What is Monkey Patch

是在运行时对已有的代码进行修改，达到 hot patch 的目的。Eventlet 中大量使用了该技巧，以替换标准库中的组件，比如 socket。首先来看一下最简单的 monkey patch 的实现。

```
class Foo(object):
    def bar(self):
        print 'Foo.bar'
def bar(self):
    print 'Modified bar'
Foo().bar()
Foo.bar = bar
Foo().bar()
```

由于 Python 中的名字空间是开放，通过 dict 来实现，所以很容易就可以达到 patch 的目的。

Python namespace

Python 有几个 namespace，分别是

- locals
- globals
- builtin

其中定义在函数内声明的变量属于 locals，而模块内定义的函数属于 globals。

Python module Import & Name Lookup

当我们 import 一个 module 时，python 会做以下几件事情

- 导入一个 module
- 将 module 对象加入到 sys.modules，后续对该 module 的导入将直接从该 dict 中获得
- 将 module 对象加入到 globals dict 中

当我们引用一个模块时，将会从 globals 中查找。这里如果要替换掉一个标准模块，我们得做以下两件事情

1. 将我们自己的 module 加入到 sys.modules 中，替换掉原有的模块。如果被替换模块还没加载，那么我们得先对其进行加载，否则第一次加载时，还会加载标准模块。（这里有一个 import hook 可以用，不过这需要我们自己实现该 hook，可能也可以使用该方法 hook module import）
2. 如果被替换模块引用了其他模块，那么我们也需要进行替换，但是这里我们可以修改 globals dict，将我们的 module 加入到 globals 以 hook 这些被引用的模块。

Eventlet Patcher Implementation

现在我们先来看一下 eventlet 中的 Patcher 的调用代码吧，这段代码对标准的 ftplib 做 monkey patch，将 eventlet 的 GreenSocket 替换标准的 socket。

```
from eventlet import patcher
# *NOTE: there might be some funny business with the "SOCKS" module
# if it even still exists
```

```
from eventlet.green import socket
patcher.inject('ftplib', globals(), ('socket', socket))
del patcher
```

inject 函数会将 eventlet 的 socket 模块注入标准的 ftplib 中，globals dict 被传入以做适当的修改。

让我们接着来看一下 inject 的实现。

```
__exclude = set(('__builtins__', '__file__', '__name__'))

def inject(module_name, new_globals, *additional_modules):
    """Base method for "injecting" greened modules into an imported module.  It
    imports the module specified in *module_name*, arranging things so
    that the already-imported modules in *additional_modules* are used when
    *module_name* makes its imports.

    *new_globals* is either None or a globals dictionary that gets populated
    with the contents of the *module_name* module.  This is useful when creating
    a "green" version of some other module.

    *additional_modules* should be a collection of two-element tuples, of the
    form (, ).  If it's not specified, a default selection of
    name/module pairs is used, which should cover all use cases but may be
    slower because there are inevitably redundant or unnecessary imports.
    """
    if not additional_modules:
        # supply some defaults
        additional_modules = (
            _green_os_modules() +
            _green_select_modules() +
            _green_socket_modules() +
            _green_thread_modules() +
            _green_time_modules())

    ## Put the specified modules in sys.modules for the duration of the import
    saved = {}
    for name, mod in additional_modules:
```

```
        saved[name] = sys.modules.get(name, None)
        sys.modules[name] = mod

    ## Remove the old module from sys.modules and reimport it while
    ## the specified modules are in place
    old_module = sys.modules.pop(module_name, None)
    try:
        module = __import__(module_name, {}, {}, module_name.split('.')[:-1])
        if new_globals is not None:
            ## Update the given globals dictionary with everything from this new module
            for name in dir(module):
                if name not in __exclude:
                    new_globals[name] = getattr(module, name)
        ## Keep a reference to the new module to prevent it from dying
        sys.modules['__patched_module_' + module_name] = module
    finally:
        ## Put the original module back
        if old_module is not None:
            sys.modules[module_name] = old_module
        elif module_name in sys.modules:
            del sys.modules[module_name]
        ## Put all the saved modules back
        for name, mod in additional_modules:
            if saved[name] is not None:
                sys.modules[name] = saved[name]
            else:
                del sys.modules[name]
    return module
```

注释比较清楚的解释了代码的意图。代码还是比较容易理解的。这里有一个函数__import__，这个函数提供一个模块名（字符串），来加载一个模块。而我们 import 或者 reload 时提供的名字是对象。

```
        if new_globals is not None:
            ## Update the given globals dictionary with everything from this new module
            for name in dir(module):
                if name not in __exclude:
                    new_globals[name] = getattr(module, name)
```

这段代码的作用是将标准的 ftplib 中的对象加入到 eventlet 的 ftplib 模块中。因为我们在 eventlet.ftplib 中调用了 inject，传入了 globals，而 inject 中我们手动__import__了这个 module，只得到了一个模块对象，所以模块中的对象不会被加入到 globals 中，需要手动添加。

这里为什么不用 from ftplib import *的缘故，应该是因为这样无法做到完全替换 ftplib 的目的。因为 from ... import *会根据__init__.py 中的__all__列表来导入 public symbol，而这样对于下划线开头的 private symbol 将不会导入，无法做到完全 patch。