event | Pygame中文文档

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（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（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。

在做调试和实验时，你可以打印事件对象以及相应的类型和成员。来自系统的事件都有一个事件类型和对应的成员属性，下边是每个事件类型以及对应的成员属性列表：



事件支持等值比较。如果两个事件具有相同的类型和属性值，那么认为两个事件是相等的。

函数详解

pygame.event.pump()

让 Pygame 内部自动处理事件。

*pump() -> None*

对于游戏中的每一帧，你都需要通过某种形式去调用事件队列，这将确保你的程序在内部可以与操作系统的其他部分进行交互。如果你不打算使用其他事件函数，那么你应该调用 pygame.event.pump()，这将允许 Pygame 内部自动处理事件。

如果你的程序始终通过其他 event 模块的函数处理队列中的事件，那么该函数是没必要的。

事件队列中的内部处理是非常重要的事情。主窗口可能需要重新绘制或对系统做出响应。如果你太长时间没有调用事件队列，系统可能会认定你的程序已锁定（假死）。

pygame.event.get()

从队列中获取事件。

*get() -> Eventlist*

*get(type) -> Eventlist*

*get(typelist) -> Eventlist*

这将获取并从队列中删除事件。如果指定一个或多个 type 参数，那么只获取并删除指定类型的事件。

请注意，如果你只从队列中获取和删除指定的事件，那么久而久之，队列可能被你不关注的事件所填满。

pygame.event.poll()

从队列中获取一个事件。

*poll() -> EventType instance*

从队列中返回并删除一个事件。

如果事件队列为空，那么会立刻返回类型为 pygame.NOEVENT 的事件。

pygame.event.wait()

等待并从队列中获取一个事件。

*wait() -> EventType instance*

从队列中返回并删除一个事件。如果队列为空，那么该函数将持续等待直至队列中有一个事件。当程序在等待时，它将保持睡眠状态。这对于希望与其他应用程序共享系统来说，是非常重要的。

pygame.event.peek()

检测某类型事件是否在队列中。

*peek(type) -> bool*

*peek(typelist) -> bool*

如果参数指定的类型的事件存在于队列中，返回 True。

如果参数指定多个类型的事件，则只需队列中拥有其中的任何一个事件便返回 True。

pygame.event.clear()

从队列中删除所有的事件。

*clear() -> None*

*clear(type) -> None*

*clear(typelist) -> None*

从队列中删除所有的事件，如果通过参数指定事件的类型，则删除该类型的所有事件。该函数的效果跟pygame.event.get() 相同，只是没有返回任何东西。当处理完关注的事件后，清空整个队列可以提高一些效率。

pygame.event.event\_name()

通过 id 获得该事件的字符串名字。

*event\_name(type) -> string*

Pygame 通过整数 id 代表事件类型。如果你需要将这些类型的事件展示给用户看，那么你需要将它们转换成字符串（一堆数字谁知道你想表示啥？）。该函数将返回事件类型对应的字符串名字。返回值是以单词大写的样式（提示：DanCiDaXieDe）。

pygame.event.set\_blocked()

控制哪些事件禁止进入队列。

*set\_blocked(type) -> None*

*set\_blocked(typelist) -> None*

*set\_blocked(None) -> None*

参数指定的类型的事件均不允许出现在事件队列中。默认是允许所有事件进入队列。多次禁止同一类型的事件并不会引发什么问题。

如果传入 None，则表示允许所有的事件进入队列。

pygame.event.set\_allowed()

控制哪些事件允许进入队列。

*set\_allowed(type) -> None*

*set\_allowed(typelist) -> None*

*set\_allowed(None) -> None*

参数指定的类型的事件均允许出现在事件队列中。默认是允许所有事件进入队列。多次允许同一类型的事件并不会引发什么问题。

如果传入 None，则表示禁止所有的事件进入队列。

pygame.event.get\_blocked()

检测某一类型的事件是否被禁止进入队列。

*get\_blocked(type) -> bool*

如果参数指定类型的事件被禁止进入队列，则返回 True。

pygame.event.set\_grab()

控制输入设备与其他应用程序的共享。

*set\_grab(bool) -> None*

当你的程序运行在窗口环境中，它将与其他拥有焦点的应用程序分享鼠标和键盘设备的输入。如果你的程序设置事件独占为True，那么你的程序将锁定所有的输入（提示：不共享给其他程序了）。

最好不要经常独占输入，因为这将阻止用户在操作系统上的其他操作。

pygame.event.get\_grab()

检测程序是否共享输入设备。

*get\_grab() -> bool*

当程序独占输入事件时，返回 True。使用 pygame.event.set\_grab() 函数控制这一状态。

pygame.event.post()

放置一个新的事件到队列中。

*post(Event) -> None*

该函数将放置一个新的事件到事件队列的末端。这些事件将最迟被其他队列函数获取。

该函数通常用于放置 pygame.USEREVENT（用户自定义事件）事件到队列中。尽管你可以放置所有类型的事件，但你需要确保为系统事件类型相应的属性传递合适的值。

如果 SDL 事件队列已满，将抛出 pygame.error 异常。

pygame.event.Event()

创建一个新的事件对象。

*Event(type, dict) -> EventType instance*

*Event(type, \*\*attributes) -> EventType instance*

根据参数给定的类型创建一个新的事件。dict 参数指定事件的属性以及相应的值。

class pygame.event.EventType

代表 SDL 事件的 Pygame 对象。

*pygame.event.EventType.type — SDL event type identifier.*

*pygame.event.EventType.\_\_dict\_\_ — vent object attribute dictionary*

用于代表 SDL 事件的 Pygame 对象。通过 pygame.event.Event() 创建用户自定义事件。EventType 类型并不是直接可以

被调用的。EventType 实例对象支持属性赋值和删除。

type

SDL 事件类型标识符。

*type -> int*

只读。预定义事件标识符是 QUIT 和 MOUSEMOTION 等。对于用于创建的事件对象，这是传递给

pygame.event.Event() 的 type 参数。

\_\_dict\_\_

事件对象的属性字典。

*\_\_dict\_\_ -> dict*

只读。事件类型指定的属性。例如，KEYDOWN 事件包含 unicode，key 和 mod 属性。