

## Fazendo Mais com o Admin do Django

### Três Maneiras de Você Customizar este Poderoso Aplicativo para Atender às Suas Necessidades

Liza Daly

Software Engineer and Owner  
Threepress Consulting Inc.

26/Mai/2009

O console de administração integrado fornecido pelo Django é um de seus maiores pontos de vendas. E se você precisar customizar mais do que apenas uma aparência e um comportamento e alguns campos modelos? Saiba como estender o aplicativo de administração existente sem modificar a origem.

## O Admin do Django

O Django oferece muitos recursos para possíveis desenvolvedores: uma biblioteca padrão madura, uma comunidade de usuários ativa e todos os benefícios da linguagem Python. Enquanto outras estruturas da Web podem fazer reclamações semelhantes, o recurso exclusivo do Django é seu aplicativo de administração integrado — o admin.

O admin oferece a funcionalidade Create-Read-Update-Delete (CRUD) avançada, padrão, eliminando horas de trabalho repetitivo. Isso é fundamental para muitos aplicativos da Web no desenvolvimento, quando programadores podem explorar rapidamente seus modelos de banco de dados, e na implementação, quando usuários finais não técnicos podem utilizar o admin para incluir e editar conteúdo do site.

### Versões de Software Utilizadas neste Artigo

- Django V1.0.2
- SQLite V3
- Python V2.4-2.6 (Django ainda não suporta Python V3)
- IPython (para a saída de amostra)

O Object-Relational Mapper (ORM) do Django suporta muitos back ends de banco de dados, mas o SQLite é o mais fácil de instalar e é fornecido com vários sistemas operacionais. Esses exemplos devem funcionar com qualquer back end. Para obter uma lista dos bancos de dados suportados pelo Django, consulte [Recursos](#).

### Sobre as Amostras de Código

O Django fornece um atalho acessível para a configuração de um ambiente de trabalho em código independente: através da execução do shell `python manage.py`. Todas as amostras de código neste artigo pressupõem que o ambiente tenha sido invocado dessa forma.

Na linguagem Django supõe-se o seguinte:

- Este é projeto do Django chamado `more_with_admin`.
- O projeto `more_with_admin` contém um aplicativo chamado `examples`.

O aplicativo `examples` modela um sistema básico em forma de blog de documentos e zero ou mais comentários sobre esses documentos.

Todos os exemplos de linha de comando são da raiz do projeto—o diretório principal `more_with_admin`.

No mundo real, há sempre uma customização para ser feita. A documentação do Django fornece inúmeras diretrizes para você reconstruir a aparência e o comportamento básicos do admin, e o Django em si inclui alguns métodos simples para você substituir um subconjunto de comportamentos administrativos. E se você precisar fazer mais? Por onde começar? Este artigo fornece algumas diretrizes para customização administrativa avançada.

## Um Tour Rápido pelo Admin

A maioria dos desenvolvedores Django está familiarizada com os recursos padrão do admin. Para fazer uma revisão rapidamente, comece ativando o admin na Listagem 1, editando o `urls.py` de nível superior.

### Listagem 1. Ativando o Admin no `urls.py`

```
from django.conf.urls.defaults import *

# Remova o comentário das duas próximas linhas para ativar o admin:
from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns("",
    # Remova o comentário da próxima linha para ativar o admin:
    (r'^admin/(.*)', admin.site.root),
)
```

Você também precisa incluir o aplicativo `django.contrib.admin` em `settings.INSTALLED_APPS`.

Antes de prosseguir, qualquer pessoa que estiver planejando customizar extensivamente o admin deve se familiarizar com o código de origem. Para sistemas operacionais que suportam atalhos ou symlinks, talvez seja útil criar um para o aplicativo admin. O admin encontra-se dentro do pacote Django. Supondo que ele tenha sido instalado com ferramentas de configuração, o admin está nos pacotes de site em `django/contrib/admin`. Veja aqui um exemplo de um link simbólico de um projeto para a origem de administração do Django que você pode customizar com base em seu sistema operacional e no local da instalação do Django para fácil cópia e referência:

```
$ ln -s /path/to/Python/install/site-packages/django/contrib/admin admin-source
```

O método `admin.autodiscover()` é iterado através de cada aplicativo em `settings.INSTALLED_APPS` e procura um arquivo chamado `admin.py`. Geralmente, ele fica na parte superior do diretório de aplicativo, no mesmo nível que `models.py`.

O aplicativo `examples` precisa de um `models.py`, fornecido na Listagem 2. Um `admin.py` correspondente é mostrado abaixo.

## Listagem 2. Um models.py de Amostra para este Aplicativo

```
from django.db import models
class Document(models.Model):
    """Document é uma postagem em blog ou uma entrada de wiki com algum conteúdo de texto"""
    name = models.CharField(max_length=255)
    text = models.TextField()

    def __unicode__(self):
        return self.name

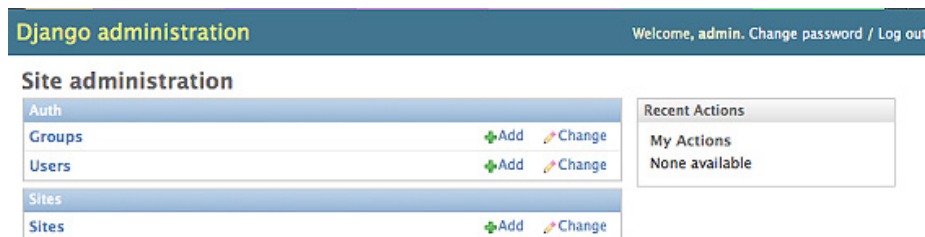
class Comment(models.Model):
    """Comment é algum texto sobre um determinado Document"""
    document = models.ForeignKey(Document, related_name='comments')
    text = models.TextField()
```

Neste ponto, você pode invocar o admin executando o servidor de desenvolvimento Django:

```
python manage.py runserver
```

O admin está disponível no local padrão em <http://localhost:8000/admin/>. Após efetuar login, você verá uma tela de administração básica mostrada abaixo.

## Figura 1. A Tela de Administração Básica do Django



### Alterando Código em admin.py

Ao contrário de outros arquivos em um aplicativo Django, se você fizer mudanças no admin.py utilizando o servidor da Web de desenvolvimento Django, talvez seja necessário reiniciar manualmente o servidor.

Observe que seus modelos não estão disponíveis ainda porque você não criou um admin.py. A Listagem 3 demonstra o código que permite o trabalho com os modelos no admin.

## Listagem 3. Um admin.py de Amostra

```
from django.contrib import admin
from more_with_admin.examples import models

class DocumentAdmin(admin.ModelAdmin):
    pass

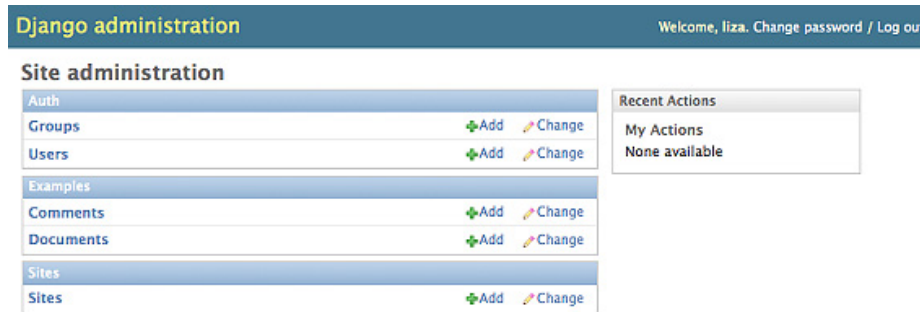
class CommentAdmin(admin.ModelAdmin):
    pass

admin.site.register(
    (models.Document, DocumentAdmin)

admin.site.register(
    (models.Comment, CommentAdmin)
```

Agora, ao reinserir a página principal no admin, você vê os novos modelos disponíveis para uso, conforme mostrado abaixo.

## Figura 2. O Admin do Django Pronto para Suportar Modelos Customizados



## Customizando as Páginas Modelo do Admin

A chave para você entender como customizar o admin sem perder a origem do Django é se lembrar de que o admin é um aplicativo Django comum como outro qualquer. Em primeiro lugar, isso significa que o sistema de herança de modelo do Django se aplica.

A ordem de procura de modelo do Django sempre prioriza seus próprios modelos do projeto sobre os de qualquer sistema. Além disso, o admin tenta procurar modelos codificados permanentemente que correspondam a cada modelo antes de reclassificar para os padrões. Isso fornece um ponto de entrada para facilitar a customização.

Primeiro, certifique-se de que o Django irá conhecer os diretórios de modelo editando o settings.py do projeto.

## Listagem 4. Editando o settings.py para Conhecer Seus Diretórios de Modelo

```
TEMPLATE_DIRS = (
    "/path/to/project/more_with_admin/templates",
    "/path/to/project/more_with_admin/examples/templates",
)
```

### Nomes de Diretórios nas Pastas de Administração

Observe que nos nomes dos modelos utilizo letras minúsculas. Isso está de acordo com o que é normal as páginas de administração trabalham quando geram URLs. O Django chama esses formulários apropriados para URL de espaçadores. Se você não estiver certo de qual é o espaçador correto de um determinado modelo, explore o admin primeiro antes de criar seus diretórios e anote os nomes que aparecem na URL.

Em seguida, crie os seguintes diretórios em seu projeto:

```
$ mkdir templates/admin/examples/document/
$ mkdir templates/admin/examples/comment/
```

O comportamento especial do admin do Django irá verificar um diretório com o nome do seu aplicativo (aqui, `examples`), e depois o nome do modelo (`documento` e `comentário`) antes de utilizar os modelos de sistema para renderizar a página de administração.

## Substituindo uma Única Página Incluir/Editar Modelo

O nome da página que o admin utiliza para incluir e editar uma instância de modelo é `change_form.html`. Comece criando uma nova página chamada `templates/admin/examples/document/change_form.html` no diretório de modelo `Documento` e coloque uma única linha de herança de modelo do Django nele: `{% extends "admin/change_form.html" %}`.

Agora você está pronto para customizar. Tire um tempo para se familiarizar com o conteúdo do admin real/`change_form.html`. Ele é razoavelmente bem organizado em blocos de modelos que você pode substituir, mas alguns tipos de customizações podem exigir a cópia global dos blocos. Todavia, o uso da substituição de modelo baseada em bloco é sempre preferível à cópia da página inteira.

Que tipos de customizações você pode querer fazer na página de inclusão/edição? Talvez, para cada `Documento` no sistema, você gostaria de mostrar uma visualização dos cinco comentários mais recentes.

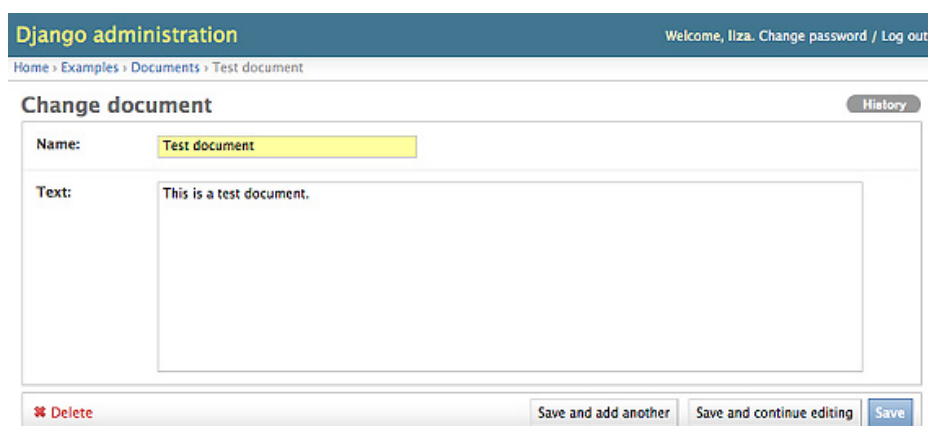
Primeiro, crie um certo conteúdo de amostra, conforme mostrado abaixo.

## Listagem 5. Utilizando o Shell do Django para Criar um Documento de Amostra com vários Comentários

```
$ python manage.py shell
In [1]: from examples import models
In [2]: d = models.Document.objects.create(name='Test document',
      ...:                                     text='This is a test document.')
In [3]: for c in range(0, 10):
      ...:     models.Comment.objects.create(text='Comment number %s' % c, document=d)
```

Agora a página da lista administrativa mostra um `Documento`. Selecione esse `Documento` para apresentar a página de inclusão/edição padrão mostrada abaixo.

## Figura 3. Página de Inclusão/Edição Padrão Apresentando um Documento



The screenshot shows the Django administration interface. At the top, there's a blue header with 'Django administration' on the left and 'Welcome, Ilza. Change password / Log out' on the right. Below the header is a breadcrumb trail: 'Home > Examples > Documents > Test document'. The main content area is titled 'Change document' and contains a form with two fields: 'Name' (with the value 'Test document') and 'Text' (with the value 'This is a test document.'). Below the form, there are four buttons: 'Delete' (with a trash icon), 'Save and add another', 'Save and continue editing', and 'Save'.

Observe que os comentários relacionados não são mostrados de forma nenhuma. A forma padrão de se exibir modelos relacionados no admin é através das poderosas classes `Inline`. As classes `Inline` permitem que

usuários administrativos editem ou incluam vários modelos relacionados em uma única página. Para ver as `inlines` em ação, edite o aplicativo `admin.py` para corresponder à Listagem 6.

## Listagem 6. Incluindo o Comentário do Modelo Relacionado ao Admin do Documento

```
from django.contrib import admin from more_with_admin.examples import models
class CommentInline(admin.TabularInline):
    model = models.Comment

class DocumentAdmin(admin.ModelAdmin):
    inlines = [CommentInline,]

class CommentAdmin(admin.ModelAdmin):
    pass

admin.site.register(models.Document, DocumentAdmin)
admin.site.register(models.Comment, CommentAdmin)
```

A Figura 4 mostra a nova página de inclusão/edição após a inclusão do controle `TabularInline`.

### Figura 4. Página de Inclusão/Edição do Documento após a Inclusão do Modelo de Comentário como `Inline`

The screenshot displays the Django administration interface. At the top, there's a blue header with 'Django administration' on the left and 'Welcome, Iiza. Change password / Log out' on the right. Below the header is a breadcrumb trail: 'Home > Examples > Documents > Test document'. The main content area is titled 'Change document' and includes a 'History' button. The form has two main sections: 'Name' with a text input containing 'Test document', and 'Text' with a large text area containing 'This is a test document.' Below these is a 'Comments' section, which is a tabular inline. It has a 'Text' header and a 'Delete?' column. There are two rows of comments, each with a 'Comment object' label and a 'Comment number' (0 and 1 respectively). Each row has a text input for the comment and a delete button (a square icon with an 'x').

Certamente, ela é poderosa, mas pode ser arrasadora se você quiser ter apenas uma visualização rápida dos comentários.

Você pode utilizar duas abordagens aqui. Uma é editar os widgets HTML associados à `inline` usando a interface do widget do admin do Django. A documentação do Django descreve widgets em detalhes. A outra abordagem é modificar o modelo de inclusão/edição diretamente. Essa abordagem é mais útil quando você não quer usar nenhum recurso específico do admin.

Se você não quiser permitir nenhuma edição dos comentários (talvez porque os usuários não tenham permissões suficientes), mas certamente quer que eles consigam ver os comentários, utilize a abordagem de modificação de `change_form.html`.

## Variáveis Fornecidas pelo Admin do Django

Para incluir funcionalidade a uma página de instância do modelo, você precisa saber quais dados já estão disponíveis a partir do admin. As duas principais variáveis são descritas abaixo.

**Tabela 1. Variáveis Necessárias para Customização de um Modelo do Admin**

Variável	Descrição
<code>object_id</code>	Esta é a chave principal para o objeto sendo editado. Se você estiver customizando uma página de instância específica (como Documento), isso é tudo que você precisa.
<code>content_type_id</code>	Se você estiver substituindo vários tipos de páginas de modelo, utilize esta variável para consultar a estrutura <code>ContentTypes</code> para obter o nome do modelo. Consulte <a href="#">Recursos</a> para obter mais informações sobre tipos de conteúdo.

## Criando uma Tag de Modelo para Inclusão na Página do Admin

Listar os comentários relacionados requer código que não pode ser inserido diretamente em um modelo do Django. A melhor solução para isso é utilizar uma tag de modelo. Primeiro, crie o diretório da tag de modelo e o arquivo `__init__.py`:

```
$ mkdir examples/templatetags/
$ touch examples/templatetags/__init__.py
```

Crie um novo arquivo chamado `examples/templatetags/example_tags.py` e inclua o código mostrado abaixo.

### Listagem 7. Tag de Modelo para Recuperação de Comentários para um Determinado ID de Documento

```
from django import template
from examples import models

register = template.Library()

@register.inclusion_tag('comments.html')
def display_comments(document_id):
    document = models.Document.objects.get(id__exact=document_id)
    comments = models.Comment.objects.filter(document=document)[0:5]
    return { 'comments': comments }
```

Como essa é uma tag de inclusão, você precisa criar o arquivo de modelo correspondente: `comments.html`. Edite o arquivo `examples/templates/comments.html` e insira o código da listagem 8.

## Listagem 8. Modelo para Exibição de um Conjunto de Visualizações de Comentário

```
{% for comment in comments %}
<blockquote>{{ comment.text }}</blockquote>
{% endfor %}
```

Agora é hora de incluir isso na página administrativa. Comente as referências a `CommentInline` em `admin.py` e faça as mudanças mostradas na Listagem 9 para sua versão local de `change_form.html`.

## Listagem 9. Incluindo a Tag de Modelo na Página de Inclusão/Edição

```
{% extends "admin/change_form.html" %}

{% load example_tags %}

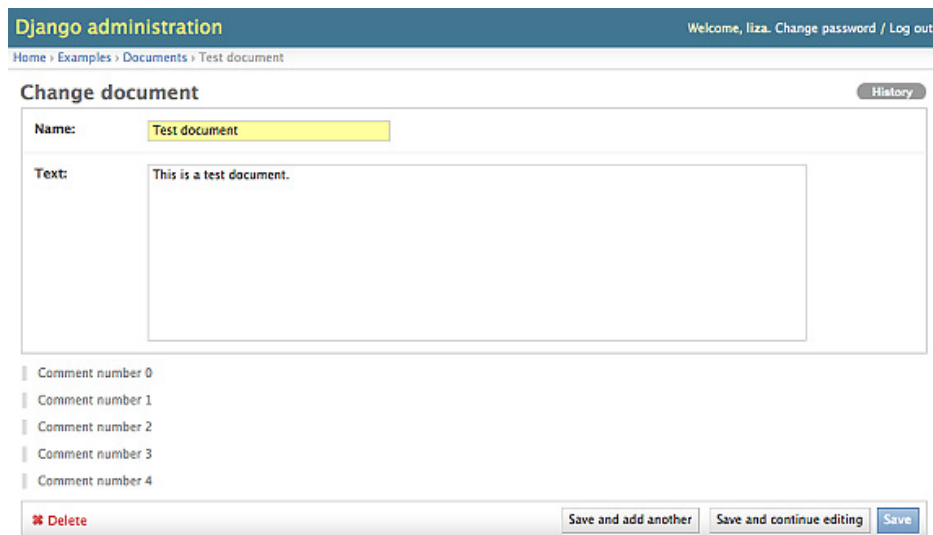
{% block after_field_sets %}
    {% if object_id %}{% display_comments object_id %}{% endif %}
{% endblock %}
```

É importante verificar a existência de `object_id` antes de tentar utilizá-lo, pois `change_form.html` também é utilizado para a criação de novas instâncias, em cujo caso `object_id` ainda não está disponível.

O bloco `after_field_sets` é apenas um dos vários fornecidos como pontos de extensão no admin. Consulte a página de origem `change_form.html` para conhecer outros.

A Figura 5 mostra o formulário atualizado.

## Figura 5. Página de Inclusão/Edição de Documento após Inclusão da Tag de Modelo Customizado



Django administration

Welcome, liza. Change password / Log out

Home > Examples > Documents > Test document

Change document History

Name: Test document

Text: This is a test document.

Comment number 0

Comment number 1

Comment number 2

Comment number 3

Comment number 4

Delete Save and add another Save and continue editing Save

## Modificando o Comportamento Administrativo

As substituições de modelos só podem fazer isso. E se você quiser alterar o comportamento e o fluxo reais do admin? Remover a origem é uma possibilidade, mas isso o bloqueia na versão específica do Django que você está utilizando no momento da atualização.



## Substituindo Métodos AdminModel

Por padrão, um clique em **Salvar** no admin retorna o usuário à página de listas. Geralmente isso é bom, mas e se você quiser ir diretamente a uma página de visualização para um objeto que está fora do admin? Esse é um caso de uso comum quando você está desenvolvendo um content management system (CMS).

### Fornecendo o Método `get_absolute_url()`

A [Listagem 10](#) supõe que o Documento tenha sido modificado para incluir um método `get_absolute_url()`, que é a forma recomendada para um modelo Django especificar sua representação canônica. Se isso for especificado, o admin do Django também fornecerá um botão útil **Visualizar no Site** em cada página desse modelo.

A maioria da funcionalidade no aplicativo administrativo está conectada à classe `admin.ModelAdmin`. Essa é a classe da qual os objetos herdam em `admin.py`. Existem muitos, mas muitos, métodos públicos que você pode substituir. Verifique a origem em `admin-source/options.py` para obter a definição de classe.

Existem duas maneiras de se alterar o comportamento do botão **Salvar**: Você pode substituir `admin.ModelAdmin.response_add`, que é responsável pelo redirecionamento real após um salvamento, ou pode substituir `admin.ModelAdmin.change_view`. A segunda opção é um pouco mais simples e é ilustrada abaixo.

## Listagem 10. Substituindo a Página para a qual os Usuários São Direcionados após um Evento de Salvamento

```
class DocumentAdmin(admin.ModelAdmin):

    def change_view(self, request, object_id, extra_context=None):

        result = super(DocumentAdmin, self).change_view(request, object_id, extra_context)

        document = models.Document.objects.get(id__exact=object_id)

        if not request.POST.has_key('_addanother') and
            not request.POST.has_key('_continue'):
            result['Location'] = document.get_absolute_url()
        return result
```

Agora, quando os usuários clicam em **Salvar**, eles são redirecionados para a página de visualização, e não para a página de lista mostrando todos os Documentos.

## Incluindo Recursos no Admin com Sinais

Sinais são um recurso pouco utilizado no Django que melhora a modularidade de seu código. Os sinais definem eventos, como salvar um modelo ou inserir um modelo, que funcionam em qualquer lugar onde o projeto Django possa atender e responder. Isso significa que você pode melhorar facilmente o comportamento dos aplicativos, sem ter que modificá-los diretamente.

O admin fornece um recurso que os desenvolvedores de aplicativo muitas vezes querem alterar: o gerenciamento de usuários via classe `django.contrib.auth.models.User`. Muitas vezes, o admin é o único lugar onde os usuários do Django são incluídos ou modificados, dificultando a customização dessa classe útil.

Imagine que você queira que o administrador do site receba um e-mail cada vez que um novo objeto `User` for criado. Como o modelo `User` não está diretamente disponível no projeto, pode parecer que a única maneira de

se fazer isso é criando a subclasse de `User` ou utilizando um método indireto, como a criação de um objeto de perfil simulado para modificar.

Em vez disso, a Listagem 11 demonstra como é fácil incluir uma função que é executada quando uma instância de `User` é salva. Sinais geralmente são incluídos no `models.py`.

## Listagem 11. Utilizando Sinais do Django para Notificar Quando um Novo Usuário For Incluído

```
from django.db import models
from django.db.models import signals
from django.contrib.auth.models import User
from django.core.mail import send_mail

class Document(models.Model):
    [...]

class Comment(models.Model):
    [...]

def notify_admin(sender, instance, created, **kwargs):
    """Notifique o administrador de que um novo usuário foi incluído."""
    if created:
        subject = 'New user created'
        message = 'User %s was added' % instance.username
        from_addr = 'no-reply@example.com'
        recipient_list = ('admin@example.com',)
        send_mail(subject, message, from_addr, recipient_list)

signals.post_save.connect(notify_admin, sender=User)
```

O sinal `post_save` é fornecido pelo Django e dispara sempre que um modelo é salvo ou criado. O método `connect()` aqui está usando dois argumentos: um retorno de chamada (`notify_admin`) e o argumento `sender`, que especifica que esse retorno de chamada só está interessado em salvar eventos do modelo `User`.

Dentro do retorno de chamada, o sinal `post_save` passa o emissor (a classe de modelo), a instância desse modelo e um booleano, `criado`, que indica se a instância acabou de ser criada. Nesse exemplo, o método envia uma mensagem de e-mail se o `User` estiver sendo criado; caso contrário, ele não faz nada.

Uma lista de outros sinais fornecidos pelo Django é fornecida em [Recursos](#), bem como a documentação sobre como gravar seus próprios sinais.

## Modificações Intensas: Incluindo Permissões no Nível da Linha

Um recurso solicitado com frequência do admin do Django é o referente a que seu sistema de permissão seja estendido para incluir permissões de nível de linha. Por padrão, o admin permite um controle de granularidade fina de funções e direitos, mas essas funções se aplicam apenas no nível da classe: um usuário pode modificar todos os Documentos ou nenhum.

Muitas vezes, é desejável permitir que os usuários modifiquem apenas objetos específicos. Eles costumam ser chamados de permissões de nível de linha porque refletem a capacidade de modificar apenas linhas específicas de uma tabela de banco de dados, e não uma permissão geral para modificar qualquer registro na tabela.

Um caso de uso no aplicativo examples pode ser o que você queira que os usuários consigam ver apenas os Documentos que eles criaram.

Primeiro, atualize models.py para incluir um atributo gravando quem criou o `Documento`, conforme mostrado abaixo.

### Por que blank=True?

Talvez não fique imediatamente óbvio porque um campo `ForeignKey` seria configurado com `blank=True` quando ele não é um campo de texto. Nesse caso, é porque o admin do Django utiliza em branco em vez de nulo para determinar se o valor deve ser configurado manualmente antes de o modelo ser salvo.

Se você fornecer apenas `null=True` ou nada, o admin do Django forçará o usuário a selecionar manualmente um valor "incluído por" antes do salvamento, quando você quiser que o comportamento seja padronizado para o usuário atual no salvamento.

## Listagem 12. Atualizando models.py para Gravar o Usuário que Criou cada Documento

```
from django.db import models
from django.db.models import signals
from django.contrib.auth.models import User
from django.core.mail import send_mail

class Document(models.Model):
    name = models.CharField(max_length=255)
    text = models.TextField()
    added_by = models.ForeignKey(User, null=True, blank=True)

    def get_absolute_url(self):
        return 'http://example.com/preview/document/%d/' % self.id

    def __unicode__(self):
        return self.name
[...]
```

Em seguida, você precisa incluir código para gravar automaticamente qual usuário criou o `Documento`. Sinais não funcionam para isso porque não têm acesso ao objeto do usuário. Entretanto, a classe `ModelAdmin` fornece um método que inclui o pedido e, portanto, o usuário atual como um parâmetro.

Modifique o método `save_model()` em `admin.py`, conforme mostrado abaixo.

## Listagem 13. Substituindo um Método em DocumentAdmin para Salvar o Usuário Atual no Banco de Dados Quando Criado

```
from django.contrib import admin

class DocumentAdmin(admin.ModelAdmin):
    def save_model(self, request, obj, form, change):
        if getattr(obj, 'added_by', None) is None:
            obj.added_by = request.user
            obj.last_modified_by = request.user
            obj.save()
[...]
```

Se o valor de `added_by` for `None`, esse será um novo registro que não foi salvo. (Você também poderia verificar se `change` é `false`, o que indica que o registro está sendo incluído, mas verificar se `added_by` está vazio significa que ele também preenche registros que foram incluídos fora do admin.)

A próxima parte das permissões no nível da linha é restringir a lista de documentos apenas aos usuários que as criaram. A classe `ModelAdmin` fornece um gancho para isso através de um método chamado `queryset()`, que determina o conjunto de consultas padrão retornado por qualquer página de lista.

Conforme mostrado na Listagem 14, substitua `queryset()` para restringir a listagem apenas aos Documentos criados pelo usuário atual. Os superusuários podem ver todos os documentos.

## Listagem 14. Substituindo o Conjunto de Consultas Retornado pelas Páginas de Lista

```
from
django.contrib import admin
from more_with_admin.examples import models
class DocumentAdmin(admin.ModelAdmin):

    def queryset(self, request):
        qs = super(DocumentAdmin, self).queryset(request)

        # Se for superusuário, mostre todos os comentários
        if request.user.is_superuser:
            return qs

        return qs.filter(added_by=request.user)

[...]
```

Agora quaisquer pedidos para a página de lista `Documento` no admin mostram apenas aqueles criados pelo usuário atual (a menos que o usuário atual seja um superusuário, em cujo caso, todos os documentos são mostrados).

É claro que, atualmente, nada impede um determinado usuário de acessar uma página de edição para um documento desautorizado sabendo seu ID. As permissões de nível de linha realmente seguras requerem mais substituição de métodos. Como os usuários admin geralmente são confiáveis de alguma forma, às vezes as permissões básicas são suficientes para se fornecer um fluxo de trabalho simplificado.

## Conclusão

A customização do admin do Django não requer conhecimento do código de origem do admin, mas sim uma certa pesquisa minuciosa. O admin é estruturado para ser extensível utilizando-se herança Python normal e alguns recursos exclusivos do Django, como sinais.

As vantagens da customização do admin sobre a criação de uma interface de administração totalmente nova são muitas:

- Seu aplicativo se beneficia dos avanços no Django conforme o desenvolvimento continua.
- O admin já suporta os casos de uso mais comuns.
- Aplicativos externos incluídos em seu projeto são automaticamente administráveis lado a lado com seu próprio código.

Pensando no futuro para o Django V1.1 (com release planejado para abril de 2009), o admin fornece dois novos recursos que muitas vezes são solicitados: a capacidade de edição de campos sequenciais nas páginas

de lista e ações administrativas, as quais permitem atualizações em massa em muitos objetos de uma vez. Ambas as inclusões removerão a necessidade de gravar esses recursos comuns do zero durante a inclusão de pontos de extensão para customização adicional.

## Recursos

### Aprender

- Os desenvolvedores que estão apenas começando no Django ou admin devem começar com o excelente [Tutorial do Django](#).
- A [documentação administrativa principal](#) e o [código de origem administrativo](#) são as melhores referências completas para tudo no admin.
- As referências para sinais incluem a [lista de sinais fornecidos pelo Django](#) e sua [documentação sobre a definição de seus sinais](#).
- Aprenda sobre os dois novos recursos administrativos do Django V1.1: [ações administrativas](#) e [listas editáveis](#).
- Os exemplos neste artigo demonstraram a edição de páginas específicas de modelos. Se você precisar editar páginas através de vários modelos, familiarize-se com a [estrutura contenttypes](#) do Django.
- A documentação do Django inclui [uma lista completa de todos os mecanismos de banco de dados suportados pelo Django](#). Desde o Django V1.0, é possível definir novos mecanismos externos também.
- Saiba mais sobre a técnica de [substituir o comportamento de redirecionamento no admin](#).
- Navegue pela [livraria tecnológica](#) para localizar livros sobre esses e outros tópicos técnicos.
- Para ouvir entrevistas e discussões interessantes para desenvolvedores de software, consulte [podcasts do developerWorks](#).
- Mantenha-se atualizado com os eventos Técnicos do developerWorks' e [webcasts](#).
- Siga o [developerWorks no Twitter](#).
- Consulte novas conferências, exposições, webcasts e outros [Eventos](#) em todo o mundo que são extremamente interessantes para desenvolvedores de software livre da IBM.
- Visite a [Zona de software livre](#) do developerWorks para obter informações sobre instruções extensivas, ferramentas e atualizações de projetos para ajudar você a desenvolver tecnologias de software livre e a utilizá-las com os produtos da IBM.
- Assista e aprenda sobre as tecnologias IBM e de software livre e funções de produtos gratuitamente com as [Demos On Demand do developerWorks](#).

### Obter produtos e tecnologias

- Muitas distribuições Linux®, bem como Mac OS® X, são fornecidas com SQLite, mas caso não seja esse seu sistema, você pode fazer o download do [SQLite](#) no site do projeto.
- Desde a V2.5, o Python é um [pacote configurável com suporte para SQLite](#) e não requer nenhum outro driver. Para versões anteriores do Python, você deve fazer o download do [pysqlite](#) diretamente.
- Inove seu próximo projeto de desenvolvimento de software livre com [software de avaliação IBM](#), disponível para download ou em DVD.
- Download [Versões de avaliação do produto IBM](#) ou [explore as avaliações on-line no IBM SOA Sandbox](#) e tenha em mãos as ferramentas de desenvolvimento de aplicativos e os produtos de middleware do DB2®, Lotus®, Rational®, Tivoli® e WebSphere®.

### Discutir

- Participe dos [blogs do developerWorks](#) e envolva-se na comunidade do developerWorks.

## Sobre o autor

### Liza Daly



Liza Daly é uma engenheira de software especializada em aplicativos para o segmento de mercado de publicações. Ela é desenvolvedora líder dos principais produtos on-line da Oxford University Press, O'Reilly Media, e de outros publicadores. Atualmente, ela é uma consultora independente e fundadora da Threepress, um projeto de software livre desenvolvendo aplicativos ebook.

© Copyright IBM Corporation 2009. Todos os direitos reservados.

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

**Marcas Registradas**

([www.ibm.com/developerworks/br/ibm/trademarks/](http://www.ibm.com/developerworks/br/ibm/trademarks/))