Splitting a Large Class and Multiple Inheritance in Python

作者：blueset

原文链接：https://ld246.com/article/1586761463199

来源网站：链滴

<!-- wp:paragraph -->

When I started refactoring EFB Telegram Master Channel (ETM) for 2.0 updates, I was inv
stigating ways to organize code into different files in a decent manner. In this article I’ d like
o talk about the strategy I used, comparing to another codebase I was reading back then, <a
ria-label="itchat (opens in a new tab)" rel="noreferrer noopener" href="https://github.com/lit
lecodersh/ItChat/" target="_blank" class="ek-link"><code>itchat</code>.
<!-- /wp:paragraph -->

<!-- wp:more -->

<!--more-->

<!-- /wp:more -->

<!-- wp:paragraph -->

In ETM version 1, most of the code is written the heavy and ugly 1675-line-long <code>
a href="https://github.com/blueset/ehForwarderBot/blob/v1/plugins/eh_telegram_master/__i
it__.py" class="ek-link">__init__.py</code>. As more features planned to be added to E
M, it was really hard for me to navigate through the code, which have brought up my need of
refactoring this huge thing.
<!-- /wp:paragraph -->

<!-- wp:paragraph -->

Back then <small>(which, surprisingly, was <a href="https://github.com/blueset/ehForwa
derBot/commit/652262099425e47824504c231a17f5f1763220c1" class="ek-link">over 2 years
ago)</small>, the main reference I had on a large enough project was <code>itchat</c
de>. Their code structure hasn’ t been changing much since then. <code>itchat</code> did
have a reasonably large code repository, but the way it splits its functions is rather unideal. <
p>
<!-- /wp:paragraph -->

<!-- wp:paragraph -->

The way itchat did to have all functioned defined at root level of each file, and have a loa
er function that “loads” these methods to an object called <code>core</code> which cont
ins some configuration data. To the Python interpreter, this method indeed works, thanks to i
s dynamic typing. But this looks really bad when you were trying to work with the code, as IDE
usually can’ t give any hint with objects defined in this way. That also happens when you try
o work on the library itself, despite every function starts with a <code>self</code> in their ar
uments.
<!-- /wp:paragraph -->

<!-- wp:paragraph -->

Then I went on looking for other common practices on breaking down a large class, some
suggested importing functions inside a function, other using multiple inheritance. <sup>[<a

ref="https://stackoverflow.com/questions/9638446/is-there-any-python-equivalent-to-partial classes" class="ek-link">Ref.]</sup> The former is not much different from what <code itchat</code> was doing, and the latter looked promising at the beginning. I went on to do s me experiment with multiple inheritance, and found that it does provide better autocomplete with IDE, but only in the main class. I can’ t see one subclass from another one in the IDE. Th t is still reasonable as all those subclasses only comes together in the main class, they are not ware of each other.</p>
<!-- /wp:paragraph -->

<!-- wp:ghostkit/tabs-v2 {"tabActive":"tab-corepy","tabsData":[{"slug":"tab-corepy","title":"\u 03ccode\u003ecore.py\u003c/code\u003e"},{"slug":"tab-componentsinitpy","title":"\u003cco e\u003ecomponents/__init__.py\u003c/code\u003e"},{"slug":"tab-componentscomponent1py" "title":"\u003ccode\u003ecomponents/component_1.py\u003c/code\u003e"},{"slug":"tab-co ponentscomponent2py","title":"\u003ccode\u003ecomponents/component_2.py\u003c/code u003e"}],"className":"is-style-pills"} -->

<code>core.py</code><code>components/__init__.py</code><code>components/component_1.py</ ode><ode>components/component_2.py</code> !-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-corepy"} -->
<!-- wp:loos-hcb/code-block {"langType":" ython","langName":"Python","fileName":"core.py","preClass":"prism undefined-numbers lang python"} -->
<pre class="prism undefined-numbers lang-python" data-file="core. y" data-lang="Python"><code>from .components import load_components

class Core:

def method_1(self, param_1, param_2, param_3):

"""Doc string goes here."
quot;"

raise NotImplementedError()

def method_2(self):
    """Doc string goes here."""
    raise NotImplementedError()

def method_3(self, param_1):
    """Doc string goes here."""
    raise NotImplementedError()

load_components(Core)
</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-componentsinitpy"} -->

<!-- wp:loos-hcb/code-block {"langType":"python","langName":"Python","fileName":"components/__init__.py","preClass":"prism undefined-numbers lang-python"} -->
<pre class="prism undefined-numbers lang-python" data-file="components/__init__.py" data-lang="Python"><code>from .component_1 import load_component1
from .component_2 import load_component_2

def load_components(core):

load_component_1(core)

load_component_2(core)
</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-componentscomponent1py"} -->

<!-- wp:loos-hcb/code-block {"langType":"python","langName":"Python","fileName":"components/component_1.py","preClass":"prism undefined-numbers lang-python"} -->
<pre class="prism undefined-numbers lang-python" data-file="components/component_1.py" data-lang="Python"><code>def load_contact(core):
    core.method_1 = method_1
    core.method_2 = method_2

def method_1(self, param_1, param_2, param_3):

Actual implementation

...

def method_2(self):

Actual implementation

...
</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-componentscomponent2py"} -->

<!-- wp:loos-hcb/code-block {"langType":"python","langName":"Python","fileName":"components/component_2.py","preClass":"prism undefined-numbers lang-python"} -->

<pre class="prism undefined-numbers lang-python" data-file="components/component_2.py" data-lang="Python"><code>def load_contact(core):
    core.method_3 = method_3

def method_3(self, param_1):
# Actual implementation
...
</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- /wp:ghostkit/tabs-v2 -->

<!-- wp:paragraph -->

I thought to myself, why can’ t I just make some more classes and let them reference each other? Turns out that worked pretty well for me. I split my functions into several different manager” classes, each of which is initialized with a reference to the main class. These classes are instantiated in topological order such that classes being referred to by others are created arlier. In ETM, the classes that are being referred to are usually those data providers utilities, namely <code>ExperimentalFlagsManager</code>, <code>DatabaseManager</code>, and <code>TelegramBotManager</code>.
<!-- /wp:paragraph -->

<!-- wp:ghostkit/tabs-v2 {"tabActive":"tab-initpy","tabsData":[{"slug":"tab-initpy","title":"\u003ccode\u003e__init__.py\u003c/code\u003e"},{"slug":"tab-flagspy","title":"\u003ccode\u003eflags.py\u003c/code\u003e"},{"slug":"tab-dbpy","title":"\u003ccode\u003edb.py\u003c/code\u003e"},{"slug":"tab-chatbindingpy","title":"\u003ccode\u003echat_binding.py\u003c/code\u003e"}],"className":"is-style-pills"} -->

<code>__init__.py</code><code>flags.py</code><code>db.py</code><code>chat_binding.py</code> <!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-initpy"} -->
<!-- wp:loos-hcb/code-block {"langType":"python","langName":"Python","fileName":"__init__.py","preClass":"prism undefined-numbers lang-python"} -->
<pre class="prism undefined-numbers lang-python" data-file="__init_.py" data-lang="Python"><code>from .flags import ExperimentalFlagsManager
from .db import DatabaseManager
from .chat_binding import ChatBindingManager

class TelegramChannel():
def **init**(self):

self.flags: ExperimentalFlagsManager = ExperimentalFlagsManager(self)

self.db: DatabaseManager = DatabaseManager(self)

self.chat_binding: ChatBindingManager = ChatBindingManager(self)</code></pre>
/div>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-flagspy"} -->

<!-- wp:loos-hcb/code-block {"langType":
python","langName":"Python","fileName":"flags.py","preClass":"prism undefined-numbers lan
-python"} -->
<pre class="prism undefined-numbers lang-python" data-file="flags
py" data-lang="Python"><code>from typing import TYPE_CHECKING

if TYPE_CHECKING:

Avoid cycle import for type checking

from . import TelegramChannel

class ExperimentalFlagsManager:

def **init**(channel: 'TelegramChannel'
:

self.channel = channel

...</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-dbpy"} -->

<!-- wp:loos-hcb/code-block {"langType":"py
hon","langName":"Python","fileName":"db.py","preClass":"prism undefined-numbers lang-pyt
on"} -->
<pre class="prism undefined-numbers lang-python" data-file="db.p
" data-lang="Python"><code>from typing import TYPE_CHECKING
from .flags import ExperimentalFlagsManager

if TYPE_CHECKING:

Avoid cycle import for type checking

from . import TelegramChannel

class DatabaseManager:

def **init**(channel: 'TelegramChannel'

:

self.channel: 'TelegramChannel'
= channel

self.flags: ExperimentalFlagsManager = channel.flags

...</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- wp:ghostkit/tabs-tab-v2 {"slug":"tab-chatbindingpy"} -->

<!-- wp:loos-hcb/code-block {"lan
Type":"python","langName":"Python","fileName":"chat_binding.py","preClass":"prism undefin
d-numbers lang-python"} -->
<pre class="prism undefined-numbers lang-python" data-file="chat
binding.py" data-lang="Python"><code>from typing import TYPE_CHECKING
from .chat_binding import ChatBindingManager
from .db import DatabaseManager

if TYPE_CHECKING:

Avoid cycle import for type checking

from . import TelegramChannel

class ChatBindingManager:

def **init**(channel: 'TelegramChannel'
:

self.channel: 'TelegramChannel'
= channel

self.flags: ExperimentalFlagsManager = channel.flags

self.db: DatabaseManager = channel.db

...</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- /wp:ghostkit/tabs-tab-v2 -->

<!-- /wp:ghostkit/tabs-v2 -->

<!-- wp:paragraph -->

While going on refactoring ETM, I learnt that multiple inheritance in Python is also used i
another way – mixins. Mixins are classes that are useful when you want to add a set of featur
s to many other classes. This has enlightened me when I was trying to deal with constantly ad
ing references of the <code>gettext</code> translator in all manager classes.
<!-- /wp:paragraph -->

<!-- wp:paragraph -->

I added a mixin called <code>LocaleMixin</code> that extracts the translator functions <code>gettext</code> and <code>ngettext</code>) from the main class reference (assumi g they are guaranteed to be there), and assign a local property that reflects these methods.<
p>
<!-- /wp:paragraph -->

<!-- wp:loos-hcb/code-block {"langType":"python","langName":"Python","preClass":"prism u defined-numbers lang-python"} -->

<pre class="prism undefined-numbers lang-python" data-lang="Pyt on"><code>class LocaleMixin:
    channel: 'TelegramChannel'

    @property
    def _(self):
        return self.channel.gettext

    @property
    def ngettext(self):
        return self.channel.ngettext</code></pre>

<!-- /wp:loos-hcb/code-block -->

<!-- wp:paragraph -->

When the mixin classes is added to the list of inherited classes, the IDE can properly reco nise these helper properties, and their definitions are consolidated in the same place. I find it more organised that the previous style.
<!-- /wp:paragraph -->

<!-- wp:separator {"className":"is-style-default"} -->

<hr class="wp-block-separator is-style-default"/>
<!-- /wp:separator -->

<!-- wp:paragraph -->

In the end, I find that simply creating classes for each component of my code turns out to be the most organised, and IDE-friendly way to breakdown a large class, and mixins are helpf l to make references or helper functions available to multiple classes.
<!-- /wp:paragraph -->