**Pygame 官方文档 - pygame.sprite**

pygame.sprite

带有基本游戏对象类的pygame模块  
[pygame.sprite.Sprite](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite) - 可见游戏对象的简单基类。  
[pygame.sprite.DirtySprite](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#DirtySprite%28%29) - 具有更多属性和功能的Sprite的子类。  
[pygame.sprite.Group](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group) - 用于保存和管理多个Sprite对象的容器类。  
[pygame.sprite.RenderPlain](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#RenderPlain) - 与pygame.sprite.Group相同  
[pygame.sprite.RenderClear](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#RenderClear) - 与pygame.sprite.Group相同  
[pygame.sprite.RenderUpdates](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#RenderUpdates) - 跟踪dirty更新的组子类  
[pygame.sprite.OrderedUpdates](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#OrderedUpdates%28%29) - RenderUpdates（渲染更新）按添加顺序绘制Sprite的子类。  
[pygame.sprite.LayeredUpdates](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates) - LayeredUpdates（分层更新）是一个精灵组，它处理图层并像OrderedUpdates（顺序更新）一样绘制。  
[pygame.sprite.LayeredDirty](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty) - LayeredDirty组用于DirtySprite对象。子类（分层更新）LayeredUpdates。  
[pygame.sprite.GroupSingle](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#GroupSingle%28%29) - 包含单个sprite的组容器。  
[pygame.sprite.spritecollide](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#spritecollide%28%29) - 在与另一个精灵相交的组中查找精灵。  
[pygame.sprite.collide\_rect](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#collide_rect%28%29) - 使用rects检测两个精灵之间的碰撞。  
[pygame.sprite.collide\_rect\_ratio](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#collide_rect_ratio%28%29) - 使用按比例缩放的rects检测两个精灵之间的碰撞。  
[pygame.sprite.collide\_circle](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#collide_circle%28%29) - 使用圆圈检测两个精灵之间的碰撞。  
[pygame.sprite.collide\_circle\_ratio](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#collide_circle_ratio%28%29) - 使用按比例缩放的圆圈检测两个精灵之间的碰撞。  
[pygame.sprite.collide\_mask](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#collide_mask%28%29) - 使用蒙版在两个精灵之间进行碰撞检测。  
[pygame.sprite.groupcollide](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#groupcollide%28%29) - 查找在两个组之间发生碰撞的所有精灵。  
[pygame.sprite.spritecollideany](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#spritecollideany%28%29) - 如果精灵与组中的任何内容相交，则进行简单测试。

        该模块包含几个在游戏中使用的简单类。有一个主要的Sprite类和几个包含Sprite的Group类。使用pygame时，这些类的使用完全是可选的。这些类相当轻量级，只为大多数游戏常见的代码提供了一个起始点。  
        Sprite类旨在用作游戏中不同类型对象的基类。还有一个基本的Group类，它只存储sprite对象。游戏可以创建新类型的Group类，这些类在它们包含的特殊定制的Sprite实例上运行。  
        基本的Sprite类可以将它包含的Sprite对象绘制到Surface对象。 Group.draw()方法要求每个Sprite都有一个Surface.image属性和一个Surface.rect。 Group.clear()方法需要这些相同的属性，并可用于擦除所有具有背景的Sprite。还有更高级的组：pygame.sprite.RenderUpdates()和pygame.sprite.OrderedUpdates()。  
        最后，该模块包含多个碰撞功能。这些有助于在具有交叉边界矩形的多个组内查找精灵。要查找碰撞，Sprite对象必须分配Surface.rect属性。  
        这些组旨在高效地删除和添加Sprite对象。它们还允许进行简单的测试，以查看Sprite对象中是否已存在Group中。给定的Sprite可以存在于任意数量的组中。游戏可以使用一些组来控制对象渲染，以及一组完全独立的组来控制交互或玩家移动。不要将类型属性或bool添加到派生的Sprite类，而应考虑将Sprite保留在有组织的Group中。这将允许在游戏中稍后更加容易的查找。  
        Sprite和Groups使用add()和remove()方法管理它们之间的关系。对于它们的成员资格，这些方法可以接受单个或多个目标。这些类的默认初始值设定项还为初始成员资格采用单个或目标列表。从组中重复添加和删除相同的Sprite对象是安全的。  
        虽然可以不从下面的Sprite和AbstractGroup类派生设计sprite和group类，但强烈建议您在添加Sprite或Group类时扩展它们。  
精灵不是线程安全的。因此，如果使用线程，请自行锁定。

pygame.sprite.Sprite [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

可见游戏对象的简单基类。  
Sprite(\*groups) -> Sprite  
[pygame.sprite.Sprite.update](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.update%28%29) - 控制精灵行为的方法  
[pygame.sprite.Sprite.add](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.add%28%29) - 将精灵添加到组中  
[pygame.sprite.Sprite.remove](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.remove%28%29) - 从组中删除精灵  
[pygame.sprite.Sprite.kill](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.kill%28%29) - 从所有组中删除Sprite  
[pygame.sprite.Sprite.alive](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.alive%28%29) - 精灵属于任何组的检测  
[pygame.sprite.Sprite.groups](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite.groups%28%29) - 包含此Sprite的组列表  
可见游戏对象的基类。 派生类将要覆盖Sprite.update()并分配Sprite.image和Sprite.rect属性。 初始值设定项可以接受要添加的任意数量的Group实例。  
在对Sprite进行子类化时，请务必在Sprite添加到Groups之前调用基本初始值设定项。 例如：

class Block(pygame.sprite.Sprite):

#构建函数 传入颜色和它的x、y坐标

def \_\_init\_\_(self, color, width, height):

#调用父类（Sprite）的构建函数

pygame.sprite.Sprite.\_\_init\_\_(self)

#创建一个图像，用某个颜色填充

#这也可以是从磁盘加载的图像

self.image = pygame.Surface([width, height])

self.image.fill(color)

＃获取具有图像尺寸的矩形对象

＃通过设置rect.x和rect.y的值来更新该对象的位置

self.rect = self.image.get\_rect()

pygame.sprite.Sprite.update() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

控制精灵行为的方法  
update(\*args) -> None  
此方法的默认实现不执行任何操作; 它只是一个方便的“钩子”，你可以覆盖。 Group.update()使用您提供的任何参数调用此方法。  
如果没有在Group类中使用同名的方便方法，则无需使用此方法。  
[搜索pygame.sprite.Sprite.update的示例](https://github.com/search?q=pygame.sprite.Sprite.update&type=Code)

pygame.sprite.Sprite.add() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

将精灵添加到组中  
add(\*groups) -> None  
可以将任意数量的Group实例作为参数传递。 Sprite将被添加到它还不是其成员的组中。  
[搜索pygame.sprite.Sprite.add的示例](https://github.com/search?q=pygame.sprite.Sprite.add&type=Code)

pygame.sprite.Sprite.remove() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

从组中删除精灵  
remove(\*groups) -> None  
可以将任意数量的Group实例作为参数传递。 Sprite将从其当前所属的组中删除  
[搜索pygame.sprite.Sprite.remove的示例](https://github.com/search?q=pygame.sprite.Sprite.remove&type=Code)

pygame.sprite.Sprite.kill() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

从所有组中删除Sprite  
kill() -> None  
Sprite将从包含它的所有组中删除。 这不会改变关于Sprite状态的任何信息。 调用此方法后，可以继续使用Sprite，包括将其添加到Groups。  
[搜索pygame.sprite.Sprite.kill的示例](https://github.com/search?q=pygame.sprite.Sprite.kill&type=Code)

pygame.sprite.Sprite.alive() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

精灵属于任何组的检测  
alive() -> bool  
当Sprite属于一个或多个组时返回True。  
[搜索pygame.sprite.Sprite.alive的示例](https://github.com/search?q=pygame.sprite.Sprite.alive&type=Code)

pygame.sprite.Sprite.groups() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Sprite)

包含此Sprite的组列表  
groups() -> group\_list  
返回包含此Sprite的所有组的列表。  
[搜索pygame.sprite.Sprite.groups的示例](https://github.com/search?q=pygame.sprite.Sprite.groups&type=Code)

pygame.sprite.DirtySprite() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

具有更多属性和功能的Sprite的子类。  
DirtySprite(\*groups) -> DirtySprite  
额外的DirtySprite属性及其默认值：  
dirty = 1

if set to 1, it is repainted and then set to 0 again

if set to 2 then it is always dirty ( repainted each frame,

flag is not reset)

0 means that it is not dirty and therefore not repainted again

如果设置为1，则重新绘制，然后再次设置为0

如果设置为2，那么它总是dirty（重新绘制每一帧，标志未重置）

0表示它不dirty，因此不再重新绘制

blendmode = 0

its the special\_flags argument of blit, blendmodes

它是blit、blendmodes的special\_flags 参数

source\_rect = None

source rect to use, remember that it is relative to

topleft (0,0) of self.image

源rect 使用，记住它是相对于self.image的左上角（0,0）坐标

visible = 1

normally 1, if set to 0 it will not be repainted

(you must set it dirty too to be erased from screen)

101/5000

通常为1，如果设置为0，则不会重新绘制

（你必须将它设置为dirty以便从屏幕上删除）

layer = 0

(READONLY value, it is read when adding it to the

LayeredDirty, for details see doc of LayeredDirty)

（只读的值，将其添加到LayeredDirty时被读取，

有关详细信息，请参阅LayeredDirty的文档）

[搜索pygame.sprite.DirtySprite的示例](https://github.com/search?q=pygame.sprite.DirtySprite&type=Code)

pygame.sprite.Group [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

用于保存和管理多个Sprite对象的容器类。  
Group(\*sprites) -> Group  
[pygame.sprite.Group.sprites](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.sprites%28%29) - 此组包含的Sprite列表  
[pygame.sprite.Group.copy](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.copy%28%29) - 复制组  
[pygame.sprite.Group.add](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.add%28%29) - 将Sprite添加到此组  
[pygame.sprite.Group.remove](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.remove%28%29) - 从组中删除Sprite  
[pygame.sprite.Group.has](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.has%28%29) - 测试一个Group是否包含Sprite  
[pygame.sprite.Group.update](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.update%28%29) - 在包含的Sprite上调用update方法  
[pygame.sprite.Group.draw](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.draw%28%29) - blit Sprite的图像  
[pygame.sprite.Group.clear](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.clear%28%29) - 在Sprites上绘制背景  
[pygame.sprite.Group.empty](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group.empty%28%29) - 删除所有Sprite  
Sprite对象的简单容器。 可以继承此类以创建具有更多特定行为的容器。 构造函数将任意数量的Sprite参数添加到Group中。 该组支持以下标准Python操作：

in test if a Sprite is contained

len the number of Sprites contained

bool test if any Sprites are contained

iter iterate through all the Sprites

in 测试是否包含Sprite

len 包含的精灵数量

bool 测试是否包含任何精灵

iter 遍历所有精灵

组中的Sprite不是有序的，因此绘制和迭代Sprites没有特别的顺序。

pygame.sprite.Group.sprites() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

此组包含的Sprite列表  
sprites() -> sprite\_list  
返回此组包含的所有Sprite的列表。 您也可以从组中获取迭代器，但在修改它时不能对组进行迭代。  
[搜索pygame.sprite.Group.sprites的示例](https://github.com/search?q=pygame.sprite.Group.sprites&type=Code)

pygame.sprite.Group.copy() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

复制组  
copy() -> Group  
创建一个包含与原始Sprites相同的Sprit的新组。 如果您有子类Group，则新对象将具有与原始对象相同的（子）类。 这仅在派生类的构造函数采用与Group类相同的参数时才有效。  
[搜索pygame.sprite.Group.copy的示例](https://github.com/search?q=pygame.sprite.Group.copy&type=Code)

pygame.sprite.Group.add() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

将Sprite添加到此组  
add(\*sprites) -> None  
向此组添加任意数量的Sprite。 这只会添加尚未成为该组成员的Sprite。  
每个sprite参数也可以是包含Sprite的迭代器。  
[搜索pygame.sprite.Group.add的示例](https://github.com/search?q=pygame.sprite.Group.add&type=Code)

pygame.sprite.Group.remove() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

从组中删除Sprite  
remove(\*sprites) -> None  
从Group中删除任意数量的精灵。 这将仅删除已经是该组成员的Sprite。  
每个sprite参数也可以是包含Sprite的迭代器。  
[搜索pygame.sprite.Group.remove的示例](https://github.com/search?q=pygame.sprite.Group.remove&type=Code)

pygame.sprite.Group.has() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

测试一个Group是否包含Sprite  
has(\*sprites) -> None  
如果组包含所有给定的精灵，则返回True。 这类似于在Group上使用“in”运算符（“if sprite in group：…”），它测试单个Sprite是否属于一个Group。  
每个sprite参数也可以是包含Sprite的迭代器。  
[搜索pygame.sprite.Group.has的示例](https://github.com/search?q=pygame.sprite.Group.has&type=Code)

pygame.sprite.Group.update() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

在包含的Sprite上调用update方法  
update(\*args) -> None  
在组中的所有Sprite上调用update()方法。 基础Sprite类有一个更新方法，它接受任意数量的参数并且什么也不做。 传递给Group.update()的参数将传递给每个Sprite。  
无法从Sprite.update()方法获取返回值。  
[搜索pygame.sprite.Group.update的示例](https://github.com/search?q=pygame.sprite.Group.update&type=Code)

pygame.sprite.Group.draw() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

blit Sprite的图像  
draw(Surface) -> None  
将包含的Sprite绘制到Surface参数。 这使用源surface的Sprite.image属性和Sprite.rect的位置。  
本Group不会以任何顺序保留精灵，因此绘制顺序是任意的。  
[搜索pygame.sprite.Group.draw的示例](https://github.com/search?q=pygame.sprite.Group.draw&type=Code)

pygame.sprite.Group.clear() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

在Sprites上绘制背景  
clear(Surface\_dest, background) -> None  
删除最后一次Group.draw()调用中使用的Sprite。 通过用背景填充绘制的Sprite位置来清除目标Surface。  
背景通常是与目标Surface相同尺寸的Surface图像。 但是，它也可以是一个带有两个参数的回调函数; 目的地Surface和要清除的区域。 背景的回调函数将被每次清除时调用多次。  
这是一个示例回调，它将使用纯红色清除Sprite：

def clear\_callback(surf, rect):

color = 255, 0, 0

surf.fill(color, rect)

[搜索pygame.sprite.Group.clear的示例](https://github.com/search?q=pygame.sprite.Group.clear&type=Code)

pygame.sprite.Group.empty() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#Group)

删除所有Sprite  
empty() -> None  
从该组中删除所有Sprite。  
[搜索pygame.sprite.Group.empty的示例](https://github.com/search?q=pygame.sprite.Group.empty&type=Code)

pygame.sprite.RenderPlain [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

与pygame.sprite.Group相同  
该类是pygame.sprite.Group()的别名。 它没有其他功能。  
[搜索pygame.sprite.RenderPlain的示例](https://github.com/search?q=pygame.sprite.RenderPlain&type=Code)

pygame.sprite.RenderClear [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

与pygame.sprite.Group相同  
该类是pygame.sprite.Group()的别名。 它没有其他功能。  
[搜索pygame.sprite.RenderClear的示例](https://github.com/search?q=pygame.sprite.RenderClear&type=Code)

pygame.sprite.RenderUpdates [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

跟踪dirty更新的组子类  
RenderUpdates(\*sprites) -> RenderUpdates  
[pygame.sprite.RenderUpdates.draw](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#RenderUpdates.draw%28%29) — blit Sprite图像并跟踪更改的区域  
该类派生自pygame.sprite.Group()。 它有一个扩展的draw()方法，可以跟踪屏幕的变化区域。

pygame.sprite.RenderUpdates.draw() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#RenderUpdates)

blit Sprite图像并跟踪更改的区域  
draw(surface) -> Rect\_list  
将所有Sprite绘制到surface，与Group.draw()相同。 此方法还返回屏幕上已更改的矩形区域列表。 返回的更改包括已受先前Group.clear()调用影响的屏幕区域。  
返回的Rect列表应该传递给pygame.display.update()。 这将有助于软件驱动显示模式的性能。 这种类型的更新通常仅对具有非动画背景的目的地有用。  
[搜索pygame.sprite.RenderUpdates.draw的示例](https://github.com/search?q=pygame.sprite.RenderUpdates.draw&type=Code)

pygame.sprite.OrderedUpdates() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

RenderUpdates（渲染更新）按添加顺序绘制Sprite的子类。  
OrderedUpdates(\*spites) -> OrderedUpdates  
该类派生自pygame.sprite.RenderUpdates()。 它维持Sprite被添加到Group进行渲染的顺序。 这使得从组中添加和删除Sprites比常规组慢一点。  
[搜索pygame.sprite.OrderedUpdates的示例](https://github.com/search?q=pygame.sprite.OrderedUpdates&type=Code)

pygame.sprite.LayeredUpdates [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

LayeredUpdates（分层更新）是一个精灵组，它处理图层并像OrderedUpdates（顺序更新）  
一样绘制。  
LayeredUpdates(\*spites, \*\*kwargs) -> LayeredUpdates  
[pygame.sprite.LayeredUpdates.add](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.add%28%29) - 向一个组添加sprite或sprite序列  
[pygame.sprite.LayeredUpdates.sprites](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.sprites%28%29) - 返回精灵的有序列表（第一个底部，最后一个顶部）。  
[pygame.sprite.LayeredUpdates.draw](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.draw%28%29) - 以正确的顺序将所有精灵绘制到传递的surface上。  
[pygame.sprite.LayeredUpdates.get\_sprites\_at](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_sprites_at%28%29) - 返回包含该位置所有精灵的列表。  
[pygame.sprite.LayeredUpdates.get\_sprite](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_sprite%28%29) - 从组sprites中返回索引idx处的sprite  
[pygame.sprite.LayeredUpdates.remove\_sprites\_of\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.remove_sprites_of_layer%28%29) - 从图层中删除所有精灵并将其作为列表返回。  
[pygame.sprite.LayeredUpdates.layers](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.layers%28%29) - 返回定义的图层列表（唯一），从下往上排序。  
[pygame.sprite.LayeredUpdates.change\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.change_layer%28%29) - 更改精灵层  
[pygame.sprite.LayeredUpdates.get\_layer\_of\_sprite](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_layer_of_sprite%28%29) - 返回sprite当前所在的图层。  
[pygame.sprite.LayeredUpdates.get\_top\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_top_layer%28%29) - 返回顶层  
[pygame.sprite.LayeredUpdates.get\_bottom\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_bottom_layer%28%29) - 返回底层  
[pygame.sprite.LayeredUpdates.move\_to\_front](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.move_to_front%28%29) - 将精灵带到前面  
[pygame.sprite.LayeredUpdates.move\_to\_back](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.move_to_back%28%29) - 将精灵移动到底层  
[pygame.sprite.LayeredUpdates.get\_top\_sprite](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_top_sprite%28%29) - 返回最顶层的精灵  
[pygame.sprite.LayeredUpdates.get\_sprites\_from\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.get_sprites_from_layer%28%29) - 返回图层中的所有精灵，按其添加位置排序  
[pygame.sprite.LayeredUpdates.switch\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates.switch_layer%28%29) - 将sprite从layer1切换到layer2

该组与可见游戏对象的pygame.sprite.SpriteSimple基类完全兼容。  
您可以使用’default\_layer’通过kwargs设置默认图层，并为图层设置整数。 默认图层为0。  
如果您添加的精灵具有属性图层，则将使用该图层。 如果\*\* kwarg包含’layer’，那么传递的sprite将被添加到该层（覆盖sprite.layer属性）。 如果sprite既没有属性层也没有\*\* kwarg，那么默认层用于添加精灵。  
pygame 1.8.0中的新功能

pygame.sprite.LayeredUpdates.add() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

向一个组添加sprite或sprite序列  
add(*sprites, \*\*kwargs) -> None*  
如果精灵有一个属性层，那么用于该层。 如果\* kwargs包含’layer’，则sprite将被添加到该参数（覆盖sprite layer属性）。 如果两者都没有传递，那么精灵将被添加到默认层。  
[搜索pygame.sprite.LayeredUpdates.add的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.add&type=Code)

pygame.sprite.LayeredUpdates.sprites() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回精灵的有序列表（第一个底部，最后一个顶部）。  
sprites() -> sprites  
[搜索pygame.sprite.LayeredUpdates.sprites的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.sprites&type=Code)

pygame.sprite.LayeredUpdates.draw() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

以正确的顺序将所有精灵绘制到传递的surface上。  
draw(surface) -> Rect\_list  
[搜索pygame.sprite.LayeredUpdates.draw的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.draw&type=Code)

pygame.sprite.LayeredUpdates.get\_sprites\_at() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回包含该位置所有精灵的列表。  
get\_sprites\_at(pos) -> colliding\_sprites  
底部精灵优先，顶部精灵最后。  
[搜索pygame.sprite.LayeredUpdates.get\_sprites\_at的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_sprites_at&type=Code)

pygame.sprite.LayeredUpdates.get\_sprite() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

从组sprites中返回索引idx处的sprite  
get\_sprite(idx) -> sprite  
如果idx不在范围内，则引发IndexOutOfBounds异常。  
[搜索pygame.sprite.LayeredUpdates.get\_sprite的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_sprite&type=Code)

pygame.sprite.LayeredUpdates.remove\_sprites\_of\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

从图层中删除所有精灵并将其作为列表返回。  
remove\_sprites\_of\_layer(layer\_nr) -> sprites  
[搜索pygame.sprite.LayeredUpdates.remove\_sprites\_of\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.remove_sprites_of_layer&type=Code)

pygame.sprite.LayeredUpdates.layers() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回定义的图层列表（唯一），从下往上排序。  
layers() -> layers  
[搜索pygame.sprite.LayeredUpdates.layers的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.layers&type=Code)

pygame.sprite.LayeredUpdates.change\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

更改精灵层  
change\_layer(sprite, new\_layer) -> None  
精灵必须已添加到渲染器中。 它没有被检查。  
[搜索pygame.sprite.LayeredUpdates.change\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.change_layer&type=Code)

pygame.sprite.LayeredUpdates.get\_layer\_of\_sprite() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回sprite当前所在的图层。  
get\_layer\_of\_sprite(sprite) -> layer  
如果未找到精灵，则它将返回默认图层。  
[搜索pygame.sprite.LayeredUpdates.get\_layer\_of\_sprite的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_layer_of_sprite&type=Code)

pygame.sprite.LayeredUpdates.get\_top\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回顶层  
get\_top\_layer() -> layer  
[搜索pygame.sprite.LayeredUpdates.get\_top\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_top_layer&type=Code)

pygame.sprite.LayeredUpdates.get\_bottom\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回底层  
get\_bottom\_layer() -> layer  
[搜索pygame.sprite.LayeredUpdates.get\_bottom\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_bottom_layer&type=Code)

pygame.sprite.LayeredUpdates.move\_to\_front() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

将精灵带到前面  
move\_to\_front(sprite) -> None  
将精灵带到前面，将精灵层更改为最顶层（在该层的末尾添加）。  
[搜索pygame.sprite.LayeredUpdates.move\_to\_front的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.move_to_front&type=Code)

pygame.sprite.LayeredUpdates.move\_to\_back() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

将精灵移动到底层  
move\_to\_back(sprite) -> None  
将精灵移动到底层，将其移动到所有其他图层后面并添加一个额外的图层。  
[搜索pygame.sprite.LayeredUpdates.move\_to\_back的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.move_to_back&type=Code)

pygame.sprite.LayeredUpdates.get\_top\_sprite() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回最顶层的精灵  
get\_top\_sprite() -> Sprite  
[搜索pygame.sprite.LayeredUpdates.get\_top\_sprite的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_top_sprite&type=Code)

pygame.sprite.LayeredUpdates.get\_sprites\_from\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

返回图层中的所有精灵，按其添加位置排序  
get\_sprites\_from\_layer(layer) -> sprites  
返回图层中的所有精灵，按其添加位置排序。 它使用线性搜索，并且不会从图层中删除精灵。  
[搜索pygame.sprite.LayeredUpdates.get\_sprites\_from\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.get_sprites_from_layer&type=Code)

pygame.sprite.LayeredUpdates.switch\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredUpdates)

将sprite从layer1切换到layer2  
switch\_layer(layer1\_nr, layer2\_nr) -> None  
图层编号必须存在，不进行检查。  
[搜索pygame.sprite.LayeredUpdates.switch\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredUpdates.switch_layer&type=Code)

pygame.sprite.LayeredDirty [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

用于DirtySprite对象的LayeredDirty组。子类（分层更新）LayeredUpdates。  
LayeredDirty(\*spites, \*\*kwargs) -> LayeredDirty  
[pygame.sprite.LayeredDirty.draw](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.draw%28%29) - 以正确的顺序将所有精灵绘制到传递的surface上。  
[pygame.sprite.LayeredDirty.clear](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.clear%28%29) - 用于设置背景  
[pygame.sprite.LayeredDirty.repaint\_rect](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.repaint_rect%28%29) - 重新绘制给定区域  
[pygame.sprite.LayeredDirty.set\_clip](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.set_clip%28%29) - 剪裁绘制区域。 只需传递None（默认）即可重置剪辑  
[pygame.sprite.LayeredDirty.get\_clip](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.get_clip%28%29) - 剪裁绘制区域。 只需传递None（默认）即可重置剪辑  
[pygame.sprite.LayeredDirty.change\_layer](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.change_layer%28%29) - 更改精灵层  
[pygame.sprite.LayeredDirty.set\_timing\_treshold](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty.set_timing_treshold%28%29) - 以毫秒为单位设置阈值

该组需要具有pygame.sprite.DirtySprite 或任何具有以下属性的精灵：

image, rect, dirty, visible, blendmode (看DirtySprite的文档).

它使用dirty标志技术，因此比跟踪dirty更新的pygame.sprite.RenderUpdatesGroup子类更快。 如果你有很多静态精灵。 它还会在dirty的rect更新和全屏绘制之间自动切换，因此您不必担心会更快。  
与pygame.sprite.Group类相同。您可以通过kwargs指定一些其他属性：

\_use\_update: True/False default is False

\_default\_layer: default layer where sprites without a layer are added.

\_time\_threshold: threshold time for switching between dirty rect mode

and fullscreen mode, defaults to 1000./80 == 1000./fps

\_use\_update：True / False 默认值为False

\_default\_layer：默认图层，其中添加了没有图层的精灵。

\_time\_threshold：在dirty rect 模式之间切换的阈值时间

和全屏模式，默认为1000./80 == 1000./fps

pygame 1.8.0的新特性

pygame.sprite.LayeredDirty.draw() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

以正确的顺序将所有精灵绘制到传递的surface上。  
draw(surface, bgd=None) -> Rect\_list  
你也可以传递背景。 如果已设置背景，则bgd参数无效。  
[搜索pygame.sprite.LayeredDirty.draw的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.draw&type=Code)

pygame.sprite.LayeredDirty.clear() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

用于设置背景  
clear(surface, bgd) -> None  
[搜索pygame.sprite.LayeredDirty.clear的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.clear&type=Code)

pygame.sprite.LayeredDirty.repaint\_rect() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

重新绘制给定区域  
repaint\_rect(screen\_rect) -> None  
screen\_rect在屏幕坐标中。  
[搜索pygame.sprite.LayeredDirty.repaint\_rect的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.repaint_rect&type=Code)

pygame.sprite.LayeredDirty.set\_clip() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

剪裁绘制区域。 只需传递None（默认）即可重置剪辑  
set\_clip(screen\_rect=None) -> None  
[搜索pygame.sprite.LayeredDirty.set\_clip的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.set_clip&type=Code)

pygame.sprite.LayeredDirty.get\_clip() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

剪裁绘制区域。 只需传递None（默认）即可重置剪辑  
get\_clip() -> Rect  
[搜索pygame.sprite.LayeredDirty.get\_clip的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.get_clip&type=Code)

pygame.sprite.LayeredDirty.change\_layer() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

更改精灵层  
change\_layer(sprite, new\_layer) -> None  
精灵必须已添加到渲染器中。 它没有被检查。  
[搜索pygame.sprite.LayeredDirty.change\_layer的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.change_layer&type=Code)

pygame.sprite.LayeredDirty.set\_timing\_treshold() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#LayeredDirty)

以毫秒为单位设置阈值  
set\_timing\_treshold(time\_ms) -> None  
默认值为1000./80，其中80是我要切换到全屏模式的fps。 此方法的名称是拼写错误，应该修复。  
[搜索pygame.sprite.LayeredDirty.set\_timing\_treshold的示例](https://github.com/search?q=pygame.sprite.LayeredDirty.set_timing_treshold&type=Code)

pygame.sprite.GroupSingle() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

包含单个sprite的组容器。  
GroupSingle(sprite=None) -> GroupSingle  
GroupSingle容器只包含一个Sprite。 添加新Sprite后，将删除旧Sprite。  
有一个特殊属性GroupSingle.sprite，用于访问该组包含的Sprite。 当组为空时，它可以是None。 还可以指定该属性以将Sprite添加到GroupSingle容器中。  
[搜索pygame.sprite.GroupSingle的示例](https://github.com/search?q=pygame.sprite.GroupSingle&type=Code)

pygame.sprite.spritecollide() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

在与另一个精灵相交的组中查找精灵。  
spritecollide(sprite, group, dokill, collided = None) -> Sprite\_list  
返回一个列表，其中包含与另一个Sprite相交的Group中的所有Sprite。 通过比较每个Sprite的Sprite.rect属性来确定交集。  
dokill参数是一个布尔值。 如果设置为True，则将从组中删除所有碰撞的Sprite。  
碰撞参数是一个回调函数，用于计算两个精灵是否发生碰撞。 它应该将两个精灵作为值，并返回一个bool值，指示它们是否发生碰撞。 如果未传递碰撞，则所有精灵必须具有“rect”值，该值是sprite区域的矩形，将用于计算碰撞。

collided callabled：

collide\_rect, collide\_rect\_ratio, collide\_circle,

collide\_circle\_ratio, collide\_mask

#查看Sprite块是否与Group block\_list中的任何内容发生冲突

#True标志将删除block\_list中的精灵

blocks\_hit\_list = pygame.sprite.spritecollide(player, block\_list, True)

#检查碰撞精灵列表，并为每个精灵添加一个score

for block in blocks\_hit\_list:

score +=1

[搜索pygame.sprite.spritecollide的示例](https://github.com/search?q=pygame.sprite.spritecollide&type=Code)

pygame.sprite.collide\_rect() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

使用rects检测两个精灵之间的碰撞。  
collide\_rect(left, right) -> bool  
测试两个精灵之间的碰撞。 使用pygame rect colliderect函数计算碰撞。 作为碰撞回调函数传递给\*collide函数。 精灵必须具有“rect”属性。  
pygame 1.8.0中的新功能  
[搜索pygame.sprite.collide\_rect的示例](https://github.com/search?q=pygame.sprite.collide_rect&type=Code)

pygame.sprite.collide\_rect\_ratio() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

使用按比例缩放的rects检测两个精灵之间的碰撞。  
collide\_rect\_ratio(ratio) -> collided\_callable  
一个可调用的类，使用sprites 的 rects的缩放版本来检查两个sprite之间的冲突。  
使用ratio创建，然后将实例作为碰撞回调函数传递给\*collide函数。  
ratio是浮点数 - 1.0是相同的大小，2.0是两倍大，0.5是大小的一半。  
pygame 1.8.1中的新功能  
[搜索pygame.sprite.collide\_rect\_ratio的示例](https://github.com/search?q=pygame.sprite.collide_rect_ratio&type=Code)

pygame.sprite.collide\_circle() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

使用圆圈检测两个精灵之间的碰撞。  
collide\_circle(left, right) -> bool  
测试两个精灵之间的碰撞，通过测试以查看精灵中心的两个圆是否重叠。 如果精灵具有“radius（半径）”属性，用于创建圆，否则会创建一个足够大的圆，以完全包围由“rect”属性给出的精灵矩形。 作为碰撞回调函数传递给\*collide函数。 精灵必须具有“rect”和可选的“radius”属性。  
pygame 1.8.1中的新功能  
[搜索pygame.sprite.collide\_circle的示例](https://github.com/search?q=pygame.sprite.collide_circle&type=Code)

pygame.sprite.collide\_circle\_ratio() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

使用按比例缩放的圆圈检测两个精灵之间的碰撞。  
collide\_circle\_ratio(ratio) -> collided\_callabl  
一个可调用类，使用精灵半径的缩放版本检查两个精灵之间的碰撞。  
使用浮点数ratio创建，然后将实例作为碰撞回调函数传递给\*collide函数。  
ratio是浮点数 - 1.0是相同的大小，2.0是两倍大，0.5是大小的一半。  
两个精灵之间的碰撞创建的可调用测试，通过测试以查看以精灵为中心的两个圆是否重叠，在通过存储的比例缩放圆半径之后。 如果精灵具有“radius半径”属性，用于创建圆，否则会创建一个足够大的圆，以完全包围由“rect”属性给出的精灵矩形。 打算作为碰撞回调函数传递给\*collide函数。 精灵必须具有“rect”和可选的“radius”属性。  
pygame 1.8.1中的新功能  
[搜索pygame.sprite.collide\_circle\_ratio的示例](https://github.com/search?q=pygame.sprite.collide_circle_ratio&type=Code)

pygame.sprite.collide\_mask() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

使用蒙版在两个精灵之间进行碰撞检测。  
collide\_mask(SpriteLeft, SpriteRight) -> point  
返回masks碰撞的mask上的第一个点，如果没有碰撞，则返回None。  
通过测试它们的bitmasks是否重叠来测试两个精灵之间的碰撞。 如果精灵具有“mask”属性，该属性用作mask，否则将从精灵图像创建mask。 作为碰撞回调函数传递给\*collide函数。 精灵必须具有“rect”和可选的“mask”属性。  
如果要多次检查碰撞，应该考虑在加载时为精灵创建一个遮罩。 这将提高性能，否则这可能是一个昂贵的功能，因为它会在每次检查碰撞时创建mask。  
sprite.mask = pygame.mask.from\_surface(sprite.image)  
pygame 1.8.0中的新功能  
[搜索pygame.sprite.collide\_mask的示例](https://github.com/search?q=pygame.sprite.collide_mask&type=Code)

pygame.sprite.groupcollide() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

查找在两个组之间发生碰撞的所有精灵。  
groupcollide(group1, group2, dokill1, dokill2, collided = None) -> Sprite\_dict  
这将在两组中找到所有精灵之间的碰撞。 通过比较每个Sprite的Sprite.rect属性或使用碰撞函数（如果它不是None）来确定碰撞。  
group1中的每个Sprite都被添加到返回字典中。 每个项的值是group2中相交的Sprite列表。  
如果dokill参数为True，则将从各自的组中删除碰撞的Sprite。  
碰撞参数是一个回调函数，用于计算两个精灵是否发生碰撞。 它应该将两个精灵作为值并返回一个bool值，指示它们是否发生碰撞。 如果未传递碰撞，则所有精灵必须具有“rect”值，该值是精灵区域的矩形，将用于计算碰撞。  
[搜索pygame.sprite.groupcollide的示例](https://github.com/search?q=pygame.sprite.groupcollide&type=Code)

pygame.sprite.spritecollideany() [↶](https://blog.csdn.net/Enderman_xiaohei/article/details/88218773#title)

如果精灵与组中的任何内容相交，则进行简单测试。  
spritecollideany(sprite, group, collided = None) -> Sprite 与返回的精灵碰撞。  
spritecollideany(sprite, group, collided = None) -> None 没有碰撞  
如果精灵与组中的任何一个精灵发生碰撞，则返回该组中的一个精灵。 无碰撞时返回None。  
如果您不需要pygame.sprite.spritecollide()函数的所有功能，则此函数会更快一些。  
碰撞参数是一个回调函数，用于计算两个精灵是否发生碰撞。 它应该将两个精灵作为值并返回一个bool值，指示它们是否发生碰撞。 如果未传递碰撞，则所有精灵必须具有“rect”值，该值是精灵区域的矩形，将用于计算碰撞。  
[搜索pygame.sprite.spritecollideany的示例](https://github.com/search?q=pygame.sprite.spritecollideany&type=Code)

以上文档，自己翻译，可能有误，可参考：[pygame.sprite](https://www.pygame.org/docs/ref/sprite.html)