ps,px\_int x,px\_int y,px\_color color); |
| 说明 | **绘制一个像素** |
| 参数 | ps 指向表面  x，y 坐标  color 颜色 |
| 返回值 |  |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SurfaceDrawPixelFaster(px\_surface \*ps,px\_int x,px\_int y,px\_color color); |
| 说明 | **绘制一个像素,注意,该函数不会检查表面边界!!!确保在边界内避免内存越界访问!!** |
| 参数 | ps 指向表面  x，y 坐标  color 颜色 |
| 返回值 |  |

|  |  |
| --- | --- |
| 函数名 | PX\_SUFACECOLOR(Surface,X,Y) |
| 说明 | **查找像素颜色** |
| 参数 | Surface指向表面  X，Y 坐标 |
| 返回值 | 对应px\_color |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SetPixel(px\_surface \*ps,px\_int x,px\_int y,px\_color color); |
| 说明 | **设置一个像素** |
| 参数 | ps 指向表面  x，y 坐标  color 颜色 |
| 返回值 |  |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SurfaceSetRect(px\_surface \*psurface, px\_int left, px\_int top, px\_int right, px\_int bottom,px\_color color) |
| 说明 | **将一个渲染表面的一个矩形区域设置为某种颜色** |
| 参数 | psurface 渲染表面  left top right bottom 位置描述  color 颜色 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_SurfaceMemorySize(px\_uint width,px\_uint height); |
| 说明 | **预估一个表面的占用内存(实际占用依据内存池结构应适当增大)** |
| 参数 | width 表面宽度  height 表面高度 |
| 返回值 | - |

#### 纹理

px\_texture是PainterEngine的纹理对象结构，建议所有的图像数据都最终加载为px\_texture

其本质上仍然是一个surface,其指针可以和px\_surface互换使用.

##### 创建纹理

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCreate(px\_memorypool \*mp,px\_texture \*tex,px\_int width,px\_int height); |
| 说明 | **创建一个Texture表面** |
| 参数 | mp 内存池指针  tex 输出表面指针  width 宽  height 高 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCreateFromMemory(px\_memorypool \*mp,px\_void \*data,px\_int size,px\_texture \*tex); |
| 说明 | **从支持的数据创建一个Texture表面（如bmp或TRaw）** |
| 参数 | mp 内存池指针  data 数据指针  size 宽  tex 纹理格式 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 缩放纹理

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCreateScale(px\_memorypool \*mp,px\_texture \*resTexture,px\_int newWidth,px\_int newHeight,px\_texture \*out); |
| 说明 | **使用窗采样缩放一个纹理并创造缩放后的纹理** |
| 参数 | mp 内存池指针  resTexture 原纹理指针  newWidth newHeight 缩放后的大小  out 缩放后的纹理指针,在使用完毕后改指针必须被释放 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 拷贝纹理

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCopy(px\_memorypool \*mp,px\_texture \*resTexture, px\_texture \*out); |
| 说明 | **深拷贝一个纹理到目标纹理中** |
| 参数 | mp 内存池指针  resTexture 原纹理指针  out 拷贝到目标纹理 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 渲染纹理

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRender(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend); |
| 说明 | **渲染一个纹理到表面** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND; |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderRotation(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_int Angle); |
| 说明 | **渲染一个纹理到表面,并顺时针旋转一个角度** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  Angle 旋转的角度  注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用PX\_TextureRenderEx,以较低的显示损失获得8倍左右的性能 |
| 返回值 | - |
| 函数名 | px\_void PX\_TextureRenderRotation\_vector(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_point p\_vector); |
| 说明 | 渲染一个纹理到表面,并顺时针旋转一个角度 |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  P\_vector 旋转向量,以x轴正方向为基准  注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用PX\_TextureRenderEx,以较低的显示损失获得8倍左右的性能 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderRotation\_sincos(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float sin,px\_float cos); |
| 说明 | 渲染一个纹理到表面,并顺时针旋转一个角度 |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  Sin cos 旋转矩阵对于sin cos值  注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用PX\_TextureRenderEx,以较低的显示损失获得8倍左右的性能 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderEx(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_float rotation); |
| 说明 | **渲染一个纹理到表面,注意,这个渲染方式的速度不如PX\_TextureRender,但提供了纹理的缩放和旋转(同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  scale 缩放大小  rotation 旋转角度 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderEx\_sincos(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_float sin,px\_float cos); |
| 说明 | **渲染一个纹理到表面,注意,这个渲染方式的速度不如PX\_TextureRender,但提供了纹理的缩放和旋转(以旋转矩阵sin cos形式,同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  scale 缩放大小  sin 旋转矩阵sin值  cos 旋转矩阵 cos值 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderEx(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_float rotation); |
| 说明 | **渲染一个纹理到表面,注意,这个渲染方式的速度不如PX\_TextureRender,但提供了纹理的缩放和旋转(同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  scale 缩放大小  rotation 旋转角度 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureRenderEx\_vector(px\_surface \*psurface,px\_texture \*tex,px\_int x,px\_int y ,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_point p\_vector); |
| 说明 | **渲染一个纹理到表面,注意,这个渲染方式的速度不如PX\_TextureRender,但提供了纹理的缩放和旋转(以旋转矩阵vector形式,同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)**  **默认的指向方向为** |
| 参数 | psurface 渲染到的表面  px\_texture需要渲染的纹理  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  typedef struct  {  float hdr\_R; //HDR of Red  float hdr\_G; //HDR of Green  float hdr\_B; //HDR of Blue  float alpha; //Blend of alpha  }PX\_TEXTURERENDER\_BLEND;  scale 缩放大小  指向方向 |
| 返回值 | - |

##### 创建旋转纹理

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCreateRotationAngle(px\_memorypool \*mp,px\_texture \*resTexture,px\_float Angle,px\_texture \*out); |
| 说明 | **将一个纹理按角度进行顺时针旋转并创建一个新纹理** |
| 参数 | mp 内存池指针  resTexture 原纹理  Angle 旋转的角度  out 输出纹理 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureRotationAngleToTexture(px\_texture \*resTexture,px\_float Angle,px\_texture \*out); |
| 说明 | **将一个纹理按角度进行顺时针旋转并拷贝到目标纹理** |
| 参数 | resTexture 原纹理  Angle 旋转的角度  out 输出纹理 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TextureCreateRotationRadian(px\_memorypool \*mp,px\_texture \*resTexture,px\_float Rad,px\_texture \*out); |
| 说明 | **将一个纹理按弧度进行顺时针旋转并创建一个新纹理** |
| 参数 | mp 内存池指针  resTexture 原纹理  Rad旋转的弧度  out 输出纹理 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 释放纹理

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TextureFree(px\_texture \*tex); |
| 说明 | **释放纹理内存** |
| 参数 | tex 纹理指针 |
| 返回值 | - |

#### 轮廓

PX\_Shape是painterEngine内建的轮廓格式,轮廓可以被认为是仅带有灰度及位置信息的纹理,轮廓的每个位置信息都由一个256阶的灰度表示(1字节)在渲染轮廓的时候,需要制定一个颜色信息以对轮廓进行渲染.

##### 创建轮廓

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ShapeCreate(px\_memorypool \*mp,px\_shape \*shape,px\_int width,px\_int height); |
| 说明 | **创建一个轮廓** |
| 参数 | mp 内存池指针  shape 输出轮廓  width 宽  height 高 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 创建纹理轮廓

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ShapeCreateFromTexture(px\_memorypool \*mp,px\_shape \*shape,px\_texture \*texture); |
| 说明 | **从纹理创建一个轮廓** |
| 参数 | mp 内存池指针  shape 输出轮廓  texture 映射纹理 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ShapeCreateFromMemory(px\_memorypool \*mp,px\_void \*data,px\_int size,px\_shape \*shape); |
| 说明 | **从内存创建一个轮廓(支持TRaw内建格式)** |
| 参数 | mp 内存池指针  shape 输出轮廓  data 数据  size 数据大小 |
| 返回值 | 若成功返回PX\_TRUE 否者PX\_FALSE |

##### 渲染轮廓

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ShapeRender(px\_surface \*psurface,px\_shape \*shape,px\_int x,px\_int y,PX\_TEXTURERENDER\_REFPOINT refPoint,px\_color blendColor); |
| 说明 | **渲染一个轮廓到表面** |
| 参数 | psurface 渲染到的表面  px\_shape需要渲染的轮廓  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  blendcolor 渲染颜色 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ShapeRenderEx(px\_surface \*psurface,px\_shape \*shape,px\_int x,px\_int y,PX\_TEXTURERENDER\_REFPOINT refPoint,px\_color blendColor,px\_float scale,px\_float Angle); |
| 说明 | **旋转缩放后渲染一个轮廓到表面** |
| 参数 | psurface 渲染到的表面  shape 需要渲染的轮廓  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  blendcolor 渲染颜色  scale 缩放大小  rotation 旋转角度 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ShapeRenderEx\_sincos(px\_surface \*psurface,px\_shape \*shape,px\_int x,px\_int y,PX\_TEXTURERENDER\_REFPOINT refPoint,px\_color blendColor,px\_float scale,px\_float sinx,px\_float cosx); |
| 说明 | **旋转后渲染一个轮廓到表面** |
| 参数 | psurface 渲染到的表面  shape 需要渲染的轮廓  x，y偏移量(该坐标以纹理左上角为参照)  refPoint 参考中心点，用于表示纹理绘制的相对位置  blend blend类型结构体，用于调整绘制纹理的alpha，hdr值  当它为PX\_NULL时，表示采用默认blend值  blendcolor 渲染颜色  sinx cosx,旋转矩阵对应sinx cosx位置值,这个函数在以向量为标准的旋转标定渲染中尤为有用 |
| 返回值 | - |

##### 释放轮廓

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ShapeFree(px\_shape \*shape); |
| 说明 | **释放轮廓** |
| 参数 | shape 轮廓指针 |
| 返回值 | - |

#### 位图

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_BitmapGetHeight(void \*BitmapBuffer); |
| 说明 | **取得位图高度** |
| 参数 | BitmapBuffer 位图数据 |
| 返回值 | 位图的高度(可能为负数) |
|  |  |

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_BitmapGetWidth(void \*BitmapBuffer); |
| 说明 | **取得位图宽度** |
| 参数 | BitmapBuffer 位图数据 |
| 返回值 | 位图的宽度(可能为负数) |
|  |  |

|  |  |
| --- | --- |
| 函数名 | px\_word PX\_BitmapGetBitCount(void \*BitmapBuffer); |
| 说明 | **取得位图像素数据** |
| 参数 | BitmapBuffer 位图数据 |
| 返回值 | 位图的位图像素数据 |

|  |  |
| --- | --- |
| 函数名 | void PX\_BitmapRender(px\_surface surface,void \*BitmapBuffer,px\_int BufferSize,int x,int y); |
| 说明 | **渲染位图到表面** |
| 参数 | surface 比阿姆  BitmapBuffer 位图数据  BufferSize 数据长度  x,y偏移量 |
| 返回值 | 位图的位图像素数据 |

#### TRaw纹理格式

typedef struct \_PX\_TRaw\_Header

{

px\_dword Magic;//TRAW 0x57415254

px\_int Width;

px\_int Height;

}PX\_TRaw\_Header;

TRaw文件头，TRaw文件是一个后缀为tex的文件

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TRawVerify(px\_void \*data,px\_int size); |
| 说明 | **判断是否为TRaw合法文件格式** |
| 参数 | data 数据指针 size 数据大小 |
| 返回值 | 返回TRUE表示合法数据否者非合法数据 |

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_TRawGetWidth(px\_void \*data); |
| 说明 | **取得TRaw文件的图像宽度** |
| 参数 | data 数据指针 |
| 返回值 | 返回图像宽度 |

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_TRawGetHeight(px\_void \*data); |
| 说明 | **取得TRaw文件的图像高度** |
| 参数 | data 数据指针 |
| 返回值 | 返回图像宽度 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TRawRender(px\_surface \*psurface,px\_void \*data,px\_int x,px\_int y); |
| 说明 | **绘制TRaw数据** |
| 参数 | psurface 表面指针  data TRaw数据  x，y偏移量 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TRawBuild(px\_surface \*psurface,px\_byte \*TRawBuffer,px\_int \*size); |
| 说明 | **将pSurface转换为TRAW格式数据** |
| 参数 | psurface 表面指针  TRawBuffer 输出数据(当TRAWBuffer不为PX\_NULL时)  size 输出大小 |
| 返回值 | - |

### 字体

PainterEngine 支持GBK编码及ASCII码的字符绘制，提供如下接口

#define \_\_PX\_FONT\_GBKSIZE 13 //中文字符宽度

#define \_\_PX\_FONT\_ASCSIZE 8 //英文字符宽度

#define \_\_PX\_FONT\_HEIGHT 12 //字符高度

#### 绘制字体

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_FontDrawChar(px\_surface \*psurface,int x,int y,px\_char \*Text,px\_color Color,PX\_FONT\_ALIGN align); |
| 说明 | **绘制一个中/英字体** |
| 参数 | psurface 表面  x,y偏移坐标  Text GBK多字符集文本指针  Color 颜色  Align 字体对齐模式  PX\_FONT\_ALIGN\_LEFTTOP, 左上角对齐  PX\_FONT\_ALIGN\_MIDTOP,居中顶部对齐  PX\_FONT\_ALIGN\_RIGHTTOP,右上角对齐  PX\_FONT\_ALIGN\_LEFTMID,靠左居中对齐  PX\_FONT\_ALIGN\_CENTER,中心对齐  PX\_FONT\_ALIGN\_RIGHTMID,靠右居中对齐  PX\_FONT\_ALIGN\_LEFTBOTTOM,靠左底部对齐 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_FontDrawText(px\_surface \*psurface,int x,int y,PX\_FONT\_ALIGN align,const px\_char \*Text,px\_color Color); |
| 说明 | **绘制ANSI文本** |
| 参数 | psurface 表面  x,y偏移坐标  align 对齐模式  PX\_FONT\_ALIGN\_LEFTTOP, 左上角对齐  PX\_FONT\_ALIGN\_MIDTOP,居中顶部对齐  PX\_FONT\_ALIGN\_RIGHTTOP,右上角对齐  PX\_FONT\_ALIGN\_LEFTMID,靠左居中对齐  PX\_FONT\_ALIGN\_CENTER,中心对齐  PX\_FONT\_ALIGN\_RIGHTMID,靠右居中对齐  PX\_FONT\_ALIGN\_LEFTBOTTOM,靠左底部对齐  PX\_FONT\_ALIGN\_MIDBOTTOM,居中底部对齐  PX\_FONT\_ALIGN\_RIGHTBOTTOM,靠右底部对齐  Text ANSI字符集文本指针  Color 颜色 |
| 返回值 | - |

#### 取绘制字体像素宽度

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_GetFontTextPixelsWidth( px\_char \*Text ) |
| 说明 | **取得绘制文本的像素绘制宽度** |
| 参数 | Text 文本 |
| 返回值 | 文本宽度 |

#### 字模初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_FontModuleInitialize(px\_memorypool \*mp,PX\_FontModule \*module,PX\_FONTMODULE\_CODEPAGE codepage) |
| 说明 | **初始化一个字模库** |
| 参数 | Mp 内存池  Module 字模库  codepage 字模代码页 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 加载字模

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_FontModuleLoad(PX\_FontModule \*module,px\_byte \*buffer,px\_int size); |
| 说明 | **加载pxf字模到字模库,重复的字模以之前加载的为准** |
| 参数 | module 字模库  buffer pxf数据  size pxf数据大小 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 释放字幕库

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_FontModuleFree(PX\_FontModule \*module); |
| 说明 | **释放字模库** |
| 参数 | module 字模库 |
| 返回值 | - |

#### 以字模库绘制文本

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_FontModuleDrawText(px\_surface \*psurface,PX\_FontModule \*mod,int x,int y,PX\_FONT\_ALIGN align,const px\_char \*Text,px\_color Color) |
| 说明 | **以该字模库绘制文本,注意,输入的字模必须是UTF-16 littleendia编码的** |
| 参数 | Psurface 目标表面  X 原点x坐标  Y 原点y坐标    Text 文本  Color字颜色  Mod 字模库  Align 字体对齐模式  PX\_FONT\_ALIGN\_LEFTTOP, 左上角对齐  PX\_FONT\_ALIGN\_MIDTOP,居中顶部对齐  PX\_FONT\_ALIGN\_RIGHTTOP,右上角对齐  PX\_FONT\_ALIGN\_LEFTMID,靠左居中对齐  PX\_FONT\_ALIGN\_CENTER,中心对齐  PX\_FONT\_ALIGN\_RIGHTMID,靠右居中对齐  PX\_FONT\_ALIGN\_LEFTBOTTOM,靠左底部对齐 |
| 返回值 | 绘制文本的像素宽度 |

#### 设置字模库字间间隔

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_FontModuleSetXYSpace(PX\_FontModule \*module,int x,int y); |
| 说明 | **设置字模库绘制时字间间隔** |
| 参数 | module 字模库  x 空格的间隔  y 换行的间隔 |
| 返回值 | - |

### 信号处理

#### 离散傅里叶变换（DFT）

**复数结构体**

typedef struct \_\_complex

{

float re;// really

float im;// imaginary

}px\_complex;

|  |  |
| --- | --- |
| 函数名 | void PX\_DFT(\_IN complex x[],\_OUT complex X[],int N); |
| 说明 | **对复数信号进行DFT正变换** |
| 参数 | x 复信号  X 变换结果  N 复信号长度 |

|  |  |
| --- | --- |
| 函数名 | void PX\_IDFT(\_IN complex X[],\_OUT complex x[],int N); |
| 说明 | **对复数信号进行IDFT即逆变换** |
| 参数 | X 复信号  x 逆变换结果  N 复信号长度 |

#### 离散余弦变换（DCT）

**复数结构体**

|  |  |
| --- | --- |
| 函数名 | void PX\_DCT(\_IN px\_double x[],\_OUT px\_double X[],int N); |
| 说明 | **对信号进行DCT正变换** |
| 参数 | x 复信号  X 变换结果  N 复信号长度 |

|  |  |
| --- | --- |
| 函数名 | void PX\_IDFT(\_IN px\_double X[],\_OUT px\_double x[],int N); |
| 说明 | **对信号进行IDCT即逆变换** |
| 参数 | X 复信号  x 逆变换结果  N 复信号长度 |

#### 快速傅里叶变换（FFT）

|  |  |
| --- | --- |
| 函数名 | void PX\_FFT(\_IN complex x[],\_OUT complex X[],int N); |
| 说明 | **对复数信号进行FFT正变换** |
| 参数 | x 复信号  X 变换结果  N 复信号长度（必须是2基数） |

|  |  |
| --- | --- |
| 函数名 | void PX\_IFFT(\_IN complex X[],\_OUT complex x[],int N); |
| 说明 | **对复数信号进行IFFT即逆变换** |
| 参数 | X 复信号  x 逆变换结果  N 复信号长度（必须是2基数） |

|  |  |
| --- | --- |
| 函数名 | void PX\_FFT\_2(\_IN complex x[],\_OUT complex X[],int N\_N); |
| 说明 | **对复数信号进行二维FFT正变换** |
| 参数 | x二维复信号矩阵  X 变换结果  N\_N 复信号边长（复信号必须是2基边长的正方矩阵） |

|  |  |
| --- | --- |
| 函数名 | void PX\_IFFT\_2(\_IN complex X[],\_OUT complex x[],int N\_N); |
| 说明 | **对复数信号进行IFFT即逆变换** |
| 参数 | X 二维复信号矩阵  x 逆变换结果  N\_N 复信号边长（复信号必须是2基边长的正方矩阵） |

|  |  |
| --- | --- |
| 函数名 | void PX\_FFT\_2\_Shift(\_IN complex \_in[],\_OUT complex \_out[],int N\_N); |
| 说明 | **对复数信号进行FFTShift** |
| 参数 | \_in 二维复信号矩阵  \_out 逆变换结果  N\_N 复信号边长（复信号必须是2基边长的正方矩阵） |

#### 强制共轭对称

|  |  |
| --- | --- |
| 函数名 | void PX\_FT\_Symmetry(\_IN px\_complex X[],\_OUT px\_complex x[],px\_int N); |
| 说明 | **对一个傅里叶变换频域信号强制共轭对称(参考前半段数据)** |
| 参数 | x输入信号  X 输出信号  N 长度 |

#### 上采样(插值, UpSampled)

|  |  |
| --- | --- |
| 函数名 | void PX\_UpSampled(\_IN px\_complex x[],\_OUT px\_complex X[],px\_int N,px\_int L) |
| 说明 | **对一个信号进行上采样** |
| 参数 | x输入信号  X 输出信号  N 原长度  L 上采样倍数(1/M)  注意:输出数据缓存的长度应该至少是原长度的2\*L倍  如果原信号或输出信号长度不是2基数,将使用较慢的DFT进行上采样 |

#### 下采样(抽取, SubSampled, DownSampled)

|  |  |
| --- | --- |
| 函数名 | void PX\_DownSampled (\_IN px\_complex x[],\_OUT px\_complex X[],px\_int N,px\_int M) |
| 说明 | **对一个信号进行下采样(SubSampled)** |
| 参数 | x输入信号  X 输出信号  N 原长度  M 下采样倍数(1/M)  注意:输出数据缓存的长度应该至少是原长度的两倍  如果原信号或输出信号长度不是2基数,将使用较慢的DFT进行下采样 |

#### tukey窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_tukey(px\_double data[],px\_int N); |
| 说明 | **构造一个tukey窗函数** |
| 参数 | data 数据缓存  N 长度 |
| 返回值 |  |

#### triangular窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_ triangular (px\_double data[],px\_int N); |
| 说明 | **构造一个triangular窗函数** |
| 参数 | data 数据缓存  N 长度 |
| 返回值 |  |

#### blackMan窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_blackMan(px\_double data[],px\_int N); |
| 说明 | **构造一个blackman窗函数** |
| 参数 | data 数据缓存  N 长度 |
| 返回值 |  |

#### hamming窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_ hamming (px\_double data[],px\_int N); |
| 说明 | **构造一个hamming窗函数** |
| 参数 | data 数据缓存  N 长度 |
| 返回值 |  |

#### hanning窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_ hanning (px\_double data[],px\_int N); |
| 说明 | **构造一个hanning窗函数** |
| 参数 | data 数据缓存  N 长度 |
| 返回值 |  |

#### kaiser窗

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_kaiser(px\_double beta,px\_double data[],px\_int N); |
| 说明 | **构造一个kaiser窗函数** |
| 参数 | beta 就是字面的意思  data 数据缓存  N 长度 |
| 返回值 |  |

#### 应用窗函数

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_WindowFunction\_Apply(px\_double data[],px\_double window[],px\_int N); |
| 说明 | **将一个窗函数应用于离散信号** |
| 参数 | data 数据缓存  window 窗函数缓存  N 长度 |
| 返回值 |  |

#### 计算系统幅频响应和相频响应

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_gain(px\_double b[],px\_double a[],px\_int m,px\_int n,px\_double x[],px\_double y[],px\_int len,px\_int sign); |
| 说明 | **计算系统幅频响应和相频响应** |
| 参数 | b存放滤波器分子多项式系数b[i],长度为m+1  a存放滤波器分母多项式系数a[i],长度为n+1  m 滤波器分子多项式阶数  n滤波器分母多项式阶数  x,y 输出数组,长度为len,当sign为1时存放滤波器幅频响应|H(w)|(x为实部,y为虚部),当sign为2时存放分贝表示的滤波器幅频相频  len 见上一项说明  sign 见上一项说明 |
| 返回值 |  |

#### 计算级联型系统幅频响应和相频响应

|  |  |
| --- | --- |
| 函数名 | px\_double b[],px\_double a[],px\_int n,px\_int ns,px\_double x[],px\_double y[],px\_int len,px\_int sign |
| 说明 | **计算级联型系统幅频响应和相频响应** |
| 参数 | b存放滤波器分子多项式系数b[i],矩阵,体积为ns\*(n+1)  a存放滤波器分母多项式系数a[i],矩阵,体积为ns\*(n+1)  n级联滤波器每节多项式阶数  ns 级联滤波器n阶节数L  x,y 输出数组,长度为len,当sign为1时存放滤波器幅频响应|H(w)|(x为实部,y为虚部),当sign为2时存放分贝表示的滤波器幅频相频  len 见上一项说明  sign 见上一项说明 |
| 返回值 |  |

#### 倒谱

|  |  |
| --- | --- |
| 函数名 | void PX\_Cepstrum(\_IN px\_complex x[],\_OUT px\_complex X[],px\_int N, PX\_CEOSTRUM\_TYPE type); |
| 说明 | **计算一个信号的倒谱域** |
| 参数 | x输入信号  X倒谱域  N信号长度  Type 倒谱类型, PX\_CEOSTRUM\_TYPE\_COMPLEX表示复倒谱  PX\_CEOSTRUM\_TYPE\_REAL 表示实倒谱 |
| 返回值 |  |

#### 人声基音频率估算

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_PitchEstimation(\_IN px\_complex x[],px\_int N,px\_int sampleRate); |
| 说明 | **估算人声基音频率** |
| 参数 | x输入信号  N信号长度  sampleRate 信号采样率 |
| 返回值 | 估算的基音频率 |

#### 估算瞬时频率

|  |  |
| --- | --- |
| 函数名 | px\_sine PX\_PhaseVocoder(px\_sine src,px\_double p2,px\_double delta\_t) |
| 说明 | **估算瞬时频率** |
| 参数 | src 原正弦波信号  p2 第二相位  delta\_t 取样时间 |
| 返回值 | 带有瞬时频率的正弦曲线 |

#### 预加重

|  |  |
| --- | --- |
| 函数名 | void PX\_PreEmphasise(const px\_double \*data, int len, px\_double \*out, px\_double preF) |
| 说明 | **对一个实信号进行预加重** |
| 参数 | data 输入信号  len 长度  out 输出预加重信号  preF 加重权值 范围为0.9-1.0 一般取0.97 |
| 返回值 |  |

#### WAV文件格式

##### 取得wav声道数目

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_WAVEGetChannel(px\_byte \*buffer,px\_int size); |
| 说明 | **取得一个wav数据的声道数量** |
| 参数 | buffer指向wav数据指针  size wav数据长度 |
| 返回值 | 声道数量 |

##### 验证wav数据

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_WAVEVerify(px\_byte \*buffer,px\_int size); |
| 说明 | **验证一段数据是否是可以加载的wav数据** |
| 参数 | buffer指向wav数据指针  size wav数据长度 |
| 返回值 | 如果可以返回PX\_TRUE,否者返回PX\_FALSE |

##### 取得wav数据的PCM流大小

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_WAVEGetPCMSize(px\_byte \*buffer,px\_int size); |
| 说明 | **取得wav数据的PCM流大小** |
| 参数 | buffer指向wav数据指针  size wav数据长度 |
| 返回值 | 返回数据长度 |

#### PCM混音器

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SoundInit(px\_memorypool \*mp,PX\_SoundPlay \*pSound); |
| 说明 | **初始化一个声音混音器** |
| 参数 | mp 内存池  pSoundPlay sound结构体 |

##### 加载播放数据

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SoundAdd(PX\_SoundPlay \*pSound,PX\_Sound sounddata); |
| 说明 | **加载一个播放数据混音器中** |
| 参数 | pSoundPlay sound结构体  sounddata 播放数据 |
| 返回值 | 返回PX\_TRUE表示成功,否者返回PX\_FALSE |

##### 读取混音PCM数据流

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SoundRead(PX\_SoundPlay \*pSound,px\_byte \*pBuffer,px\_int readSize); |
| 说明 | **读取混音后的PCM流** |
| 参数 | pSoundPlay sound结构体  buffer PCM数据流  readsize 读取长度 |
| 返回值 | 返回PX\_TRUE表示成功,否者返回PX\_FALSE |

##### 释放混音器

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SoundFree(PX\_SoundPlay \*pSound); |
| 说明 | **释放混音器** |
| 参数 | pSoundPlay sound结构体 |
| 返回值 | - |

##### 清除混音器所有音效

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SoundClear(PX\_SoundPlay \*pSound); |
| 说明 | **清除混音器所有音效** |
| 参数 | pSoundPlay sound结构体 |
| 返回值 | - |

##### 加载静态音源

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SoundStaticDataCreate(PX\_SoundStaticData \*sounddata,px\_memorypool \*mp,px\_byte \*data,px\_int datasize); |
| 说明 | **加载一个静态音源,在使用结束后,这个音源需要手动释放** |
| 参数 | sounddata 静态音源  mp 存储内存池  data 原始解析数据(例如WAV格式)  data 原始解析数据大小 |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 释放静态音源

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SoundStaticDataFree(PX\_SoundStaticData \*sounddata); |
| 说明 | **释放静态音源** |
| 参数 | sounddata 静态音源  mp 存储内存池  data 原始解析数据(例如WAV格式)  data 原始解析数据大小 |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 相位声码器

##### 时域延拓

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TuningTimeScale(px\_double timescale,px\_double in[],px\_int count,px\_memory \*out); |
| 说明 | **基于相位声码器的变速不变调TSM(时域压拓)系统** |
| 参数 | **timescale 时域压拓系数**  **in 输入信号(建议归一化处理)**  count 信号数量  out 输出信号,该内存变量需要被初始化 |
| 返回值 | 成功返回PX\_TRUE,否者PX\_FALSE |

##### 变调不变速

###### 调音台初始化

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningInitialize(px\_memorypool \*mp,PX\_Tuning \*tuning,px\_double pitchShift,px\_double window[],px\_double filter[],px\_double fix[],PX\_TUNING\_WINDOW\_SIZE windowsize); |
| 说明 | **基于重采样和相位声码器的变调不变速系统调音台初始化** |
| 参数 | **mp 计算内存池**  **tuning 调音台指针**  **pitchshift 调变系数,例如音调提升2倍则为2,降低一倍则为0.5**  window 窗函数(建议为能量相等的处理窗,50% overlap)  filter 滤波器,长度等于windowsize指示长度  fix 增量值,长度等于windowsize指示长度(归一化后)  windowsize 窗函数类型  PX\_TUNING\_WINDOW\_SIZE\_8, 长度为8的窗,主要用于测试  PX\_TUNING\_WINDOW\_SIZE\_128, 长度为128的窗  PX\_TUNING\_WINDOW\_SIZE\_256, 长度为256的窗  PX\_TUNING\_WINDOW\_SIZE\_512, 长度为512的窗,建议值  PX\_TUNING\_WINDOW\_SIZE\_1024, 长度为1024的窗,建议值  PX\_TUNING\_WINDOW\_SIZE\_2048, 长度为2048的窗,建议值  PX\_TUNING\_WINDOW\_SIZE\_4096, 长度为4096的窗  PX\_TUNING\_WINDOW\_SIZE\_8192, 长度为8192的窗 |
| 返回值 |  |

###### 调音台滤波器系统

|  |  |
| --- | --- |
| 函数名 | px\_int PX\_TuningFilter(PX\_Tuning \*tuning,\_IN px\_double frame\_unit[],px\_int Size,\_OUT px\_double out\_unit[]); |
| 说明 | **基于重采样和相位声码器的变调不变速系统调音台滤波器系统**  **\*该滤波器为FIR类型滤波器**  **\*输入信号必须是连续的** |
| 参数 | **tuning 调音台指针**  frame\_unit[] 输入数据(建议归一化处理)  Size 输入数据长度  out\_unit 输出数据 |
| 返回值 | 输出长度 |

###### 调音台TSM

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningTimeScale(px\_double timescale,px\_double in[],px\_int count,px\_memory \*out); |
| 说明 | **调音台TSM算法** |
| 参数 | **timescale 压拓系数**  **in 输入信号(归一化建议)**  **count 输入信号长度**  **out 输出内存** |
| 返回值 |  |

###### 设置调音台的PitchShift

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningSetPitchShift(PX\_Tuning \*tuning,px\_double pitchShift); |
| 说明 | **设置调音台的PitchShift** |
| 参数 | **tuning 调音台指针**  PitchShift 移调系数 |
| 返回值 |  |

###### 设置调音台的滤波器

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningSetFilter(PX\_Tuning \*tuning,px\_double filter[]); |
| 说明 | **设置调音台的滤波器** |
| 参数 | **tuning 调音台指针**  filter滤波器系数 |
| 返回值 |  |

###### 设置调音台的频域补偿系数

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningSetFix(PX\_Tuning \*tuning,px\_double fix[]); |
| 说明 | **设置调音台的频域补偿系数** |
| 参数 | **tuning 调音台指针**  fix频域补偿系数 |
| 返回值 |  |

###### 设置调音台的ZCR阈值

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TuningSetZCR(px\_double low,px\_double high); |
| 说明 | **设置调音台的ZCR阈值,当输入信号的ZCR不在阈值范围内时,将不对信号进行变调处理** |
| 参数 | **tuning 调音台指针**  low 最低ZCR  high 最高ZCR |
| 返回值 |  |

### 反向传播神经网络

PainterEngine 提供一个基础的反向传播神经网络框架,集成了linear tanh sigmod reLU等激活函数,regularzation方式及权值初始化方式.

#### 初始化神经网络框架

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ANNInit(px\_memorypool \*mp,PX\_ANN \*ann,px\_double learningRate,PX\_ANN\_REGULARZATION regularzation,px\_double regularization\_rate); |
| 说明 | **初始化一个反向传播神经网络框架** |
| 参数 | **mp 运行内存池**  **ann 神经网络结构**  **learningRate 学习率**  regularization 正则化方式  参考:  typedef enum  {  PX\_ANN\_REGULARZATION\_NONE,  PX\_ANN\_REGULARZATION\_L1,  PX\_ANN\_REGULARZATION\_L2  }PX\_ANN\_REGULARZATION;  regularization\_rate 正则化速率 |
| 返回值 | 返回PX\_TRUE表示成功,否者返回PX\_FALSE |

#### 添加网络层

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ANNAddLayer(PX\_ANN \*pAnn,px\_int Neurals,px\_double bias,PX\_ANN\_ACTIVATION\_FUNCTION activation,PX\_ANN\_LAYER\_WEIGHT\_INITMODE mode,px\_double weight\_c); |
| 说明 | **添加一个网络层(其中第一层为输入层,最后一层为输出层)** |
| 参数 | **ann 神经网络结构**  Neurals 神经元数量  bias 偏置  activation 激活函数类型,参照  typedef enum  {  PX\_ANN\_ACTIVATION\_FUNCTION\_SIGMOID, //sigmoid  PX\_ANN\_ACTIVATION\_FUNCTION\_TANH,//tanh  PX\_ANN\_ACTIVATION\_FUNCTION\_LINEAR,//linear  PX\_ANN\_ACTIVATION\_FUNCTION\_RELU,//ReLU  }PX\_ANN\_ACTIVATION\_FUNCTION;  mode 权值初始方式,参照  typedef enum  {  PX\_ANN\_LAYER\_WEIGHT\_INITMODE\_CONST, //常量  PX\_ANN\_LAYER\_WEIGHT\_INITMODE\_RAND,//随机数  PX\_ANN\_LAYER\_WEIGHT\_INITMODE\_GAUSSRAND,//正态分布随机数  }PX\_ANN\_LAYER\_WEIGHT\_INITMODE;  weight\_c 当权值初始方式为常量时的常量值 |
| 返回值 | 返回PX\_TRUE表示成功,否者返回PX\_FALSE |

#### 训练网络

|  |  |
| --- | --- |
| 函数名 | px\_double PX\_ANNTrain(PX\_ANN \*pAnn,px\_double \*input,px\_double \*expect); |
| 说明 | **监督学习训练网络** |
| 参数 | **ann 神经网络结构**  input 输入数组(与输入层神经元对应)  expect 期望输出数组(与输出层神经元对应) |
| 返回值 | train loss值 |

#### 前向传播

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ANNForward(PX\_ANN \*pAnn,px\_double \*input); |
| 说明 | **神经网络前向传播** |
| 参数 | **ann 神经网络结构**  input 输入数组(与输入层神经元对应) |
| 返回值 |  |

#### 取得输出结果

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ANNGetOutput(PX\_ANN \*pAnn,px\_double \*result); |
| 说明 | **神经网络前向传播的输出结果** |
| 参数 | **ann 神经网络结构**  result 输出数组(与输入层神经元对应) |
| 返回值 |  |

#### 释放神经网络框架

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ANNFree(PX\_ANN \*pAnn); |
| 说明 | **释放框架** |
| 参数 | **ann 神经网络结构** |
| 返回值 |  |

#### 导出神经网络框架

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ANNExport(PX\_ANN \*pAnn,px\_void \*buffer,px\_int \*size); |
| 说明 | **将神经网络当前的训练框架导出到内存结构** |
| 参数 | **ann 神经网络结构.**  **buffer 导出buffer(当该值为PX\_NULL时表示只计算大小)**  **size 导出数据大小** |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 导入神经网络框架

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ANNImport(px\_memorypool \*mp,PX\_ANN \*pAnn,px\_void \*buffer,px\_int size); |
| 说明 | **从内存结构中导入神经网络框架** |
| 参数 | **mp 神经网络使用内存池**  **ann 未初始化的神经网络结构.**  **buffer 导入buffer**  **size 导入数据大小** |
| 返回值 | 成功返回PX\_TRUE,否者返回PX\_FALSE |

### 物理

#### 四叉树AABB碰撞检测

PainterEngine Core提供四叉树AABB碰撞检测算法,该算法用于加速矩形box区域的碰撞检测,使用自定义的树深调节对

##### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_QuadtreeCreate(px\_memorypool \*mp,PX\_Quadtree \*pQuadtree,px\_float mapStartX,px\_float mapStartY,px\_float mapWidth,px\_float mapHeight,px\_int ObjectsCount,px\_int deep); |
| 说明 | **创建四叉树根结构** |
| 参数 | mp 一次性内存池,该内存池不能够存在其他的节点,仅用于四叉树计算使用  pQuadtree, 四叉树结构体  mapStartX 需要分隔的域左上角起始坐标X  mapStartY 需要分隔的域左上角起始坐标Y  mapWidth,mapHeight 域的宽度和高度  ObjectsCount 需要计算的节点数  deep 深度,建议在2-4范围,启始深度为0,每个深度分4个区域 |
| 返回值 | 如果成功返回PX\_TRUE,否者为PX\_FALSE |

##### 添加节点

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_QuadtreeTestNode(PX\_Quadtree \*pQuadtree,px\_float x,px\_float y,px\_float width,px\_float height,PX\_Quadtree\_UserData userData) |
| 说明 | **在四叉树添加一个节点,该节点会不参与碰撞计算** |
| 参数 | pQuadtree 四叉树结构体  x,y,z,width,height,AABB描述  userData 碰撞测试的用户数据,这个数据将被附加到碰撞结果中,注意,相同的userData将被认为是重复数据 |

##### 碰撞测试

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_QuadtreeTestNode(PX\_Quadtree \*pQuadtree,px\_float x,px\_float y,px\_float width,px\_float height,PX\_Quadtree\_UserData userData) |
| 说明 | **在四叉树设置接参与碰撞计算的节点,这个节点不会被加入四叉树中** |
| 参数 | pQuadtree 四叉树结构体  x,y,z,width,height,length AABB box  userData该节点的标识号,该索引将出现在碰撞结果中,注意,如果节点的ptr相同,则该节点将标记为重复节点不计入碰撞检测中 |

### 密码学

#### AES

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AES\_Initialize(PX\_AES \*aesStruct,PX\_AES\_KeySize keySize,AES\_BYTE keyByte[]); |
| 说明 | **初始化AES编码器** |
| 参数 | aseStruct AES编码器  keySize 128,192,256加密  keyByte 对应位数长度密钥 |
|  |  |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AES\_Cipher(PX\_AES \*aesStruct,void \*input, void \*output); |
| 说明 | **编码16字节的数据** |
| 参数 | aseStruct AES编码器  input 输入  output 输出 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AES\_CipherBuffer(PX\_AES \*aesStruct,void \*input, px\_int size,void \*output); |
| 说明 | **编码一段的数据,这个数据的长度必须是16的整数倍** |
| 参数 | aseStruct AES编码器  input 输入  size 长度  output 输出 |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AES\_InvCipher(PX\_AES \*aesStruct,void \*input, void \*output); |
| 说明 | **解码16字节的数据** |
| 参数 | aseStruct AES编码器  input 输入  output 输出 |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AES\_InvCipherBuffer(PX\_AES \*aesStruct,void \*input, px\_int size,void \*output); |
| 说明 | **解码一段的数据,这个数据的长度必须是16的整数倍** |
| 参数 | aseStruct AES编码器  input 输入  size 长度  output 输出 |

#### curve25519

curve25519 是ECC加密衍生的Diffie-Hellman函数,用于生成一个公共密钥保证通讯安全

|  |  |
| --- | --- |
| 函数名 | px\_void curve25519\_donna(px\_byte \*mypublic, const px\_byte \*secret, const px\_byte \*basepoint); |
| 说明 | **解码16字节的数据** |
| 参数 | mypublic 32字节的公钥,输出  secret 32字节私钥,输入  basepoint 32字节长度的基点  私钥需要如下处理  mysecret[0] &= 248;  mysecret[31] &= 127;  mysecret[31] |= 64;  基点可以为  static const px\_byte basepoint[32] = {9}; |

#### SHA256

|  |  |
| --- | --- |
| 函数名 | PX\_Sha256Calculate (void\*Buffer,px\_uint32 BufferSize, PX\_SHA256\_HASH\* Digest ); |
| 说明 | **Sha256** |
| 参数 | buffer 输入缓存  buffersize 缓存大小  digest 输出hash值 |

### 编码及数据压缩

#### BASE64

##### 编码

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_Base64Encode(const px\_byte \*in, px\_uint inlen, px\_char \*out); |
| 说明 | **使用Base64编码一段数据** |
| 参数 | in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,不可为NULL,可以使用PX\_Base64GetEncodeLen预先计算编码后长度 |
| 返回值 | 编码长度 |

##### 解码

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_Base64Decode(const px\_char \*\_in, px\_uint input\_size, px\_byte \*out); |
| 说明 | **使用Base64解码一段数据** |
| 参数 | in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,不可为NULL,可以使用PX\_Base64GetDecodeLen预先计算解码后长度 |
| 返回值 | 解码长度,如果为0表示无法解码 |

##### 长度计算

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_Base64GetEncodeLen(px\_uint codeLen);  px\_uint PX\_Base64GetDecodeLen(px\_uint codeLen); |
| 说明 | **计算BASE64的编码/解码长度** |
| 参数 | Codelen 输入数据长度 |
| 返回值 | 编码/解码后的长度,如果为0表示无法编/解码 |

#### ARLE

ARLE(AdvanceRun Length Encoding)改进行程长度压缩算法,主要适用于图像及音频数据的压缩,最坏情况下,长度为N\*4/3,最优秀情况下,为长度的N\*2/127

##### 压缩

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ArleCompress(px\_byte \*\_in,px\_uint input\_size,px\_byte \*\_out,px\_uint \*\_outsize); |
| 说明 | **使用ARLE压缩一段数据** |
| 参数 | \_in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,可以为NULL表示仅计算输出长度  \_outsize 输出长度 |

##### 解压缩

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ArleCompress(px\_byte \*\_in,px\_uint input\_size,px\_byte \*\_out,px\_uint \*\_outsize); |
| 说明 | **使用ARLE压缩一段数据** |
| 参数 | \_in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,可以为NULL表示仅计算输出长度  \_outsize 输出长度 |

#### Huffman

##### 压缩

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_HuffmanCompress(px\_byte \*\_in,px\_uint input\_size,px\_byte \*\_out,px\_uint \*\_outsize); |
| 说明 | **使用Huffman压缩一段数据** |
| 参数 | \_in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,可以为NULL表示仅计算输出长度  \_outsize 输出长度 |

##### 解压缩

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_Huffmandecompress(px\_byte \*\_in,px\_uint input\_size,px\_byte \*\_out,px\_uint \*\_outsize); |
| 说明 | **使用Huffman压缩一段数据** |
| 参数 | \_in 输入数据缓存  input\_size 输入数据长度  \_out 输出数据缓存,可以为NULL表示仅计算输出长度  \_outsize 输出长度 |

### 3D渲染管线

PainterEngine提供一个轻量级软件3D渲染管线,其中大部分参数都以默认方式部署(质量优先),可用于3D动画渲染,静态模型渲染,但因为渲染性能消耗较大,并不建议使用该3D渲染管线直接开发需要动态渲染的复杂游戏,3D渲染管线是作为一个模块存在于PainterEngine中的,PainterEngine并不仅作为3D渲染器,不建议将3D渲染模式作为PainterEngine中的主要使用模式中.

##### 世界矩阵

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_WorldInitialize(PX\_3D\_World \*world,px\_float x,px\_float y,px\_float z,px\_float rotX,px\_float rotY,px\_float rotZ,px\_float scale); |
| 说明 | **初始化一个世界坐标,在渲染中该世界矩阵将直接作用于渲染列表** |
| 参数 | world 世界矩阵  x,y,z 世界矩阵的平移坐标  rotX,rotY,rotZ 绕X,Y,Z轴的旋转角度  scale 缩放比例 |

##### 欧拉相机矩阵

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_CameraEulerInitialize(px\_memorypool \*mp,PX\_3D\_Camera \*camera,px\_point4D cameraPosition,px\_float rotX,px\_float rotY,px\_float rotZ,px\_float near,px\_float far,px\_float fov,px\_float viewPortWidth,px\_float viewPortHeight); |
| 说明 | **初始化一个欧拉相机矩阵,在渲染中该矩阵将直接作用于渲染列表** |
| 参数 | mp 内存池,用于分配Z缓存  camera 相机矩阵  camera 相机位置  rotX,rotY,rotZ 绕X,Y,Z轴的旋转角度  far 远裁剪面  near 近裁剪面,这个值不能小于1.0  fov 视锥角度  viewportWidth 窗口宽度(像素为单位)  viewPortHeight 窗口高度(像素为单位) |

##### UVN相机矩阵

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_CameraUVNInitialize(px\_memorypool \*mp,PX\_3D\_Camera \*camera,px\_point4D cameraPosition,px\_point4D cameraTarget,px\_float near,px\_float far,px\_float fov,px\_float viewPortWidth,px\_float viewPortHeight); |
| 说明 | **初始化一个UVN相机矩阵,在渲染中该矩阵将直接作用于渲染列表** |
| 参数 | mp 内存池,用于分配Z缓存  camera 相机矩阵  cameraposition 相机位置  target 相机指向位置  far 远裁剪面  near 近裁剪面,这个值不能小于1.0  fov 视锥角度  viewportWidth 窗口宽度(像素为单位)  viewPortHeight 窗口高度(像素为单位) |

##### 设置UVN相机位置

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_CameraSetPosition(PX\_3D\_Camera \*camera,px\_point4D cameraPosition,px\_point4D cameraTarget); |
| 说明 | **设置uvn相机位置** |
| 参数 | cameraposition 相机位置  target 相机指向位置 |

##### 释放相机矩阵

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_CameraFree(PX\_3D\_Camera \*camera) |
| 说明 | **释放相机矩阵** |
| 参数 | camera 相机矩阵 |

##### 渲染列表

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_3D\_RenderListInitialize(px\_memorypool \*mp,PX\_3D\_RenderList \*list,px\_dword PX\_3D\_PRESENTMODE,PX\_3D\_CULLMODE cullmode,px\_texture \*ptexture); |
| 说明 | **渲染列表是PainterEngine渲染管线的基本渲染单元**  **所有的面都必须加载到该渲染列表中进行渲染** |
| 参数 | mp 使用内存池  list 渲染列表  PX\_3D\_PRESENTMODE 显示方式  包括:  #define PX\_3D\_PRESENTMODE\_LINE 1 线段渲染  #define PX\_3D\_PRESENTMODE\_TEXTURE 2 纹理渲染  #define PX\_3D\_PRESENTMODE\_PURE 4 纯色渲染  cullmode 裁剪模式  PX\_3D\_CULLMODE\_NONE, 不裁剪  PX\_3D\_CULLMODE\_CW, 顺时针裁剪  PX\_3D\_CULLMODE\_CCW, 逆时针裁剪  ptexture 纹理 |

##### 渲染列表添加三角形面

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_3D\_RenderListPush(PX\_3D\_RenderList \*list,PX\_3D\_Face face); |
| 说明 | **将一个三角形面添加到渲染列表当中** |
| 参数 | list 渲染列表  face 三角形面  typedef struct  {  px\_point4D position;//顶点位置  px\_vector4D normal;//法线(由渲染管线计算)  px\_color clr;//颜色  px\_float u,v;//uv纹理映射坐标  }PX\_3D\_Vertex;  typedef struct  {  px\_dword state;//由渲染管线控制  PX\_3D\_Vertex vertex[3];//顶点  PX\_3D\_Vertex transform\_vertex[3];//变换后顶点,由渲染管线控制  }PX\_3D\_Face;  每次使用PX\_3D\_Face建议对其内存清0 |

##### 渲染列表矩阵变换

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_RenderListTransform(PX\_3D\_RenderList \*list,px\_matrix mat,PX\_3D\_RENDERLIST\_TRANSFORM t); |
| 说明 | **将渲染列表中每一个顶点都使用mat矩阵变换** |
| 参数 | list 渲染列表  mat 变换矩阵  PX\_3D\_RENDERLIST\_TRANSFORM  PX\_3D\_RENDERLIST\_TRANSFORM\_LOCAL\_TO\_LOCAL,变换原始顶点并替换原始顶点  PX\_3D\_RENDERLIST\_TRANSFORM\_LOACL\_TO\_GLOBAL,变换原始顶点,并将结果放置在变换顶点中  PX\_3D\_RENDERLIST\_TRANSFORM\_GLOBAL\_TO\_GLOBAL,变换变换顶点,并将结果放置在变换顶点中 |

##### 渲染列表清空

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_RenderListReset(PX\_3D\_RenderList \*list); |
| 说明 | **清空渲染列表,注意:该函数并不清理内存,该列表仍然可用** |
| 参数 | list 渲染列表 |

##### 释放列表清空

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_RenderListFree(PX\_3D\_RenderList \*list); |
| 说明 | **释放渲染列表,并清理内存,该列表将不可用** |
| 参数 | list 渲染列表 |

##### 开始渲染

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_Scene(PX\_3D\_RenderList \*list,PX\_3D\_World \*world,PX\_3D\_Camera \*camera) |
| 说明 | **开始渲染(在Present之前调用)** |
| 参数 | list 渲染列表  word 世界矩阵  camera 相机矩阵 |

##### 渲染列表

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_3D\_Present(px\_surface \*psurface, PX\_3D\_RenderList \*list); |
| 说明 | **渲染列表到surface** |
| 参数 | psurface 渲染表面  list 渲染列表  world 世界矩阵  camera 相机矩阵 |

## Kernel

### 词法分析机

PainterEngine Kernel 提供通用词法分析机,用于对常用文本类型的脚本,数据格式语言的词法分析,在使用词法分析机之前,都应该使用PX\_LexerInit对词法分析机进行初始化操作,在词法分析机使用完毕后,应该使用PX\_LexerFree释放其占用资源.

#### 初始化词法分析机

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerInit(px\_lexer \*lexer,px\_memorypool \*mp); |
| 说明 | **初始化词法分析机** |
| 参数 | lexer px\_lexer结构指针  mp 词法分析内存池 |

#### 注册注释块

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerRegisterComment(px\_lexer \*lexer,px\_char Begin[],px\_char End[]); |
| 说明 | **注册注释块** |
| 参数 | lexer px\_lexer结构指针  begin 注释块的起始字符串  end 注释块的结束字符串 |
| 备注 | 如果注释块的结束字符串是回车,则该注释块在解析后为一个回车符 |

#### 注册包含块

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_LexerRegisterContainer(px\_lexer \*lexer,px\_char Begin[],px\_char End[]); |
| 说明 | **注册注释块** |
| 参数 | lexer px\_lexer结构指针  begin 包含块的起始字符串  end 包含块的结束字符串 |
| 返回值 | 该包含块的索引ID |

#### 注册空白符

多个连续的空白符在解析过程中将被解析为单个空白符

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerRegisterSpacer(px\_lexer \*lexer,px\_char Spacer); |
| 说明 | **注册空白符** |
| 参数 | lexer px\_lexer结构指针  Spacer 空白符 |

#### 注册分隔符

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_LexerRegisterDelimiter(px\_lexer \*lexer,px\_char Delimiter); |
| 说明 | **注册分隔符** |
| 参数 | lexer px\_lexer结构指针  Delimiter分隔符 |
| 返回值 | 该分隔符的ID |

#### 取得分隔符ID

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_LexerGetDelimiterType(px\_lexer \*lexer,px\_char Delimiter); |
| 说明 | **取得分隔符ID** |
| 参数 | lexer px\_lexer结构指针  Delimiter分隔符 |
| 返回值 | 该分隔符的注册ID |

#### 取得字符串包含块类型

|  |  |
| --- | --- |
| 函数名 | px\_uint PX\_LexerGetContainerType(px\_lexer \*lexer,px\_char \*pContainerText); |
| 说明 | **取得字符串包含块类型** |
| 参数 | lexer px\_lexer结构指针  pContainerText解释文本 |
| 返回值 | 该包含块的注册ID |

#### 释放词法分析机资源

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerFree(px\_lexer \*lexer); |
| 说明 | **释放词法分析机资源** |
| 参数 | lexer px\_lexer结构指针 |

#### 加载分析文本

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_LexerSortText(px\_lexer \*lexer,px\_char \*SourceText);  px\_bool PX\_LexerLoadSourceFromMemory(px\_lexer \*lexer,px\_byte \*buffer); |
| 说明 | **释放词法分析机资源** |
| 参数 | lexer px\_lexer结构指针  SourceText Buffer 分析文本 |
| 返回值 | 如果成功解析返回PX\_TRUE,否则返回PX\_FALSE(词法错误) |

#### 读取文本

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_LexerReadString(px\_lexer \*lexer,px\_string \*str,px\_uint size); |
| 说明 | **在当前词法分析状态下读取文本,该文本不置入词法分析** |
| 参数 | lexer px\_lexer结构指针  str 字符串指针  size 大小 |
| 返回值 | 如果成功解析返回PX\_TRUE,否则返回PX\_FALSE(词法错误) |

#### 取得特殊符号

|  |  |
| --- | --- |
| 函数名 | px\_char PX\_LexerGetSymbol(px\_lexer \*lexer); |
| 说明 | **取得当前分析的符号(分隔符,空白符或回车符)** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 当前分析的符号 |

#### 取得当前词

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerGetLexemeString(px\_lexer \*lexer,px\_string \*str); |
| 说明 | **取得当前词文本** |
| 参数 | lexer px\_lexer结构指针  str px\_string指针 |
| 返回值 |  |

可以通过访问px\_lexer-> CurLexeme 直接取得词字符串

#### 取得包含文本

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerGetIncludedString(px\_lexer \*lexer,px\_string \*str); |
| 说明 | **如果当前词为包含类型,则取得其包含文本** |
| 参数 | lexer px\_lexer结构指针  str px\_string指针 |
| 返回值 |  |

#### 设置词大小写敏感

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerSetTokenCase(px\_lexer \*lexer,PX\_LEXER\_LEXEME\_CASE \_case); |
| 说明 | **设置返回词大小写,为PX\_LEXER\_LEXEME\_CASE枚举类型,如果被设置,读取的词除包含类型外都会转换为对应大小写** |
| 参数 | lexer px\_lexer结构指针  \_case大小写类型 |
| 返回值 |  |

#### 取得词法分析机当前状态

|  |  |
| --- | --- |
| 函数名 | PX\_LEXER\_STATE PX\_LexerGetState(px\_lexer \*lexer); |
| 说明 | **取得当前词法分析机状态,可以在之后重设该状态** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 词法分析机当前状态 |

#### 设置词法分析机当前状态

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_LexerRestoreState(PX\_LEXER\_STATE state); |
| 说明 | **重新设置词法分析机状态** |
| 参数 | state 词法分析机状态结构体 |
| 返回值 | - |

#### 判断当前词是否是合法数字

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_LexerIsLememeIsNumeric(px\_lexer \*lexer); |
| 说明 | **判断当前词是否是合法数字** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 如果是返回PX\_TRUE如果不是返回PX\_FALSE |

#### 读取下一个字母

|  |  |
| --- | --- |
| 函数名 | px\_char PX\_LexerGetNextChar(px\_lexer \*lexer); |
| 说明 | **直接读取下一个字母** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 下一个字母 |

#### 读取下一个词

|  |  |
| --- | --- |
| 函数名 | PX\_LEXER\_LEXEME\_TYPE PX\_LexerGetNextLexeme(px\_lexer \*lexer); |
| 说明 | **直接读取下一个词** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 该词的词性,参考PX\_LEXER\_LEXEME\_TYPE  PX\_LEXER\_LEXEME\_TYPE\_END =0, 结束  PX\_LEXER\_LEXEME\_TYPE\_SPACER =1,空白符  PX\_LEXER\_LEXEME\_TYPE\_DELIMITER =2,分割符  PX\_LEXER\_LEXEME\_TYPE\_CONATINER =3,包含块  PX\_LEXER\_LEXEME\_TYPE\_NEWLINE =4,换行符  PX\_LEXER\_LEXEME\_TYPE\_TOKEN =5,词块  PX\_LEXER\_LEXEME\_TYPE\_ERR = -1,错误的词性 |

#### 取得当前词性

|  |  |
| --- | --- |
| 函数名 | PX\_LEXER\_LEXEME\_TYPE PX\_LexerGetCurrentLexeme(px\_lexer \*lexer); |
| 说明 | **取得当前词词性** |
| 参数 | lexer px\_lexer结构指针 |
| 返回值 | 该词的词性,参考PX\_LEXER\_LEXEME\_TYPE  PX\_LEXER\_LEXEME\_TYPE\_END =0, 结束  PX\_LEXER\_LEXEME\_TYPE\_SPACER =1,空白符  PX\_LEXER\_LEXEME\_TYPE\_DELIMITER =2,分割符  PX\_LEXER\_LEXEME\_TYPE\_CONATINER =3,包含块  PX\_LEXER\_LEXEME\_TYPE\_NEWLINE =4,换行符  PX\_LEXER\_LEXEME\_TYPE\_TOKEN =5,词块  PX\_LEXER\_LEXEME\_TYPE\_ERR = -1,错误的词性 |

### StoryScript编译器

#### 脚本语法

Story Script是PainterEngine所支持下的编译型脚本,是一个文件系统无关的支持脚本.

Story Script以下简称为SS,其具有如下的语法特性.

Story Script是大小写无关的语言.

Story Script是文件系统无关的语言,所有SS脚本都不以文件系统作为标识,所有的SS源文件必须以一个#name “id”作为标识

其中id可以为任意合法字符,在一个Story Script项目中,这个id必须是唯一的

Story Script是一个弱类型语言,类型有 整数 浮点数 字符串类型 内存类型

Story Script使用双引号表示一个字符串类型

Story Script使用单引号表示一个字符类型

SS 使用大括号表示一个内存类型

SS 以分号作为语句结束

以下以BNF文法来描述Story Script语法结构

##### 语法规则(BNF文法)

定义

Token::=符合命名规范的词或数字;

vName::= 符合命名规范的词

Numeric::=一个合法的数字,包括以0x开头的16进制数(最大不超过32位)

##### 预处理

###### #define name Token

#define用于将一个标识符替换为另一个标识符

所有name 类型都会被替换为Token

<预定义>::=”#define” <Token> <Token>

例如

#define PI 3.14159265

表示之后的PI都会被替换为3.14159265

###### #include “name”

#include用于将一个以name命名的源文件替换到当前预定义处

#include\_STATEMENT::= “#include” <”name”>

###### #runtime thread count

#runtime thread 用于定义最大支持线程数量

###### #runtime stack count

#runtime stack 用于定义默认的栈大小

##### 注释

StoryScript的注释方式与C语言一致

一种为双斜杠,代表本行注释,例如,下面是一个合法的注释

//note

同时,StoryScript支持/\*\*/类型注释.例如下文也是一个合法的注释

/\*note

note

\*/

##### 常量

StoryScript有以下几种常量

###### 数字

例如123,456及3.14159都是合法的数字常量,同时,StoryScript支持十六进制数常量

例如0x01,0xffffffff,其大小不应该超过32位数的定义

最后,字符ascii的表达方式也被解释为数字,例如

‘a’表示a的ascii码,其也作为一种数字常量

###### 字符串

用引号来包含一个字符串运算,例如

“hello world”

“Story Script”

都是一个合法的字符串常量

###### 数据流

使用一对@来定义一个数据流常量类型

例如

@FF0102030405060708090a0b0c0f@

数据流中,每字节数据都由成对的十六进制符定义完成,这代表序列中的数据必须被严格的配对定义

##### 变量类型

StoryScript支持以下几种类型

###### int

在StoryScript中,int类型被定义为一个32位的有符号整数.

###### float

在StoryScript中,float类型被定义为一个32位的有符号浮点数,该类型需要在符合IEEE754标准的编译器中被定义,否者,它将使用Q0.15的定点数标准进行实现..

###### string

字符串类型

###### memory

数据流类型

###### set

用户自定义集合类型

**在StoryScript中,所有变量必须预声明后使用,一个变量名在其有效域范围内仅能够被定义一次**

其BNF描述为

VARIABLE\_TYPE::=int|float|string|memory

VARIABLE\_DEFINE\_STATEMENT::=< VARIABLE\_TYPE > {<vName>[,]}

##### 变量定义

###### 常规变量定义

如下皆为合法常规变量定义

定义一个整形

int I;

连续定义多个整形

int a,b,c;

定义集合类型和集合数组

set a,b[10];

###### 指针类型

storyScript支持一级指针,所有的变量类型皆可定义为一个一级指针类型

POINTER\_STATEMENT::=<VARIABLE\_TYPE> \* <vName>;

eg:如下代码定义了一个int类型指针

int \*p;

###### 数组

storyScript支持一维数组,其BNF文法描述为

ARRAY\_STATEMENT::= <VARIABLE\_TYPE> <vName>”[“<Numeric>”]”;

eg:使用如下方式定义一个int类型数组

int a[10];

定义一个一级整形指针数组,和一个整形数组

int \*a[10],b[100];

###### 集合

StoryScript是参照C语言设计的一门脚本语言,在StoryScript中,使用数学描述Set(集合)来替代C语言的struct,同时,StoryScript吸取了C++中结构体的定义方式,定义一个集合文法如下

SET\_STATEMENT::= set <vName> “{” <….> ”}”

例如,下面是一个合法的定义例子

set MySet

{

int a,b,c;

float d;

string e[2];

memory f;

}

结构体运行相互嵌套,例如,下面的结构体定义也是合法的

set MySet2

{

MySet a,b[5];

int c;

}

##### 表达式

在StoryScript中,脚本由逻辑结构及表达式组成:

###### 运算符及优先级表

StoryScript参照了C语言的运算符进行设计,支持以下运算符

其中有所区别的是,sizeof被解析为返回内存类型或字符串类型的长度

|  |  |  |
| --- | --- | --- |
| 运算符 | 解释 | 结合方式 |
| () [] -> . | 括号（函数等），数组，两种结构成员访问 | 由左向右 |
| ! ~ ++ -- + -  \* & (类型) | 否定，按位否定，增量，减量，正负号，  间接，取地址，类型转换 | 由右向左 |
| \* / % | 乘，除，取模 | 由左向右 |
| + - | 加，减 | 由左向右 |
| << >> | 左移，右移 | 由左向右 |
| < <= >= > | 小于，小于等于，大于等于，大于 | 由左向右 |
| == != | 等于，不等于 | 由左向右 |
| & | 按位与 | 由左向右 |
| ^ | 按位异或 | 由左向右 |
| | | 按位或 | 由左向右 |
| && | 逻辑与 | 由左向右 |
| || | 逻辑或 | 由左向右 |
| = | 赋值运算 | 由右向左 |
| , | 逗号（顺序） | 由左向右 |

其运算符优先级同样参照C语言运算符的优先级表进行设计

1. ()  []  .  ->
2. !  ~   -（负号） ++  --   & \*

3   \* / %

4   + -

5   >> <<

6   > >= < <=

7   == !=

8   &

9   ^

10  |

11  &&

12  ||

13  ?

14   =

15  ,

###### 类型访问

int,float 类型访问

int及float变量类型可以直接对其助记符进行访问

例如

int a;

a=123;

a=a+3;

a++;

都是合法的访问

int 及 float的相互赋值将导致隐式转换,例如

int a;

a=3.14;

那么a的值将转换为3

a=5;

a=a/2;a的值为2;

string类型访问

string类型仅允许string类型及字符串常量间的相互操作

例如下面都是合法的语句

string foo;

foo=”abc”;

string类型允许加法运算

例如

string foo;

foo=”Hello”;

foo=foo+”World”;

foo=foo+foo;

//foo最终为HelloWorldHelloWorld;

同时,string类型允许访问其元素,例如

string foo=”HelloWolrd”;

int a;

a=foo[0];

//a的值为H的ase码

foo[0]=’M’;

//foo变换为MelloWorld;

memory类型访问

memory类型仅允许memory类型及字符串常量间的相互操作

例如下面都是合法的语句

memoryfoo;

foo=@010203@

memory类型允许加法运算,但加法仅接受一个范围为0x00-0xff的int类型,如果大于0xff的值将被直接截断

例如

memory foo;

foo=foo+0xff;

foo=foo+foo;

//foo最终为0xffff;

同时, memory类型允许访问其元素,例如

memoryfoo=@0102@;

int a;

a=foo[0];

//a的值为1

foo[0]=0xff;

//foo变换为ff 02;

set类型访问

直接访问set类型元素使用符合.,访问指针类型set使用->运算符

例如定义下列集合

set mySet

{

int a,b;

}

可以进行如下访问

mySet theSet;

theSet.a=99;

mySet \*pSet=&thenSet;

pSet->b=99;

###### 关键字类型

1. int(<Expression>) 转换为int类型,参数可以是一个整形,浮点,字符串类型
2. float(<Expression>)转换为float类型, 参数可以是一个整形,浮点,字符串类型
3. string(<Expression>)转换为string类型, 参数可以是一个整形,浮点,字符串,或内存类型
4. memory(<Expression>)转换为memory类型,参数可以是一个字符串或内存类型;
5. strlen(<Expression>) 内建函数,返回string的长度
6. memlen(<Expression>) 内建函数,返回memory的长度
7. sin(<Expression>) 内建函数,sin三角函数
8. cos(<Expression>) 内建函数,cos三角函数
9. break 跳出当前的循环结构,可以是一个while for compare结构
10. return <Expression> 函数返回值

###### 函数调用

StoryScript的函数调用方式为

FUNCTIONCALL\_STATEMENT::=<Function>(<…..>);

例如下面是一个合法的函数调用

add(1,2);

##### 语法结构

###### if条件判断语句

if语句的设计与C语言保持一致

IF\_STATEMENT::=if(<Expression>)

{<….>}

else

{

<…..>

}

或者是

if(<Expression>)

<Expression>

else

<Expression>

当if判断块中的Expression计算结果不为0时,执行对于语句块,否者跳转到else中执行,例如,下面是一个合法的if语句

if(a==1)

{

print(“a is 1”);

}

else

print (“a is not 1”);

###### while循环语句

while语句的设计与C语言保持一致

WHILE\_STATEMENT::=while(<Expression>)

{<….>}

或者是

while(<Expression>)

<Expression>

当while判断块中的Expression计算结果不为0时,循环执行器对应语句块,例如下面是合法的while语句

while(1)

{

//do something

}

###### for循环语句

for语句的设计与C语言保持一致

FOR\_STATEMENT::=for(<init>;<condition>;<extend>)

{<….>}

或者是

for(<init>;<condition>;<extend>)

<Expression>

其中,init表达式会先执行一次,然后判断condition表达式是否为1,如果是,执行对于语句块,否者跳出for循环,最后执行extend,再次判断condition

###### compare语句

compare语句是StoryScript中独有的语法结构其语法格式为

compare(<VARIABLE>)

{

with <Expression1;Expression2;….>

{

<……>

}

}

即VARIABLE将会与with对于的表达式进行比较,如果满足with语句中任意表达式对等,则执行with语句块中的语句,compare语句设计用于简化if-elseif-else..结构,并设计为switch语句的替代语句.

下面的compare语句都是合法的

int a;

compare(a) with(1;2;3) print(“a is 1 or 2 or 3.”);

compare(a)

{

with(1)

{

print(“a is 1”);

}

with(2;1+2)

{

print(“a is 2 or 3”);

}

print(“compare done”);

}

##### 函数定义

StoryScript的函数定义与C语言基本一致

FUNCTION\_STATEMENT::=<VARIABLE\_TYPE> <FunctionName>(<….>)

在StoryScript中,函数必须被前置声明后使用,同时在StoryScript中,函数返回值不允许使用Set类型,其参数也不允许使用set类型,如果需要使用set类型引用,必须使用set指针进行传递,使用set指针作为返回值是合法的但需要注意局部set在函数执行结束后其占用的资源将被释放,这将应用该set将导致访问不可预知的数据.

storyScript中,函数分三种类型(内定函数,导入函数(远程函数),导出函数),并使用两个关键字进行修饰,如果不使用修饰关键字,该函数将被当做是一个内定函数，作为一个内定函数，其在StoryScript中必须被实现

例如,下面是一个合法的内定函数定义

int MyFunction(int a,float b,string c,MySet \*pSet);

内定函数仅允许StoryScript脚本内部进行调用,虚拟机无法找到内定函数,如果你希望虚拟机可以调用该函数,它必须被声明为导出函数;

导入函数(远程函数)即以host进行修饰的函数,例如,下面是一个导入函数的声明

host void print(string msg);

导入函数即虚拟机中实现的函数,所有的导入函数都不能在StoryScript编写实现,例如

host void print(string msg)

{

//do something

}

是不合法的,导入函数的名称必须与虚拟机中注册的名称保持一致,如果调用一个不存在的导入函数,将会导致虚拟机报错.

导出函数即以export进行修饰的函数,导入函数可以在StoryScript中被内部调用或者被虚拟机中调用,例如,下面是一个导入函数的声明

export void main(string msg);

如果你希望虚拟机可以调用该函数,那么它必须被声明为导出函数,否者它无法被虚拟机所发现调用。作为一个导出函数，其在StoryScript中必须被实现。

下面是一个函数的示范代码

int add(int a,int b)

{

return a+b;

}

export int ss\_main()

{

return add(1,2);

}

##### 内联汇编

StoryVM允许使用内联汇编,格式为

\_asm

{

//内联的汇编语句

}

在内联汇编中,允许对变量进行访问,例如

int I;

\_asm

{

Mov I,1;

}

是被允许的

### SctoryAsm汇编器

#### 标志

汇编引擎以如下标号作为跳转标志

Name:

其中,Name为合法变量助记符

可以对其前置FUNCTION关键字

FUNCTION Name:

其效果和Name:等价

如果需要对标志进行导出以方便其它语言进行调用,必须前缀EXPORT关键字

语法规则为

EXPORT FUNCTION Name:

#### 汇编指令集

##### mov

赋值指令,将op2的值赋值给op1

mov [reg,local,global],[num,string,reg,local,global]

·

##### add

加法指令,op1=op1+op2

add [reg,local,global],[num,reg,local,global]

·

##### sub

减法指令,op1=op1-op2

sub [reg,local,global],[num,reg,local,global]

·

##### neg

符号求反指令,op1=-op1

neg [reg,local,global]

##### div

除法指令,op1=op1/op2

div [reg,local,global],[num,reg,local,global]

·

##### mul

乘法指令,op1=op1\*op2

mul [reg,local,global],[num,reg,local,global]

·

##### mod

余数指令,两个操作数必须为整数 op1=op1%op2

mod [reg,local,global],[int,reg,local,global]

##### shl

左移位指令,两个操作数必须为整数 op1=op1<<op2

shl [reg,local,global],[int,reg,local,global]

##### shr

右移位指令,两个操作数必须为整数 op1=op1>>op2

shr [reg,local,global],[int,reg,local,global]

##### and

与运算指令,op1=op1&op2

and [reg,local,global],[num,reg,local,global]

##### or

或运算指令,op1=op1|op2

or [reg,local,global],[num,reg,local,global]

·

##### xor

异或运算指令op1=op1^op2

xor [reg,local,global],[num,reg,local,global]

·

##### inv

取反指令,op1=~op1

inv [reg,local,global]

##### not

逻辑非指令 op1=!op1

not [reg,local,global]

##### andl

逻辑与指令 op1=op1&&op2

andl [reg,local,global],[num,reg,local,global]

##### orl

逻辑或指令 op1=op1||op2

andl [reg,local,global],[num,reg,local,global]

##### pow

阶乘指令(op1为底数,op2为指数,结果在op1中) op1=op1\_op2

pow [reg,local,global],[num,reg,local,global]

##### sin

正弦函数op1=sin(op2)

sin [reg,local,global],[num,reg,local,global]

##### cos

余弦函数 op1=cos(op2)

cos [reg,local,global],[num,reg,local,global]

##### int

强制类型转换为int型(原类型float)

int [reg,local,global]

##### flt

强制类型转换为float型

flt [reg,local,global]

##### strlen

字符型长度指令

op1=strlen(op2)

strlen [reg,local,global],[reg,local,global,string]

##### strcat

字符型拼接指令

strcat(op1,op2)

strcat [reg,local,global],[int,reg,local,global,string]

##### strrep

字符串替换函数

将op1存在的op2字符串替换为op3中的字符串, 注意:op2 op3必须为字符串类型

strrep [reg,local,global],[reg,local,global,string],[reg,local,global,string]

##### strchr

将op2在索引op3中的字存储在op1中, 注意:op2必须为字符串类型

strchr [reg,local,global],[reg,local,global,string],[reg,local,global,int]

##### strtoi

将op2转换为整数保存在op1中,注意:op2必须为字符串类型

strtoi [reg,local,global],[reg,local,global,string]

##### strtof

将op2转换为浮点数保存在op1中,注意:op2必须为字符串类型

strtof [reg,local,global],[reg,local,global,string]

##### strfri

将op2整数类型转换为字符串类型保存在op1中

strfri [reg,local,global],[reg,local,global,int]

##### strfrf

将op2浮点类型转换为字符串类型保存在op1中

strfrf [reg,local,global],[reg,local,global,float]

##### strset

将op1所在字符串索引为op2 int的字符置换为op3

如果op3为一个int,则取asc码(第八位1字节),如果op3为一个字符串,则取第一个字母

strset [reg,local,global],[reg,local,global,int],[reg,local,global,string,int]

##### strfind

在op2字符串中查找op3子字符串,如果找到了op1等于其开始索引,否者等于-1

如果op3为一个int,则取asc码(第八位1字节),如果op3为一个字符串,则取第一个字母

strset [reg,local,global],[reg,local,global,string],[reg,local,global,string]

##### strtmem

将op1字符串类型转换为内存类型

strfrf [reg,local,global]

##### asc

将op2的第一个字母以asc码的形式

asc [reg,local,global],[reg,local,global,string]

##### membyte

将op3 内存类型对应op2索引复制到op1中,这个类型是一个int类型(小于256)

membyte [reg,local,global],[reg,local,global,int],[reg,local,global,memory]

##### memset

设置op1对应op2索引的内存为op3

memset [reg,local,global],[reg,local,global,int],[reg,local,global ,int]

##### memtrm

将op1内存进行裁剪,其中,op2为开始位置,op2为大小

memcpy [reg,local,global],[reg,local,global,int],[reg,local,global,memory]

##### memfind

查找op2对应于op3内存所在的索引位置,返回结果存储在op1中,如果没有找到,op1将会置为-1

memfind [reg,local,global],[reg,local,global,memory],[reg,local,global,memory]

##### memlen

将op2的内存长度存储在op1中

memlen [reg,local,global],[reg,local,global,memory]

##### memcat

将op2的内存拼接到op1的尾部

memcat [reg,local,global],[int,reg,local,global,memory]

##### memtstr

将op1内存类型转换为字符串类型,如果op1的内存结尾不为0,将会被强制置为0

memtstr [reg,local,global]

##### datacpy

复制虚拟机data数据,从地址op2到地址op1,长度为op3

datacpy [reg,local,global,int], [reg,local,global,int], [reg,local,global,int]

##### jmp

跳转指令 跳转到op1地址

jmp [reg,num,local,global,label]

##### je

条件跳转,当op1等于op2,跳转到op3

je [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### jne

条件跳转,当op1不等于op2,跳转到op3

jne [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### jl

条件跳转,当op1小于op2,跳转到op3

jl [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### jle

条件跳转,当op1小于等于op2,跳转到op3

jle [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### jg

条件跳转,当op1大于op2,跳转到op3

jg [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### jge

条件跳转,当op1大于等于op2,跳转到op3

jge [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

##### lge

逻辑比较指令,当op2等于op3时将op1置1,否则为0

lge [reg,local,global], [num,string,reg,local,global] , [num,string,reg,local,global]

##### lgne

逻辑比较指令,当op2等于op3时将op1置0,否则为1

lge [reg,local,global], [num,string,reg,local,global] , [num,string,reg,local,global]

##### lgz

逻辑比较指令,当op1等于0时将op1置1,否则为0

lgz [reg,local,global]

##### lggz

逻辑比较指令,当op1大于0时将op1置1,否则为0

lggz [reg,local,global]

##### lggez

逻辑比较指令,当op1大于等于0时将op1置1,否则为0

lggez [reg,local,global]

##### lglz

逻辑比较指令,当op1小于0时将op1置1,否则为0

lglz [reg,local,global]

##### lglez

逻辑比较指令,当op1小于等于0时将op1置1,否则为0

lglez [reg,local,global]

##### call

调用指令,如果op1是本地地址则将当期下一条指令地址压栈,然后跳转到op1,如果op1是一个host地址,则该call为一个hostcall,hostcall不会将返回地址压栈

call [reg,int,local,global,label,host]

\*Host Call的返回值在r[0]中

\*由被调用者清理堆栈

##### push

将op1压栈 sp-1,stack[0]=op1

push [num,reg,local,global,string,label]

##### pop

出栈，并将该值

pop [reg,local,global]

##### adr

取堆栈的绝对地址,返回该堆栈的绝对地址

ADR [reg,local,global], [local,global]

##### popn

将op1个元素出栈

popn [reg,local,global]

##### ret

返回,pop一个返回地址,跳转到该地址.

##### wait

等待一个信号量置为0,否者这个虚拟机实例将被暂时挂起(但并不不影响suspend标准位),在每个虚拟机实例中都有16个信号量,通过signal指令对这些信号量进行设置

##### signal

等待op1对应索引的信号量置为op2, 在每个虚拟机实例中都有16个信号量,这意味着op1的范围是0-15,当一个信号量被设置为非0值时,执行wait指令后改虚拟机实例会被阻塞,直到这个信号量被置为0时才能继续执行后续指令

##### bpx

如果启动了调试器,将会在该指令上断点,否者作为一个空指令

##### nop

空指令

#### 表达式

在文件开头以

.stack m

.global n

指定堆栈元素大小

注:以下参数表达reg表示寄存器,local表示局部变量,global表示全局变量,num表示数值,label表示标签或函数,host表示引擎API,int表示整数,float表示浮点数,string表示字符串类型

其中local表示对当前堆栈操作local[0]表示栈顶,local[1]表示栈顶往后的第一个指针

global 表示对当前堆操作,按照global[n] 表示，n为索引量

op1表示参数1,op2表示参数2….以此类推

在Label中以$标志做前缀,表示这是一个hostcall

{}包含符表示为一内存类型例如{02AFEFCCDDEE},两个字符为一字节,必须2字节对齐

“”包含符表示为一字符串类型,支持\n换行\t对齐转义

‘’表示这个字符代表的asc码值

#### 汇编编译器

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptAsmCompile(px\_memorypool \*mp,px\_byte \*asmcode,px\_memory \*binmemory); |
| 说明 | **编译一个脚本汇编文件** |
| 参数 | mp 临时内存池  asmcode 以\0结束的汇编代码  binmemory 输出bin内存指针 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

### StoryVM虚拟机

#### vm实例

typedef struct \_\_PX\_ScriptVM\_Instance

{

px\_int VM\_memsize; //虚拟机使用元内存大小

px\_int stringsize; //字符串常量内存

px\_int memorysize; //二进制常量内存大小

px\_int binsize; //shellcode内存大小

px\_int funcCount; //导出函数数量

px\_int hostCount; //host函数数量

px\_bool Suspend; //是否被挂起

px\_int T; //当前执行线程索引

px\_byte \* \_bin; //shellcode指针

px\_byte \*\_memory; //二进制常量指针

px\_char \*\_string; //字符串常量指针

px\_memorypool \*mp; //归属内存池

PX\_SCRIPT\_EXPORT\_FUNCTION \*\_func;//导出函数表

PX\_SCRIPT\_ASM\_HOST\_NODE \*\_host;//hostcall表

PX\_SCRIPTVM\_VARIABLE \*\_mem;//元内存指针

px\_bool debug; //是否处于调试状态

px\_bool bp\_next;//下一条语句断点

px\_int bp\_IP;//断点指令地址

}PX\_ScriptVM\_Instance;

#### host函数定义

|  |  |
| --- | --- |
| 函数名 | typedef px\_bool (\*PX\_ScriptVM\_Function\_Modules)(PX\_ScriptVM\_Instance \*Ins); |
| 说明 | **StoryScript host函数声明标准** |
| 参数 | PX\_ScriptVM\_Instance 实例 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |
| 备注 | 通过  #define PX\_ScriptVM\_STACK(Ins,i) ((Ins)->\_mem[(Ins)->BP+i])  访问传入参数,其中,第一个参数下标为0,第二个为1以此类推  通过  #define PX\_ScriptVM\_RETURN(Ins) ((Ins)->R[1])  设定host函数返回类型 |

**实例初始化**

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceInit(PX\_ScriptVM\_Instance \*Ins,px\_memorypool \*mp,px\_byte \*code,px\_int size); |
| 说明 | **初始化一个虚拟机实例** |
| 参数 | PX\_ScriptVM\_Instance 实例  mp 内存池  code 虚拟机指令流  size 指令流大小 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 执行虚拟机函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceRunFunction(PX\_ScriptVM\_Instance \*Ins,px\_int threadID,px\_void \*runParam,px\_char \*func,px\_int paramcount,...); |
| 说明 | **通过函数名执行脚本函数,这个函数被调用后,对应函数会立即执行直到其结束或被信号量中断** |
| 参数 | PX\_ScriptVM\_Instance 实例  thread 由thread索引线程执行该函数  runparam 运行时参数  func 函数名  paramcount 参数个数  … 由特定函数初始化的函数类型 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceRunFunctionIndex(PX\_ScriptVM\_Instance \*Ins,px\_int threadID,px\_void \*runParam,px\_int funcIndex,px\_int paramcount,...); |
| 说明 | **通过函数索引执行脚本函数, 这个函数被调用后,对应函数会立即执行直到其结束或被信号量中断** |
| 参数 | PX\_ScriptVM\_Instance 实例  thread 由thread索引线程执行该函数  runparam 运行时参数  funcIndex 函数索引  paramcount 参数个数  … 由特定函数初始化的函数类型 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceBeginThreadFunction(PX\_ScriptVM\_Instance \*Ins,px\_int threadID,px\_void \*runParam,px\_char \*func,px\_int paramcount,...); |
| 说明 | **通过函数名创建线程函数,这个函数被调用后,对应函数并不会立即执行,需要调用PX\_ScriptVM\_InstanceRunThread来执行对应线程** |
| 参数 | PX\_ScriptVM\_Instance 实例  thread 由thread索引线程执行该函数  runparam 运行时参数  func 函数名  paramcount 参数个数  … 由特定函数初始化的函数类型 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceBeginThreadFunctionIndex(PX\_ScriptVM\_Instance \*Ins,px\_int threadID,px\_void \*runParam,px\_int \*funcIndex,px\_int paramcount,...); |
| 说明 | **通过函数索引创建线程函数,这个函数被调用后,对应函数并不会立即执行,需要调用PX\_ScriptVM\_InstanceRunThread来执行对应线程** |
| 参数 | PX\_ScriptVM\_Instance 实例  thread 由thread索引线程执行该函数  runparam 运行时参数  funcIndex 函数索引  paramcount 参数个数  … 由特定函数初始化的函数类型 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 运行虚拟机

|  |  |
| --- | --- |
| 函数名 | PX\_SCRIPTVM\_RUNRETURN PX\_ScriptVM\_InstanceRunThread(PX\_ScriptVM\_Instance \*Ins ,px\_int tick); |
| 说明 | **执行虚拟机实例直到当前线程其时间片耗尽** |
| 参数 | PX\_ScriptVM\_Instance 实例  tick 时间片,如果这个值小于0,该线程将一直执行到线程退出或信号量中断 |
| 返回值 | PX\_SCRIPTVM\_RUNRETURN有以下几种返回方式  PX\_SCRIPTVM\_RUNRETURN\_ERROR =0,虚拟机执行发生错误  PX\_SCRIPTVM\_RUNRETURN\_TIMEOUT,时间片耗尽返回  PX\_SCRIPTVM\_RUNRETURN\_END,线程结束  PX\_SCRIPTVM\_RUNRETURN\_SUSPEND,线程被挂起  PX\_SCRIPTVM\_RUNRETURN\_WAIT,信号量中断返回 |

#### 线程上下文切换

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceThreadSwitch(PX\_ScriptVM\_Instance \*Ins,px\_int T); |
| 说明 | **切换线程上下文** |
| 参数 | PX\_ScriptVM\_Instance 实例  T 线程ID |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 终止线程

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_ThreadStop(PX\_ScriptVM\_Instance \*Ins,px\_int ThreadId); |
| 说明 | **强制终止实例的一个线程,这个函数将在脚本函数最后ret时调用,它不会清除寄存器中的值.** |
| 参数 | PX\_ScriptVM\_Instance 实例  T 线程ID |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_ThreadClear(PX\_ScriptVM\_Instance \*Ins,px\_int ThreadId); |
| 说明 | **清理一个线程中的堆栈和寄存器,调用该函数必须在该线程没有执行时调用** |
| 参数 | PX\_ScriptVM\_Instance 实例  T 线程ID |
| 返回值 | - |

#### 挂起线程

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_ThreadSuspend (PX\_ScriptVM\_Instance \*Ins,px\_int ThreadId); |
| 说明 | **挂起实例的一个线程** |
| 参数 | PX\_ScriptVM\_Instance 实例  T 线程ID |
| 返回值 | - |

#### 恢复线程

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_ThreadResume (PX\_ScriptVM\_Instance \*Ins,px\_int ThreadId); |
| 说明 | **恢复被挂起实例的一个线程** |
| 参数 | PX\_ScriptVM\_Instance 实例  T 线程ID |
| 返回值 | - |

#### 注册host函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_RegistHostFunction(PX\_ScriptVM\_Instance \*Ins,px\_char \*name,PX\_ScriptVM\_Function\_Modules funcModules); |
| 说明 | **注册一个host函数** |
| 参数 | PX\_ScriptVM\_Instance 实例  name 注册函数名  funcModules 函数指针 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 释放虚拟机

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ScriptVM\_InstanceFree(PX\_ScriptVM\_Instance \*Ins); |
| 说明 | **释放一个虚拟机实例** |
| 参数 | PX\_ScriptVM\_Instance 实例 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 类型访问

#define PX\_ScriptVM\_STACK(Ins,i) 访问虚拟栈元

#define PX\_ScriptVM\_HOSTPARAM(Ins,i) 访问虚拟栈元,该宏用于Host访问hostcall函数调用参数,其中索引0表示第一个参数

#define PX\_ScriptVM\_LOCALPARAM(Ins,i) 访问虚拟栈元,该宏用于访问脚本内部函数调用参数,其中索引0表示第一个参数

#define PX\_ScriptVM\_GLOBAL(Ins,i) 访问虚拟堆元

#define PX\_ScriptVM\_RETURN(Ins) 访问寄存器R1

##### 弹出栈

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_POPN(PX\_ScriptVM\_Instance \*Ins,px\_int n); |
| 说明 | **弹出n字节栈元** |
| 参数 | PX\_ScriptVM\_Instance 实例  n 栈元数目 |
| 返回值 | - |

##### host返回值

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_RET(PX\_ScriptVM\_Instance \*Ins,PX\_SCRIPTVM\_VARIABLE ret); |
| 说明 | **设置R0寄存器(返回值寄存器),该函数主要用于hostcall中的返回值** |
| 参数 | PX\_ScriptVM\_Instance 实例  ret 值 |
| 返回值 | - |

##### 压栈

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ScriptVM\_PUSH(PX\_ScriptVM\_Instance \*Ins,PX\_SCRIPTVM\_VARIABLE val); |
| 说明 | **元数据压栈** |
| 参数 | PX\_ScriptVM\_Instance 实例  val 元数据 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_int(px\_int \_int);  PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_float(px\_float \_float);  PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_string(px\_string \_ref\_string);  PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_memory(px\_memory \_ref\_memory);  PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_const\_string(px\_char \*buffer);  PX\_SCRIPTVM\_VARIABLE PX\_ScriptVM\_Varibale\_const\_memory(px\_byte \*buffer,px\_int \_size); |
| 说明 | **构造对应类型元数据** |
| 参数 | ~ |
| 返回值 | 构造的元数据 |

#### 命令调试系统

PainterEngine StoryVM 提供一个基于命令的简单调试器,提供如下三个接口函数供使用

##### 1.启用调试器

px\_int PX\_ScriptVM\_DebuggerEnable(PX\_ScriptVM\_Instance \*Ins,PX\_ScriptVM\_DebuggerPrint \_printFunc,PX\_ScriptVM\_DebuggerCommand \_cmdFunc);

通过调用该函数来启用一个调试器,其中PX\_ScriptVM\_DebuggerPrint是一个指向printf函数的函数指针,用于输出调试信息

PX\_ScriptVM\_DebuggerCommand 指向一个类似于scanf的函数指针用于中断输入命令

##### 2.调试器命令

调试器支持以下命令

B IP断点,BP后接addr 例如 B 100

N 单步执行下一条指令

C 继续执行

U 反汇编代码,U后接接下来的反汇编指令数目 例如 U 1400

R 查看寄存器,例如R 1表示查看寄存器1的值

G 查看Global数据,例如G 1表示查看Global[1]的数据

L 查看local数据,例如L 1表示查看Local[1]数据,Local数据本质是Global[BP+oft]

### 2dx动画描述语言

2dx动画文件是PainterEngine专门为逐帧动画设计的动画文件,在有文件的文件系统中,其常常以.2dx作为后缀,同时2dx makechain必须运行在有文件系统的操作系统中

2dx动画文件是一个依据简单描述脚本经过编译而成的一个动画文件,下面是一个2dx脚本

texture "1.traw" f1

texture "2.traw" f2

texture "3.traw" f3

animation clock

loop -1

tag begin

frame f1

sleep 100

frame f2

sleep 90

frame f3

sleep 80

goto begin

end

#### 指令

2dx脚本支持以下指令

**Texture 纹理映射指令**,其负责将脚本存在目录下的纹理文件映射为一助记符,在脚本中就可以通过该助记符来使用该纹理,其格式为

Texture “文件路径” 助记符

**Frame 纹理显示指令**,将助记符映射的纹理显示到当前帧上,其格式为

Frame 助记符

**Sleep 延迟指令**,延迟一段时间,其单位是毫秒,其格式为

Sleep 毫秒时间数

**Tag 标记**,其不是一个指令,它将搭配Goto指令进行使用,其使用方式为

Tag 标记符

**Animation 标记**,其不是一个指令,它将搭配Goto指令进行使用,实际上他和tag是同一作用的不同写法,用法为

Animation 标记符

**Loop指令**,修改Loop寄存器的值,他配合Goto 指令进行使用,其用法为

Loop 值

**Goto条件跳转指令**,跳转到标号的指令执行,在执行前其会检查Loop寄存器,如果Loop寄存器的值不为0,则跳转,同时假如loop寄存器的值不为-1,其会将loop寄存器的值-1,格式为

Goto 标记符

**END 结束指令**,用于通知动画播放器动画已经播放结束,是一个特殊指令,用法为

END

#### 编译

要编译一起2dx动画文件,你需要一个2dx脚本及对应纹理,并且将该脚本与纹理放在同一个目录中(否者你需要使用相对路径重新定位其在文件系统中的具体位置)

PainterEngine\_2dxMakeTool是PainterEngine附带的2dx动画编译器工具配件,其可以在带文件系统的windows及Linux衍生版本中编译成可执行的二进制文件并用于2dx动画编译

其使用方法为

PainterEngine\_2dxMakeTool 脚本路径 编译输出路径

如果脚本正确它将会输出一个2dx动画文件,你可以使用painterEngine将它加载到引擎当中

(注意,因为该工具并不附带脚本错误提示器,所以你必须确保你的脚本编写正确,不过不用担心,2dx的脚本语言本身较为简单,并不会给你编写带来多大困扰).

### 2dx动画

#### 2dx library

需要创建一个2dx动画,首先要加载2dx动画数据,并通过该数据创建一个库文件

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AnimationLibraryCreateFromMemory(px\_memorypool \*mp,px\_animationlibrary \*panimationLib,px\_byte \*\_2dxBuffer,px\_uint size); |
| 说明 | **通过2dx数据创建一个动画库** |
| 参数 | mp内存池  panimationLib px\_animationlibrary 库指针  \_2dxBuffer 2dx数据  size 2dx数据的长度 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationLibraryFree(px\_animationlibrary \*panimationLib); |
| 说明 | **释放动画库文件** |
| 参数 | panimation |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 2dx animation

2dx animation动画是2dx library的实例化,每一个2dx animation都有自己独立的动画实例

每一个2dx animation都必须连接到一个2dx animation,并且通过update进行其动画的帧更新,在此之后,它就能渲染到对应的surface实现动画的播放了

##### 创建一个动画

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AnimationCreate(px\_animation \*animation,px\_animationlibrary \*linker); |
| 说明 | **通过2dx库数据创建一个动画** |
| 参数 | animation 动画对象  linker px\_animationlibrary 连接的库指针 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 释放2dx动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationFree(px\_animation \*animation); |
| 说明 | **释放2dx动画对象** |
| 参数 | animation 动画对象 |
| 返回值 | - |

##### 重置该2dx动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationReset(px\_animation \*animation); |
| 说明 | **重置该2dx动画对象** |
| 参数 | animation 动画对象 |
| 返回值 | - |

##### 更新该动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationUpdate(px\_animation \*panimation,px\_uint elpased); |
| 说明 | **更新该动画对象** |
| 参数 | animation 动画对象  elapsed 距离上一次更新经过的时间(毫秒级) |
| 返回值 | - |

##### 渲染动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationRender(px\_surface \*psurface,px\_animation \*animation, px\_point position,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend); |
| 说明 | **渲染动画对象** |
| 参数 | animation 动画对象  psurface 渲染表面  position 偏移量  refPoint 渲染参考点  blend blend属性,参考PX\_TEXTURERENDER\_BLEND定义 |
| 返回值 | - |

##### 以一个旋转角度渲染动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationRenderRotation(px\_surface \*psurface,px\_animation \*animation, px\_point position,px\_int angle,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend); |
| 说明 | **渲染动画对象** |
| 参数 | animation 动画对象  psurface 渲染表面  position 偏移量  angle 旋转角度  refPoint 渲染参考点  blend blend属性,参考PX\_TEXTURERENDER\_BLEND定义 |
| 返回值 | - |

##### 以一个比例和方向渲染动画对象

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationRender(px\_surface \*psurface,px\_animation \*animation, px\_point position,px\_float scale,px\_point direction,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend); |
| 说明 | **渲染动画对象** |
| 参数 | animation 动画对象  psurface 渲染表面  position 偏移量  scale 比例  direction 方向,默认方向为(1,0,0)即x方向  refPoint 渲染参考点  blend blend属性,参考PX\_TEXTURERENDER\_BLEND定义 |
| 返回值 | - |

##### 以一个向量方向渲染动画对象(初始方向是x方向)

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationRender\_vector(px\_surface \*psurface,px\_animation \*animation, px\_point position,px\_point directionPX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend); |
| 说明 | **以一个向量方向渲染动画对象(初始方向是x方向)** |
| 参数 | animation 动画对象  psurface 渲染表面  position 偏移量  direction 方向向量, 默认方向为(1,0,0)即x方向  refPoint 渲染参考点  blend blend属性,参考PX\_TEXTURERENDER\_BLEND定义 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_AnimationRenderEx(px\_surface \*psurface,px\_animation \*animation, px\_point position,PX\_TEXTURERENDER\_REFPOINT refPoint,PX\_TEXTURERENDER\_BLEND \*blend,px\_float scale,px\_float rotation); |
| 说明 | **渲染动画对象** |
| 参数 | animation 动画对象  psurface 渲染表面  position 偏移量  refPoint 渲染参考点  blend blend属性,参考PX\_TEXTURERENDER\_BLEND定义  scale 缩放比例  rotation 旋转角度 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_AnimationIsEnd(px\_animation \*panimation); |
| 说明 | **判断当前动画是否已经结束** |
| 参数 | panimation 动画对象 |
| 返回值 | - |

|  |  |
| --- | --- |
| 函数名 | px\_rect PX\_AnimationGetSize(px\_animation \*panimation) |
| 说明 | **取得动画的建议最大宽高** |
| 参数 | panimation 动画对象 |
| 返回值 | rect类型,其中x,y必定为0  width,height为建议宽度 |

### 粒子系统

#### 粒子系统脚本描述

PainterEngine的粒子系统是一个基于脚本运算及内嵌物理计算的粒子系统,其使用StoryScript作为其粒子的描述语言,下面是一个标准的粒子系统脚本

#name "main"

host float partical\_rand();//host函数,返回值0-1.0

set vector

{

float x,y,z; //三维点坐标或一个向量

}

set hdr\_color

{

int g,b,r,a;//hdr color

}

set Partical\_Init\_Info //粒子系统初始环境

{

int generateDuration; //粒子生成间隔

int maxCount;//最大的数量

vector force;//力

float resistanceK;//阻力系数

int launchCount;//该系统生成的粒子数量

}

set PARTICAL\_ATOM\_INFO//粒子描述集合

{

float size;//粒子大小,默认为1.0

float rotation;//旋转角度

float mass;//质量

float alpha;//alpha混合系数

int aliveTime;//粒子存活时间(毫秒)

int elpasedTime;//粒子生成到当前的时间(毫秒)

vector position;//粒子当前位置

vector velocity;//粒子当前速度

float hdrR;//hdr Red

float hdrG;//hdr Green

float hdrB;//hdr Blue

float rotationSpeed;//旋转速度(每秒)

float sizeIncrement;//尺寸增量(每秒)

float alphaIncrement;//alpha增量(每秒)

}

Partical\_Init\_Info g\_particalInitInfo;

PARTICAL\_ATOM\_INFO atomInfo;

export Partical\_Init\_Info \*Partical\_Init() //示范:每次初始化粒子环境会调用该函数

{

g\_particalInitInfo.generateDuration=25;

g\_particalInitInfo.maxCount=128;

g\_particalInitInfo.force.x=0;

g\_particalInitInfo.force.y=0;

g\_particalInitInfo.force.z=0;

g\_particalInitInfo.resistanceK=0;

g\_particalInitInfo.launchCount=-1;

return &g\_particalInitInfo;//填写该环境的参数然后返回

}

export PARTICAL\_ATOM\_INFO \*Partical\_Atom\_Create(int index)//示范:每次生成粒子会调用该函数,index为当前生成索引

{

atomInfo.size=0.5+partical\_rand()/2;

atomInfo.rotation=partical\_rand()\*360;

atomInfo.mass=1.0;

atomInfo.alpha=1;

atomInfo.aliveTime=3000;

atomInfo.elpasedTime=0;

atomInfo.position.x=0;

atomInfo.position.y=0;

atomInfo.position.z=0;

atomInfo.velocity.x=300\*partical\_rand()-150;

atomInfo.velocity.y=300\*partical\_rand()-150;

atomInfo.velocity.z=0;

atomInfo.hdrR=0.75+partical\_rand()/4;

atomInfo.hdrG=0.75+partical\_rand()/4;

atomInfo.hdrB=0.75+partical\_rand()/4;

atomInfo.rotationSpeed=360\*partical\_rand();

atomInfo.alphaIncrement=-0.45;

atomInfo.sizeIncrement=-0.35;

return &atomInfo;//填写该粒子属性后返回

}

export void Partical\_Atom\_Update(PARTICAL\_ATOM\_INFO \*atomInfo)//每次更新粒子时调用该函数,可以依据参数进行修改,该调用发生在系统物理计算之后,可以对粒子进行进一步调整,如果该函数不写,粒子将完全使用自带的物理运算

{

}

#### 创建粒子系统

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ParticalLauncherCreate(PX\_Partical\_Launcher \*launcher,px\_memorypool \*mp,px\_texture \*tex,PX\_ScriptVM\_Instance \*pIns, ,px\_char \*Initfunc); |
| 说明 | **设创建一个粒子环境** |
| 参数 | launcher 发射器  mp 内存池  tex 粒子纹理  pIns StoryVM实例  Initfunc 初始化环境的脚本函数名 |
| 返回值 | - |

#### 更新粒子系统

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ParticalUpdate(PX\_Partical\_Launcher \*launcher,px\_dword elpased); |
| 说明 | **更新粒子系统** |
| 参数 | launcher 发射器  elapsed 距离上次更新经过的时间(毫秒) |
| 返回值 | TRUE |

#### 渲染粒子系统

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ParticalRender(PX\_Partical\_Launcher \*launcher,px\_surface \*surface,px\_point position); |
| 说明 | **渲染粒子系统** |
| 参数 | launcher 发射器  surface 渲染表面  position 偏移量 |
| 返回值 |  |

#### 释放粒子系统

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ParticalFree(PX\_Partical\_Launcher \*launcher); |
| 说明 | **释放一个粒子发射器** |
| 参数 | launcher 发射器 |
| 返回值 |  |

#### 设置粒子系统喷射方向

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ParticalLauncherSetDirection(PX\_Partical\_Launcher \*launcher,px\_point direction) |
| 说明 | **设置粒子系统喷射方向** |
| 参数 | launcher 发射器  direction 方向,这个方向是以局部坐标为准的 |
| 返回值 |  |

#### 设置粒子创建脚本函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ParticalLauncherSetCreateFuncIndex(PX\_Partical\_Launcher \* launcher,px\_char \*func\_Name); |
| 说明 | **设置该发射机的粒子创建脚本函数** |
| 参数 | launcher 发射器  func\_name 脚本函数名 |
| 返回值 |  |

#### 设置粒子更新脚本函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ParticalLauncherSetUpdateFuncIndex(PX\_Partical\_Launcher launcher,px\_char \*func\_Name); |
| 说明 | **设置该发射机的粒子更新脚本函数** |
| 参数 | launcher 发射器  func\_name 脚本函数名 |
| 返回值 |  |

### 可靠传输协议

PainterEngine可靠传输协议,是一套独立于传输系统的传输协议,在用于传输大量数据时尤为有效,使用此传输协议需要定义读写接口,同时也支持对传输数据的额外编码解码(加密/解密)方案

传输函数声明如下

typedef px\_bool (\*PX\_TransmissionProtocal\_Interface\_Write)(px\_byte \*data,px\_int size);

typedef px\_bool (\*PX\_TransmissionProtocal\_Interface\_Read)(px\_byte \*data,px\_int bufferSize,px\_int \*readsize);

typedef px\_bool (\*PX\_TransmissionProtocal\_Interface\_Encode)(px\_byte \*data,px\_int size);

typedef px\_bool (\*PX\_TransmissionProtocal\_Interface\_Decode)(px\_byte \*data,px\_int size);

#### 初始化传输协议

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TransmissionInit(px\_memorypool \*mp,PX\_Transmission \*Ts,px\_dword bridge\_id,PX\_TransmissionProtocal\_Interface\_Write write,PX\_TransmissionProtocal\_Interface\_Read read,PX\_TransmissionProtocal\_Interface\_Encode encode,PX\_TransmissionProtocal\_Interface\_Decode decode); |
| 说明 | **初始化传输协议** |
| 参数 | mp 内存池  Ts 传输实例  bridge\_id 桥id,在每次数据传输中,只有有同一个桥id的连接才能够进行数据传输,当桥id为0时,将会被匹配第一次传入连接的id  write 发送函数指针,注意,除非通讯出现错误,该函数都应该返回TRUE  read 读取函数指针 ,注意,除非通讯出现错误,该函数都应该返回TRUE  encode 编码函数  decode 解码函数 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 协议发送函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TransmissionSend(PX\_Transmission \*trans,px\_byte \*data,px\_dword size); |
| 说明 | **发送缓存区数据,调用该函数后数据不会立即发送,将在下一次PX\_TransmissionUpdate发送,通过检查实例中的status是否为PX\_TRANSMISSION\_STATUS\_DONE判断是否将缓存区完整发送.** |
| 参数 | Ts 传输实例  data 传输缓存  size 大小 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 协议接收函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TransmissionRecv(PX\_Transmission \*trans,px\_byte \*data,px\_dword size); |
| 说明 | **接收数据函数,当这个函数返回PX\_TRUE时,表示有一个数据已经被完整接受到缓存区当中,可以通过预先检查实例中的status是否为PX\_TRANSMISSION\_STATUS\_DONE,通过检查size属性得知接收数据的长度** |
| 参数 | Ts 传输实例  data 传输缓存  size 接收缓存大小 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 协议更新处理函数

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_TransmissionUpdate(PX\_Transmission \*trans,px\_int elpased); |
| 说明 | **协议处理函数,在更新函数中,将完成发送和接收的操作,注意,这个函数需要被调用多次来完成一个收发流程** |
| 参数 | Ts 传输实例  elpased上次更新经过时间,这个时间将影响重发及掉线处理 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE,注意,返回PX\_FALSE可能出现了不可挽回错误,可能需要修正该错误再使用该协议 |

#### 设置最大接收数据长度

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TramsmissionSetMaxRecvSize(PX\_Transmission \*Ts,px\_int size); |
| 说明 | **设置最大接收数据的长度** |
| 参数 | Ts 传输实例  size 最大接收长度,如果这个值设置为小于0的值则忽略最大接收长度 |
| 返回值 | - |

#### 协议完成

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TransmissionCompleted(PX\_Transmission \*Ts); |
| 说明 | **在没错收发完成后,都应该调用这个函数来完成传输** |
| 参数 | Ts 传输实例 |
| 返回值 | - |

#### 释放传输协议

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_TransmissionFree(PX\_Transmission \*Ts); |
| 说明 | **释放传输协议占用资源,如果不再使用该传输协议** |
| 参数 | Ts 传输实例 |
| 返回值 | - |

### 资源管理器

painterengine提供资源管理器,所有的资源从其它存储介质加载后将被以特定的格式保存在资源管理器中,资源管理器所有的对象均由一个唯一的key进行标识存储,其内部使用CRC+红黑树的形式进行索引.

资源包括以下几种类型:

typedef enum

{

PX\_RESOURCE\_TYPE\_NULL, //无类型

PX\_RESOURCE\_TYPE\_ANIMATIONLIBRARY,//动画集

PX\_RESOURCE\_TYPE\_SCRIPT,//编译脚本

PX\_RESOURCE\_TYPE\_TEXTURE,//纹理

PX\_RESOURCE\_TYPE\_SHAPE,//轮廓

PX\_RESOURCE\_TYPE\_DATA,//数据类型

}PX\_RESOURCE\_TYPE;

#### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ResourceLibraryInit(px\_memorypool \*mp,PX\_ResourceLibrary \*lib); |
| 说明 | **资源库初始化** |
| 参数 | mp 资源存放内存池  lib 资源库 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 加载一个资源

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ResourceLibraryLoad(px\_memorypool \*mptemp,PX\_ResourceLibrary \*lib,PX\_RESOURCE\_TYPE type,px\_byte \*data,px\_uint datasize,px\_char \*key); |
| 说明 | **加载资源** |
| 参数 | mptemp 用于资源剖析的临时内存池  lib 资源库  type 资源类型  data 数据缓存  datasize 数据大小  key 索引key |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 添加一个贴图资源

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ResourceLibraryAddTexture(PX\_ResourceLibrary \*lib,px\_char \*key,px\_texture \*pTexture); |
| 说明 | **加载贴图资源** |
| 参数 | lib 资源库  key 索引key  pTexture 需要加载的贴图,这个贴图数据将会被深拷贝到资源池中 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 读取资源

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ResourceLibraryGet(PX\_ResourceLibrary \*lib,PX\_Resource \*pRes,px\_char \*key); |
| 说明 | **读取对应key的资源类型** |
| 参数 | lib 资源库  pres 用于存储读取资源结构  key 对应字符索引 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 读取纹理资源

|  |  |
| --- | --- |
| 函数名 | px\_texture \*PX\_ResourceLibraryGetTexture(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个纹理** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 读取轮廓资源

|  |  |
| --- | --- |
| 函数名 | px\_shape \*PX\_ResourceLibraryGetShape(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个轮廓** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 读取动画资源

|  |  |
| --- | --- |
| 函数名 | px\_animationlibrary \*PX\_ResourceLibraryGetAnimationLibrary(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个动画集** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 读取脚本资源

|  |  |
| --- | --- |
| 函数名 | px\_animationlibrary \*PX\_ResourceLibraryGetScript(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个脚本资源** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 读取声音资源

|  |  |
| --- | --- |
| 函数名 | px\_sounddata \*PX\_ResourceLibraryGetSound(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个脚本资源** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 读取数据资源

|  |  |
| --- | --- |
| 函数名 | px\_memory \*PX\_ResourceLibraryGetData(PX\_ResourceLibrary \*lib,px\_char \*key); |
| 说明 | **从资源中读取一个脚本资源** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | 如果成功返回该资源指针,否者返回PX\_NULL |

#### 删除资源

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_ResourceLibraryGet(PX\_ResourceLibrary \*lib, px\_char \*key); |
| 说明 | **删除对应key的资源类型** |
| 参数 | lib 资源库  key 对应字符索引 |
| 返回值 | - |

#### 释放资源库

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_ResourceLibraryGet(PX\_ResourceLibrary \*lib,PX\_Resource \*pRes,px\_char \*key); |
| 说明 | **读取对应key的资源类型** |
| 参数 | lib 资源库  pres 用于存储读取资源结构  key 对应字符索引 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

### 同步协议

painterEngine提供两种网络同步协议

一种是可用于多人游戏中帧同步或状态同步的开发,提供高容错,延迟优化,断线重连,数据平滑,第三方数据扰乱,数据攻击,等优化

在默认情况下,该同步协议的平滑延迟为66毫秒,约每秒15帧,在所有客户端连接后,服务端开始以这个频率对所有游戏客户端进行同步

一种是数据同步协议,该协议将保证每一个客户端的数据和服务端数据同步.

#### 帧同步服务端

##### 服务端初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SyncFrameServerInit(PX\_Sync\_Server \*sync,px\_memorypool \*mp,px\_dword updateDuration,px\_sync\_server\_read read,px\_sync\_server\_write write,px\_void \*user); |
| 说明 | **同步器服务端初始化** |
| 参数 | sync同步器结构体  mp 内存池  updateduration 平滑延迟  read 服务器读数据函数指针,该指针可以使用任何不可靠通信IO实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高PX\_SYNC\_CLIENT\_SEND\_DURATION和PX\_SYNC\_SERVER\_SEND\_TIMES减轻该丢包造成的延迟,但这也意味着将占有更大的带宽  write服务器写数据函数指针,该指针可以使用任何不可靠通信IO实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高PX\_SYNC\_CLIENT\_SEND\_DURATION和PX\_SYNC\_SERVER\_SEND\_TIMES减轻该丢包造成的延迟,但这也意味着将占有更大的带宽  user 用户自定义数据指针,可以访问结构体获得 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 服务端添加客户端

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SyncFrameServerAddClient(PX\_Sync\_Server \*sync,px\_dword verify\_id,px\_dword client\_verify\_id,px\_dword c\_id); |
| 说明 | **服务端添加客户端** |
| 参数 | sync同步器结构体  verify\_id 验证id,这个id需要以安全随机数产生不可预测,与client\_verify\_id成对使用  client\_verify\_id 客户端验证id, 这个id需要以安全随机数产生不可预测,与verify\_id成对使用,该表示是验证客户端的唯一标准方式  c\_id 用于客户端的玩家标识,将会随着指令一通写入到指令流当中 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 服务端继续运行

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameServerRun(PX\_Sync\_Server \*sync); |
| 说明 | **服务端继续运行** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

##### 服务端暂停运行

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameServerStop(PX\_Sync\_Server \*sync); |
| 说明 | **服务端暂停运行** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

##### 服务端更新循环

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameServerUpdate(PX\_Sync\_Server \*sync,px\_dword elpased); |
| 说明 | **服务端更新** |
| 参数 | sync同步器结构体  elapsed 距离上次更新经过的时间,注意,该值应该做到尽可能的小,否者可能导致过大的抖动 |
| 返回值 | - |

##### 释放服务端

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameServerFree(PX\_Sync\_Server \*sync); |
| 说明 | **释放服务端资源** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

##### 取得数据累加和校验

|  |  |
| --- | --- |
| 函数名 | px\_uint32 PX\_SyncFrameServerSum32(PX\_Sync\_Server \*sync); |
| 说明 | **计算当前同步数据的累加和,注意,只有当一帧结束后进行累加校验才有参考意义** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

#### 帧客户端

##### 客户端初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_SyncFrameClientInit(PX\_Sync\_Client \*client,px\_memorypool \*mp,px\_dword updateDuration,PX\_Sync\_Port serverport,px\_dword server\_verify\_id,px\_dword verify\_id,px\_sync\_client\_read read,px\_sync\_client\_write write,px\_void \*user)); |
| 说明 | **同步器客户端初始化** |
| 参数 | client同步器结构体  mp 内存池  verify\_id 验证id,这个id需要以安全随机数产生不可预测,与server\_verify\_id成对使用  erver\_verify\_id 客户端验证id, 这个id需要以安全随机数产生不可预测,与verify\_id成对使用,该表示是验证客户端的唯一标准方式  read客户端读数据函数指针,该指针可以使用任何不可靠通信IO实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高PX\_SYNC\_CLIENT\_SEND\_DURATION和PX\_SYNC\_SERVER\_SEND\_TIMES减轻该丢包造成的延迟,但这也意味着将占有更大的带宽  write客户端写数据函数指针,该指针可以使用任何不可靠通信IO实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高PX\_SYNC\_CLIENT\_SEND\_DURATION和PX\_SYNC\_SERVER\_SEND\_TIMES减轻该丢包造成的延迟,但这也意味着将占有更大的带宽  user 用户自定义数据指针,可以访问结构体获得 |
| 返回值 | 如果成功返回PX\_TRUE,否者返回PX\_FALSE |

##### 添加指令

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameClientAddInstr(PX\_Sync\_Client \*client,px\_void \*instr,px\_int size); |
| 说明 | **添加客户端指令流,该指令流会被所有客户端同步,同步时间由服务端确定** |
| 参数 | client同步器结构体  instr 指令流指针  size 指令流长度 |
| 返回值 | - |

##### 更新客户端

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameClientUpdate(PX\_Sync\_Client \*sync,px\_dword elpased); |
| 说明 | **更新客户端** |
| 参数 | client同步器结构体  elapsed 距离上次更新经过的时间,注意,该值应该做到尽可能的小,否者可能帧率不稳定 |
| 返回值 | - |

##### 释放客户端

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_SyncFrameClientFree(PX\_Sync\_Client \*sync); |
| 说明 | **释放客户端资源** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

##### 取得数据累加和校验

|  |  |
| --- | --- |
| 函数名 | px\_uint32 PX\_SyncFrameClientSum32(PX\_Sync\_Client \*sync); |
| 说明 | **计算当前同步数据的累加和,注意,只有当一帧结束后进行累加校验才有参考意义** |
| 参数 | sync同步器结构体 |
| 返回值 | - |

### JSON

PainterEngine支持对Json格式的解析,但不支持字符串转义,如果需要转义请自行处理

typedef struct \_PX\_Json\_Object

{

px\_list values;//用于存储PX\_Json\_Value的链表

}PX\_Json\_Object;

typedef struct

{

PX\_JSON\_VALUE\_TYPE type;//值数据类型

px\_string name;//值名

union

{

px\_string \_string;//字符串数据

px\_double \_number;//数值数据

px\_bool \_boolean;//布尔类型数据

PX\_Json\_Object json\_Object;//对象数据

px\_list \_Array;//数组数据

};

}PX\_Json\_Value;

#### 初始化

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_JsonInitialize(px\_memorypool \*mp,PX\_Json \*pjson); |
| 说明 | **初始化一个JSON结构体** |
| 参数 | Mp 内存池  Pjson json结构体 |
| 返回值 | -如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 读取JSON数据

|  |  |
| --- | --- |
| 函数名 | PX\_Json\_Value \* PX\_JsonGetValue(PX\_Json \*json,const px\_char \_payload[]) |
| 说明 | **读取JSON数据** |
| 参数 | Pjson json结构体  payload 加载文本  payload示范  读取类成员 Human.lily.age  支持数组读取 human.score[10].point |
| 返回值 | -如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 解析JSON数据

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_JsonParse(PX\_Json \*pjson,const px\_char \*json\_content); |
| 说明 | **解析JSON数据** |
| 参数 | Pjson json结构体  Json\_content JSON文本 |
| 返回值 | -如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 将JSON结构转换为文本

|  |  |
| --- | --- |
| 函数名 | px\_bool PX\_JsonBuild(PX\_Json \*pjson,px\_string \*\_out); |
| 说明 | **解析JSON数据** |
| 参数 | Pjson json结构体  Json\_content JSON文本 |
| 返回值 | -如果成功返回PX\_TRUE,否者返回PX\_FALSE |

#### 释放JSON

|  |  |
| --- | --- |
| 函数名 | px\_void PX\_JsonFree(PX\_Json \*pjson); |
| 说明 | **释放JSON结构数据** |
| 参数 | Pjson json结构体 |
| 返回值 |  |

### PainterEngine 基本对象

#### 使用PainterEngine绘制对象

PainterEngine的对象是由PainterEngine图形渲染函数进一步封装的结果，在PainterEngine中，所有的显示物都是以对象的形式存在的，对象是PainterEngine显示的基本形式，通过对对象的派生而衍生出不同的功能类型。

#### PainterEngine对象机制

PainterEngine的对象是所有控件的基础类型，一个对象当中包含了以下几种类型

1. 对象的父对象
2. 对象的前一个兄弟对象
3. 对象的后一个兄弟对象
4. 对象的子对象

用下图表示对象之间的级联关系



依照图所示，每一个Object都拥有4个链接点，向上方向指向该对象的父对象，向下指向该对象的子对象，向左指向该对象的前一个兄弟对象，向右指向改对象的后一个兄弟对象，当方向上不存在这个对象指向的对应节点的时候，这个节点将会也必须被赋值为NULL表示空。也就是，一个对象的四个方向节点，要么有效存在，要么为NULL。

当父对象为NULL且只存在子对象的时候，我们称之为对“根对象”，

根对象不存在父对象，也不存在兄弟对象，例如下图当中，被红色标志的Object即为根对象



对象的级联关系中，我们将上下级联的对象称之为“父子对象”，如下图所示，Object1与Object2互为父子



Object1是Object2的父对象，Object2是Object1的子对象

同时可以从上图看到Object2拥有2个兄弟对象，分别是Object3与Object4，在广义上Object3与Object4也是Object1的子对象，但是，Object3与Object4的父向量并不指向Object1而指向了NULL，这也意味着无法通过Object3或Object4直接寻找到其父对象而必须经过Object2.这也就意味着Object2是对象的入口，在同一层级的对象中，有且仅有一个入口对象，在上图中可以看到，Object2是Object1子对象的入口对象，Object5是Object3子对象的入口对象

PainterEngine提供了以下几个函数，用于对象的操作与使用

**PX\_Object \*PX\_ObjectCreate(PX\_Object \*Parent,int x,int y,int Width,int Height);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectCreate |
| 功能 | 创建一个对象 |
| 参数Parent | 创建对象的父对象 |
| 参数x | 对象的左上角坐标x |
| 参数y | 对象的左上角坐标y |
| 参数width | 对象的宽度 |
| 参数height | 对象的高度 |
| 返回值PX\_ObjectCreate \* | 创建对象的指针 |

当Parent为NULL的时候，表示这个对象是一个根对象，当对象不为NULL的时候，这个对象创建如下

创建对象的连接规则

1.当父对象没有子对象的时候，创建的对象将作为其子对象



2.当父对象拥有子对象的时候，这个对象将被插入到父对象的子对象层次的最后一个节点



**void PX\_ObjectInit(PX\_Object \*Object,PX\_Object \*Parent,int x,int y,int Width,int Height);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectInit |
| 功能 | 初始换一个对象 |
| 参数Object | 需要初始化的对象 |
| 参数Parent | 初始化该对象的父对象 |
| 参数x | 对象的左上角坐标x |
| 参数y | 对象的左上角坐标y |
| 参数width | 对象的宽度 |
| 参数height | 对象的高度 |

这个函数用于初始化自定义的一个对象，并将对象连接到父对象当中，连接的规则和创建对象的连接规则一样。

**void PX\_ObjectDelete(PX\_Object \*\*ppObject);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectDelete |
| 功能 | 删除一个对象 |
| 参数Object | 指向该对象的二级指针 |

使用该函数将会将这个对象断开原有的对象链并删除，同时，这个对象所有的子对象都会被删除

如下图所示，删除Object将导致如下对象都被删除

删除完成后，Object2的下一个兄弟向量将被重新定位连接到Object4



如果删除的是Object2，则Object1的子对象将被重新定位到Object3，而Object3将作为Object1子对象的入口对象



**void PX\_DeleteLinkerObject(PX\_Object \*\*ppObject);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectLinkerDelete |
| 功能 | 删除该对象和该对象后续的所有兄弟对象 |
| 参数Object | 指向该对象的二级指针 |

**void PX\_ObjectSetPosition(PX\_Object \*Object,int x,int y);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetPosition |
| 功能 | 设置对象的位置 |
| 参数Object | 该对象指针 |
| 参数x | 对象的新横坐标x |
| 参数y | 对象的新纵坐标y |

**void PX\_ObjectSetSize(PX\_Object \*Object,int Width,int Height);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetSize |
| 功能 | 设置对象的大小 |
| 参数Object | 该对象的指针 |
| 参数width | 设置对象的新宽度 |
| 参数height | 设置对象的新高度 |

**void PX\_ObjectFree(PX\_Object \*Object);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetVisible |
| 功能 | 设置该对象可见性（将影响其子对象） |
| 参数Object | 该对象指针 |
| 参数Visible | 可见性，不为0表示可见，为0不可见 |

#### 创建对象

非根对象是除根对象的附属对象，就如同对象机制中描述的那样，非根对象必须被链接到一个根对象当中，同时，可以进一步指派该对象的长宽高坐标位置等具体属性类型。

**PX\_Object \*PX\_ObjectCreate(px\_memorypool \*mp,PX\_Object \*Parent,px\_float x,px\_float y,px\_float z,px\_float Width,px\_float Height,px\_float Lenght);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectCreate |
| 功能 | 创建一个对象 |
| mp | 对象使用的内存池 |
| parent | 父对象(如果为NULL表示这是一个根对象) |
| x,y,z | 坐标信息,注意z坐标信息,z的值越大,这个对象在渲染时将被最先绘制,这也就意味着z值小的对象将会覆盖在z值较大的对象中 |
| Width,Heigth,Lenght | 宽度,高度,长度 |
| 返回值 | 返回创建的对象指针 |

注意,其它的非根对象需要被链接到一个根对象中,如果该对象是一个实际绘制对象,可以在创建完根对象后对其的具体属性进行进一步的赋值操作。

#### 设置对象的usercode

**px\_void PX\_ObjectSetUserCode(PX\_Object \*pObject,px\_int user);**

**px\_void PX\_ObjectSetUserPointer(PX\_Object \*pObject,px\_void \*user\_ptr);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetUserCode  PX\_ObjectSetUserPointer |
| 功能 | 设置对象的user字段,这个字段为用户自定义的类型,可以将这个值赋值为需要的数据类型,在对象更新渲染或者是事件回调时,利用该函数获取额外信息 |
| Object | 需要设置的对象指针 |
| user\_int  user\_ptr | 设置的值,注意,每个对象的user\_int和User\_ptr为共享内存的数据类型.建议仅能设置其中一个 |

#### 删除对象

**px\_void PX\_ObjectDelete(PX\_Object \*pObject);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectDelete |
| 功能 | 删除一个对象,注意,当需要删除对象的时候不能够使用MP\_Free函数,必须调用该函数来删除一个对象,该函数将会断开对象与父对象与兄弟对象的链接,整理对象关系 |
| Object | 需要删除的对象指针 |

**px\_void PX\_ObjectDeleteChilds( PX\_Object \*pObject );**

|  |  |
| --- | --- |
| 函数名 | PX\_ ObjectDeleteChilds |
| 功能 | 删除一个对象的所有子对象 |
| Object | 需要删除子对象的对象指针 |

#### 对象属性设置

**px\_void PX\_ObjectSetPosition(PX\_Object \*Object,px\_float x,px\_float y);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetPosition |
| 功能 | 设置一个对象的坐标 |
| Object | 需要设置的对象指针 |
| x,y,z | 三维坐标 |

**px\_void PX\_ObjectSetSize(PX\_Object \*Object,px\_float Width,px\_float Height,px\_float Length);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetSize |
| 功能 | 设置一个对象的宽度和高度 |
| Object | 需要设置的对象指针 |
| Width,Height,Length | 宽,高,长 |

**px\_void PX\_ObjectSetVisible(PX\_Object \*Object,px\_bool visible);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectSetVisible |
| 功能 | 设置一个对象的可见性 |
| Object | 需要设置的对象指针 |
| visible | 可见性,如果为FALSE这个对象将不被渲染 |
|  |  |

**px\_float PX\_ObjectGetHeight(PX\_Object \*Object);**

**px\_float PX\_ObjectGetWidth(PX\_Object \*Object);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectGetHeight  PX\_ObjectGetWidth |
| 功能 | 取得对象的高宽 |
| Object | 对象指针 |
| 返回值 | 对象的高宽 |

#### PainterEngine更新/绘制流水线

PainterEngine的绘制流程按照

先父对象，后子对象，先左兄弟对象，后右兄弟对象的流程进行绘制

PainterEngine提供两个对象绘制函数进行对象的绘制，分别是

**void PX\_ObjectUpdate(struct \_PX\_Object \*,px\_uint elpased);**

|  |  |
| --- | --- |
| 函数名 | ObjectUpdate |
| 功能 | 更新该对象及该对象的所有子对象 |
| 参数Object | 需要更新的对象指针 |
| 参数 elapsed | 上一次更新经过的时间 |

**void PX\_ObjectRender(px\_surface \*psurface,struct \_PX\_Object \*,px\_uint elpased);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectRender |
| 功能 | 绘制该对象及该对象的所有子对象 |
| 参数 psurface | 将对象渲染到的目标表面 |
| 参数Object | 需要绘制的对象指针 |
| 参数 elapsed | 上一次更新经过的时间 |

依照下图所示更新/绘制的绘制流程如下



Start🡪Object1🡪Object2🡪Object3🡪object5🡪object6🡪Object7🡪Object4🡪End;

#### 对象事件机制

在PainterEngine中,每个对象都可以为其注册触发事件,当使用PX\_ObjectPostEvent函数对一个对象进行事件投递后,该对象及其子对象都会受到该事件信息,如果这个对象有注册这个事件的处理函数,将会回调该处理函数.

在默认的情况下,PainterEngine默认内置以下几种事件

#define PX\_OBJECT\_EVENT\_ANY 0

//任意事件

#define PX\_OBJECT\_EVENT\_CURSORMOVE 1

//收到指针移动事件,在PC中常常是鼠标移动触发

#define PX\_OBJECT\_EVENT\_CURSORUP 2

//收到指针抬起事件,在PC中是鼠标左键点击抬起,在移动端是触摸抬起

#define PX\_OBJECT\_EVENT\_CURSORRDOWN 3

//收到指针按下事件,在PC中是鼠标右键点击按下

#define PX\_OBJECT\_EVENT\_CURSORDOWN 4

//收到指针按下事件,在PC中是鼠标左键点击按下,在移动端是触摸按下

#define PX\_OBJECT\_EVENT\_CURSORRUP 5

//收到指针抬起事件,在PC中是鼠标右键点击抬起

#define PX\_OBJECT\_EVENT\_STRING 6

//字符串事件,常常是输入法输入文字触发该事件

#define PX\_OBJECT\_EVENT\_CURSORCLICK 7

//指针点击事件,在PC中是鼠标点击,在移动端是手指触摸点击

#define PX\_OBJECT\_EVENT\_CURSORDRAG 8

//指针拖拽事件,在PC中是鼠标左键按下拖动,移动端是手指触摸按下移动

#define PX\_OBJECT\_EVENT\_VALUECHAGE 9

//对象值改变事件,针对部分内设对象,当其值改变时会触发该事件对象

#define PX\_OBJECT\_EVENT\_DRAGFILE 10

//文件拖动事件,当一个文件被拖动到客户端中

#define PX\_OBJECT\_EVENT\_KEYDOWN 11

//按键事件,当键盘某个按键按下时触发

一个事件的由以下结构体组成

typedef struct \_PX\_Object\_Event

{

px\_uint Event;

union

{

px\_uint PARAM[4];//内建参数

px\_void \* user;

};

}PX\_Object\_Event;

需要注意的是,当其为指针事件时PARAM[0] PARAM[1] PARAM[2]分别为指针的X,Y,Z坐标(Z不一定用到)

当其为字符串事件时,user指向输入字符串指针

当为拖拽文件事件时,user指向拖拽文件路径

#### 响应处理函数

响应处理函数是对象接收到对应事件调用的回调函数

其定义为

**px\_void (\*ProcessFunc)(PX\_Object \*,PX\_Object\_Event e);**

|  |  |
| --- | --- |
| 函数名 | ProcessFunc |
| 功能 | 事件回调函数 |
| 参数Object | 响应对象指针 |
| 参数Event | 响应的事件类型 |

#### 注册对象事件

**px\_int PX\_ObjectRegisterEvent(PX\_Object \*Object,px\_uint Event,px\_void (\*ProcessFunc)(PX\_Object \*,PX\_Object\_Event e),px\_void \*ptr);**

|  |  |
| --- | --- |
| 函数名 | PX\_ObjectRegisterEvent |
| 功能 | 为一个对象注册响应事件 |
| 参数Object | 需要绘制的对象指针 |
| 参数Event | 响应的事件类型 |
| ProcessFunc | 响应处理函数 |
| ptr | 用户指针 |

#### 派分事件

**px\_void PX\_ObjectPostEvent(PX\_Object \*pPost,PX\_Object\_Event Event);**

|  |  |
| --- | --- |
| 函数名 | ObjectPostEvent |
| 功能 | 派分事件,该事件将会被派分到该对象及其子对象当中,如果该对象注册了该事件,将会执行该事件的回调函数 |
| 参数Object | 派分的对象指针 |
| 参数Event | 响应的事件 |