3.16 结合时区的日期操作¶

问题¶

你有一个安排在2012年12月21日早上9:30的电话会议，地点在芝加哥。 而你的朋友在印度的班加罗尔，那么他应该在当地时间几点参加这个会议呢？

解决方案¶

对几乎所有涉及到时区的问题，你都应该使用 `pytz` 模块。这个包提供了Olson时区数据库， 它是时区信息的事实上的标准，在很多语言和操作系统里面都可以找到。

`pytz` 模块一个主要用途是将 `datetime` 库创建的简单日期对象本地化。 比如，下面如何表示一个芝加哥时间的示例：

```
>>> from datetime import datetime
>>> from pytz import timezone
>>> d = datetime(2012, 12, 21, 9, 30, 0)
>>> print(d)
2012-12-21 09:30:00
>>>

>>> # Localize the date for Chicago
>>> central = timezone('US/Central')
>>> loc_d = central.localize(d)
>>> print(loc_d)
2012-12-21 09:30:00-06:00
>>>
```

一旦日期被本地化了， 它就可以转换为其他时区的时间了。 为了得到班加罗尔对应的时间，你可以这样做：

```
>>> # Convert to Bangalore time
>>> bang_d = loc_d.astimezone(timezone('Asia/Kolkata'))
>>> print(bang_d)
2012-12-21 21:00:00+05:30
>>>
```

如果你打算在本地化日期上执行计算，你需要特别注意夏令时转换和其他细节。 比如，在2013年，美国标准夏令时时间开始于本地时间3月13日凌晨2:00(在那时，时间向前跳过一小时)。 如果你正在执行本地计算，你会得到一个错误。比如：

```
>>> d = datetime(2013, 3, 10, 1, 45)
>>> loc_d = central.localize(d)
>>> print(loc_d)
2013-03-10 01:45:00-06:00
>>> later = loc_d + timedelta(minutes=30)
>>> print(later)
2013-03-10 02:15:00-06:00 # WRONG! WRONG!
>>>
```

结果错误是因为它并没有考虑在本地时间中有一小时的跳跃。 为了修正这个错误，可以使用时区对象 `normalize()` 方法。比如：

```
>>> from datetime import timedelta
>>> later = central.normalize(loc_d + timedelta(minutes=30))
>>> print(later)
2013-03-10 03:15:00-05:00
>>>
```

讨论¶

为了不让你被这些东东弄的晕头转向，处理本地化日期的通常的策略先将所有日期转换为UTC时间， 并用它来执行所有的中间存储和操作。比如：

```
>>> print(loc_d)
2013-03-10 01:45:00-06:00
>>> utc_d = loc_d.astimezone(pytz.utc)
>>> print(utc_d)
2013-03-10 07:45:00+00:00
>>>
```

一旦转换为UTC，你就不用去担心跟夏令时相关的问题了。 因此，你可以跟之前一样放心的执行常见的日期计算。 当你想将输出变为本地时间的时候，使用合适的时区去转换下就行了。比如：

```
>>> later_utc = utc_d + timedelta(minutes=30)
>>> print(later_utc.astimezone(central))
2013-03-10 03:15:00-05:00
>>>
```

当涉及到时区操作的时候，有个问题就是我们如何得到时区的名称。 比如，在这个例子中，我们如何知道“Asia/Kolkata”就是印度对应的时区名呢？ 为了查找，可以使用ISO 3166国家代码作为关键字去查阅字典 `pytz.country_timezones` 。比如：

```
>>> pytz.country_timezones['IN']
['Asia/Kolkata']
>>>
```

注：当你阅读到这里的时候，有可能 `pytz` 模块已经不再建议使用了，因为PEP431提出了更先进的时区支持。 但是这里谈到的很多问题还是有参考价值的(比如使用UTC日期的建议等)。