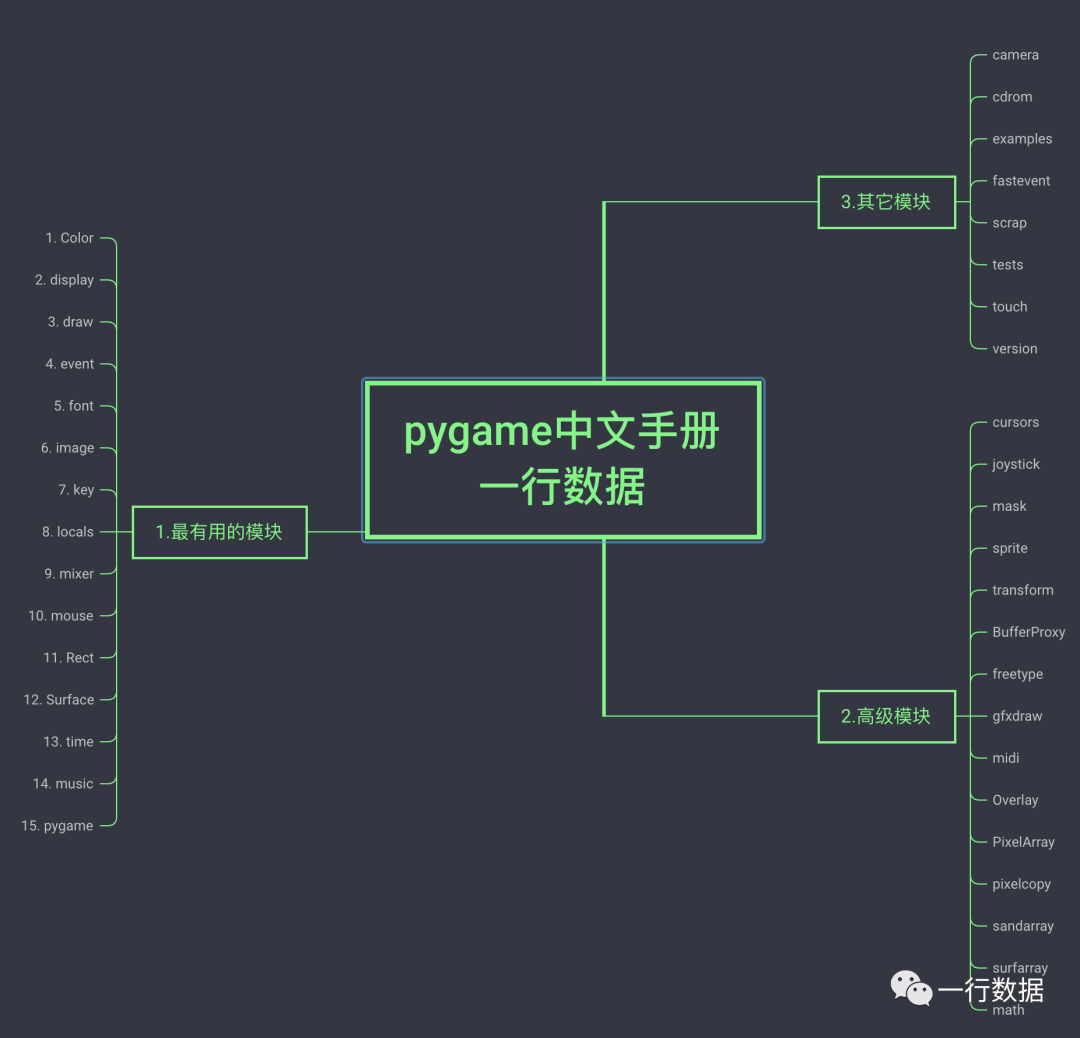
Python游戏工具包---Pygame最常用的15个模块详解（附pdf版本）

最近很多读者想学习使用python来制作游戏，其中最经典的工具包就是Pygame ，如果单纯只看之前分享的是个游戏代码案例，直接来制作一款游戏有些难度



python制作十款游戏案例

所以今天行哥对pygame工具包进行了总结，并分享了pygame工具包最常用15个模块的详细方法，如果想获取pdf版本可见文末链接



pygame中文手册pdf版本：https://yihang.cowtransfer.com/s/7b027011c9cd4a

**1.Color类**

**class pygame.Color**

Pygame 中用于描述颜色的对象。Color(name) -> Color Color(r, g, b, a) -> Color Color(rgbvalue) -> Color

方法 & 属性

* pygame.Color.r — 获取或设置 Color 对象的红色值
* pygame.Color.g — 获取或设置 Color 对象的绿色值
* pygame.Color.b — 获取或设置 Color 对象的蓝色值
* pygame.Color.a — 获取或设置 Color 对象的 alpha 值
* pygame.Color.cmy — 获取或设置 Color 对象表示的 CMY 值
* pygame.Color.hsva — 获取或设置 Color 对象表示的 HSVA 值
* pygame.Color.hsla — 获取或设置 Color 对象表示的 HSLA 值
* pygame.Color.i1i2i3 — 获取或设置 Color 对象表示的 I1I2I3 值
* pygame.Color.normalize() — 返回 Color 对象的标准化 RGBA 值
* pygame.Color.correct\_gamma() — 应用一定的伽马值调整 Color 对象
* pygame.Color.set\_length() — 设置 Color 对象的长度（成员数量）

Pygame 使用 Color 类表示 RGBA 颜色值，每个颜色值的取值范围是 0 ~ 255。允许通过基本的算术运算创造新的颜色值，支持转换为其他颜色空间，例如 HSV 或 HSL，并让你调整单个颜色通道。当没有给出 alpha 的值默认是 255（不透明）。

“RGB值”可以是一个颜色名，一个 HTML 颜色格式的字符串，一个 16 进制数的字符串，或者一个整型像素值。HTML 格式是 "#rrggbbaa"，其中 "rr"，"gg"，"bb"，"aa" 都是 2 位的 16 进制数。代表 alpha 的 "aa" 是可选的。16 进制数的字符串组成形式为 "0xrrggbbaa"，当然，其中的 "aa" 也是可选的。

**2.display模块**

**pygame.display**

Pygame 中用于控制窗口和屏幕显示的模块。

注：为了适应语境，display 在该文档中有时翻译为“显示”，有时翻译为“显示界面”。

**函数**

* pygame.display.init() — 初始化 display 模块
* pygame.display.quit() — 结束 display 模块
* pygame.display.get\_init() — 如果 display 模块已经初始化，返回 True
* pygame.display.set\_mode() — 初始化一个准备显示的窗口或屏幕
* pygame.display.get\_surface() — 获取当前显示的 Surface 对象
* pygame.display.flip() — 更新整个待显示的 Surface 对象到屏幕上
* pygame.display.update() — 更新部分软件界面显示
* pygame.display.get\_driver() — 获取 Pygame 显示后端的名字
* pygame.display.Info() — 创建有关显示界面的信息对象
* pygame.display.get\_wm\_info() — 获取关于当前窗口系统的信息
* pygame.display.list\_modes() — 获取全屏模式下可使用的分辨率
* pygame.display.mode\_ok() — 为显示模式选择最合适的颜色深度
* pygame.display.gl\_get\_attribute() — 获取当前显示界面 OpenGL 的属性值
* pygame.display.gl\_set\_attribute() — 设置当前显示模式的 OpenGL 属性值
* pygame.display.get\_active() — 当前显示界面显示在屏幕上时返回 True
* pygame.display.iconify() — 最小化显示的 Surface 对象
* pygame.display.toggle\_fullscreen() — 切换全屏模式和窗口模式
* pygame.display.set\_gamma() — 修改硬件显示的 gama 坡道
* pygame.display.set\_gamma\_ramp() — 自定义修改硬件显示的 gama 坡道
* pygame.display.set\_icon() — 修改显示窗口的图标
* pygame.display.set\_caption() — Set the current window caption
* pygame.display.get\_caption() — Get the current window caption
* pygame.display.set\_palette() — Set the display color palette for indexed displays 这个模块提供控制 Pygame 显示界面（display）的各种函数。Pygame 的 Surface 对象即可显示为一个窗口，也可以全屏模式显示。当你创建并显示一个常规的 Surface 对象后，在该对象上的改变并不会立刻反映到可见屏幕上，你必须选择一个翻转函数来显示改动后的画面。

显示的原点是 (x=0, y=0) 的位置，及屏幕的左上角，坐标轴向右下角增长。

Pygame 的 display 事实上可以有几种初始化的方式。默认情况下，display 作为一个软件驱动的帧缓冲区。除此之外，你可以使用硬件加速和 OpenGL 支持的特殊模块。这些是通过给 pygame.display.set\_mode() 传入 flags 参数来控制的。

Pygame 在任何时间内都只允许有一个显示界面。使用 pygame.display.set\_mode() 创建的新显示界面会自动替换掉旧的。如果需要精确控制像素格式或显示分辨率，使用 pygame.display.mode\_ok()，pygame.display.list\_modes()，和 pygame.display.Info() 来查询显示界面相关的信息。

一旦 Surface 对象的显示界面被创建出来，这个模块的函数就只影响当前的显示界面。如果该模块未初始化， Surface 对象也会变为“非法”。如果新的显示模式被设置，当前的 Surface 对象将会自动切换到新的显示界面。

当一个新的显示模式被设置，会在 Pygame 的事件队列中放入几个相关事件。当用于希望关闭程序时，pygame.QUIT 事件会被发送；当显示界面获得和失去焦点时，窗口会得到 pygame.ACTIVEEVENT 事件；如果显示界面设置了 pygame.RESIZABLE 标志，那么当用户调整窗口尺寸时，pygame.VIDEORESIZE 事件会被发送；硬件显示指当接收到 pygame.VIDEOEXPOSE 事件时，将部分需要被重绘的窗口直接绘制到屏幕上。

一些显示环境拥有自动拉伸所有窗口的选项。当该选项被启动时，自动拉伸会扭曲 Pygame 窗口的外观。在 Pygame 的例子目录中，有一个演示代码（prevent\_display\_stretching.py）展示如何在微软系统（Vista 以上系统）中关闭 Pygame 显示的自动拉伸属性。

**函数详解pygame.display.init()**初始化 display 模块。

init() -> None

初始化 Pygame 的 display 模块。在初始化之前，display 模块无法做任何事情。但当你调用更高级别的 pygame.init()，变会自动调用 pygame.display.init() 进行初始化。

初始化后，Pygame 将自动从几个内部的显示后端中选择一个。显示模式由平台和当前用户权限决定。在 display 模块被初始化之前，可以通过环境变量 SDL\_VIDEODRIVER 设置哪一个显示后端将被使用。具有多种显示后端的系统如下：

Windows : windib, directx

Unix : x11, dga, fbcon, directfb, ggi, vgl, svgalib, aalib

复制

在一些平台上，可以将 Pygame 的 display 嵌入到已经存在的窗口中。如果这么做，环境变量 SDL\_WINDOWID 必须被设置为一个包含窗口 ID 或句柄的字符串。当 Pygame 的 display 被初始化的时候，将检测环境变量。注意，在一个运行的窗口嵌入 display 会产生许多奇怪的副作用。

多次调用该函数并没有任何问题，但也不会有什么效果。

**pygame.display.quit()**

结束 display 模块。

quit() -> None

这个函数会关闭整个 display 模块。这将意味着任何一个活跃的显示界面都将被关闭。当主程序退出时，该函数也会被自动调用。

多次调用该函数并没有任何问题，但也不会有什么效果。

**pygame.display.get\_init()**

如果 display 模块已经初始化，返回 True。

get\_init() -> bool

如果 display 模块已经初始化，返回 True。

**pygame.display.set\_mode()**初始化一个准备显示的窗口或屏幕。

set\_mode(resolution=(0,0), flags=0, depth=0) -> Surface

这个函数将创建一个 Surface 对象的显示界面。传入的参数用于指定显示类型。最终创建出来的显示界面将是最大可能地匹配当前操作系统。

resolution 参数是一个二元组，表示宽和高。flags 参数是附件选项的集合。depth 参数表示使用的颜色深度。

返回的 Surface 对象可以像常规的 Surface 对象那样去绘制，但发生的改变最终会显示到屏幕上。

如果没有传入 resolution 参数，或者使用默认设置 (0, 0)，且 Pygame 使用 SDL1.2.10 以上版本，那么创建出来的 Surface 对象将与当前屏幕用户一样的分辨率。如果只有宽或高其中一项被设置为 0，那么 Surface 对象将使用屏幕分辨率的宽或高代替它。如果 SDL 版本低于 1.2.10，那么将抛出异常。

通常来说，最好的做法是不要传递 depth 参数。因为默认 Pygame 会根据当前操作系统选择最好和最快的颜色深度。如果你的游戏确实需要一个特殊的颜色格式，那么你可以通过控制 depth 参数来实现。Pygame 将为模拟一个非现成的颜色深度而耗费更多的时间。

当使用全屏显示模式的时候，有时候无法完全匹配到需要的分辨率。在这种情况下，Pygame 将自动选择最匹配的分辨率使用，而返回的 Surface 对象将保持与需求的分辨率一致。

flags 参数指定你想要的显示类型。提供几个选择给你，你可以通过位操作同时使用多个类型（管道操作符 "|"）。如果你传入 0 或没有传入 flags 参数，默认会使用软件驱动窗口。这儿是 flags 参数提供的几个可选项：

| 选项 | 含义 |
| --- | --- |
| pygame.FULLSCREEN | 创建一个全屏显示 |
| pygame.DOUBLEBUF | 1. 双缓冲模式 2. 推荐和 HWSURFACE 或 OPENGL 一起使用 |
| pygame.HWSURFACE | 硬件加速，只有在 FULLSCREEN 下可以使用 |
| pygame.OPENGL | 创建一个 OPENGL 渲染的显示 |
| pygame.RESIZABLE | 创建一个可调整尺寸的窗口 |
| pygame.NOFRAME | 创建一个没有边框和控制按钮的窗口 |

举个例子：

# 在屏幕中创建一个 700 \* 400 的窗口

screen\_width=700

screen\_height=400

screen=pygame.display.set\_mode([screen\_width, screen\_height])

复制

**pygame.display.get\_surface()**获取当前显示的 Surface 对象。

get\_surface() -> Surface

返回当前显示的 Surface 对象。如果没有设置任何显示模式，那么返回 None。

**pygame.display.flip()**更新整个待显示的 Surface 对象到屏幕上。

flip() -> None

这个函数将更新整个显示界面的内容。如果你的显示模式使用了 pygame.HWSURFACE（硬件加速）和 pygame.DOUBLEBUF（双缓冲）标志，那么将等待垂直会扫并切换显示界面。如果你使用不同类型的显示模式，那么它将简单的更新整个显示界面的内容。

当使用 pygame.OPENGL（使用 OPENGL 渲染）显示模式时，将创建一个 gl 缓冲切换区。

温馨提示：垂直回扫是与视频显示相关的时间测量，它代表了一个帧的结束和下一帧的开始时间之间的时间间隔。

**pygame.display.update()**更新部分软件界面显示。

update(rectangle=None) -> None

update(rectangle\_list) -> None

这个函数可以看作是 pygame.display.flip() 函数在软件界面显示的优化版。它允许更新屏幕的部分内容，而不必完全更新。如果没有传入任何参数，那么该函数就像 pygame.display.flip() 那样更新整个界面。

你可以传递一个或多个矩形区域给该函数。一次性传递多个矩形区域比多次传递更有效率。如果传入的是一个空列表或者 None，那么将忽略参数。

该函数不能在 pygame.OPENGL 显示模式下调用，否则会抛出异常。

**pygame.display.get\_driver()**获取 Pygame 显示后端的名字。

get\_driver() -> name

初始化的时候，Pygame 会从多个可用的显示后端中选择一个。这个函数返回显示后端内部使用的名字。可以用来提供有关显示性能加速的一些信息。可以参考 pygame.display.set\_mode() 的 SDL\_VIDEODRIVER 环境变量。

**pygame.display.Info()**创建有关显示界面的信息对象。

Info() -> VideoInfo

创建一个对象，包含对当前图形环境一些属性的描述。在一些平台上，如果这个函数在 pygame.display.set\_mode() 前被调用，可以提供一些关于默认显示模式的信息。也可以在设置完显示模式后调用该函数，以确认显示选项是否如愿以偿。

返回的 VideoInfo 对象包含以下这些属性：

| 属性 | 含义 |
| --- | --- |
| hw | 如果是 True，则表示启用硬件加速 |
| wm | 如果是 True，则表示显示窗口模式 |
| video\_mem | 表示显存是多少兆字节（mb），0 表示不清楚 |
| bitsize | 表示每个像素存放多少位 |
| bytesize | 表示每个像素存放多少字节 |
| masks | 4 个值用于打包像素的 RGBA 值 |
| shifts | 4 个值用于打包像素的 RGBA 值 |
| losses | 4 个值用于打包像素的 RGBA 值 |
| blit\_hw | 如果是 True，则表示加速硬件驱动的 Surface 对象绘制 |
| blit\_hw\_CC | 如果是 True，则表示加速硬件驱动的 Surface 对象 colorkey 绘制 |
| blit\_hw\_A | 如果是 True，则表示加速硬件驱动的 Surface 对象 pixel alpha 绘制 |
| blit\_sw | 如果是 True，则表示加速软件驱动的 Surface 对象绘制 |
| blit\_sw\_CC | 如果是 True，则表示加速软件驱动的 Surface 对象 colorkey 绘制 |
| blit\_sw\_A | 如果是 True，则表示加速软件驱动的Surface 对象 pixel alpha 绘制 |
| current\_w, current\_h | 1. 表示当前显示模式的宽和高（如果在 display.set\_mode() 前被调用，则表示当前桌面的宽和高）2. current\_w, current\_h 在 Pygame 1.8.0 以后，SDL 1.2.10 以后才支持3. -1 表示错误，或者 SDL 版本太旧 |

**pygame.display.get\_wm\_info()**获取关于当前窗口系统的信息。

get\_wm\_info() -> dict

创建一个由操作系统填充数据的字典。一些操作系统可能不会往里边填充信息，则返回一个空字典。大多数平台将返回一个 "window" 键，对应的值是当前显示界面的系统 ID。

Pygame 1.7.1 新增加的。

pygame.display.list\_modes() 获取全屏模式下可使用的分辨率。

list\_modes(depth=0, flags=pygame.FULLSCREEN) -> list

这个函数返回一个列表，包含指定颜色深度所支持的所有分辨率。如果显示模式非全屏，则返回一个空列表。如果返回 -1 表示支持任何分辨率（类似于窗口模式）。返回的列表由大到小排列。

如果颜色深度是 0，SDL 将选择当前/最合适的颜色深度显示。flags 参数默认值是 pygame.FULLSCREEN，但你可能需要添加额外的全屏模式标志。

**pygame.display.mode\_ok()**为显示模式选择最合适的颜色深度。

mode\_ok(size, flags=0, depth=0) -> depth

这个函数使用与 pygame.display.set\_mode() 函数一样的参数。一般用于判断一个显示模式是否可用。如果显示模式无法设置，则返回 0。正常情况下将会返回显示需求的像素深度。

通常不用理会 depth 参数，除非一些支持多个显示深度的平台，它会提示哪个颜色深度是更合适的。

最有用的 flags 参数是 pygame.HWSURFACE，pygame.DOUBLEBUF 和 pygame.FULLSCREEN。如果这些标志不支持，那么该函数会返回 0。

**pygame.display.gl\_get\_attribute()**获取当前显示界面 OpenGL 的属性值。

gl\_get\_attribute(flag) -> value

在调用设置了 pygame.OPENGL 标志的 pygame.display.set\_mode() 函数之后，检查 OpenGL 的属性值不失为一个好的习惯。参考 pygame.display.gl\_set\_attribute() 关于合法标志的列表。

**pygame.display.gl\_set\_attribute()**设置当前显示模式的 OpenGL 属性值。

gl\_set\_attribute(flag, value) -> None

当调用设置了 pygame.OPENGL 标志的 pygame.display.set\_mode() 函数时，Pygame 会自动设置 OpenGL 的一些属性值，例如颜色和双缓冲区。OpenGL 其实还提供了其他一些属性值供你控制。在 flag 参数中传入属性名，并将其值设置在 value 参数中。这个函数必须在 pygame.display.set\_mode() 前设置。

这些 OPENGL 标志是：

GL\_ALPHA\_SIZE, GL\_DEPTH\_SIZE, GL\_STENCIL\_SIZE, GL\_ACCUM\_RED\_SIZE, GL\_ACCUM\_GREEN\_SIZE, GL\_ACCUM\_BLUE\_SIZE, GL\_ACCUM\_ALPHA\_SIZE, GL\_MULTISAMPLEBUFFERS, GL\_MULTISAMPLESAMPLES, GL\_STEREO

**pygame.display.get\_active()**当前显示界面显示在屏幕上时返回 True。

get\_active() -> bool

pygame.display.set\_mode() 函数被调用之后，Surface 对象将被显示在屏幕上。大多数窗口都支持隐藏，如果显示的 Surface 对象被隐藏和最小化，那么该函数将返回 False。

**pygame.display.iconify()**最小化显示的 Surface 对象。

iconify() -> bool

将显示的Surface 对象最小化或隐藏。并不是所有的操作系统都支持最小化显示界面。如果该函数调用成功，返回 True。

当显示界面最小化时，pygame.display.get\_active() 返回 False。事件队列将接收到 ACTIVEEVENT 事件。

pygame.display.toggle\_fullscreen() 切换全屏模式和窗口模式。

toggle\_fullscreen() -> bool

切换全屏模式和窗口模式。这个函数只在 unix x11 显示驱动下工作。在大多数情况下，建议调用 pygame.display.set\_mode() 创建一个新的显示模式进行切换。

**pygame.display.set\_gamma()**修改硬件显示的 gama 坡道。

set\_gamma(red, green=None, blue=None) -> bool

设置硬件驱动显示的红色、绿色和蓝色伽马值。如果没有传递 green 和 blue 参数，它们将与 red 值相等。不是所有的操作系统和硬件都支持伽马坡道。如果函数修改成功，则返回 True。

伽马值为 1.0 创建一个线性颜色表，较低的值会使屏幕变暗，较高的值会使屏幕变量。

**pygame.display.set\_gamma\_ramp()**自定义修改硬件显示的 gama 坡道

set\_gamma\_ramp(red, green, blue) -> bool

使用自定义表设置硬件驱动显示的红色、绿色和蓝色伽马坡道。每个参数必须是 256 位整数的列表。每位整数应该在 0 和 0xffff 之间。不是所有的操作系统和硬件都支持伽马坡道。如果函数修改成功，则返回 True。

**pygame.display.set\_icon()**修改显示窗口的图标。

set\_icon(Surface) -> None

设置显示窗口执行时的图标。所有的操作系统默认都是以简单的 Pygame LOGO 作为图标。

你可以传入任何 Surface 对象作为图标，但大多数操作系统要求图标的大小是 32 \* 32。图标可以设置 colorkey 透明度。

一些操作系统不允许修改显示中的窗口图标。对于这类操作系统，该函数需要再调用 pygame.display.set\_mode() 前先创建并设置图标。

**pygame.display.set\_caption()**设置当前窗口的标题栏。

set\_caption(title, icontitle=None) -> None

如果显示窗口拥有一个标题栏，这个函数将修改窗口标题栏的文本。一些操作系统支持最小化窗口时切换标题栏，通过设置 icontitle 参数实现。

**pygame.display.get\_caption()**获取当前窗口的标题栏。

get\_caption() -> (title, icontitle)

返回当前窗口的标题栏和最小化标题栏，通常这两个值是一样的。

**pygame.display.set\_palette()**设置显示界面的调色板。

set\_palette(palette=None) -> None

这个函数将修改显示界面的 8 位调色板。这不会改变Surface 对象实际的调色板，仅用于 Surface 对象的显示。如果没有传入参数，将恢复系统默认调色板。调色板是一组 RGB 三元组序列。

**3.draw 模块**

pygame.draw Pygame 中绘制图形的模块。

函数

* pygame.draw.rect() — 绘制矩形
* pygame.draw.polygon() — 绘制多边形
* pygame.draw.circle() — 根据圆心和半径绘制圆形
* pygame.draw.ellipse() — 根据限定矩形绘制一个椭圆形
* pygame.draw.arc() — 绘制弧线
* pygame.draw.line() — 绘制线段
* pygame.draw.lines() — 绘制多条连续的线段
* pygame.draw.aaline() — 绘制抗锯齿的线段
* pygame.draw.aalines() — 绘制多条连续的线段（抗锯齿）

该模块用于在 Surface 对象上绘制一些简单的形状。这些函数将渲染到任何格式的 Surface 对象上。硬件渲染会比普通的软件渲染更耗时。

大部分函数用 width 参数指定图形边框的大小，如果 width = 0 则表示填充整个图形。

所有的绘图函数仅能在 Surface 对象的剪切区域生效。这些函数返回一个 Rect，表示包含实际绘制图形的矩形区域。

大部分函数都有一个 color 参数，传入一个表示 RGB 颜色值的三元组，当然也支持 RGBA 四元组。其中的 A 是 Alpha 的意思，用于控制透明度。不过该模块的函数并不会绘制透明度，而是直接传入到对应 Surface 对象的 pixel alphas 中。color 参数也可以是已经映射到 Surface 对象的像素格式中的整型像素值。

当这些函数在绘制时，必须暂时锁定 Surface 对象。许多连续绘制的函数可以通过一次性锁定直到画完再解锁来提高效率。

**函数详解pygame.draw.rect()**绘制矩形。

rect(Surface, color, Rect, width=0) -> Rect

在 Surface 对象上绘制一个矩形。Rect 参数指定矩形的位置和尺寸。width 参数指定边框的宽度，如果设置为 0 则表示填充该矩形。

**pygame.draw.polygon()**绘制多边形。

polygon(Surface, color, pointlist, width=0) -> Rect 在 Surface 对象上绘制一个多边形。pointlist 参数指定多边形的各个顶点。width 参数指定边框的宽度，如果设置为 0 则表示填充该矩形。

绘制一个抗锯齿的多边形，只需要将 aalines() 的 closed 参数设置为 True 即可。

**pygame.draw.circle()**根据圆心和半径绘制圆形。

circle(Surface, color, pos, radius, width=0) -> Rect

在 Surface 对象上绘制一个圆形。pos 参数指定圆心的位置，radius 参数指定圆的半径。width 参数指定边框的宽度，如果设置为 0 则表示填充该矩形。

**pygame.draw.ellipse()**根据限定矩形绘制一个椭圆形。

ellipse(Surface, color, Rect, width=0) -> Rect

在 Surface 对象上绘制一个椭圆形。Rect 参数指定椭圆外围的限定矩形。width 参数指定边框的宽度，如果设置为 0 则表示填充该矩形。

**pygame.draw.arc()**绘制弧线。

arc(Surface, color, Rect, start\_angle, stop\_angle, width=1) -> Rect

在 Surface 对象上绘制一条弧线。Rect 参数指定弧线所在的椭圆外围的限定矩形。两个 angle 参数指定弧线的开始和结束位置。width 参数指定边框的宽度。

**pygame.draw.line()**绘制线段。

line(Surface, color, start\_pos, end\_pos, width=1) -> Rect

在 Surface 对象上绘制一条线段。两端以方形结束。

**pygame.draw.lines()**绘制多条连续的线段。

lines(Surface, color, closed, pointlist, width=1) -> Rect

在 Surface 对象上绘制一系列连续的线段。pointlist 参数是一系列短点。如果 closed 参数设置为 True，则绘制首尾相连。

**pygame.draw.aaline()**绘制抗锯齿的线段。

aaline(Surface, color, startpos, endpos, blend=1) -> Rect

在 Surface 对象上绘制一条抗锯齿的线段。blend 参数指定是否通过绘制混合背景的阴影来实现抗锯齿功能。该函数的结束位置允许使用浮点数。

**pygame.draw.aalines()**绘制多条连续的线段（抗锯齿）。

aalines(Surface, color, closed, pointlist, blend=1) -> Rect

在 Surface 对象上绘制一系列连续的线段（抗锯齿）。如果 closed 参数为 True，则首尾相连。blend 参数指定是否通过绘制混合背景的阴影来实现抗锯齿功能。该函数的结束位置允许使用浮点数。

image

# 公众号：一行数据

import pygame

import sys

import math

from pygame.locals import \*

pygame.init()

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

GREEN = (0, 255, 0)

RED = (255, 0, 0)

BLUE = (0, 0, 255)

points = [(200, 175), (300, 125), (400, 175), (450, 125), (450, 225), (400, 175), (300, 225)]

size = width, height = 640, 1000

screen = pygame.display.set\_mode(size)

pygame.display.set\_caption("Python Demo")

clock = pygame.time.Clock()

while True:

for event in pygame.event.get():

if event.type == QUIT:

sys.exit()

screen.fill(WHITE)

pygame.draw.rect(screen, BLACK, (50, 30, 150, 50), 0)

pygame.draw.rect(screen, BLACK, (250, 30, 150, 50), 1)

pygame.draw.rect(screen, BLACK, (450, 30, 150, 50), 10)

pygame.draw.polygon(screen, GREEN, points, 0)

pygame.draw.circle(screen, RED, (320, 400), 25, 1)

pygame.draw.circle(screen, GREEN, (320, 400), 75, 1)

pygame.draw.circle(screen, BLUE, (320, 400), 125, 1)

pygame.draw.ellipse(screen, BLACK, (100, 600, 440, 100), 1)

pygame.draw.ellipse(screen, BLACK, (220, 550, 200, 200), 1)

pygame.draw.arc(screen, BLACK, (100, 800, 440, 100), 0, math.pi, 1)

pygame.draw.arc(screen, BLACK, (220, 750, 200, 200), math.pi, math.pi \* 2, 1)

pygame.display.flip()

clock.tick(10)

复制

**4.event模块**

**pygame.event**用于处理事件与事件队列的 Pygame 模块。

**函数**

* pygame.event.pump() — 让 Pygame 内部自动处理事件
* pygame.event.get() — 从队列中获取事件
* pygame.event.poll() — 从队列中获取一个事件
* pygame.event.wait() — 等待并从队列中获取一个事件
* pygame.event.peek() — 检测某类型事件是否在队列中
* pygame.event.clear() — 从队列中删除所有的事件
* pygame.event.event\_name() — 通过 id 获得该事件的字符串名字
* pygame.event.set\_blocked() — 控制哪些事件禁止进入队列
* pygame.event.set\_allowed() — 控制哪些事件允许进入队列
* pygame.event.get\_blocked() — 检测某一类型的事件是否被禁止进入队列
* pygame.event.set\_grab() — 控制输入设备与其他应用程序的共享
* pygame.event.get\_grab() — 检测程序是否共享输入设备
* pygame.event.post() — 放置一个新的事件到队列中
* pygame.event.Event() — 创建一个新的事件对象
* pygame.event.EventType — 代表 SDL 事件的 Pygame 对象 Pygame 通过事件队列控制所有的时间消息。该模块中的程序将帮你管理事件队列。输入队列很大程度依赖于 pygame 的 display 模块。如果 display 没有被初始化，显示模式没有被设置，那么事件队列就还没有开始真正工作。

常规的队列是由 pygame.event.EventType 定义的事件对象的组成，有多种方法来访问里边的事件对象：从简单的检测事件是否存在，到直接从栈中获取它们。

所有事件都有一个类型标识符，这个标识符对应的值定义在 NOEVENT 到 NUMEVENTS 之间（温馨提示：类似于 C 语言的宏定义，明白？）。用户可以自行定义事件，但类型标识符的值应该高于或等于 USEREVENT。

获取各种输入设备的状态，推荐你直接通过它们相应的模块（mouse，key 和 joystick）提供的函数访问，而不是通过事件队列；如果你使用此函数，请记住，Pygame 需要通过一些方式与系统的窗口管理器和平台的其他部分进行通信。为了保持 Pygame 和系统同步，你需要调用 pygame.event.pump() 确保实时更新，你将在游戏的每次循环中调用这个函数。

事件队列提供了一些简单的过滤。通过阻止某些事件进入事件队列，可以略微提高游戏的性能（温馨提示：因为这样事件队列的尺寸就会小一些，所以说可以略微提升性能）。使用 pygame.event.set\_allowed() 和 pygame.event.set\_blocked() 来控制某些事件是否允许进入事件队列。默认所有事件都会进入事件队列。

事件子系统应该在主线程被调用。如果你希望从其他线程中投递事件消息进入事件队列，请使用 fastevent 包。

Joysticks（游戏手柄）只有在设备初始化后才会发送事件。

一个 EventType 事件对象包含一个事件类型标识符和一组成员数据（事件对象不包含方法，只有数据）。EventType 对象从 Python 的事件队列中获得，你也可以使用 pygame.event.Event() 函数创建自定义的新事件。

由于 SDL 的事件队列限制了事件数量的上限（标准的 SDL 1.2 限制为 128），所以当队列已满时，新的事件将会被扔掉。为了防止丢失事件消息，尤其是代表退出的输入事件（因为当用户点击退出按钮没有反应，往往会被认为“死机”了），你的程序必须定期检测事件，并对其进行处理。

为了加快事件队列的处理速度，可以使用 pygame.event.set\_blocked() 函数阻止一些我们不关注的事件进入队列中。

所有的 EventType 实例对象都拥有一个事件类型标识符，属性名是 type。你也可以通过事件对象的 **dict** 属性来完全访问其他属性。所有其他成员属性的值都是通过事件对象的字典来传递。

**5.font模块**

**pygame.font**Pygame 中加载和表示字体的模块。

**函数**

* pygame.font.init() —— 初始化字体模块
* pygame.font.quit() —— 还原字体模块
* pygame.font.get\_init() —— 检查字体模块是否被初始化
* pygame.font.get\_default\_font() —— 获得默认字体的文件名
* pygame.font.get\_fonts() —— 获取所有可使用的字体
* pygame.font.match\_font() —— 在系统中搜索一种特殊的字体
* pygame.font.SysFont() —— 从系统字体库创建一个 Font 对象 类
* pygame.font.Font —— 从一个字体文件创建一个 Font 对象 字体模块可以在一个新的 Surface 对象上表示 TrueType 字体。它接受所有 UCS-2 字符（'u0001' 到 'uFFFF'）。此模块为可选择模块，并且依赖于 SDL\_ttf。在使用之前，你需要先测试该模块是否可用，而且对其进行初始化。

通过使用现有的 Font 对象，可以完成大多数与字体有关的工作。Pygame.font 模块自身仅可以完成常规的初始化以及通过 pygame.font.Font() 创建 Font 对象。

你可以通过使用 pygame.font.SysFont() 函数从系统内加载字体。另外还有其他几个函数可以帮助你搜索系统的字体。

Pygame 配备了内建的默认字体。通过传递 "None" 为文件名访问此字体。

在 pygame 第一次导入之前，当pygame.font 模块确定环境变量 PYGAME\_FREETYPE 时使用基于 pygame.ftfont 的 pygame.freetype 模块。Pygame.ftfont 是一个pygame.font 可兼容模块，兼容绝大部分，除开其中某个字体模块单元测试：Pygame.ftfont 并没有基于字体模块的 SDL\_ttf 的 UCS-2 字符限制，所以对于大于 'uFFFF' 的码点会产生异常。如果 pygame.freetype 是不可使用的，那么 SDL\_ttf 字体模块将会被加载用于替代。

**常用函数详解**类 class pygame.font.Font 从一个字体文件创建一个 Font 对象。

Font(filename, size) -> Font

Font(object, size) -> Font

方法

* pygame.font.Font.render() —— 在一个新 Surface 对象上绘制文本
* pygame.font.Font.size() —— 确定多大的空间用于表示文本
* pygame.font.Font.set\_underline() —— 控制文本是否用下划线渲染
* pygame.font.Font.get\_underline() —— 检查文本是否绘制下划线
* pygame.font.Font.set\_bold() —— 启动粗体字渲染
* pygame.font.Font.get\_bold() —— 检查文本是否使用粗体渲染
* pygame.font.Font.set\_italic() —— 启动斜体字渲染
* pygame.font.Font.metrics() —— 获取字符串参数每个字符的参数
* pygame.font.Font.get\_italic() —— 检查文本是否使用斜体渲染
* pygame.font.Font.get\_linesize() —— 获取字体文本的行高
* pygame.font.Font.get\_height() —— 获取字体的高度
* pygame.font.Font.get\_ascent() —— 获取字体顶端到基准线的距离
* pygame.font.Font.get\_descent() —— 获取字体底端到基准线的距离
* 根据提供的文件名或者 python 文件对象加载一个新的字体。字体的高度是以像素为单位。如果文件名是 "None"，则加载 Pygame 的默认字体。如果一个字体无法由给定的参数加载，将会产生一个异常。一旦字体已经创建完毕，那么字体的尺寸将不能修改。

字体对象主要被用于在新 Surface 对象中渲染文本。文本可以渲染为仿真的粗体或者斜体特征，但最好是加载的字体本身就带有粗体或者斜体字形。可以用普通字符串或者 Unicode 编码字符来渲染文本。

**方法详解**

pygame.font.Font.render() 在一个新 Surface 对象上绘制文本。

render(text, antialias, color, background=None) -> Surface

该函数创建一个新的 Surface 对象，并在上边渲染指定的文本。Pygame 没有提供直接的方式在一个现有的 Surface 对象上绘制文本，取而代之的方法是：使用 Font.render() 函数创建一个渲染了文本的图像（Surface 对象），然后将这个图像绘制到目标 Surface 对象上。

仅支持渲染一行文本：“换行”字符不会被渲染。空字符（'x00'）被渲染将产生一个 TypeError 错误。Unicode 和 char（字节）字符串都可以被接受。对于 Unicode 字符串，仅 UCS-2 字符范围（'u0001' 到 'uFFFF'）被认为是有效的。任何编码值更大字符的字符会产生一个 UnicodeError 的错误；对于 char 字符串，默认的是使用 LATIN1 编码。color 参数决定的是文本的颜色（例如：(0, 0, 255) 表示蓝色）。可选参数 background 决定了文本的背景颜色。如果没有传递 background 参数，则对应区域内表示的文本背景将会被设置为透明。

返回的 Surface 对象将保持表示文本所需要的尺寸（与 Font.size() 所返回的尺寸相同）。如果将一个空字符串渲染为文本，将会返回一个空白 Surface 对象，它仅有一个像素点的宽度，但高度与字体高度一样。

由于取决于文本背景的类型和抗锯齿功能的使用，该函数将会返回不同类型的 Surface 对象。出于性能上的考虑，了解何种类型的图像会被使用是很有帮助的：如果抗锯齿功能没有被使用，返回的图像将采用二元调色的 8 位图像。此时如果背景是透明的，只设置一个 colorkey 来实现；抗锯齿图像会被渲染为 24 位 RGB 图像。此时如果背景是透明的，每个像素都将包含一个 alpha 通道。

优化：如果你已知文本最终将绘制在一个纯色的背景上，那么文本是抗锯齿的，你可以通过指定文本的背景色来提高性能（将文本背景色设置目标 Surface 对象的颜色）。使用这个技巧，你只需用一个 colorkey 即可保持透明信息，而不需要设置每个像素的 alpha 通道值（这样效率会低很多）。

如果你尝试渲染 '\n'，通常是显示为一个矩形（未知字符）。因此，你需要自己想办法处理换行。

字体渲染并不是线程安全的行为：在任何时候仅有一个线程可以渲染文本。

**6.image模块**

**pygame.image**用于图像传输的 Pygame 模块。

**函数**

* pygame.image.load() — 从文件加载新图片
* pygame.image.save() — 将图像保存到磁盘上
* pygame.image.get\_extended() — 检测是否支持载入扩展的图像格式
* pygame.image.tostring() — 将图像转换为字符串描述
* pygame.image.fromstring() — 将字符串描述转换为图像
* pygame.image.frombuffer() — 创建一个与字符串描述共享数据的 Surface 对象 image 模块包含了加载和保存图像的函数，同时转换为 Surface 对象支持的格式。

注意：没有 Image 类；当一个图像被成功载入后，将转换为 Surface 对象。Surface 对象允许你在上边画线、设置像素、捕获区域等。

Image 是 Pygame 相当依赖的一个模块，支持载入的图像格式如下：

* JPG
* PNG
* GIF（无动画）
* BMP
* PCX
* TGA（无压缩）
* TIF
* LBM（和 PBM）
* PBM（和 PGM，PPM）
* XPM 支持保存为以下格式：
* BMP
* TGA
* PNG
* JPEG 其中，保存为 PNG 和 JPEG 格式是 Pygame 1.8 新增加的。

**函数详解pygame.image.load()**从文件加载新图片。

load(filename) -> Surface

load(fileobj, namehint=””) -> Surface

从文件加载一张图片，你可以传递一个文件路径或一个 Python 的文件对象。

Pygame 将自动判断图像的格式（比如 GIF 或位图）并创建一个新的 Surface 对象。有时它可能需要知道文件的后缀名（比如 GIF 图像应该以 ".gif" 为后缀）。如果你传入原始文件对象，你需要传入它对应的文件名到 namehint 参数中。

返回的 Surface 对象将包含与源文件相同的颜色格式，colorkey 和 alpha 透明度通道。你通常需要调用 Surface.convert() 函数进行转换，这样可以使得在屏幕上绘制的速度更快。

对于含有 alpha 通道的图片（支持部分位置透明，像 PNG 图像），需要使用 Surface.convert\_alpha() 函数进行转换。

在某些环境下，Pygame 可能无法支持上述所有的图像格式，但至少无压缩的 BMP 格式是支持的。你可以调用 pygame.image.get\_extended() 函数，如果返回 True，说明可以加载上述的格式（包含 PNG，JPG 和 GIF）。

你应该使用 os.path.join() 提高代码的兼容性：

asurf = pygame.image.load(os.path.join('data', 'Python.png'))

复制

**pygame.image.save()**将图像保存到磁盘上。

save(Surface, filename) -> None

该函数将保存 Surface 对象到磁盘上，支持存储为 BMP，TGA，PNG 或 JPEG 格式的图像。如果 filename 没有指定后缀名，那么默认是保存为 TGA 格式。TGA 和 BMP 格式是无压缩的文件。

保存为 PNG 和 JPEG 格式是 Pygame 1.8 新增的。

**pygame.image.get\_extended()**检测是否支持载入扩展的图像格式。

get\_extended() -> bool

如果 Pygame 支持上述所有的扩展图像格式，则返回 True。

**pygame.image.tostring()**将图像转换为字符串描述。

tostring(Surface, format, flipped=False) -> string

将图像转换为一个字符串描述，可以被 Python 的其他图像模块通过 "fromstring" 转换回图像。一些 Python 图像模块喜欢“自下而上”的存储格式（例如 PyOpenGL）。如果 flipped 参数为 True，那么字符串将会垂直翻转以适用这类图像模块。

format 参数可以是下表中任何一个字符串。注意：只有 8 位的 Surface 对象可以使用 "P" 格式。其他格式可以用于任何 Surface 对象上。

**pygame.image.fromstring()**将字符串描述转换为图像。

fromstring(string, size, format, flipped=False) -> Surface

该函数的使用跟 pygame.image.tostring() 相似。size 参数是一对表示宽度和高度的数字。一旦新的 Surface 对象创建成功，你就可以删除字符串描述。

size 和 format 参数指定的数据需要跟字符串描述相符，否则将抛出异常。

更快地将图片转换到 Pygame，请参考 pygame.image.frombuffer() 函数。

**pygame.image.frombuffer()**创建一个与字符串描述共享数据的 Surface 对象。

frombuffer(string, size, format) -> Surface

创建一个新的 Surface 对象，与字符串描述直接共享像素数据。该函数的使用跟 pygame.image.fromstring() 类似，但没法垂直翻转原始数据。

该函数的速度会比 pygame.image.fromstring() 快很多，因为该函数不需要申请和拷贝任何像素数据。

**7.key模块**

**pygame.key**与键盘相关的 Pygame 模块。

函数

* pygame.key.get\_focused() — 当窗口获得键盘的输入焦点时返回 True
* pygame.key.get\_pressed() — 获取键盘上所有按键的状态
* pygame.key.get\_mods() — 检测是否有组合键被按下
* pygame.key.set\_mods() — 临时设置某些组合键为被按下状态
* pygame.key.set\_repeat() — 控制重复响应持续按下按键的时间
* pygame.key.get\_repeat() — 获取重复响应按键的参数
* pygame.key.name() — 获取按键标识符对应的名字 该模块包含处理与键盘操作相关的函数。当键盘按键被按下和释放时，事件队列将获得 pygame.KEYDOWN 和 pygame.KEYUP 事件消息。这两个消息均包含 key 属性，是一个整数的 id，代表键盘上具体的某个按键。

pygame.KYEDOWN 事件还有个额外的属性 unicode 和 scancode。unicode 代表一个按键翻译后的 Unicode 编码，这包含 shift 按键和组合键。scancode 是扫描码，不同键盘间该值可能不同。不过这对于特殊按键像多媒体键的选择是有用的。

温馨提示：当键盘按下的时候，键盘会发送一个扫描码给系统。扫描码是键盘反馈哪一个按键被按下的方式，不同类型的键盘扫描码不同。再由系统调用相应的函数将其转换为统一的 Unicode 编码。

key 属性的值是一个数字，为了方便使用，Pygame 将这些数字定义为以下这些常量：

| KeyASCII | ASCII | 描述 |
| --- | --- | --- |
| K\_BACKSPACE | \b | 退格键（Backspace） |
| K\_TAB | \t | 制表键（Tab） |
| K\_CLEAR |  | 清楚键（Clear） |
| K\_RETURN | \r | 回车键（Enter） |
| K\_PAUSE |  | 暂停键（Pause） |
| K\_ESCAPE | ^[ | 退出键（Escape） |
| K\_SPACE |  | 空格键（Space） |
| K\_EXCLAIM | ! | 感叹号（exclaim） |
| K\_QUOTEDBL | " | 双引号（quotedbl） |
| K\_HASH | # | 井号（hash） |
| K\_DOLLAR | $ | 美元符号（dollar） |
| K\_AMPERSAND | & | and 符号（ampersand） |
| K\_QUOTE | ' | 单引号（quote） |
| K\_LEFTPAREN | ( | 左小括号（left parenthesis） |
| K\_RIGHTPAREN | ) | 右小括号（right parenthesis） |
| K\_ASTERISK | \* | 星号（asterisk） |
| K\_PLUS | + | 加号（plus sign） |
| K\_COMMA | , | 逗号（comma） |
| K\_MINUS | - | 减号（minus sign） |
| K\_PERIOD | . | 句号（period） |
| K\_SLASH | / | 正斜杠（forward slash） |
| K\_0 | 0 | 0 |
| K\_1 | 1 | 1 |
| K\_2 | 2 | 2 |
| K\_3 | 3 | 3 |
| K\_4 | 4 | 4 |
| K\_5 | 5 | 5 |
| K\_6 | 6 | 6 |
| K\_7 | 7 | 7 |
| K\_8 | 8 | 8 |
| K\_9 | 9 | 9 |
| K\_COLON | : | 冒号（colon） |
| K\_SEMICOLON | ; | 分号（semicolon） |
| K\_LESS | < | 小于号（less-than sign） |
| K\_EQUALS | = | 等于号（equals sign） |
| K\_GREATER | > | 大于号（greater-than sign） |
| K\_QUESTION | ? | 问号（question mark） |
| K\_AT | @ | at 符号（at） |
| K\_LEFTBRACKET | [ | 左中括号（left bracket） |
| K\_BACKSLASH | |反斜杠（backslash） |  |
| K\_RIGHTBRACKET | ] | 右中括号（right bracket） |
| K\_CARET | ^ | 脱字符（caret） |
| K\_UNDERSCORE | \_ | 下划线（underscore） |
| K\_BACKQUOTE | ` | 重音符（grave） |
| K\_a | a | a |
| K\_b | b | b |
| K\_c | c | c |
| K\_d | d | d |
| K\_e | e | e |
| K\_f | f | f |
| K\_g | g | g |
| K\_h | h | h |
| K\_i | i | i |
| K\_j | j | j |
| K\_k | k | k |
| K\_l | l | l |
| K\_m | m | m |
| K\_n | n | n |
| K\_o | o | o |
| K\_p | p | p |
| K\_q | q | q |
| K\_r | r | r |
| K\_s | s | s |
| K\_t | t | t |
| K\_u | u | u |
| K\_v | v | v |
| K\_w | w | w |
| K\_x | x | x |
| K\_y | y | y |
| K\_z | z | z |
| K\_DELETE |  | 删除键（delete） |
| K\_KP0 |  | 0（小键盘） |
| K\_KP1 |  | 1（小键盘） |
| K\_KP2 |  | 2（小键盘） |
| K\_KP3 |  | 3（小键盘） |
| K\_KP4 |  | 4（小键盘） |
| K\_KP5 |  | 5（小键盘） |
| K\_KP6 |  | 6（小键盘） |
| K\_KP7 |  | 7（小键盘） |
| K\_KP8 |  | 8（小键盘） |
| K\_KP9 |  | 9（小键盘） |
| K\_KP\_PERIOD | . | 句号（小键盘） |
| K\_KP\_DIVIDE | / | 除号（小键盘） |
| K\_KP\_MULTIPLY | \* | 乘号（小键盘） |
| K\_KP\_MINUS | - | 减号（小键盘） |
| K\_KP\_PLUS | + | 加号（小键盘） |
| K\_KP\_ENTER | \r | 回车键（小键盘） |
| K\_KP\_EQUALS | = | 等于号（小键盘） |
| K\_UP |  | 向上箭头（up arrow） |
| K\_DOWN |  | 向下箭头（down arrow） |
| K\_RIGHT |  | 向右箭头（right arrow） |
| K\_LEFT |  | 向左箭头（left arrow） |
| K\_INSERT |  | 插入符（insert） |
| K\_HOME |  | Home 键（home） |
| K\_END |  | End 键（end） |
| K\_PAGEUP |  | 上一页（page up） |
| K\_PAGEDOWN |  | 下一页（page down） |
| K\_F1 |  | F1 |
| K\_F2 |  | F2 |
| K\_F3 |  | F3 |
| K\_F4 |  | F4 |
| K\_F5 |  | F5 |
| K\_F6 |  | F6 |
| K\_F7 |  | F7 |
| K\_F8 |  | F8 |
| K\_F9 |  | F9 |
| K\_F10 |  | F10 |
| K\_F11 |  | F11 |
| K\_F12 |  | F12 |
| K\_F13 |  | F13 |
| K\_F14 |  | F14 |
| K\_F15 |  | F15 |
| K\_NUMLOCK |  | 数字键盘锁定键（numlock） |
| K\_CAPSLOCK |  | 大写字母锁定键（capslock） |
| K\_SCROLLOCK |  | 滚动锁定键（scrollock） |
| K\_RSHIFT |  | 右边的 shift 键（right shift） |
| K\_LSHIFT |  | 左边的 shift 键（left shift） |
| K\_RCTRL |  | 右边的 ctrl 键（right ctrl） |
| K\_LCTRL |  | 左边的 ctrl 键（left ctrl） |
| K\_RALT |  | 右边的 alt 键（right alt） |
| K\_LALT |  | 左边的 alt 键（left alt） |
| K\_RMETA |  | 右边的元键（right meta） |
| K\_LMETA |  | 左边的元键（left meta） |
| K\_LSUPER |  | 左边的 Window 键（left windows key） |
| K\_RSUPER |  | 右边的 Window 键（right windows key） |
| K\_MODE |  | 模式转换键（mode shift） |
| K\_HELP |  | 帮助键（help） |
| K\_PRINT |  | 打印屏幕键（print screen） |
| K\_SYSREQ |  | 魔术键（sysrq） |
| K\_BREAK |  | 中断键（break） |
| K\_MENU |  | 菜单键（menu） |
| K\_POWER |  | 电源键（power） |
| K\_EURO |  | 欧元符号（euro） |

还有一个 mod 属性，用于描述组合键状态。

以下是组合键的常量定义：

| KeyASCII | 描述 |
| --- | --- |
| KMOD\_NONE | 木有同时按下组合键 |
| KMOD\_LSHIFT | 同时按下左边的 shift 键 |
| KMOD\_RSHIFT | 同时按下右边的 shift 键 |
| KMOD\_SHIFT | 同时按下 shift 键 |
| KMOD\_CAPS | 同时按下大写字母锁定键 |
| KMOD\_LCTRL | 同时按下左边的 ctrl 键 |
| KMOD\_RCTRL | 同时按下右边的 ctrl 键 |
| KMOD\_CTRL | 同时按下 ctrl 键 |
| KMOD\_LALT | 同时按下左边的 alt 键 |
| KMOD\_RALT | 同时按下右边的 alt 键 |
| KMOD\_ALT | 同时按下 alt 键 |
| KMOD\_LMETA | 同时按下左边的元键 |
| KMOD\_RMETA | 同时按下右边的元键 |
| KMOD\_META | 同时按下元键 |
| KMOD\_NUM | 同时按下数字键盘锁定键 |
| KMOD\_MODE | 同时按下模式转换键 |

温馨提示：如果 mod & KMOD\_CTRL 是真的话，表示用户同时按下了 Ctrl 键。

**函数详解pygame.key.get\_focused()**当窗口获得键盘的输入焦点时返回 True。

get\_focused() -> bool

当窗口获得键盘的输入焦点时返回 True，如果窗口需要确保不失去键盘焦点，可以使用 pygame.event.set\_grab(True) 独占所有的输入接口。

温馨提示：注意，这样做你就无法将鼠标移出窗口客户区了，但你仍然可以通过 Ctrl - Alt - Delete 热键“解围”。

**pygame.key.get\_pressed()**获取键盘上所有按键的状态。

get\_pressed() -> bools

返回一个由布尔类型值组成的序列，表示键盘上所有按键的当前状态。使用 key 常量作为索引，如果该元素是 True，表示该按键被按下。

使用该函数获取一系列按钮被按下的状态，并不能正确的获取用户输入的文本。因为你无法知道用户按键的被按下的顺序，并且快速的连续按下键盘可能无法完全被捕获（在两次调用 pygame.key.get\_pressed() 的过程中被忽略），也无法将这些按下的按键完全转化为字符值。实现此功能可以通过捕获 pygame.KEYDOWN 事件消息来实现。

**pygame.key.get\_mods()**检测是否有组合键被按下。

get\_mods() -> int

返回一个包含所有组合键位掩码的整数。使用位操作符 & 你可以检测某个组合键是否被按下。

温馨提示：假如 pygame.key.get\_mods() 返回值存放在 mods 变量中，如果 mods & KMOD\_CTRL 为 True，表示 ctrl 键正被按下。

pygame.key.set\_mods() 临时设置某些组合键为被按下状态。

set\_mods(int) -> None

创建一个位掩码整数，包含你需要设置为被按下状态的组合键。

温馨提示：比如我们需要设置 ctrl 和 alt 组合键为按下状态，则可以 mods = KMOD\_CTRL | KMOD\_ALT，然后调用 pygame.key.set\_mods(mods)，这样尽管用户没有按下 ctrl 和 alt 组合键，它们依然是显示被按下状态。

**pygame.key.set\_repeat()**控制重复响应持续按下按键的时间。

set\_repeat() -> None

set\_repeat(delay, interval) -> None

当开启重复响应按键，那么用户持续按下某一按键，就会不断产生同一 pygame.KEYDOWN 事件。delay 参数设置多久后（单位是毫秒）开始发送第一个 pygame.KEYDOWN 事件。interval 参数设置发送两个事件之间的间隔。如果不传入任何参数，表示取消重复响应按键。

**pygame.key.get\_repeat()**获取重复响应按键的参数。

get\_repeat() -> (delay, interval)

当开启重复响应按键，那么用户持续按下某一按键，就会不断产生同一 pygame.KEYDOWN 事件。返回值是一个二元组，第一个元素 delay 表示多久后（单位是毫秒）开始发送第一个 pygame.KEYDOWN 事件。第二个元素 interval 表示发送两个事件之间的间隔。

默认情况下重复响应按键是没有开启的。

Pygame 1.8 新增加的。

**pygame.key.name()**获取按键标识符对应的名字。

name(key) -> string

获取一个按键标识符对应的字符串描述

**8.locals模块**

**pygame.locals**Pygame 定义的常量。

这个模块包含了 Pygame 定义的各种常量。它的内容会被自动放入到 Pygame 模块的名字空间中。你可以使用

**from pygame.locals import**将所有的 Pygame 常量导入。

各个常量的详细描述记录在 Pygame 各个模块的相关文档中。比如 pygame.display.set\_mode() 方法用到的 HWSURFACE 常量，你就可以在 display 模块的文档中找到详细的说明；事件类型在 event 模块的文档中可以找到；当产生 KEYDOWN 或 KEYUP 事件时，key 属性描述具体哪个按键被按下，该值是以 K\_ 开头的常量（MOD\_ 开头的常量表示各种组合键被按下），在 key 模块的文档中可以找到；最后，TIME\_RESOLUTION 被定义在 time 模块中。

**9.mixer模块**

**pygame.mixer**用于加载和播放声音的pygame模块

函数

* pygame.mixer.init — 初始化混音器模块
* pygame.mixer.pre\_init — 预设混音器初始化参数
* pygame.mixer.quit — 卸载混音器模块
* pygame.mixer.get\_init — 测试混音器是否初始化
* pygame.mixer.stop — 停止播放所有通道
* pygame.mixer.pause — 暂停播放所有通道
* pygame.mixer.unpause — 恢复播放
* pygame.mixer.fadeout — 淡出停止
* pygame.mixer.set\_num\_channels — 设置播放频道的总数
* pygame.mixer.get\_num\_channels — 获取播放频道的总数
* pygame.mixer.set\_reserved — 预留频道自动使用
* pygame.mixer.find\_channel — 找到一个未使用的频道
* pygame.mixer.get\_busy — 测试混音器是否正在使用类
* pygame.mixer.Sound — 从文件或缓冲区对象创建新的Sound对象
* pygame.mixer.Channel — 创建一个Channel对象来控制播放

此模块包含用于加载 Sound 对象和控制播放的类。混音器模块是可选的，取决于SDL\_mixer。您的程序应该在使用它之前 测试 pygame.mixer 模块是否可用并进行初始化。

混音器模块具有有限数量的声音播放声道。通常程序会告诉 pygame 开始播放音频，它会自动选择一个可用的频道。默认为8个并发通道，但复杂的程序可以更精确地控制通道数量及其使用。

所有声音播放都混合在后台线程中。当您开始播放Sound对象时，它会在声音继续播放时立即返回。单个Sound对象也可以自动播放多次。

混音器还有一个特殊流通道用于音乐播放，可通过 pygame.mixer.music 模块访问。

混音器模块必须像其他 pygame 模块一样进行初始化，但它有一些额外的条件。pygame.mixer.init() 函数采用几个可选参数来控制播放速率和样本大小。Pygame将 默认为合理的值，但pygame无法执行声音重采样，因此应初始化混音器以匹配音频资源的值。

注意：不要使用较少的延迟声音，请使用较小的缓冲区大小。 默认设置为减少某些计算机上发出沙哑声音的可能性。 您可以在 pygame.mixer.init() 或者 pygame.init() 之前 通过调用pygame.mixer.pre\_init（）预设混合器初始化参数来更改默认缓冲区。 例如：pygame.mixer.pre\_init（44100，-16,2,1024）。在pygame 1.8中，默认大小从1024更改为3072。

**函数详解**pygame.mixer.init（） 初始化混音器模块 init(frequency=22050, size=-16, channels=2, buffer=4096) -> None 初始化混音器模块以进行声音加载和播放。默认参数可以被改变以提供特定的音频混合。允许使用关键字参数。对于参数设置为零的向后兼容性，使用默认值（可能由pre\_init调用更改）。

size参数表示每个音频样本使用的位数。如果值为负，则将使用带符号的样本值。正值表示将使用不带符号的音频样本。无效值会引发异常。

pygame 2中的新功能（使用SDL2编译时） - 大小可以是32（32位浮点数）。

channels参数用于指定是使用单声道还是立体声。1表示单声道，2表示立体声。不支持其他值（负值被视为1，大于2的值被视为2）。

buffer参数控制混音器中使用的内部采样数。默认值应适用于大多数情况。可以降低它以减少延迟，但可能会发生声音丢失。它可以被提升到更大的值，以确保播放永远不会跳过，但它会对声音播放施加延迟。缓冲区大小必须是2的幂（如果不是，则向上舍入到下一个最接近的2的幂）。

某些平台需要在 display 模块初始化后初始化pygame.mixer 模块。顶级pygame.init() 自动处理此问题，但无法将任何参数传递给 mixer init。为了解决这个问题，mixer 具有pygame.mixer.pre\_init() 函数在使用顶层初始化之前设置正确默认值。

多次调用是安全的，但是在初始化混音器后，如果没有先调用 pygame.mixer.quit()，则无法更改播放参数 。

**pygame.mixer.pre\_init（**预设混音器初始化参数

pre\_init(frequency=22050, size=-16, channels=2, buffersize=4096) -> None

调用 pre\_init 可以更改调用 真正的初始化 pygame.mixer.init() 使用的默认值。允许使用关键字参数。设置自定义混音器播放值的最佳方法是 在调用顶级 pygame.init() 之前调用 pygame.mixer.pre\_init()。对于向后兼容性参数，零值将替换为启动默认值。

**pygame.mixer.quit（）**退出混音器

quit() -> None

这将卸载 pygame.mixer，如果稍候重新初始化，则所有播放将停止并且任何加载的Sound对象可能与混音器不兼容。

**pygame.mixer.get\_init（）**测试混音器是否初始化

get\_init() -> (frequency, format, channels)

如果混合器已初始化，则返回正在使用的播放参数。如果混音器尚未初始化，则返回None

**pygame.mixer.stop（）**停止播放所有声道

stop() -> None

这将停止所有活动混音器通道的播放。

**pygame.mixer.pause（）**暂时停止播放所有声道

pause() -> None

这将暂时停止活动混音器通道上的所有播放。稍后可以 通过 pygame.mixer.unpause() 恢复播放

**pygame.mixer.unpause（）**恢复播放声道

unpause() -> None

这将在暂停后恢复所有活动声道。

**pygame.mixer.fadeout（）**停止前淡出所有声音的音量

fadeout(time) -> None

这将在设定时间上淡出所有活动通道上的音量，时间以毫秒为单位。声音静音后，播放将停止。

**pygame.mixer.set\_num\_channels（）**设置播放频道的总数

set\_num\_channels(count) -> None

设置调音台的可用频道数。默认值为8。可以增加或减少该值。如果该值减小，则截断的通道上播放的声音将停止。

**pygame.mixer.get\_num\_channels（）**获取播放频道的总数

get\_num\_channels() -> count

返回当前活动的播放通道数。

**pygame.mixer.set\_reserved（）**预留频道自动使用

set\_reserved(count) -> None

调音台可以保留任意数量的通道，这些通道不会被声音自动选择播放。如果声音当前正在预留频道播放，则不会停止。

这允许应用程序为重要声音保留特定数量的声道，这些声音不得被丢弃或具有可保证的频道。

**pygame.mixer.find\_channel（）**找到一个未使用的频道

find\_channel(force=False) -> Channel

这将找到并返回一个非活动的Channel对象。如果没有非活动通道，则此函数将返回None。如果没有非活动通道且force参数为True，则会找到运行时间最长的声道并返回它。

如果调音台有 pygame.mixer.set\_reserved() 保留频道，则此处不会返回这些频道。

**pygame.mixer.get\_busy（）**测试mixer 是否正忙

get\_busy() -> bool

如果混音器正忙，则返回True。如果混音器处于空闲状态，则返回False。

类 pygame.mixer.Sound 从文件或缓冲区对象创建新的Sound对象

Sound(filename) -> Sound Sound(file=filename) -> Sound Sound(buffer) -> Sound Sound(buffer=buffer) -> Sound Sound(object) -> Sound Sound(file=object) -> Sound Sound(array=object) -> Sound

* pygame.mixer.Sound.play - 开始播放声音
* pygame.mixer.Sound.stop - 停止声音播放
* pygame.mixer.Sound.fadeout - 淡出后停止声音播放
* pygame.mixer.Sound.set\_volume - 设置此声音的播放音量
  + pygame.mixer.Sound.get\_volume - 获取播放音量
* pygame.mixer.Sound.get\_num\_channels - 计算此声音播放的次数
* pygame.mixer.Sound.get\_length - 得到声音的长度
* pygame.mixer.Sound.get\_raw - 返回Sound样本的bytestring副本。从文件名，python文件对象或可读缓冲区对象加载新的声音缓冲区。将执行有限的重新采样以帮助样本匹配混音器的初始化参数。Unicode字符串只能是文件路径名。Python 2.x字符串或Python 3.x字节对象可以是路径名或缓冲区对象。使用'file'或'buffer'关键字来避免歧义; 否则Sound可能会猜错。如果使用了array关键字，则该对象应该导出版本3，C级别数组接口，或者对于Python 2.6或更高版本，导出新的缓冲区接口（首先检查该对象的缓冲区接口。）

Sound对象表示实际的声音样本数据。更改Sound对象状态的方法将是Sound播放的所有实例。Sound对象还导出数组接口，对于Python 2.6或更高版本，还会导出新的缓冲区接口。

可以从OGG音频文件或未压缩的 WAV 文件加载声音。

注意：缓冲区将在内部复制，不会在它与Sound对象之间共享数据。

目前缓冲区和数组支持与sndarray.make\_sound 数值数组一致，因为忽略了样本符号和字节顺序。这将通过正确处理符号和字节顺序或在不同时引发异常来改变。此外，截断源样本以适合音频样本大小。这不会改变。

pygame.mixer.Sound（buffer）是pygame 1.8中新增的pygame.mixer.Sound关键字参数和数组接口支持pygame 1.9.2中的新功能。

**play（）**开始播放声音

play(loops=0, maxtime=0, fade\_ms=0) -> Channel

在可用频道上开始播放声音（即，在计算机的扬声器上）。 这将强制选择一个频道，因此如有必要，播放可能会切断当前正在播放的声音。

loops参数控制第一次播放后样本重复的次数。值 5 表示声音将播放一次，然后重复播放五次，因此共播放六次。默认值（0）表示声音不重复，因此只播放一次。如果循环设置为-1，则Sound将无限循环（但是您仍然可以调用stop（）来停止它）。

maxtime参数可用于在给定的毫秒数后停止播放。

fade\_ms参数将使声音以0音量开始播放，并在给定时间内逐渐升至全音量。样本可以在淡入完成之前结束。

这将返回所选通道的Channel对象。

**stop（）**停止声音播放

stop() -> None

这将停止在任何活动频道上播放此声音。

**fadeout（）**淡出后停止声音播放

fadeout(time) -> None

这将在以毫秒为单位在时间参数上淡出后停止播放声音。Sound会在所有播放的频道上消失并停止。

**set\_volume（）**设置此声音的播放音量

set\_volume(value) -> None

这将设置此声音的播放音量（响度）。如果正在播放，这将立即影响声音。它也会影响此声音的任何未来播放。参数是从0.0到1.0的值。

**get\_volume（）**获取播放音量

get\_volume() -> value

返回0.0到1.0之间的值，表示此Sound的音量。

**get\_num\_channels（）**计算此声音播放的次数

get\_num\_channels() -> count

返回此声音正在播放的活动频道数。

**get\_length（）**得到声音的长度

get\_length() -> seconds

以秒为单位返回此声音的长度。

**get\_raw（）**返回Sound样本的bytestring副本。

get\_raw() -> bytes

将Sound对象缓冲区的副本作为字节（对于Python 3.x）或str（对于Python 2.x）对象返回。

pygame 1.9.2中的新功能。

类**pygame.mixer.Channel**创建一个Channel对象来控制播放

Channel(id) -> Channel

* pygame.mixer.Channel.play - 在特定频道播放声音
* pygame.mixer.Channel.stop - 停止在频道上播放
* pygame.mixer.Channel.pause - 暂时停止播放频道
* pygame.mixer.Channel.unpause - 恢复暂停播放频道
* pygame.mixer.Channel.fadeout - 淡出通道后停止播放
* pygame.mixer.Channel.set\_volume - 设置播放频道的音量
* pygame.mixer.Channel.get\_volume - 获得播放频道的音量
* pygame.mixer.Channel.get\_busy - 检查通道是否处于活动状态
* pygame.mixer.Channel.get\_sound - 得到当前播放的声音
* pygame.mixer.Channel.queue - 排队Sound对象以跟随当前
* pygame.mixer.Channel.get\_queue - 返回排队的任何声音
* pygame.mixer.Channel.set\_endevent - 播放停止时让频道发送事件
* pygame.mixer.Channel.get\_endevent - 获取播放停止时频道发送的事件

返回其中一个当前通道的Channel对象。id必须是从0到 值pygame.mixer.get\_num\_channels() 的值。

Channel对象可用于精确控制Sounds的播放。一个频道只能播放一个声音。使用频道完全是可选的，因为pygame默认可以管理它们。

**play（）**在特定频道上播放声音

play(Sound, loops=0, maxtime=0, fade\_ms=0) -> None

这将开始播放特定频道上的声音。 如果频道正在播放任何其他声音，它将被停止。

loops参数与Sound.play（）中的含义相同：它是第一次重复声音的次数。 如果是3，声音将播放4次（第一次，然后是三次）。 如果循环为-1，则播放将无限重复。

与Sound.play（）一样，maxtime参数可用于在给定的毫秒数后停止播放声音。

与Sound.play（）一样，fade\_ms参数可以在声音中淡入淡出。

**stop（）**停止在频道上播放声音

stop() -> None

停止在频道上播放声音。播放停止后，频道可用于播放新的声音。

**pause（）**暂时停止播放频道

pause() -> None

暂时停止在频道上播放声音。它可以在之后调用 Channel.unpause() 恢复

**unpause（）**恢复暂停播放频道

unpause() -> None

在暂停的频道上恢复播放。

**fadeout（）**淡出通道后停止播放

fadeout(time) -> None

在给定时间参数上淡出声音后，以毫秒为单位停止播放通道。

**set\_volume（）**设置播放频道的音量

set\_volume(value) -> None

set\_volume(left, right) -> None

设定播放声音的音量（响度）。当频道开始播放时，其音量值将被重置。这只会影响当前的声音。value参数介于0.0和1.0之间。

如果传递一个参数，则它将是两个发言者的音量。如果传递两个参数并且混音器处于立体声模式，则第一个参数将是左扬声器的音量，第二个参数将是右扬声器的音量。（如果第二个参数为None，则第一个参数将是两个扬声器的音量。）

如果频道正在播放set\_volume()已调用的声音，则会同时考虑这两个呼叫。例如：

sound = pygame.mixer.Sound("s.wav") channel = s.play() # Sound plays at full volume by default sound.set\_volume(0.9) # Now plays at 90% of full volume. sound.set\_volume(0.6) # Now plays at 60% (previous value replaced). channel.set\_volume(0.5) # Now plays at 30% (0.6 \* 0.5).

**get\_volume（）**获得播放频道的音量

get\_volume() -> value

返回当前播放声音的通道音量。这没有考虑到使用的立体声分离 Channel.set\_volume()。Sound对象也有自己的音量，与音频混合。

**get\_busy（）**检查通道是否处于活动状态

get\_busy() -> bool

如果通道正在主动混合声音，则返回true。如果通道空闲，则返回False。

**get\_sound（）**得到当前播放的声音

get\_sound() -> Sound

返回当前在此频道上播放的实际Sound对象。如果通道空闲，则返回None。

**queue（）**排队Sound对象以跟随当前

queue(Sound) -> None

当声音在频道上排队时，它将在当前声音结束后立即开始播放。每个通道一次只能排队一个声音。排队的声音仅在当前播放自动结束时播放。在对Channel.stop()或的任何其他呼叫中清除它 Channel.play()。

如果在频道上没有主动播放声音，则声音将立即开始播放。

**get\_queue（）**返回排队的任何声音

get\_queue() -> Sound

如果声音已在此频道上排队，则会返回该声音。一旦排队的声音开始播放，它将不再在队列中。

**set\_endevent（）**播放停止时让频道发送事件

set\_endevent() -> None

set\_endevent(type) -> None

当为某个频道设置了一个尝试时，每当一个声音在该频道上播放时（不仅仅是第一次），它就会向一个游戏队列发送一个事件。使用pygame.event.get()一旦它发送到检索ENDEVENT。

请注意，如果您调用Sound.play(n)或Channel.play(sound,n)，结束事件仅发送一次：声音播放“n + 1”次后（请参阅Sound.play文档）。

如果在声音仍然播放时调用Channel.stop()或Channel.play()调用，则会立即发布事件。

type参数将是发送到队列的事件id。这可以是任何有效的事件类型，但一个好的选择是pygame.locals.USEREVENT和之间的值 pygame.locals.NUMEVENTS。如果没有给出类型参数，那么Channel将停止发送事件。

**get\_endevent（）**获取播放停止时频道发送的事件

get\_endevent() -> type

返回每次Channel完成声音播放时要发送的事件类型。如果没有功能返回该功能 pygame.NOEVENT。

**10.mouse模块**

**pygame.mouse**Pygame 中与鼠标工作相关的模块。

函数

* pygame.mouse.get\_pressed() —— 获取鼠标按键的情况（是否被按下）
* pygame.mouse.get\_pos() —— 获取鼠标光标的位置
* pygame.mouse.get\_rel() —— 获取鼠标一系列的活动
* pygame.mouse.set\_pos() —— 设置鼠标光标的位置
* pygame.mouse.set\_visible() —— 隐藏或显示鼠标光标
* pygame.mouse.get\_focused() —— 检查程序界面是否获得鼠标焦点
* pygame.mouse.set\_cursor() —— 设置鼠标光标在程序内的显示图像
* pygame.mouse.get\_cursor() —— 获取鼠标光标在程序内的显示图像

这些函数可以用于获取目前鼠标设备的情况，也可以改变鼠标在程序内的显示光标。

当设置显示模式之后，事件队列将开始接收鼠标事件。当鼠标按键被按下时会产生 pygame.MOUSEBUTTONDOWN 事件，当鼠标按键被松开时会产生 pygame.MOUSEBUTTONUP 事件。这些事件包含了一个按键属性，用于表示具体由哪个按键所触发。

当鼠标滑轮被滚动时也会产生 pygame.MOUSEBUTTONDOWN 和 pygame.MOUSEBUTTONUP 事件。当鼠标滑轮往上滚动时，按键将会被设置成4；当鼠标滑轮向下滚动时，按键会被设置成 5。

任何时候鼠标移动都会产生一个 pygame.MOUSEMOTION 事件。鼠标的活动被拆分成小而精确的事件。当鼠标运动时，大量的运动事件会被放入相应的队列中等待处理。没有及时清除掉一些运动事件是队列被塞满的主要原因。

如果鼠标光标被隐藏并且输入被当前显示器占用，鼠标会进入虚拟输入模式，在此模式内，鼠标的相关活动不会因为屏幕的边界限制而停止。调用 pygame.mouse.set\_visible() 方法和 pygame.event.set\_grab() 方法进行设置。

**函数详解pygame.mouse.get\_pressed()**获取鼠标按键的情况（是否被按下）。

get\_pressed() -> (button1, button2, button3)

返回一个由布尔值组成的列表，代表所有鼠标按键被按下的情况。True 意味着在调用此方法时该鼠标按键正被按下。

注意1：获取所有的鼠标事件最好是使用 pygame.event.wait() 方法或者 pygame.event.get() 方法，然后检查确认所有事件是 MOUSEBUTTONDOWN、MOUSEBUTTONUP 或者 MOUSEMOTION。

注意2：在 X11 上一些 XServers 使用中间按键仿真机制。当你同时点击按键 1 和 3 时会发出一个按键 2 被按下的事件。

注意3：在使用此方法前记住要先调用 pygame.event.get() 方法，否则此方法将不会工作。

pygame.mouse.get\_pos() 获取鼠标光标的位置。

get\_pos() -> (x, y)

返回鼠标光标的坐标 (x, y)。这个坐标以窗口左上角为基准点。光标位置可以被定位于窗口之外，但是通常被强制性限制在屏幕内。

**pygame.mouse.get\_rel()**获取鼠标一系列的活动。

get\_rel() -> (x, y)

返回在调用此方法之前的一系列活动坐标 (x, y)。鼠标光标的相关活动被限制在屏幕范围内，但是通过虚拟输入模式可以突破这个限制。此页面的顶部有虚拟输入模式的描述。

**pygame.mouse.set\_pos()**设置鼠标光标的位置。

set\_pos([x, y]) -> None

通过提供相应的参数来设置当前鼠标的位置。如果鼠标光标是可视的，则光标将会跳到新的坐标上。移动鼠标将会产生一个新的 pygame.MOUSEMOTION 事件。

**pygame.mouse.set\_visible()**隐藏或显示鼠标光标。

set\_visible(bool) -> bool

如果返回的布尔值为 True，鼠标光标将会是可视的。返回光标在调用该方法之前的可视化情况。

**pygame.mouse.get\_focused()**检查程序界面是否获得鼠标焦点。

get\_focused() -> bool

当 pygame 正在接受鼠标输入事件（或者用专业术语说，鼠标正在处于“active”或“focus”状态）返回值为 True。

一般情况下此方法用于窗口模式。在全屏模式下，该方法总会返回 True。

注意：在 MS Windows 系统中，一个窗口可以同时对鼠标和键盘事件保持监听。但是在 X-Windows 系统中，需要用一个窗口监听鼠标事件而另一个窗口监听键盘事件。pygame.mouse.get\_focused() 可以表示 pygame 窗口是否在接收鼠标事件。

**pygame.mouse.set\_cursor()**设置鼠标光标在程序内的显示图像。

set\_cursor(size, hotspot, xormasks, andmasks) -> None

当鼠标光标是可视的时，它将通过我们提供的位掩码数组显示为一个黑白色的位图。size 指定光标的宽度和高度。hotspot 指定光标的热点位置。xormasks 指定一组字节，用于进行按位异或掩码的计算。andmasks 指定一组字节，用于进行按位与掩码的计算。

光标的宽度必须是 8 的倍数，并且提供的位掩码数组必须与宽度、高度匹配。否则将抛出异常。

关于如何创建一个系统光标，请查看 pygame.cursor 模块。

**pygame.mouse.get\_cursor()**获取鼠标光标在程序内的显示图像。

get\_cursor() -> (size, hotspot, xormasks, andmasks)

**11.Rect对象**

**class pygame.Rect**Rect 是用于存储矩形坐标的 Pygame 对象。

Rect(left, top, width, height) -> Rect

Rect((left, top), (width, height)) -> Rect

Rect(object) -> Rect

属性 & 方法

* pygame.Rect.copy() — 拷贝 Rect 对象
* pygame.Rect.move() — 移动 Rect 对象
* pygame.Rect.move\_ip() — 原地移动 Rect 对象
* pygame.Rect.inflate() — 放大和缩小 Rect 对象的尺寸
* pygame.Rect.inflate\_ip() — 原地放大和缩小 Rect 对象的尺寸
* pygame.Rect.clamp() — 将一个 Rect 对象移动到另一个 Rect 对象的中心
* pygame.Rect.clamp\_ip() — 原地将一个 Rect 对象移动到另一个 Rect 对象的中心
* pygame.Rect.clip() — 获取两个 Rect 对象互相重叠的部分
* pygame.Rect.union() — 将两个 Rect 对象合并
* pygame.Rect.union\_ip() — 原地将两个 Rect 对象合并
* pygame.Rect.unionall() — 将多个 Rect 对象合并
* pygame.Rect.unionall\_ip() — 原地将多个 Rect 对象合并
* pygame.Rect.fit() — 按照一定的宽高比调整 Rect 对象
* pygame.Rect.normalize() — 翻转 Rect 对象（如果尺寸为负数）
* pygame.Rect.contains() — 检测一个 Rect 对象是否完全包含在该 Rect 对象内
* pygame.Rect.collidepoint() — 检测一个点是否包含在该 Rect 对象内
* pygame.Rect.colliderect() — 检测两个 Rect 对象是否重叠
* pygame.Rect.collidelist() — 检测该 Rect 对象是否与列表中的任何一个矩形有交集
* pygame.Rect.collidelistall() — 检测该 Rect 对象与列表中的每个矩形是否有交集
* pygame.Rect.collidedict() — 检测该 Rect 对象是否与字典中的任何一个矩形有交集
* pygame.Rect.collidedictall() — 检测该 Rect 对象与字典中的每个矩形是否有交集

Pygame 通过 Rect [对象存储](https://cloud.tencent.com/product/cos?from=10680)和操作矩形区域。一个 Rect 对象可以由 left，top，width，height 几个值创建。Rect 也可以是由 Pygame 的对象所创建，它们拥有一个属性叫“rect”。

任何需要一个 Rect 对象作为参数的 Pygame 函数都可以使用以上值构造一个 Rect。这样使得作为参数传递的同时创建 Rect 成为可能。

Rect 对象中的大部分方法在修改矩形的位置、尺寸后会返回一个新的 Rect 拷贝，原始的 Rect 对象不会有任何改变。但有些方法比较特殊，它们会“原地”修改 Rect 对象（也就是说它们会改动原始的 Rect 对象），这些方法都会以 "ip" 作为后缀（小甲鱼温馨提示："ip" 即 "in-place" 的缩写，“原地”的意思）。

对了方便大家移动和对齐，Rect 对象提供以下这些虚拟属性：

x,y top, left, bottom, right topleft, bottomleft, topright, bottomright midtop, midleft, midbottom, midright center, centerx, centery size, width, height w,h 上边这些属性均可以被赋值，例如：

rect1.right = 10 rect2.center = (20,30) 给 size，width，height 属性赋值将改变矩形的尺寸；给其它属性赋值将移动矩形。注意：一些属性是整数，一些是整数对。

如果一个 Rect 对象的 width 或 height 非 0，那么将在非 0 测试中返回 True。一些方法返回尺寸为 0 的 Rect 对象，用于表示一个非法的矩形。

Rect 对象的坐标都是整数，size 的值可以是负数，但在大多数情况下被认为是非法的。

还有一些方法可以实现矩形间碰撞检测，大多数 Python 的[容器](https://cloud.tencent.com/product/tke?from=10680)可以用于检索其中的元素与某个 Rect 对象是否碰撞。

Rect 对象覆盖的范围并不包含 right 和 bottom 指定的边缘位置。

温馨提示，一图胜千言：

这样的话，如果一个 Rect 对象的 bottom 边框恰好是另一个 Rect 对象的 top 边框（即 rect1.bottom == rect2.top），那么两矩形就恰好没有重叠的显示在屏幕上，rect1.colliderect(rect2) 也将返回 False。

尽管 Rect 对象可以被继承，但 Rect 的方法返回的是一个全新的 Rect 对象，而不是其子对象。

属性 & 方法详解**copy()**拷贝 Rect 对象。

copy() -> Rect

返回一个新的 Rect 对象，拥有与该 Rect 对象相同的位置和尺寸。

Pygame 1.9 新增加的。

**move()**移动 Rect 对象。

move(x, y) -> Rect

返回一个新的 Rect 对象。x 和 y 参数可以是正数或负数，用于指定新对象的偏移地址。

**move\_ip()**原地移动 Rect 对象。

move\_ip(x, y) -> None

效果跟 move() 方法一样，区别是这个直接作用于当前 Rect 对象，而不是返回一个新的。

**inflate()**放大和缩小 Rect 对象的尺寸。

inflate(x, y) -> Rect

返回一个新的 Rect 对象。x 和 y 参数指定新的对象放大或缩小多少像素。新的对象保持与原始 Rect 对象在同一个中心上。

**inflate\_ip()**原地放大和缩小 Rect 对象的尺寸。

inflate\_ip(x, y) -> None

效果跟 inflate() 方法一样，区别是这个直接作用于当前 Rect 对象，而不是返回一个新的。

**clamp()**将一个 Rect 对象移动到另一个 Rect 对象的中心。

clamp(Rect) -> Rect

返回一个新的 Rect 对象，范围是以 Rect 参数指定的对象为中心，保持原始 Rect 对象的尺寸不变。如果原始 Rect 对象的尺寸比 Rect 参数的要大，那么保持中心重叠，尺寸不变。

**clamp\_ip()**原地将一个 Rect 对象移动到另一个 Rect 对象的中心。

clamp\_ip(Rect) -> None

效果跟 clamp() 方法一样，区别是这个直接作用于当前 Rect 对象，而不是返回一个新的。

**clip()**获取两个 Rect 对象互相重叠的部分。

clip(Rect) -> Rect

返回一个新的 Rect 对象，范围是原始 Rect 对象与 Rect 参数指定的对象互相重叠的部分。如果两个 Rect 对象没有任何重叠，则返回一个 (0, 0, 0, 0) 的 Rect 对象。

**union()**将两个 Rect 对象合并。

union(Rect) -> Rect

返回一个新的 Rect 对象，范围将包含原始 Rect 对象与 Rect 参数指定的对象。由于结果返回一个新的矩形，所以会产生一些多与的空间。

**union\_ip()**原地将两个 Rect 对象合并。

union\_ip(Rect) -> None

效果跟 union() 方法一样，区别是这个直接作用于当前 Rect 对象，而不是返回一个新的。

**unionall()**将多个 Rect 对象合并。

unionall(Rect\_sequence) -> Rect

返回一个新的 Rect 对象，范围将包含 Rect\_sequence 参数指定的序列中所有的 Rect 对象。

**unionall\_ip()**原地将多个 Rect 对象合并。

unionall\_ip(Rect\_sequence) -> None

效果跟 unionall() 方法一样，区别是这个直接作用于当前 Rect 对象，而不是返回一个新的。

**fit()**按照一定的宽高比调整 Rect 对象。

fit(Rect) -> Rect

返回一个新的 Rect 对象，范围是 Rect 参数的对象按照原始 Rect 对象的宽高比调整得来。

**normalize()**翻转 Rect 对象（如果尺寸为负数）。

normalize() -> None

如果 width 或 height 存在负数，则做出相应的翻转，使其变为正数。翻转后的 Rect 仍然在原来的位置，只是修改其相应的属性值。

**contains()**检测一个 Rect 对象是否完全包含在该 Rect 对象内。

contains(Rect) -> bool

如果 Rect 参数指定的对象完全包含在该 Rect 对象内，返回 True，否则返回 False。

**collidepoint()**检测一个点是否包含在该 Rect 对象内。

collidepoint(x, y) -> bool

collidepoint((x,y)) -> bool

如果给定的点在该 Rect 对象内，返回 True，否则返回 False。

一个点在 Rect 的 right 或 bottom 边缘上时，并不被认为包含在该矩形内。

**colliderect()**检测两个 Rect 对象是否重叠。

colliderect(Rect) -> bool

如果两个 Rect 对象有任何重叠的地方，返回 True，否则返回 False。

注意：right 和 bottom 指定的边缘位置并不属于对应的矩形。

**collidelist()**检测该 Rect 对象是否与列表中的任何一个矩形有交集。

collidelist(list) -> index

返回值是第 1 个有相交的矩形所在列表中的索引号（如果有的话），否则返回 -1。

**collidelistall()**检测该 Rect 对象与列表中的每个矩形是否有交集。

collidelistall(list) -> indices

返回一个列表，包含所有与该 Rect 对象有交集的元素；如果一个都没有，返回一个空列表。

**collidedict()**检测该 Rect 对象是否与字典中的任何一个矩形有交集。

collidedict(dict) -> (key, value)

返回值是第 1 个有相交的矩形所在字典中的键和值；如果没有找到，返回 None。

注意：因为 Rect 对象不是哈希值，所以不能作为字典的键存在，因此比较的只有值。

**collidedictall()**检测该 Rect 对象与字典中的每个矩形是否有交集。

collidedictall(dict) -> [(key, value), ...]

返回一个列表，包含所有与该 Rect 对象有交集的键值对；如果一个都没有，返回一个空字典。

注意：因为 Rect 对象不是哈希值，所以不能作为字典的键存在，因此比较的只有值

**13.time模块**

**pygame.time**Pygame 中用于监控时间的模块。

函数

* pygame.time.get\_ticks() —— 获取以毫秒为单位的时间
* pygame.time.wait() —— 暂停程序一段时间
* pygame.time.delay() —— 暂停程序一段时间
* pygame.time.set\_timer() —— 在事件队列上重复创建一个事件
* pygame.time.Clock() —— 创建一个对象来帮助跟踪时间
* Pygame中的时间以毫秒（1/1000秒）表示。大多数平台的时间分辨率有限，大约为10毫秒。该分辨率（以毫秒为单位) 以常量 TIMER\_RESLUTION 给出。

**函数详解pygame.time.get\_ticks()**获取以毫秒为单位的时间

get\_ticks() -> milliseconds

返回自 pygame\_init() 调用以来的毫秒数。在pygame初始化之前，这将始终为0。

\*\*pygame.time.wait() \*\* 暂停程序一段时间

wait(milliseconds) -> time

将暂停一段给定的毫秒数。此函数会暂停进程以与其他程序共享处理器。等待几毫秒的程序将消耗非常少的处理器时间。它比pygame.time.delay() 函数稍微准确一些。

这将返回实际使用的毫秒数。

\*\*pygame.time.delay() \*\* 暂停程序一段时间

delay(milliseconds) -> time

将暂停一段给定的毫秒数。此功能将使用处理器（而不是休眠），使用 pygame.time.wait() 以使延迟更准确。

这将返回实际使用的毫秒数。

\*\*pygame.time.set\_timer() \*\* 在事件队列上重复创建一个事件

set\_timer(eventid, milliseconds) -> None

将事件类型设置为每隔给定的毫秒数显示在事件队列中。第一个事件将在经过一段时间后才会出现。

每种事件类型都可以附加一个单独的计时器。在 pygame.USEREVENT 和 pygame.NUMEVENTS 中使用该值更好。

要禁用事件的计时器，请将milliseconds参数设置为0。

\*\*pygame.time.Clock() \*\* 创建一个对象来帮助跟踪时间

Clock() -> Clock

* pygame.time.Clock.tick() —— 更新时钟
* pygame.time.Clock.tick\_busy\_loop() —— 更新时钟
* pygame.time.Clock.get\_time() —— 在上一个tick中使用的时间
* pygame.time.Clock.get\_rawtime() —— 在上一个tick中使用的实际时间
* pygame.time.Clock.get\_fps() —— 计算时钟帧率 创建一个新的Clock对象，可用于跟踪一段时间。时钟还提供了几个功能来帮助控制游戏的帧速率。

**tick()**更新时钟

tick(framerate=0) -> milliseconds

注：

应该每帧调用一次此方法。它将计算自上一次调用以来经过的毫秒数。

如果传递可选的帧率参数，该函数将延迟以使游戏运行速度低于每秒给定的滴答数。这可以用于帮助限制游戏的运行时速度。通过每帧调用 一次 Clock.tick(40)，程序将永远不会超过每秒40帧。

请注意，此函数使用SDL\_Delay函数，该函数在每个平台上都不准确，但不会占用太多CPU。如果你想要一个准确的计时器，请使用tick\_busy\_loop，并且不介意咀嚼CPU。

**tick\_busy\_loop()**更新时钟

tick\_busy\_loop(framerate=0) -> milliseconds

注：

应该每帧调用一次此方法。它将计算自上一次调用以来经过的毫秒数。

如果您传递可选的帧率参数，该函数将延迟以使游戏运行速度低于每秒给定的滴答数。这可以用于帮助限制游戏的运行时速度。通过每帧调用 一次 Clock.tick\_busy\_ioop(40)，程序将永远不会超过每秒40帧。

请注意，此函数使用 pygame.time.delay(，在繁忙的循环中使用大量CPU以确保时间更准确。

pygame 1.8.0中的新功能。

**get\_time()** 在上一个tick中使用的时间

get\_time() -> milliseconds

前两次调用 Clock.tick() 之间传递的毫秒数。

**get\_rawtime()** 在上一个tick中使用的实际时间

get\_rawtime() -> milliseconds

类似于 Clock.get\_time()，但不包括 Clock.tick() 延迟限制帧速率时使用的任何时间。

**get\_fps()**

计算时钟帧率

get\_fps() -> float

计算游戏的帧速率（以每秒帧数为单位）。它是通过平均最后十次调用来计算的 Clock.tick() 。

**14.music模块**

pygame.mixer.music Pygame 中控制音频流的模块。

函数

* pygame.mixer.music.load() —— 载入一个音乐文件用于播放
* pygame.mixer.music.play() —— 开始播放音乐流
* pygame.mixer.music.rewind() —— 重新开始播放音乐
* pygame.mixer.music.stop() —— 结束音乐播放
* pygame.mixer.music.pause() —— 暂停音乐播放
* pygame.mixer.music.unpause() —— 恢复音乐播放
* pygame.mixer.music.fadeout() —— 淡出的效果结束音乐播放
* pygame.mixer.music.set\_volume() —— 设置音量
* pygame.mixer.music.get\_volume() —— 获取音量
* pygame.mixer.music.get\_busy() —— 检查是否正在播放音乐
* pygame.mixer.music.set\_pos() —— 设置播放的位置
* pygame.mixer.music.get\_pos() —— 获取播放的位置
* pygame.mixer.music.queue() —— 将一个音乐文件放入队列中，并排在当前播放的音乐之后
* pygame.mixer.music.set\_endevent() —— 当播放结束时发出一个事件
* pygame.mixer.music.get\_endevent() —— 获取播放结束时发送的事件

Pygame 中播放音乐的模块和 pygame.mixer 模块是密切联系的。使用音乐模块去控制在调音器上的音乐播放。

音乐（music）播放和声音（sound）播放的不同之处在于音乐是流式的，并且绝对不会在一开始就把一个音乐文件全部载入。调音系统在工作刚开始时仅支持单音乐流。

注意：对于 MP3 格式的支持是受限制的。在一些系统上，一种不受支持的格式将会是系统崩溃，例如 Debian Linux。为了游戏的稳定性，建议使用 OGG 进行替代。

函数详解**pygame.mixer.music.load()**载入一个音乐文件用于播放。

load(filename) -> None

load(object) -> None

该函数将会载入一个音乐文件名或者文件对象，并且准备播放。如果已经有音乐流正在播放，该音乐流将被停止。另外，函数不会开始播放音乐。

**pygame.mixer.music.play()**开始播放音乐流。

play(loops=0, start=0.0) -> None

该函数用于播放已载入的音乐流。如果音乐已经开始播放，则将会重新开始播放。

loops 参数控制重复播放的次数，例如 play(5) 意味着被载入的音乐将会立即开始播放 1 次并且再重复 5 次，共 6 次。如果 loops = -1，则表示无限重复播放。

start 参数控制音乐从哪里开始播放。开始的位置取决于音乐的格式。MP3 和 OGG 使用时间表示播放位置（以秒为单位）。MOD使用模式顺序编号表示播放位置。如果音乐文件无法设置开始位置，则传递了start参数后会产生一个NotImplementedError 错误。

**pygame.mixer.music.rewind()**重新开始播放音乐。

rewind() -> None

从文件开头开始重新播放音乐。

**pygame.mixer.music.stop()**结束音乐播放。

stop() -> None

如果音乐正在播放则立即结束播放。

**pygame.mixer.music.pause()**暂停音乐流的播放。

pause() -> None

通过调用 pygame.mixer.music.unpause() 函数继续播放音乐。

**pygame.mixer.music.unpause()**恢复音乐播放。

unpause() -> None

在播放暂停后使用该函数可以继续音乐流的播放。

**pygame.mixer.music.fadeout()**淡出的效果结束音乐播放。

fadeout(time) -> None

该函数将会在音乐淡出（也就是不在有声音放出）一段指定长度的时间（以毫秒为单位）后结束播放。

注意：该函数在调用后会一直处于阻塞状态，直到音乐已经淡出。

**pygame.mixer.music.set\_volume()**设置音量。

set\_volume(value) -> None

设置音乐的播放音量。

value 参数值范围为 0.0~1.0。当新的音乐文件被载入，音量会被重置。

**pygame.mixer.music.get\_volume()**获取音量。

get\_volume() -> value

返回正在播放的音乐的音量（此音量应该是调音器音量，注意与其他音量参数区分）。返回值范围为 0.0~1.0。

**pygame.mixer.music.get\_busy()**检查是否正在播放音乐。

get\_busy() -> bool

如果有音乐流正在播放，此方法返回 True。否则返回 False。

**pygame.mixer.music.set\_pos()**设置播放的位置。

set\_pos(pos) -> None

设置播放的起始位置。pos 参数是一个浮点数（或者一个可以转换为浮点数的数值），其值取决于音乐文件的格式：

对于 MOD 文件，它是模块中的整型模式号；对于 OGG 文件，它是一个以音频开头为零点的绝对时间值（以秒为单位）；对于 MP3 文件，它是以当前播放位置为零点的绝对时间值（以秒为单位）。为了对一个 MP3 文件的进行绝对定位，建议首先调用 rewind() 函数（其他文件格式不受支持）。SDL\_mixer 更新的版本提供了更好的定位支持。如果一种特殊的格式不支持定位，将会产生一个 SDLError 错误。

该函数会调用 SDL\_mixer 内的 Mix\_SetMusicPosition() 函数。

**pygame.mixer.music.get\_pos()**获取播放的位置。

get\_pos() -> time

此函数会获得音乐的播放时长（以毫秒为单数的数值）。返回值仅代表已经音乐已经播放了多久，并不考虑任何起始位置偏移量。

**pygame.mixer.music.queue()**将一个音乐文件放入队列中，并排在当前播放的音乐之后。

queue(filename) -> None

此函数将会载入一个音乐文件并将其放入队列中。当前的音乐一旦播放完毕，正在排队的音乐文件就会开始播放。如果当前音乐被人为停止或者切换到其他音乐，则正在排队的音乐会被丢弃。

下面的示例意思是先播放 6 次 Bach 然后再播放 1 次 Mozart：

pygame.mixer.music.load('bach.ogg')

pygame.mixer.music.play(5) # Plays six times, not five!

pygame.mixer.music.queue('mozart.ogg')

pygame.mixer.music.set\_endevent()

复制

当播放结束时发出一个事件。

set\_endevent() -> None

set\_endevent(type) -> None

调用此函数会使 Pygame 在音乐结束播放后发出信号（通过事件队列）。

type 参数决定了什么样的事件将被放入事件队列中。

任何时候音乐结束，都会放入指定事件到队列中（不仅仅是第一次）。调用该函数并不带任何参数，表示停止投放事件到队列中。

**pygame.mixer.music.get\_endevent()**获取播放结束时发送的事件。

get\_endevent() -> type

返回音乐结束时被放入队列的事件类型。

如果没有指定 endevent 事件，此方法会返回 pygame.NOEVENT 。

**15.pygame模块**

**pygame**Pygame 最顶层的包。

函数 & 属性

* pygame.init() — 初始化所有导入的 pygame 模块
* pygame.quit() — 卸载所有导入的 pygame 模块
* pygame.error() — 标准 pygame 异常模块
* pygame.get\_error() — 获得当前错误信息
* pygame.set\_error() — 设置当前错误信息
* pygame.get\_sdl\_version() — 获得 SDL 的版本号
* pygame.get\_sdl\_byteorder() — 获得 SDL 的字节顺序
* pygame.register\_quit() — 注册一个函数，这个函数将在 pygame 退出时被调用
* pygame.encode\_string() — 对 unicode 或字节对象编码
* pygame.encode\_file\_path() — 将 unicode 或字节对象编码为文件系统路径
* pygame 包是可供使用的最顶层的包。Pygame 被分成许多子模块，但是并不会影响程序使用 Pygame。

为了方便，在 pygame 中绝大多数的顶级变量被放入名为“pygame.locals”的模块中。意思是说这些变量可通过以下方式导入：

import pygame from pygame.locals import \* 当你导入 pygame 后，所有可用的 pygame 子模块都将自动被导入。需要注意的是，一些 pygame 模块是“可选的”，并且可能无法使用。以防万一，Pygame 将提供了一个占位符对象替代原来的模块，这个对象可用来测试某些功能（变量）是否可用。

函数 & 属性详解**pygame.init()**初始化所有导入的 pygame 模块。

init() -> (numpass, numfail)

初始化所有导入的 pygame 模块，如果有模块导入失败也不会显示异常，但是将返回一个元组，第一个元素为成功导入的模块数，第二个元素为导入失败的个数。

也许你想分开初始化不同的模块，以提高你程序的运行速度，或者不加载暂时用不到的模块。

重复调用 init() 方法是没问题的，也不会有任何负面影响。即使你已经调用了 pygame.quit() 卸载所有模块也是可以的。

**pygame.quit()**卸载所有导入的 pygame 模块。

quit() -> None

卸载所有之前被初始化的 pygame 模块。当 python 解释器关闭时，这个方法将被无条件地调用，所以你的程序并不需要调用这个方法，除非你想要终止 pygame 资源，并继续执行其他功能。多次执行这个方法也是没有问题的。

注意：调用这个方法 pygame.quit() 会结束所有模块，但不会结束你的程序。建议用正常结束 python 程序的方法来结束 pygame 程序。

**exception pygame.error**标准的 pygame 异常。

raise pygame.error(message)

当 pygame 或 SDL 操作失败时，将会引发异常。你可以捕获任何可预见的问题并处理异常。报告异常时，会同时显示问题的描述信息。

它是 RuntimeError 异常的子类，用于捕获这些异常。

**pygame.get\_error()**得到当前错误信息。

get\_error() -> errorstr

获取 SDL 维护的一个内部错误消息。当标准 pygame.error() 标准 pygame 异常引发时，这些信息将会提供给你。

其实你很少会使用到这个方法的啦。

**pygame.set\_error()**设置当前错误信息。

set\_error(error\_msg) -> None

设置 SDL 维护的一个内部错误消息。当标准 pygame.error() 标准 pygame 异常引发时，这些信息将会提供给你。

其实你很少会使用到这个方法的啦。

**pygame.get\_sdl\_version()**获得 SDL 的版本号。

get\_sdl\_version() -> major, minor, patch

返回 SDL 库有关版本的 3 个数字。这个版本是在编译时生成的。这个方法可用来得知哪个元件是不能正常使用的。

Pygame 1.7.0 新添加的方法。

**pygame.get\_sdl\_byteorder()**获得 SDL 的字节顺序。

get\_sdl\_byteorder() -> int

获得 SDL 库的字节顺序。返回 LIL\_ENDIAN 表示小端字节顺序；返回 BIG\_ENDIAN 表示大端字节顺序。

Pygame 1.8 新添加的方法。

**pygame.register\_quit()**注册一个函数，这个函数将在 pygame 退出时被调用。

register\_quit(callable) -> None

当调用 pygame.quit() 结束所有模块时，所有通过 register\_quit() 方法注册过的函数将被调用。这一切都是自动执行的。

一般的 pygame 用户用不到这个方法。

**pygame.encode\_string()**对 unicode 或字节对象进行编码。

encode\_string([obj [, encoding [, errors [, etype]]]]) -> bytes or None

obj：

传入 unicode 类型 -> 编码 传入 bytes 类型 -> 不变 传入其他类型 -> 返回 None 没有传递 obj 参数 -> 引起 SyntaxError 异常 encoding (string)：如果存在则进行编码，默认是 unicode\_escape。

errors (string)：指定如何处理无法编码的内容，默认使用反斜杠（\）代替。

etype (exception type)：指定编码错误引发的异常类型。默认为 UnicodeEncodeError，由 PyUnicode\_AsEncodedString() 返回。对于默认的编码和错误值不应该有编码错误。

这个函数被用于编码文件路径的时候，支持使用关键字参数。

Pygame 1.9.2 新增加的方法（主要用于单元测试）。

**pygame.encode\_file\_path()**将 unicode 或 bytes 对象编码为文件系统路径。

encode\_file\_path([obj [, etype]]) -> bytes or None

obj：

* 传入 unicode 类型 -> 编码
* 传入 bytes 类型 -> 不变
* 传入其他类型 -> 返回 None
* 没有传递 obj 参数 -> 引起 SyntaxError 异常 etype（异常类型）：若给出，则出现异常时报相应编码错误，默认为 UnicodeEncodeError，由 PyUnicode\_AsEncodedString() 返回。

这个函数被用于编码文件路径的时候，结果由 sys.getfilesystemencoding() 返回，支持使用关键字参数。

**参考文档**

* Pygame官网：https://www.pygame.org
* Pygame的官网是有详细的文档的：https://www.pygame.org/docs/
* pygame中文手册pdf版本：https://yihang.cowtransfer.com/s/7b027011c9cd4a

文章分享自微信公众号：

