Django

～ 情報の伝達に齟齬が発生するかもしれない。でも、聞いて ～

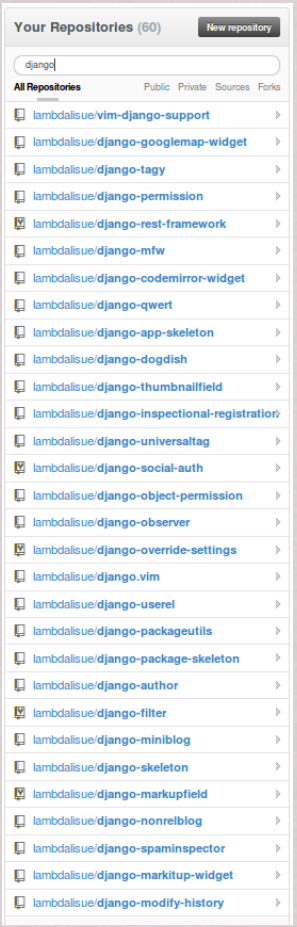
自己紹介

- Λlisue (ありすえ)
- KawazのWebサイト監督
- プログラミングは趣味です
  - 知識の偏りがひどい
  - 好きこそものの上手なれ
- 本業は生物屋さんです
  - バイオインフォじゃない
  - PCなにそれ美味しいの

今日の目的

- Django って面白いなって思ってもらう
  - スピーディーな開発
  - 豊富な拡張機能
- Kawaz って面白そうだなって思ってもらう
  - 札幌でゲームを作りたいなら、是非
  - 異種混合。絵師とプログラマが議論したりしてます
  - 異性発見。Webデザイナと3Dモデラーが出会ったりするかもしれません
- あわよくば Kawaz のポータルの作り直し時に戦力と（ry
  - 監督は京都在住
  - Pythonを使える人すら少ない
  - Ruby on rails には負けたくない

VimとDjangoと私



- GithubがDjangoで埋め尽くされる
- Djangoを快適に動かすためにVimプラグインも書いた
  - ただし、Pythonでな！ :-p
- ただの制作物に興味はない
- この中に Vim, Python, Django に当てはまる属性の人がいたら私のところに来なさい

Prologue

～ ディージャンゴじゃないよ ～

Djangoの特徴

- DRY (Don't Repeat Yourself)
  - 「再利用」を基本コンセプトとする -> スピーディーな開発
- MTV framework
  - Model, Template, View による見通しの良い設計
- Automatic Admin Interface
  - 完璧な管理ページの自動生成 -> スピーディーな開発
- All In One
  - 基本構造は外部ライブラリに依存しない

Djangoの特徴

- DRY (Don't Repeat Yourself)
  - 「再利用」を基本コンセプトとする -> スピーディーな開発
- MTV framework
  - Model, Template, View による見通しの良い設計
- Automatic Admin Interface
  - 完璧な管理ページの自動生成 -> スピーディーな開発
- All In One
  - 基本構造は外部ライブラリに依存しない

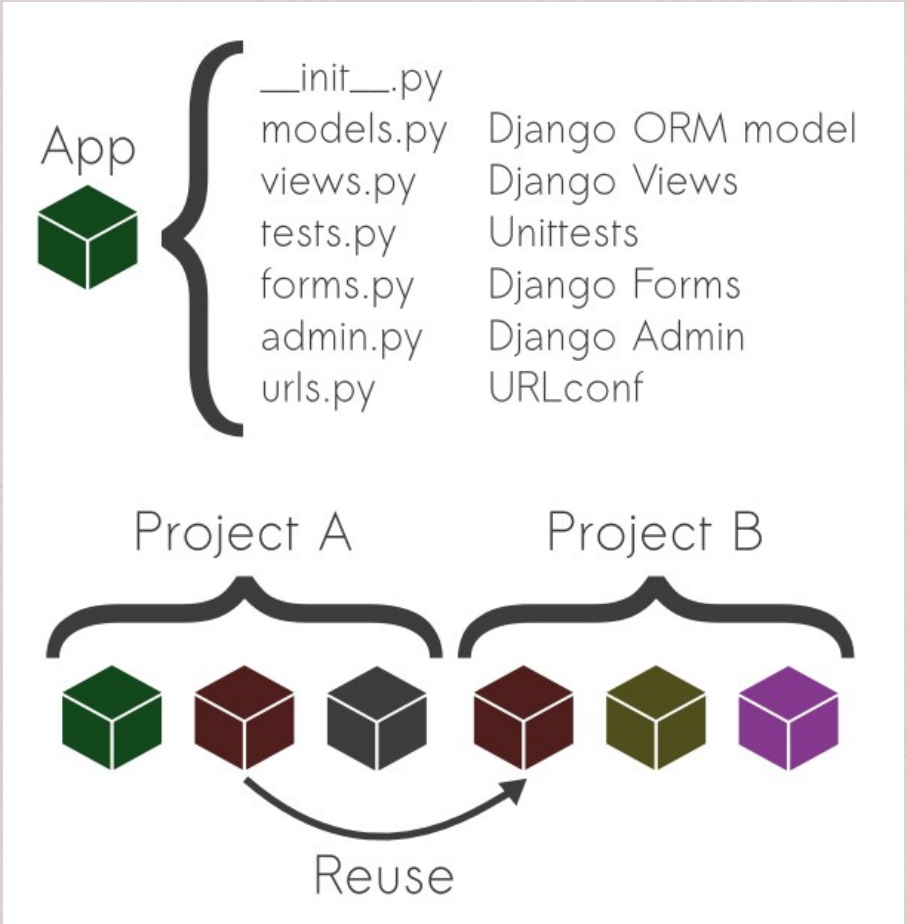
Djangoを使用している有名サイト

約4000サイトが知られている

Don't Repeat Yourself

～ おじいちゃん、夕飯はさっき食べたでしょ ～

App という機能単位は最利用可能



Django Packages

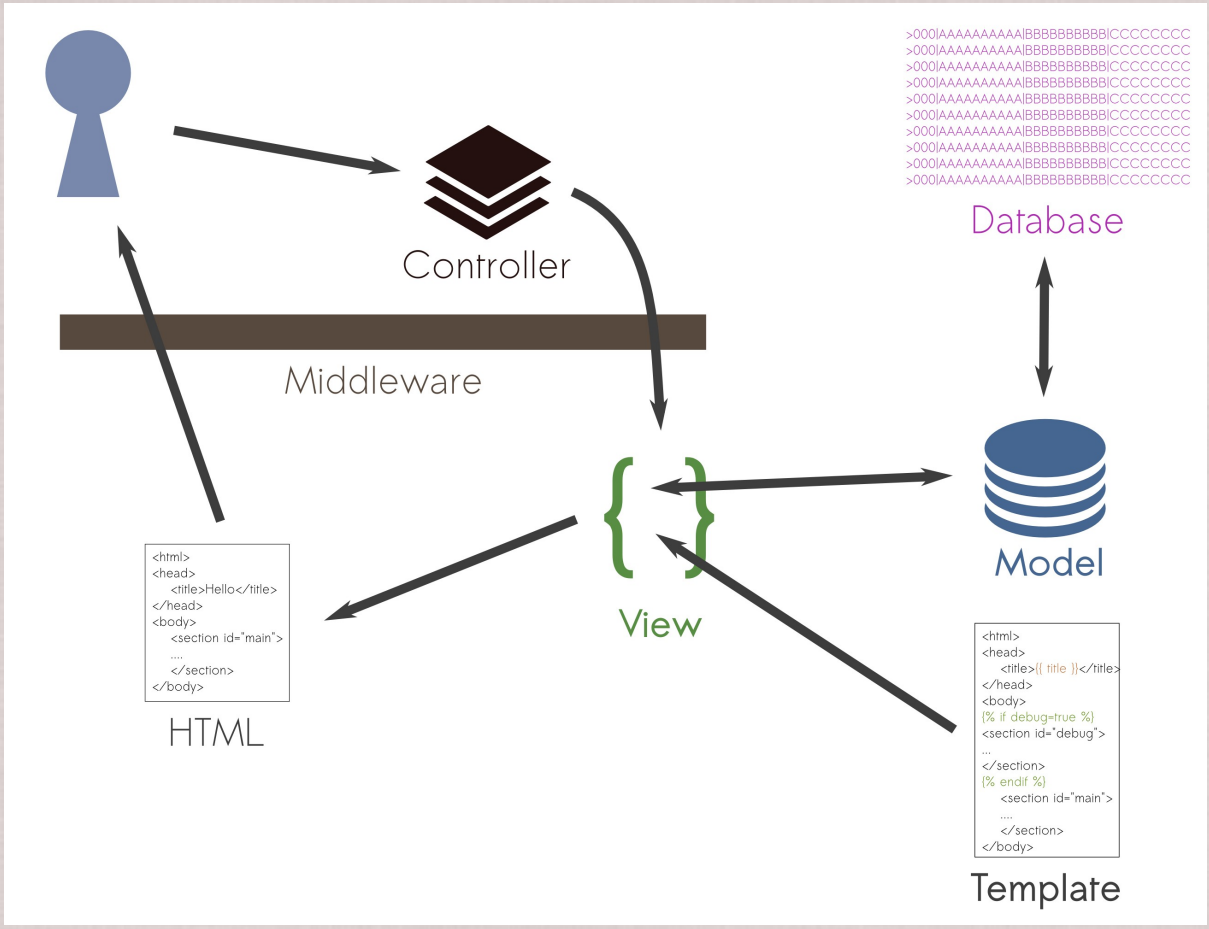
http://www.djangopackages.com/

- 1000を超えるDjangoの拡張パッケージが収録されている
- パッケージは基本的に Apps
  - Apps ~ 1103 個
  - Frameworks ~ 56 個
  - Other ~ 175 個
  - Projects ~ 49 個
- 僕も幾つかパッケージを登録して有ります :-)

MTV framework

～ 俺のMTVは革命（レボリューション）だ！ ～

MTV framework とは



Model

- Object-relational model
  - Pythonオブジェクトを扱うだけでデーターベースの操作が可能
  - オブジェクト指向においてDBを操作する際に便利
- django.db.models.Model を継承するだけ
  - DBカラムは models.XXXXXField でクラスに追加

Model 例

```python
# vim: set fileencoding=utf-8 :
from django.db import models
from django.utils.text import ugettext_lazy as _


class Article(models.Model):
    """Article model class

    Attributes:
        title
            A title of the article
        body
            A body of the article
    """
    title = models.CharField(_('title'), max_length=20)
    body = models.TextField(_('body'))

    def __unicode__(self):
        return self.title
```

- 必須部分は4行のみ
- Pythonのクラスなので
  Pythonの能力が発揮可能
- __unicode__ による
  非ASCII言語のサポート

View

- 特定のURLに対する特定の動作
- MVCにおける Controller 的存在
  - 渡されたURLからどのViewを呼び出すか？はDjangoが行う
  - URLは正規表現で登録
- Template に渡すデータの取得・加工は主にViewで
  - Template内でデータ取得・加工は極力しない
  - どうしてもしたい場合は自作Templatetag

Generic View

- Viewは決まりきった処理が多い
  - 決まりきった処理は標準で実装済み (DRY)
  - したがってDjangoにおいてViewを書くことは殆ど無い
- 以下の処理がよく用いられる
  - ListView ~ リスト表示
  - DetailView ~ 詳細表示
  - CreateView ~ 新規作成
  - UpdateView ~ 更新
  - DeleteView ~ 削除

View 例

```
1  # vim: set fileencoding=utf-8 :
2  from django.views.generic import ListView
3  from django.views.generic import DetailView
4
5  from blogs.models import Article
6
7
8  class ArticleListView(ListView):
9      model = Article
10
11
12 class ArticleDetailView(DetailView):
13     model = Article
```

- 必要部分は7行のみ
  - リスト表示・詳細表示
- Django 1.3 より Classbased に移行
  - 強力な汎用性
- ほとんど書く必要はない
  - この程度ならマッピング時に書いてもよいが…

Template

- データ -> HTML を担う部分
- Django 独自の言語を使用
  - デザイナーにもわかりやすい…？
  - 人気があったため、より強力にしたフォークが存在
    - Jinja テンプレートと呼ばれ Jinja2 は非常に強力
- テンプレートエンジンは容易に切り替え可能
  - 標準は非力なので大規模開発の場合は切り替えを推奨
  - 個人的おすすめは Jinjya2 もしくは Mako

Templatetag

- Django は基本的にテンプレート内で以下は行えない
  - データの取得
  - データの加工
- どうしても必要な場合は独自Templatetagを作る
  - 正直この辺のドキュメントは十分とは言えない
  - ただし、Djangoのソースコードはわかりやすい

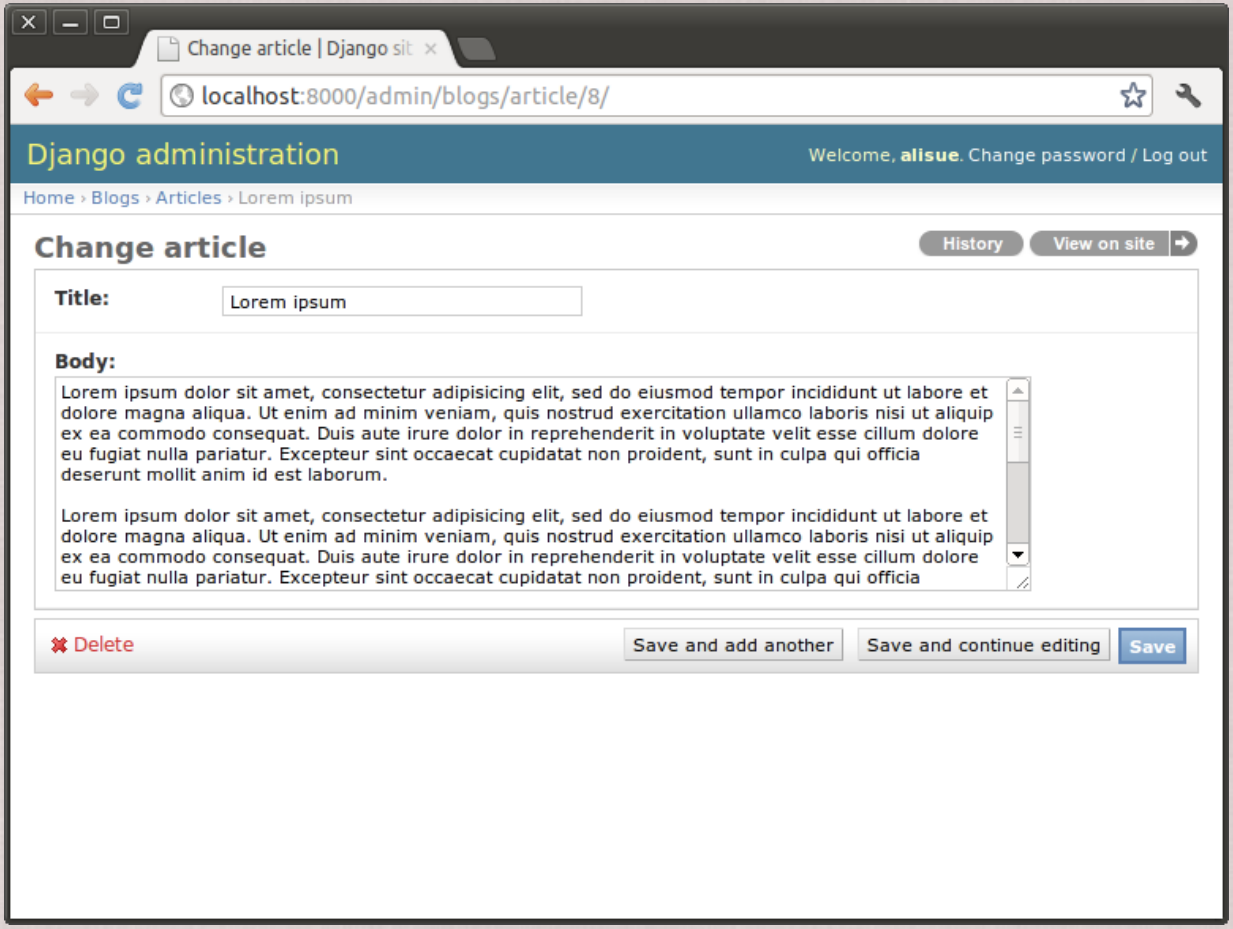
Admin Interface

～ 君は今、楽太郎の前にいるのだよ ～

Admin Interface

- Django は管理インターフェイスを全自動で作成できる
  - これが Django が爆発的人気を誇る理由
  - Rails陣から羨ましがられたのは有名（今はあるのかな？）
- 自動生成だが、カスタマイズ性が高い
  - デザインさえ気にしないのであれば正直管理インターフェイスだけでもWebサイトとして機能する
- 管理インターフェイスを作る必要がないので開発が楽 & 爆速
  - 自分専用のブログならば記事管理ページはお任せすれば良い
  - ユーザーごとに権限を変えられるため様々な役職の人間で管理可能

Admin Interface 例



All In One

～ 一人はみんなのために、みんなは一人のために ～

Non Mega framework

- 既存の良いライブラリを使い、自身は糊として働くものを目がフレームワークと呼ぶ
  - Pyramid (Pylons)
  - TurboGears
- 既存ライブラリを使用しているため使い勝手は良く、潜在能力は高い
  - PylonsはMako, SQLAlchemyなどを利用
- しかし、ライブラリ同士の依存関係に弱い
  - Pylonsで昔作ったものを動かそうとすると泣く
- Djangoは基幹部分はすべてオリジナル
  - オリジナル以外にも外部ライブラリはコピーを内部に持つなどしている
  - そのおかげで依存せずに機能を使用可能
- そのおかげでDjangoプロジェクトの信頼性は高い

Minilog

A small Django weblog sample application

作り方概要

- プロジェクトを作る
  - settings.py で初期設定
- アプリを作る
  - models.py にモデル記載
  - views.py にビュー記載
  - urls.py でマッピング
  - base.html / article_list.html / article_detail.html を書く
- プラグイン等加え最後の微調整
- CSS でスタイルをつける

新規プロジェクト作成

```
$ python manage.py startproject Minilog
```

settings.py で初期設定

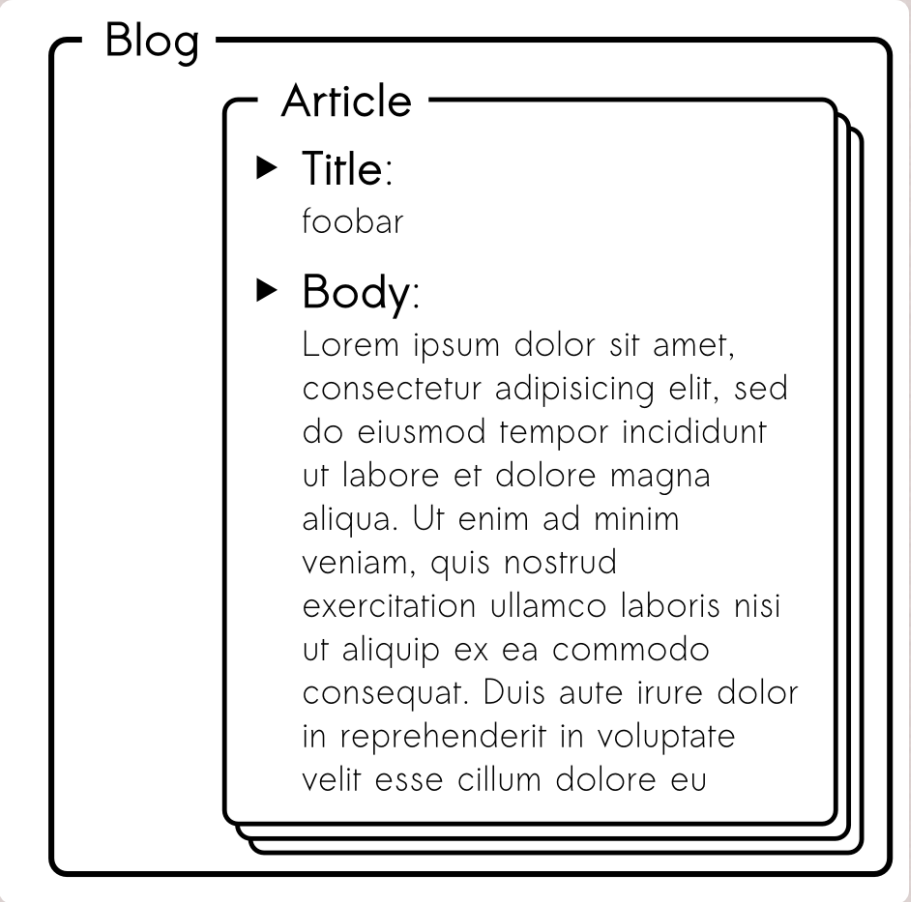
- ROOT
  - 設定ファイルないでパスが必要なことが多いので個人的に毎度実行パスから取得している
- DATABASES
  - 使用するデータベースタイプを指定
  - sqlite3 が開発には手軽で便利
- STATICFILES_DIRS
  - Django 1.3 から登場した STATIC FILES の設定
  - プロジェクトレベルでの静的ファイル保存場所を指定
- TEMPLATE_DIRS
  - プロジェクトレベルでのテンプレートファイルを探す場所を指定
  - この場所に保存されているテンプレートは最も優先される

新規アプリ作成

```
$ python manage.py startapp blogs
```

Model 作成

- 「ブログ」とは「記事」の集まりである
- 「記事」とは「題名」と「本文」を持つ



Articleモデル作成

```
# vim: set fileencoding=utf-8 :
from django.db import models
from django.utils.text import ugettext_lazy as _


class Article(models.Model):
    """Article model class

    Attributes:
        title
            A title of the article
        body
            A body of the article
    """
    title = models.CharField(_('title'), max_length=20)
    body = models.TextField(_('body'))

    def __unicode__(self):
        return self.title
```

Database更新

$ python manage.py syncdb

Would you like to create one now? (yes/no): yes

自動生成されるSQL

```
class Article(models.Model):
    title = models.CharField(max_length=20)
    body = models.TextField()
```

```
CREATE TABLE “blogs_article” (
    “id” integer NOT NULL PRIMARY KEY,
    “title” varchar(20) NOT NULL,
    “body” text NOT NULL
);
```

View作成

- ブログに絶対必要なのはCRUD
  - ブログ記事の作成
  - ブログ記事の表示
  - ブログ記事の更新
  - ブログ記事の削除
- 作成・更新・削除は Admin Interface にお任せする
- なので、作る必要があるのは表示部分。具体的には
  - ブログ記事の一覧表示 (ListView)
  - ブログ記事の詳細表示 (DetailView)

実際のコード

```
 1 # vim: set fileencoding=utf-8 :
 2 from django.views.generic import ListView
 3 from django.views.generic import DetailView
 4
 5 from blogs.models import Article
 6
 7
 8 class ArticleListView(ListView):
 9     model = Article
10
11
12 class ArticleDetailView(DetailView):
13     model = Article
```

urls.py

- URLとViewとのマッピングは urls.py 内に正規表現で記載
- 各アプリ毎に urls.py を作り、最後にプロジェクトの urls.py で include するのが作法
  - 極力プロジェクトの urls.py には include 以外は書かない
- URLがマッチするとそれ以後のマッピングは無視されるので正規表現は確実に!
  - ^ や $ の書き忘れで汎用的なURLマッピングになってハマる
  - URLのマッピング順番も重要!マッピングは上から処理される

URLにViewをマッピング

```
from django.conf.urls import patterns, url

import views
urlpatterns = patterns('',
    url(r'^$', views.ArticleListView.as_view(), name='blogs-article-list'),
    url(r'^(?P<pk>\d+)/$', views.ArticleDetailView.as_view(),
        name='blogs-article-detail'),
)
```

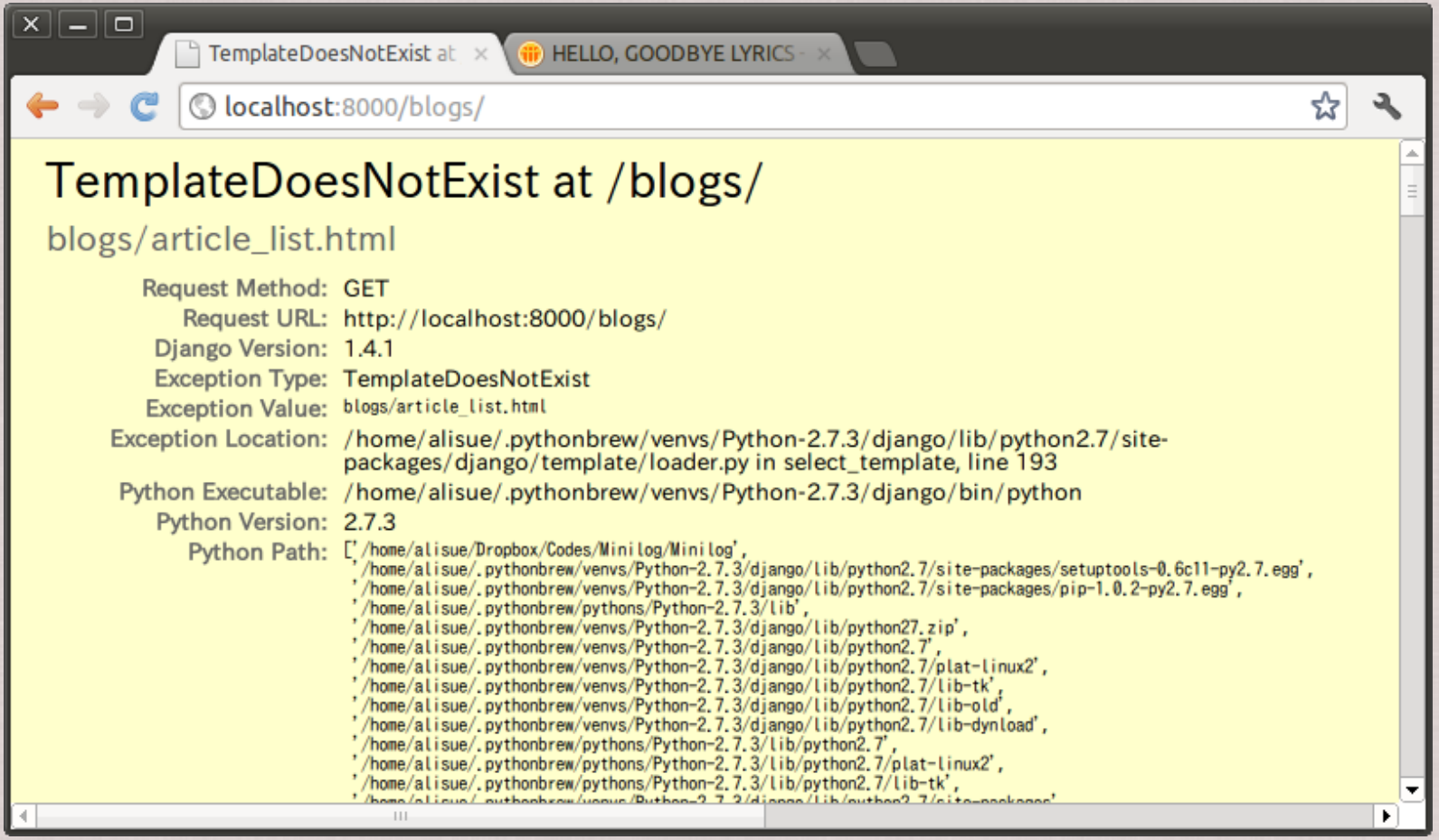
```
from django.conf.urls import patterns, include, url
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('blogs.urls')),
)
```

テストサーバ起動

```
$ python manage.py runserver
```

アクセス結果


TemplateDoesNotExist at
HELLO, GOODBYE LYRICS -
localhost:8000/blogs/
TemplateDoesNotExist at /blogs/
blogs/article_list.html
Request Method: GET
Request URL: http://localhost:8000/blogs/
Django Version: 1.4.1
Exception Type: TemplateDoesNotExist
Exception Value: blogs/article_list.html
Exception Location: /home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/site-packages/django/template/loader.py in select_template, line 193
Python Executable: /home/alisue/.pythonbrew/venvs/Python-2.7.3/django/bin/python
Python Version: 2.7.3
Python Path: ['/home/alisue/Dropbox/Codes/Minilog/Minilog',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/site-packages/setuptools-0.6c11-py2.7.egg',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/site-packages/pip-1.0.2-py2.7.egg',
'/home/alisue/.pythonbrew/pythons/Python-2.7.3/lib',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python27.zip',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/plat-linux2',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/lib-tk',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/lib-old',
'/home/alisue/.pythonbrew/venvs/Python-2.7.3/django/lib/python2.7/lib-dynload',
'/home/alisue/.pythonbrew/pythons/Python-2.7.3/lib/python2.7',
'/home/alisue/.pythonbrew/pythons/Python-2.7.3/lib/python2.7/plat-linux2',
'/home/alisue/.pythonbrew/pythons/Python-2.7.3/lib/python2.7/lib-tk',

base.html

- 基本的にWebサイト内のデザインは一緒
- 骨となる base.html を作り extends “base.html” で継承
- 個人的には各アプリ毎にも base.html を作ることを推奨

base.html の例

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>{% block title %}Minilog{% blockend %}</title>
    {% block head %}
    {% endblock %}
</head>
<body>
    <div id="wrapper">
        <header>
            <h1>Minilog</h1>
            <h2>Djangoサンプルブログ</h2>
        </header>
        <section id="main">
            {% block content %}
            {% endblock %}
        </section>
        <footer>
            <p>Copyright &copy; 2012, hashnote.net, Alisue allright reserved.</p>
        </footer>
    </div>
</body>
</html>
```

article_list.html & article_detail.html

```
{% extends 'base.html' %}
{% load markup %}
{% load pagination_tags %}

{% block title %}Minilog - ブログ記事一覧{% endblock %}

{% block content %}
<section id="article-list">
    <h1>Articles</h1>
    {% autopaginate object_list 6 %}
    {% paginate %}
    {% for object in object_list %}
    <article class="article {% cycle 'odd' 'even' %}">
        <h1 class="title"><a href="{{ object.get_absolute_url }}">{{ object.title }}</a></h1>
        <section class="body">
            {% comment %}
            英語などスペース区切りの言語では truncatewords が使用できるが
            日本語では使用できないため、代わりに truncatechars を使用。

            なお truncatechars は Django 1.4 からの機能なので、それ以前の
            Django を使用している場合は django-qwert 内に truncateletters.py
            という truncatechars とほぼ同等の処理を行うものがあるのでこちらを利用して
            いただきたい。また truncalechars は HTML に対応していないが truncateletters
            は HTML にも対応し、タグ以外の部分で truncate するという事もできる

            Ref: https://github.com/lambdalisue/django-qwert/blob/master/qwert/templatetags/truncateletters.py
            {% endcomment %}
            {{ object.body|markdown|truncatechars:200 }}
        </section>
    </article>
    {% empty %}
    <p>まだブログが投稿されていないようです</p>
    {% endfor %}
</section>
{% paginate %}
{% endblock %}
```

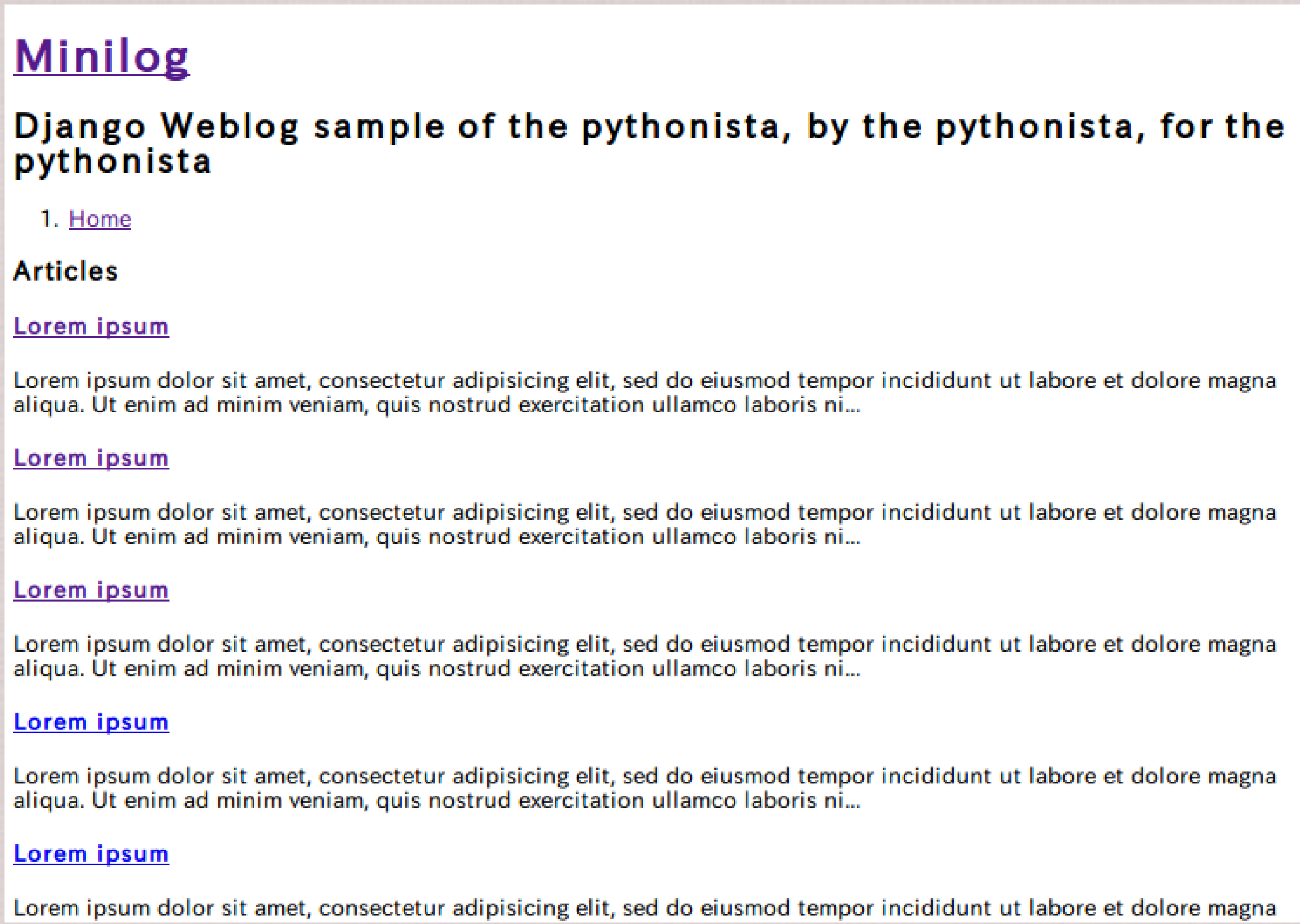
```
{% extends 'base.html' %}
{% load markup %}

{% block title %}Minilog - {{ object.title }}{% endblock %}

{% block breadcrumbs %}
<li><a href="{{ object.get_absolute_url }}">{{ object.title }}</a></li>
{% endblock %}

{% block content %}
<article class="article">
    <h1 class="title">{{ object.title }}</h1>
    <section class="body">
        {{ object.body|markdown }}
    </section>
</article>
{% endblock %}
```

アクセス結果


Minilog
Django Weblog sample of the pythonista, by the pythonista, for the pythonista
1. Home
Articles
Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris ni...
Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris ni...
Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris ni...
Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris ni...
Lorem ipsum
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna

Plugins

～ プラグインを使ってもっと便利にする ～

Pagination

~ 足切り! ~

ページネーション

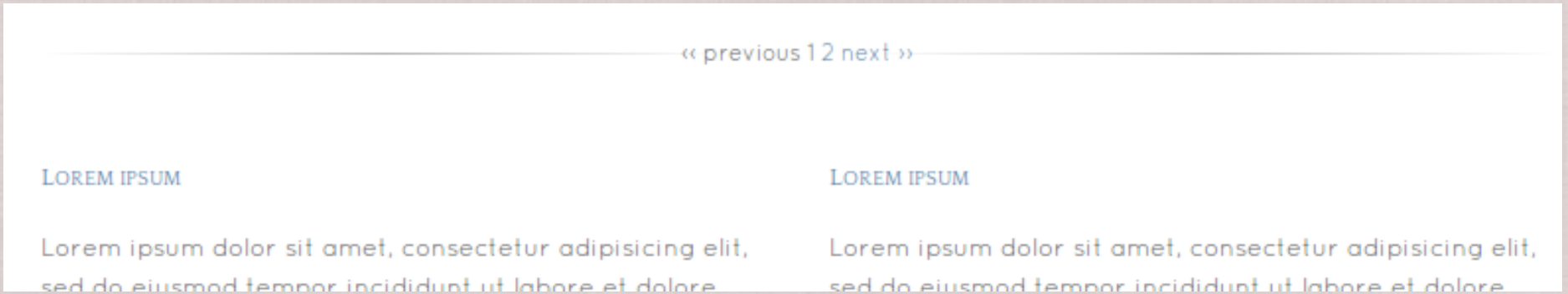
- 大量のリストを1ページに表示するのは(・A・)ｲｸﾅｲ!!
- ページネーションは地味に面倒な仕事
- Djangoにはデフォルトでページネーションがあるが…
  - DOMをデザインするのが面倒
  - 第一見た目の問題なのにViewに書くとか面倒
  - もっとシンプルなのがほしい

django-pagination

- Template内でページネーションが可能
- {% autopagination object_list 10 %} のように指定
- {% paginate %} でページネーションのDOMを構築

django-pagination 例

```
10    {% autopaginate object_list 6 %}
11    {% paginate %}
12    {% for object in object_list %}
13    <article class="article {% cycle 'odd' 'even' %}">
14        <h1 class="title"><a href="{{ object.get_absolute_url }}">{{ object.title }}</a></h1>
```



Minify と CoffeeScript と Less

~ JavaScript がダメなら CoffeeScript で書けばいいでしょ？ ~

開発陣の悩み

- JavaScript, CSS はデプロイ時には Minify するのが普通
  - minify を忘れ動かないと嘆いたり…
  - minify することで特有のバグが発生したり…
- JavaScript, CSS は辛い
  - CoffeeScriptで書くことで生産性アップ
  - Less (or Scss) を使うことで再利用性アップ

django-compressor

- 簡単な設定で Minify が可能
  - {% compress %} {% endcompress %} で囲むだけ
  - 囲まれていれば外部ファイルだろうが Minify してくれる
- CoffeeScript, Less, Sass などに事前コンパイル可能
  - type="text/coffeescript" などをブラウザに渡す前にサーバーサイドで自動的に処理
- 大規模プロジェクトには必須のプラグイン

django-compressor 例

```
{% compress css %}
<link rel="stylesheet" href="{{ STATIC_URL }}lib/WebSymbols/stylesheet.css">
<link type='text/less' rel="stylesheet" href="{{ STATIC_URL }}less/style.less">
<link type='text/less' rel="stylesheet" href="{{ STATIC_URL }}less/typography.less">
<link type='text/less' rel="stylesheet" href="{{ STATIC_URL }}less/pagination.less">
{% endcompress %}
```

- このように記載すると自動的に LESS -> CSS を行う
- CSS は 自動的に1ファイルに Minify される
  - Minify は YUI Compressor や CSSTidy など好きなものを用いることが可能
- キャッシュを使うので速度は問題ない
- JavaScript, CoffeeScript でも同様

Pagination と Less を追加したものが

最後に

- 今日のスライド・作ったブログの全ソースコードはGithub
  - Licenseとしては MIT Licenseとしておきます
  - 説明出来なかったところなど、コメントで説明しておきました
  - 参考にしたり改造して使ってもらって構いません
  - っというか改造して遊んで欲しいです、せっかくなので（笑
- URL: https://github.com/lambdalisue/Minilog

宣伝

- Kawaz知ってますか？
  - ゲーム制作を通してお互いに切磋琢磨しよう的な何かです
  - 「すごい人」がこっそり居たりするので有益です
  - 完全に自由なので気兼ねなく
- KawazはDjango製です
  - Django 1.1-1.2時代の物
  - 無理やり部分が多いので直したい
  - ポータル作成興味無いですか？

おしまい