Ask Company

리액트와 함께 장고 시작하기 / 장고 Views

장고 기본 CBV API (Generic display views)

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

Built-in CBV API

Base views

View, TemplateView, Redirect View

Generic display views

DetailView, ListView

Generic date views

ArchiveIndexView, YearArchiveView, MonthArchiveView, WeekArchiveView, DayArchiveView, TodayArchiveView, DateDetailView

Generic editing views

FormView, CreateView, UpdateView, DeleteView

https://docs.djangoproject.com/en/2.1/ref/class-based-views/

Generic display views

DetailView

- ← SingleObjectTemplateResponseMixin
  - ← TemplateResponseMixin
- ← BaseDetailView
  - ← SingleObjectMixin
  - ← View

ListView

- ← MultipleObjectTemplateResponseMixin
  - ← TemplateResponseMixin
- ← BaseListView
  - ← MultipleObjectMixin ← ContextMixin
  - ← View

https://docs.djangoproject.com/en/2.1/ref/class-based-views/generic-display/

DetailView

DetailView

1개 모델의 1개 Object에 대한 템플릿 처리

모델명소문자 이름의 Model Instance를 템플릿에 전달

지정 pk 혹은 slug에 대응하는 Model Instance

```
from django.views.generic import DetailView
from .models import Post


post_detail1 = DetailView.as_view(model=Post)


class PostDetailtView(DetailView):
    model = Post

post_detail2 = PostDetailtView.as_view()
```

DetailView 상속관계

django.views.generic.detail.DetailView

← SingleObjectTemplateResponseMixin

template_name이 지정되지 않았다면, 모델명으로 템플릿 경로 유추

← TemplateResponseMixin

← BaseDetailView

← SingleObjectMixin : url_kwarg로 지정된 Model Instance 획득

← ContextView

← View

```python
class SingleObjectMixin(ContextMixin):
    """
    Provide the ability to retrieve a single object for further manipulation.
    """
    model = None
    queryset = None
    slug_field = 'slug'
    context_object_name = None
    slug_url_kwarg = 'slug'
    pk_url_kwarg = 'pk'
    query_pk_and_slug = False

    def get_object(self, queryset=None):
        """
        Return the object the view is displaying.
        Require `self.queryset` and a `pk` or `slug` argument in the URLconf.
        Subclasses can override this to return any object.
        """
        # Use a custom queryset if provided; this is required for subclasses
        # like DateDetailView
        if queryset is None:
            queryset = self.get_queryset()

        # Next, try looking up by primary key.
        pk = self.kwargs.get(self.pk_url_kwarg)
        slug = self.kwargs.get(self.slug_url_kwarg)
        if pk is not None:
            queryset = queryset.filter(pk=pk)

        # Next, try looking up by slug.
        if slug is not None and (pk is None or self.query_pk_and_slug):
            slug_field = self.get_slug_field()
            queryset = queryset.filter(**{slug_field: slug})

        # If none of those are defined, it's an error.
        if pk is None and slug is None:
            raise AttributeError(
                "Generic detail view %s must be called with either an object "
                "pk or a slug in the URLconf." % self.__class__.__name__
            )

        try:
            # Get the single item from the filtered queryset
            obj = queryset.get()
        except queryset.model.DoesNotExist:
            raise Http404(_("No %(verbose_name)s found matching the query") %
                          {'verbose_name': queryset.model._meta.verbose_name})
        return obj
```

```python
    def get_queryset(self):
        """
        Return the `QuerySet` that will be used to look up the object.
        This method is called by the default implementation of get_object() and
        may not be called if get_object() is overridden.
        """
        if self.queryset is None:
            if self.model:
                return self.model._default_manager.all()
            else:
                raise ImproperlyConfigured(
                    "%(cls)s is missing a QuerySet. Define "
                    "%(cls)s.model, %(cls)s.queryset, or override "
                    "%(cls)s.get_queryset()." % {
                        'cls': self.__class__.__name__
                    }
                )
        return self.queryset.all()

    def get_slug_field(self):
        """Get the name of a slug field to be used to look up by slug."""
        return self.slug_field

    def get_context_object_name(self, obj):
        """Get the name to use for the object."""
        if self.context_object_name:
            return self.context_object_name
        elif isinstance(obj, models.Model):
            return obj._meta.model_name
        else:
            return None

    def get_context_data(self, **kwargs):
        """Insert the single object into the context dict."""
        context = {}
        if self.object:
            context['object'] = self.object
            context_object_name = self.get_context_object_name(self.object)
            if context_object_name:
                context[context_object_name] = self.object
            context.update(kwargs)
        return super().get_context_data(**context)
```

```python
class DetailView(SingleObjectTemplateResponseMixin, BaseDetailView):
    pass
```

https://github.com/django/django/blob/3.0.2/django/views/generic/detail.py

ListView

ListView

1개 모델에 대한 List 템플릿 처리

모델명소문자_list 이름의 QuerySet을 템플릿에 전달

페이징 처리 지원

```python
from django.views.generic import ListView
from .models import Post


post_list1 = ListView.as_view(model=Post)

post_list2 = ListView.as_view(model=Post, paginate_by=10)


class PostListView(ListView):
    model = Post
    paginate_by = 10

post_list3 = PostListView.as_view()


class PostListView(ListView):
    model = Post
    paginate_by = 10

    def get_queryset(self):
        qs = super().get_queryset()
        qs = qs.filter(...)
        return qs

post_list4 = PostListView.as_view()
```

ListView 상속관계

django.views.generic.list.ListView

← MultipleObjectTemplateResponseMixin

template_name이 지정되지 않았다면, 모델명으로 템플릿 경로 유추

← TemplateResponseMixin

← BaseListView

← MultipleObjectMixin : Paginator가 적용된 QuerySet 획득

← ContextMixin

← View

```python
class MultipleObjectMixin(ContextMixin):
    """A mixin for views manipulating multiple objects."""
    allow_empty = True
    queryset = None
    model = None
    paginate_by = None
    paginate_orphans = 0
    context_object_name = None
    paginator_class = Paginator
    page_kwarg = 'page'
    ordering = None

    def get_queryset(self):
        # ...
        return queryset

    def paginate_queryset(self, queryset, page_size):
        # ...
        return (paginator, page, page.object_list, page.has_other_pages())

    def get_context_data(self, *, object_list=None, **kwargs):
        """Get the context for this view."""
        queryset = object_list if object_list is not None else self.object_list
        page_size = self.get_paginate_by(queryset)
        context_object_name = self.get_context_object_name(queryset)
        if page_size:
            paginator, page, queryset, is_paginated = self.paginate_queryset(queryset, page_size)
            context = {
                'paginator': paginator,
                'page_obj': page,
                'is_paginated': is_paginated,
                'object_list': queryset
            }
        else:
            context = {
                'paginator': None,
                'page_obj': None,
                'is_paginated': False,
                'object_list': queryset
            }
        if context_object_name is not None:
            context[context_object_name] = queryset
            context.update(kwargs)
        return super().get_context_data(**context)
```

```python
class BaseListView(MultipleObjectMixin, View):
    """A base view for displaying a list of objects."""
    def get(self, request, *args, **kwargs):
        self.object_list = self.get_queryset()
        allow_empty = self.get_allow_empty()

        if not allow_empty:
            # When pagination is enabled and object_list is a queryset,
            # it's better to do a cheap query than to load the unpaginated
            # queryset in memory.
            if self.get_paginate_by(self.object_list) is not None \
                    and hasattr(self.object_list, 'exists'):
                is_empty = not self.object_list.exists()
            else:
                is_empty = not self.object_list
            if is_empty:
                raise Http404(_('Empty list and “%(class_name)s.allow_empty” is False.') % {
                    'class_name': self.__class__.__name__,
                })
        context = self.get_context_data()
        return self.render_to_response(context)
```

```python
class ListView(MultipleObjectTemplateResponseMixin, BaseListView):
    pass
```

https://github.com/django/django/blob/3.0.2/django/views/generic/list.py

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

-

Ask Company