|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **time 模块 -- 时间获取和转换**  time 模块提供各种时间相关的功能  在 Python 中，与时间处理有关的模块包括：time，datetime 以及 calendar   **必要说明：**   * 虽然这个模块总是可用，但并非所有的功能都适用于各个平台。 * 该模块中定义的大部分函数是调用 C 平台上的同名函数实现，所以各个平台上实现可能略有不同。   **一些术语和约定的解释：**   * 时间戳（timestamp）的方式：通常来说，时间戳表示的是从 1970 年 1 月 1 日 00:00:00 开始按秒计算的偏移量（time.gmtime(0)）此模块中的函数无法处理 1970 纪元年以前的日期和时间或太遥远的未来（处理极限取决于 C 函数库，对于 32 位系统来说，是 2038 年） * UTC（Coordinated Universal Time，世界协调时）也叫格林威治天文时间，是世界标准时间。在中国为 UTC+8 * DST（Daylight Saving Time）即夏令时的意思 * 一些实时函数的计算精度可能低于它们建议的值或参数，例如在大部分 Unix 系统，时钟一秒钟“滴答”50~100 次   **时间元祖（time.struct\_time）：**  gmtime()，localtime() 和 strptime() 以时间元祖（struct\_time）的形式返回。   |  |  |  | | --- | --- | --- | | **索引值（Index）** | **属性（Attribute）** | **值（Values）** | | 0 | tm\_year（年） | （例如：2015） | | 1 | tm\_mon（月） | 1 ~ 12 | | 2 | tm\_mday（日） | 1 ~ 31 | | 3 | tm\_hour（时） | 0 ~ 23 | | 4 | tm\_min（分） | 0 ~ 59 | | 5 | tm\_sec（秒） | 0 ~ 61（见下方注1） | | 6 | tm\_wday（星期几） | 0 ~ 6（0 表示星期一） | | 7 | tm\_yday（一年中的第几天） | 1 ~ 366 | | 8 | tm\_isdst（是否为夏令时） | 0， 1， -1（-1 代表夏令时） |   *注1：范围真的是 0 ~ 61（你没有看错哦^\_^）；60 代表闰秒，61 是基于历史原因保留。*   **time.altzone**  返回格林威治西部的夏令时地区的偏移秒数；如果该地区在格林威治东部会返回负值（如西欧，包括英国）；对夏令时启用地区才能使用。   **time.asctime([t])**  接受时间元组并返回一个可读的形式为"Tue Dec 11 18:07:14 2015"（2015年12月11日 周二 18时07分14秒）的 24 个字符的字符串。   **time.clock()**  用以浮点数计算的秒数返回当前的 CPU 时间。用来衡量不同程序的耗时，比 time.time() 更有用。  *Python 3.3 以后不被推荐，由于该方法依赖操作系统，建议使用 perf\_counter() 或 process\_time() 代替（一个返回系统运行时间，一个返回进程运行时间，请按照实际需求选择）*   **time.ctime([secs])**  作用相当于 asctime(localtime(secs))，未给参数相当于 asctime()   **time.gmtime([secs])**  接收时间辍（1970 纪元年后经过的浮点秒数）并返回格林威治天文时间下的时间元组 t（注：t.tm\_isdst 始终为 0）   **time.daylight**  如果夏令时被定义，则该值为非零。   **time.localtime([secs])**  接收时间辍（1970 纪元年后经过的浮点秒数）并返回当地时间下的时间元组 t（t.tm\_isdst 可取 0 或 1，取决于当地当时是不是夏令时）   **time.mktime(t)**  接受时间元组并返回时间辍（1970纪元后经过的浮点秒数）   **time.perf\_counter()**  返回计时器的精准时间（系统的运行时间），包含整个系统的睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。   **time.process\_time()**  返回当前进程执行 CPU 的时间总和，不包含睡眠时间。由于返回值的基准点是未定义的，所以，只有连续调用的结果之间的差才是有效的。   **time.sleep(secs)**  推迟调用线程的运行，secs 的单位是秒。   **time.strftime(format[, t])**  把一个代表时间的元组或者 struct\_time（如由 time.localtime() 和 time.gmtime() 返回）转化为格式化的时间字符串。如果 t 未指定，将传入 time.localtime()。如果元组中任何一个元素越界，将会抛出 ValueError 异常。  format 格式如下：   |  |  |  | | --- | --- | --- | | **格式** | **含义** | **备注** | | %a | 本地（locale）简化星期名称 |  | | %A | 本地完整星期名称 |  | | %b | 本地简化月份名称 |  | | %B | 本地完整月份名称 |  | | %c | 本地相应的日期和时间表示 |  | | %d | 一个月中的第几天（01 - 31） |  | | %H | 一天中的第几个小时（24 小时制，00 - 23） |  | | %l | 一天中的第几个小时（12 小时制，01 - 12） |  | | %j | 一年中的第几天（001 - 366） |  | | %m | 月份（01 - 12） |  | | %M | 分钟数（00 - 59） |  | | %p | 本地 am 或者 pm 的相应符 | *注1* | | %S | 秒（01 - 61） | *注2* | | %U | 一年中的星期数（00 - 53 星期天是一个星期的开始）第一个星期天之前的所有天数都放在第 0 周 | *注3* | | %w | 一个星期中的第几天（0 - 6，0 是星期天） | *注3* | | %W | 和 %U 基本相同，不同的是 %W 以星期一为一个星期的开始 |  | | %x | 本地相应日期 |  | | %X | 本地相应时间 |  | | %y | 去掉世纪的年份（00 - 99） |  | | %Y | 完整的年份 |  | | %z | 用 +HHMM 或 -HHMM 表示距离格林威治的时区偏移（H 代表十进制的小时数，M 代表十进制的分钟数） |  | | %Z | 时区的名字（如果不存在为空字符） |  | | %% | %号本身 |  |   *注1：“%p”只有与“%I”配合使用才有效果。* *注2：范围真的是 0 ~ 61（你没有看错哦^\_^）；60 代表闰秒，61 是基于历史原因保留。* *注3：当使用 strptime() 函数时，只有当在这年中的周数和天数被确定的时候 %U 和 %W 才会被计算。*  举个例子：   1. # I love FishC.com! 2. >>> import time as t 3. >>> t.strftime("a, %d %b %Y %H:%M:%S +0000", t.gmtime()) 4. 'a, 24 Aug 2014 14:15:03 +0000'   复制代码  **time.strptime(string[, format])**  把一个格式化时间字符串转化为 struct\_time。实际上它和 strftime() 是逆操作。  举个例子：   1. # I really love FishC.com! 2. >>> import time as t 3. >>> t.strptime("30 Nov 14", "%d %b %y") 4. time.struct\_time(tm\_year=2014, tm\_mon=11, tm\_mday=30, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=6, tm\_yday=334, tm\_isdst=-1)   复制代码  **time.time()**  返回当前时间的时间戳（1970 纪元年后经过的浮点秒数）   **time.timezone**  time.timezone 属性是当地时区（未启动夏令时）距离格林威治的偏移秒数（美洲 >0；大部分欧洲，亚洲，非洲 <= 0）   **time.tzname**  time.tzname 属性是包含两个字符串的元组：第一是当地非夏令时区的名称，第二个是当地的 DST 时区的名称。 |