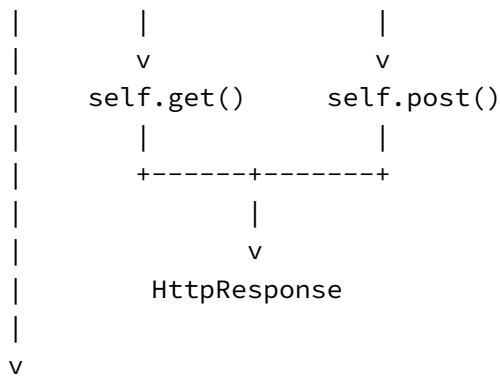

Django class-based views: the dispatch trap

Django comes with class-based views which provide convenient way to handle requests. You can overload only certain parts in request processing or the whole process part by part if needed. One such method is `dispatch`.

Dispatch method is basically the base where input is a request and output is a response. Everything in between is pure request handling. Thus this method sits (almost) on top of the whole process and delegates the work further.

Request -> response path:

```
URLconf
|
| path("example/", MyView.as_view())
|
v
MyView.as_view()                                [classmethod]
|
| Returns a callable view
| (normally created when URLconf loads)
|
|     HTTP request arrives
|         |
|         v
|     view(request, *args, **kwargs)
|         |
|         v
|     self = MyView(**initkwargs)
|         |
|         v
|     self.setup(request, *args, **kwargs)
|         |
|         | Initializes:
|         | self.request
|         | self.args
|         | self.kwargs
|         v
|     self.dispatch(request, *args, **kwargs)
|         |
|         | Selects handler using request.method
|         |
|         +-----+-----+
```



Response returned to Django

Detail on dispatch method:

dispatch() begins

```

|
+--> Before handler
|
+--> super().dispatch()
|       |
|       +--> get()
|               |
|               +--> HttpResponse
|
+--> After handler
|
+--> return response

```

Important point is `super().dispatch()`. Why? Because whatever stands/happens before this line is evaluated always. That is an important piece of information. For example for views gated with sign-in check (no anonymous user access).

You might be tempted to write following code not knowing it contains a hidden bug:

```

class MyView(LoginRequiredMixin, View): # LoginRequiredMixin is from
↳ django-braces package
    def dispatch(self, request, *args, **kwargs):
        self.user_settings = UserSettings.objects.get(user=self.request.user)

        return super().dispatch(request, *args, **kwargs)

```

This code was intended to initialize a local variable as soon as possible. The thing here is that `UserSettings` model is queried before `super().dispatch()` is called thus before `LoginRequiredMixin.dispatch()` is called which checks whether the user is authenticated or not. That

causes a bug because in case of anonymous user (which certainly doesn't have any `UserSettings` record) where the `get()` method raises an exception.

The solution here is to put the logic into:

- a) cached property (request safe, cached per view instance):

```
class MyView(...):

    def get(self, request, *args, **kwargs):

        x = self.user_settings
        # do stuff ...
        y = self.user_settings
        # do stuff ...

        return HttpResponse()

    @cached_property
    def user_settings(self):
        return UserSettings.objects.get(user=self.request.user)
```

- b) explicit method called in

```
class MyView(...):

    def get(self, request, *args, **kwargs):
        self.load_user_settings()

        # do stuff ...

        return HttpResponse()

    def load_user_settings(self):
        self.user_settings = UserSettings.objects.get(user=self.request.user)
```

The conclusion here is that you should be aware of what you override and how it affects the lifecycle of what you are overriding.