DJANGO GENERIC DISPLAY VIEW

江琳

CENERIC VIEW

GENERIC VIEW

- **In some cases, writing views, as we have seen earlier is really heavy.**
- **Generic views take certain common idioms and patterns found in view development and abstract them so that you can quickly write common views of data without having to write too much code.**

GENERIC VIEW

- **Unlike classic views, generic views are classes not functions.**
- **Django offers a set of classes for generic views in django.views.generic, and every generic view is one of those classes or a class that inherits from one of them.**

GENERIC VIEW

- **Open the terminal and type below:**

```
>>> import django.views.generic
>>> dir(django.views.generic)
['ArchiveIndexView', 'CreateView',
'DateDetailView', 'DayArchiveView', 'DeleteView',
'DetailView', 'FormView', 'GenericViewError',
'ListView', 'MonthArchiveView', 'RedirectView',
'TemplateView', 'TodayArchiveView', 'UpdateView',
'View', 'WeekArchiveView', 'YearArchiveView',
'__all__', '__builtins__', '__cached__',
'__doc__', '__file__', '__loader__', '__name__',
'__package__', '__path__', '__spec__', 'base',
'dates', 'detail', 'edit', 'list']
```

GENERIC DISPLAY VIEW

GENERIC DISPLAY VIEW

- **Generic display views are designed to display data. On many projects they are typically the most commonly used views.**
  - **ListView**
  - **DetailView**

DETAILVIEW

DETAILVIEW

- **class django.views.generic.detail.DetailView**
- **DetailView uses to get and return a object from a model.**

DETAILVIEW

- **Let's modify the BookDetail view using DetailView**

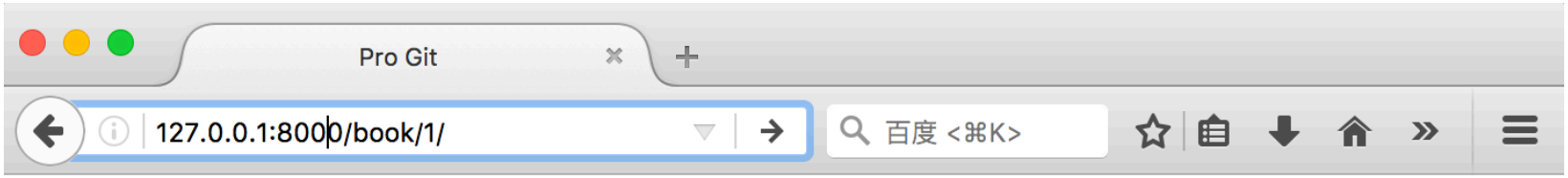
```
class BookDetail(DetailView):
    model = Book
    template_name = 'detail.html'

# in url.py
url(r'^book/(?P<pk>\d+)/', BookDetail.as_view(),
name='book_detail'),
```

DETAILVIEW

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>{{ object.name }}</title>
</head>
<body>
<h1>{{ object.name }}</h1>
<span>
    <b>Author:</b> {{ object.author }}
    <b>Publish by:</b>{{ object.publisher.name }}
    <b>Release at</b>:{{ object.pub_year }}
</span>
<h2>Introduction</h2>
<div>{{ object.intro }}</div>
</body>
</html>
```

DETAILVIEW



Pro Git

Author: Scott Chacon **Publish by:**Apress **Release at**:2009

Introduction

Git is the version control system developed by Linus Torvalds for Linux kernel development. It took the open source world by storm since its inception in 2005, and is used by small development shops and giants like Google, Red Hat, and IBM, and of course many open source projects. * A book by Git experts to turn you into a Git expert * Introduces the world of distributed version control * Shows how to build a Git development workflow What you'll learn * Use Git as a programmer or a project leader. * Become a fluent Git user. * Use distributed features of Git to the full. * Acquire the ability to insert Git in the development workflow. * Migrate programming projects from other SCMs to Git. * Learn how to extend Git. This book is for all open source developers: you are bound to encounter it somewhere in the course of your working life. Proprietary software developers will appreciate Git's enormous scalability, since it is used for the Linux project, which comprises thousands of developers and testers.

DETAILVIEW

- **Methods:**

dispatch()	http_method_not_allowed()
get_template_names()	get_slug_field()
get_queryset()	get_object()
get_context_object_name()	get_context_data()
get()	render_to_response()

DETAILVIEW

- **get_context_date()**
  - **Returns context data for displaying the object.**
- **It returns a dictionary with these contents:**
  - **object: The object that this view is displaying (self.object).**
  - **context_object_name: self.object will also be stored under the name returned by get_context_object_name(), which defaults to the lowercased version of the model name.**

DetailView

- In default

```
class BookDetail(DetailView):
    model = Book
    template_name = 'detail.html'

    def get_context_data(self, **kwargs):
        context = super(BookDetail,
self).get_context_data(**kwargs)
        return context
```

DetailView

- **If we want to passing extra field to template, we can write like below: (passing now to template)**

```
class BookDetail(DetailView):
    model = Book
    template_name = 'detail.html'

    def get_context_data(self, **kwargs):
        context = super(BookDetail,
self).get_context_data(**kwargs)
        context['now'] = datetime.now()
        return context
```

LISTVIEW

LIBRARY SYSTEM

- **Part one: Library System**
  - **Homepage**
  - **Book list page**
  - **Book detail page**
- **Part two: Library Management System**
  - **Login page**
  - **Create book, author, publisher, category**
  - **Book list**
  - **Update book**

LISTVIEW

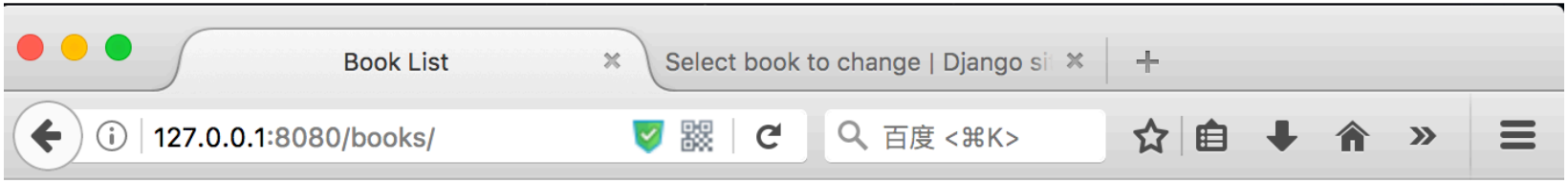
- **Let's write a page to list of books using ListView**

```
class BookList(ListView):
    model = Book
    template_name = 'list.html'
```

LISTVIEW

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Book List</title>
</head>
<body style="width: 80%">
<div style="text-align: center"><h1>Book List</h1></div>
{% for obj in object_list %}
    <span style="font-size: 20px;
                 font-weight: bold">
       {{ obj.name }}</span>
    <span>--{{ obj.author.name }}</span><br>
{% endfor %}
</body>
</html>
```

LISTVIEW



Book List

Pro Git --Scott Chacon

Learn Python the Hard Way --Zed A. Shaw

Python Web Development with Django --Jeff Forcier

Data Wrangling With Python --Jacqueline Kazil

Introduction to Machine Learning with Python --Sarah Guido

Foundations of Python Network Programming --Brandon Rhodes

Introducing Python --Introducing Python

Web Scraping with Python --Ryan Mitchell

LISTVIEW

- **methods**

dispatch()	**http_method_not_allowed()**
get_template_names()	**get()**
get_queryset()	**render_to_response()**
get_context_object_name()	**get_context_data()**

LISTVIEW

- **get_context_data()**
  - **Returns context data for displaying the list of objects.**

```
class BookList(ListView):
    model = Book
    template_name = 'list.html'

    def get_context_data(self, **kwargs):
        context = super(BookList,
self).get_context_data(**kwargs)
        context['now'] = datetime.now()
        return context
```

LISTVIEW

- **Context:**
  - **object_list: The list of objects that this view is displaying. If context_object_name is specified, that variable will also be set in the context, with the same value as object_list.**
  - **is_paginated: A boolean representing whether the results are paginated. Specifically, this is set to False if no page size has been specified, or if the available objects do not span multiple pages.**
  - **paginator: An instance of django.core.paginator.Paginator. If the page is not paginated, this context variable will be None.**
  - **page_obj: An instance of django.core.paginator.Page. If the page is not paginated, this context variable will be None.**

LISTVIEW

- **get_queryset()**
  - **Get the list of items for this view. This must be an iterable and may be a queryset**

```
class BookList(ListView):
    model = Book
    template_name = 'list.html'

    def get_queryset(self):
        queryset = super(BookList, self).get_queryset()
        return queryset
```

LISTVIEW

- **Now, we modify BookList view to get book list based on name of book**

```
class BookList(ListView):
    model = Book
    template_name = 'list.html'

    def get_queryset(self):
        name = self.request.GET.get('name')
        queryset = super(BookList, self).get_queryset()
        if name:
            queryset =
Book.objects.filter(name__icontains=name)
        return queryset
```

LISTVIEW

- **Now open link: 127.0.0.1/books/?name=python**

Book List

Learn Python the Hard Way --Zed A. Shaw

Python Web Development with Django --Jeff Forcier

Data Wrangling With Python --Jacqueline Kazil

Introduction to Machine Learning with Python --Sarah Guido

Foundations of Python Network Programming --Brandon Rhodes

Introducing Python --Introducing Python

Web Scraping with Python --Ryan Mitchell

LIBRARY SYSTEM

HOMEPAGE

- **Let modify the home view and html**

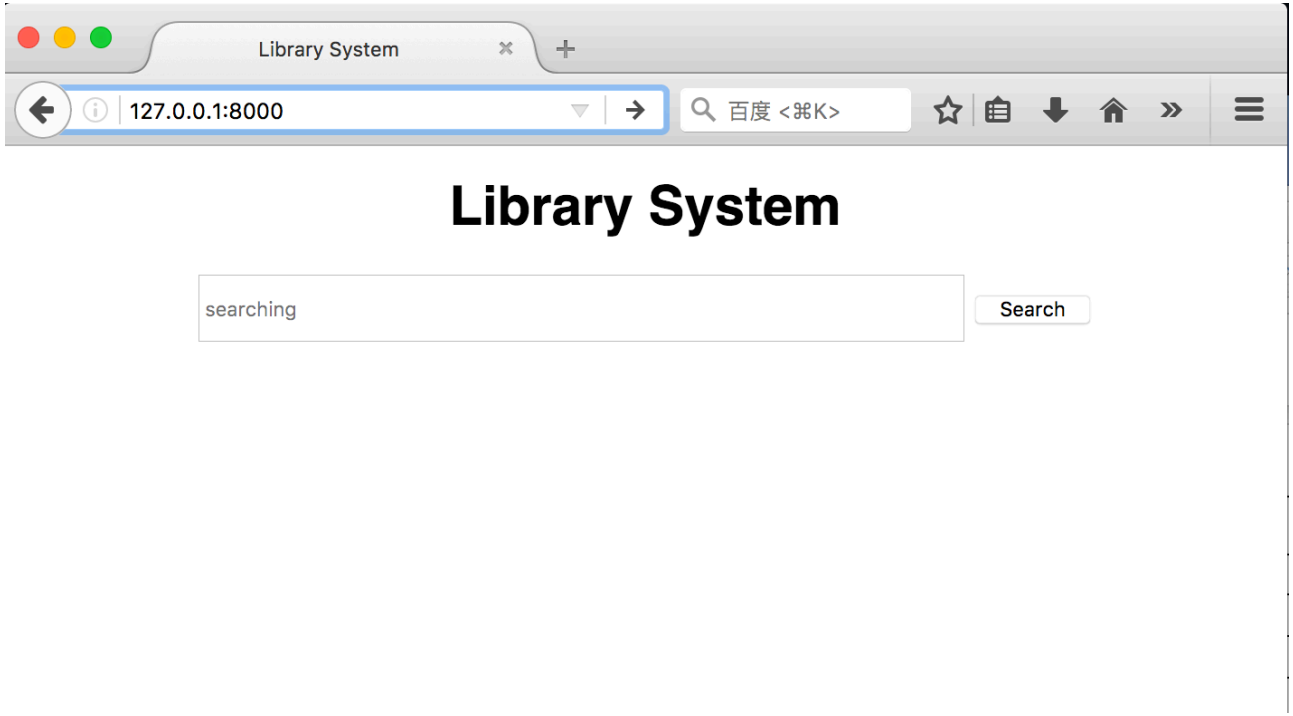
```
# home view
def home(request):
    return render(request, 'homepage.html')
```

HOMEPAGE

```
# homepage.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Library System</title>
</head>
<body style="text-align: center">
<h1>Library System</h1>
<form method="get" action="">
    <input type="text" style="width: 60%;height: 30px"
placeholder="searching">
    <input type="submit" value="Search">
</form>
</body>
</html>
```

HOMEPAGE

- **now we can search book using book's name on Homepage.**



NAMESPACING URL NAMES

- **Modify homepage.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Library System</title>
</head>
<body style="text-align: center">
<h1>Library System</h1>
<form method="get" action="{% url 'book_list' %}">
    <input type="text" style="width: 60%;height: 30px"
placeholder="searching" name="name">
    <input type="submit" value="Search">
</form>
</body>
</html>
```

NAMESPACING URL NAMES

- **Modify list.html**

```
<body>
<div style="text-align: center"><h1>Book
List</h1></div>
{% for obj in object_list %}
    <a href="{% url 'book_detail' pk=obj.id %}">
        <span style="font-size: 20px;font-weight:
bold">{{ obj.name }}</span>
        <span>--{{ obj.author.name }}</span><br>
    </a>
{% endfor %}
</body>
```

Questions?