In [1]:
```
# http://docs.python-requests.org/zh_CN/latest/
```

Blocking Or Non-Blocking? With the default Transport Adapter in place, Requests does not provide any kind of non-blocking IO. The Response.content property will block until the entire response has been downloaded. If you require more granularity, the streaming features of the library (see Streaming Requests) allow you to retrieve smaller quantities of the response at a time. However, these calls will still block. If you are concerned about the use of blocking IO, there are lots of projects out there that combine Requests with one of Python's asynchronicity frameworks. Some excellent examples are requests-threads, grequests, and requests-futures.

阻塞和非阻塞 使用默认的传输适配器，Requests 不提供任何形式的非阻塞 IO。 Response.content 属性会阻塞，直到整个响应下载完成。 如果你需要更多精细控制，该库的数据流功能（见流式请求）允许你每次接受少量的一部分响应，不过这些调用依然是阻塞式的。 如果你对于阻塞式 IO 有所顾虑，还有很多项目可以供你使用，它们结合了 Requests 和 Python 的某个异步框架。 典型的优秀例子是 grequests 和 requests-futures。

In [3]:
```
N = 5
```

In [4]:
```
import requests

def f():
    sess = requests.session()
    p = sess.get('http://www.baidu.com')
    sess.close()

for i in range(N):
    %time f()
```

```
CPU times: user 5.25 ms, sys: 885 µs, total: 6.13 ms
Wall time: 28.9 ms
CPU times: user 3.03 ms, sys: 4 µs, total: 3.04 ms
Wall time: 38.6 ms
CPU times: user 2.43 ms, sys: 0 ns, total: 2.43 ms
Wall time: 33.2 ms
CPU times: user 2.37 ms, sys: 29 µs, total: 2.4 ms
Wall time: 41.2 ms
CPU times: user 2.52 ms, sys: 0 ns, total: 2.52 ms
Wall time: 39.2 ms
```

In [5]:
```
import requests
sess = requests.session()
for i in range(N):
    %time p = sess.get('http://www.baidu.com')

sess.close()
```

```
CPU times: user 1.68 ms, sys: 1.81 ms, total: 3.49 ms
Wall time: 28.4 ms
CPU times: user 2.79 ms, sys: 0 ns, total: 2.79 ms
Wall time: 15.4 ms
CPU times: user 2.46 ms, sys: 68 µs, total: 2.53 ms
Wall time: 15.8 ms
CPU times: user 2.36 ms, sys: 132 µs, total: 2.49 ms
Wall time: 15.7 ms
CPU times: user 2.17 ms, sys: 127 µs, total: 2.3 ms
Wall time: 15.3 ms
```

In [6]:
```
import requests
import json
sess = requests.session()
for i in range(N):
    %time sess.get('http://www.baidu.com')

sess.close()
```

```
CPU times: user 3.58 ms, sys: 83 µs, total: 3.66 ms
Wall time: 26.7 ms
CPU times: user 2.35 ms, sys: 48 µs, total: 2.39 ms
Wall time: 15.2 ms
CPU times: user 2.21 ms, sys: 451 µs, total: 2.66 ms
Wall time: 16.4 ms
CPU times: user 2.5 ms, sys: 108 µs, total: 2.61 ms
Wall time: 15.5 ms
CPU times: user 2.58 ms, sys: 0 ns, total: 2.58 ms
Wall time: 15.8 ms
```

In [7]:
```
import grequests
tasks = []
for i in range(N):
    %time grequests.get('http://www.baidu.com')

grequests.map(tasks, size=4)
```

```
CPU times: user 126 µs, sys: 18 µs, total: 144 µs
Wall time: 149 µs
CPU times: user 74 µs, sys: 0 ns, total: 74 µs
Wall time: 77.2 µs
CPU times: user 57 µs, sys: 9 µs, total: 66 µs
Wall time: 69.9 µs
CPU times: user 64 µs, sys: 0 ns, total: 64 µs
Wall time: 67.7 µs
CPU times: user 61 µs, sys: 0 ns, total: 61 µs
Wall time: 65.1 µs

/data/soft/py3/lib/python3.6/site-packages/grequests.py:21: MonkeyPatchWarning: Monkey-patching ssl aft
er ssl has already been imported may lead to errors, including RecursionError on Python 3.6. Please mon
key-patch earlier. See https://github.com/gevent/gevent/issues/1016
  curious_george.patch_all(thread=False, select=False)
```

Out[7]:
```
[]
```

By default a ThreadPoolExecutor is created with 2 workers. If you would like to adjust that value or share a executor across multiple sessions you can provide one to the FuturesSession constructor. from concurrent.futures import ThreadPoolExecutor from requests_futures.sessions import FuturesSession

```
session = FuturesSession(executor=ThreadPoolExecutor(max_workers=10))
...
```

As a shortcut in case of just increasing workers number you can pass max_workers straight to the FuturesSession constructor: from requests_futures.sessions import FuturesSession

```
session = FuturesSession(max_workers=10)
```

FutureSession will use an existing session object if supplied:

```
from requests import session
from requests_futures.sessions import FuturesSession
my_session = session()
future_session = FuturesSession(session=my_session)
```

That's it. The api of requests.Session is preserved without any modifications beyond returning a Future rather than Response. As with all futures exceptions are shifted (thrown) to the future.result() call so try/except blocks should be moved there. ```

In [9]:
```
from requests import session
from requests_futures.sessions import FuturesSession
my_session = session()
sess = FuturesSession(session = my_session)

for i in range(N):
    %time sess.get('http://www.baidu.com')

sess.close()
```

```
CPU times: user 2.56 ms, sys: 945 µs, total: 3.51 ms
Wall time: 3.2 ms
CPU times: user 1.24 ms, sys: 32 µs, total: 1.27 ms
Wall time: 1.18 ms
CPU times: user 44 µs, sys: 0 ns, total: 44 µs
Wall time: 49.6 µs
CPU times: user 28 µs, sys: 0 ns, total: 28 µs
Wall time: 31.5 µs
CPU times: user 23 µs, sys: 4 µs, total: 27 µs
Wall time: 30.3 µs
```