Flask-Login

`Release 0.4.1`
`Matthew Frazier`

该文档是吴依杰根据`Flask-Login`官方文档翻译整理而来, 文档所有权归原作者所有

`Flask-login`为Flask提供了用户会话管理, 它处理登录、注销和长时间记住用户会话等常见任务

Flask-Login提供

- 将活动用户的ID存储在会话中, 并允许您轻松地对其进行登录和注销
- 允许您将视图限制为登录(或注销)用户
- 处理通常需要技巧的`"remember me"`功能
- 帮助保护您的用户会话不被cookie窃贼窃取
- 稍后可能与`Flask-Principal`或其他授权扩展集成

但是, Flask-Login不会

- 将特定的数据库或其他存储方法强加于您. 你完全可以决定用户的状态加载
- 限制您使用用户名和密码、OpenIDs或任何其他身份验证方法
- 处理"是否登录"以外的权限
- 处理用户注册或帐户恢复

安装

```
pip install flask-login
```

配置应用程序

使用`Flask-Login`的应用程序中最重要的部分是`LoginManager`类, 你应该为你的应用程序在你的代码某处创建一个实例, 像这样:

```
login_manager = LoginManager()
```

登录管理器包含让应用程序和`Flask-Login`协同工作的代码, 例如如何从ID加载用户, 当用户需要登录时向何处发送, 等等

一旦创建了实际的应用程序对象, 就可以配置它来进行登录

```
login_manager.init_app(app)
```

默认情况下,Flask-Login使用会话进行身份验证. 这意味着您必须在您的应用程序上设置密钥,否则Flask将给您一个错误消息,告诉您这样做,可以查
看https://flask.palletsprojects.com/en/1.1.x/quickstart/#sessions 来查看如何设置一个密钥

Flask-Login如何工作

您将需要提供一个user_loader回调函数. 此回调用于从存储在会话中的用户ID重新加载用户对象. 它应该接受用户的 unicode ID, 并返回相应的user对象, 例如:

```
@login_manager.user_loader
def load_user(user_id):
    return User.get(user_id)
```

如果ID无效, 它应该返回None(而不是引发异常), 在这种情况下, 将手动从会话中删除ID, 并继续处理

你的User类

用来表示用户的类需要实现这些属性和方法:

- is_authenticated:如果用户经过身份验证, 此属性应该返回True. 即他们提供了有效的证件(只有经过身份验证的用户才能满足login_required的条件).
- is_active:如果这是一个活动用户, 这个属性应该返回True——除了要进行身份验证之外, 他们也激活了他们的帐户, 没有被暂停, 或任何条件你的申请拒绝帐户,不活跃的帐户可能无法登录(当然不是强制的)
- is_anonymous:如果这是一个匿名用户, 该属性应该返回True(实际用户应该返回False)
- get_id():此方法必须返回唯一标识此用户的unicode标识符, 并可用于从user_loader回调加载用户, 请注意, 这必须是一个unicode标识符, 如果ID是一个整数或其他类型, 您将需要将它转换为unicode类型

为了使实现user类更容易, 您可以继承UserMixin, 它为所有这些属性和方法提供了默认实现. (但这不是必须的)

Login示例

一旦用户通过了身份验证, 就可以使用login_user函数对其进行登录:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    # Here we use a class of some kind to represent and validate our
    # client-side form data. For example, WTForms is a library that will
```

```
    # handle this for us, and we use a custom LoginForm to validate.
    form = LoginForm()
    if form.validate_on_submit():
        # Login and validate the user.
        # user should be an instance of your `User` class
        login_user(user)
        flask.flash('Logged in successfully.')
        next = flask.request.args.get('next')
        # is_safe_url should check if the url is safe for redirects.
        # See http://flask.pocoo.org/snippets/62/ for an example.
        if not is_safe_url(next):
            return flask.abort(400)
        return flask.redirect(next or flask.url_for('index'))
    return flask.render_template('login.html', form=form)
```

警告:您必须验证`next`参数的值. 如果不这样做, 您的应用程序将容易受到打开重定向的攻击. 有关`is_safe_url`的示例实现，请参阅 http://flask.pocoo.org/snippets/62/

就是这么简单. 然后, 你可以使用`current_user`代理访问已登录的用户, 它在每个模板中都可用:

```
{% if current_user.is_authenticated %}
    Hi {{ current_user.name }}!
{% endif %}
```

需要用户登录的视图可以使用`login_required`装饰器进行装饰

```
@app.route("/settings")
@login_required
def settings():
    pass
```

当用户准备退出时:

```
@app.route("/logout")
@login_required
def logout():
    logout_user()
    return redirect(somewhere)
```

他们将被注销, 他们会话的所有cookie将被清除

自定义登录过程

默认情况下, 当用户试图在未登录的情况下访问`login_required`视图时, `Flask-Login`将闪烁一条消息并将其重定向到`login`视图 (如果没有设置login视图, 它将以`401`错误中止)

login视图的名称可以设置为`LoginManager.login_view` 例如:

```
login_manager.login_view = "users.login"
```

所显示的默认消息是`Please log in to access this page.` 要自定义消息, 请设置`LoginManager.login_message`:

```
login_manager.login_message = u"Bonvolu ensaluti por uzi tiun paˆ gon."
```

要自定义消息类别, 请设置`LoginManager.login_message_category`:

```
login_manager.login_message_category = "info"
```

当`log in`视图被重定向到该视图时, 查询字符串中将有一个`next`变量, 即用户试图访问的页面, 或者, 如果`USE_SESSION_FOR_NEXT`为`True`, 则页面存储在会话中的键`next`下

如果您想进一步定制流程, 用`LoginManager.unauthorized_handler` 装饰一个函数

```
@login_manager.unauthorized_handler
def unauthorized():
# do stuff
    return a_response
```

使用授权头登录

注意:此方法将被弃用; 而是使用下面的`request_loader`

有时, 您希望使用授权头(如api请求)支持基本的Auth登录.要支持通过`header登录`,您需要提供一个`header_loader` 回调函数.这个回调应该与`user_loader`回调的行为相同,只是它接受一个头值而不是用户`id`, 例如`:

```
@login_manager.header_loader
def load_user_from_header(header_val):
    header_val = header_val.replace('Basic ', '', 1)
    try:
        header_val = base64.b64decode(header_val)
    except TypeError:
        pass
    return User.query.filter_by(api_key=header_val).first()

```

默认情况下, 授权头的值传递给`header_loader`回调, 您可以更改与`AUTH_HEADER_NAME`配置一起使用的头

使用请求加载程序自定义登录

有时你想在不使用cookie的情况下登录用户, 例如使用头值或作为查询参数传递的api键,在这些情况下,您应该使用request_loader回调,这个回调应该与user_loader回调的行为相同,只不过它接受的是Flask请求, 而不是user_id例如, 支持登录从一个url参数和从基本的Auth使用授权头:

```python
@login_manager.request_loader
def load_user_from_request(request):
    # first, try to login using the api_key url arg
    api_key = request.args.get('api_key')
    if api_key:
        user = User.query.filter_by(api_key=api_key).first()
        if user:
            return user
    # next, try to login using Basic Auth
    api_key = request.headers.get('Authorization')
    if api_key:
        api_key = api_key.replace('Basic ', '', 1)
        try:
            api_key = base64.b64decode(api_key)
        except TypeError:
            pass
        user = User.query.filter_by(api_key=api_key).first()
        if user:
            return user
# finally, return None if both methods did not login the user
    return None
```

匿名用户

默认情况下, 当用户没有实际登录时, current_user被设置为一个AnonymousUserMixin对象. 它有以下属性和方法:

- is_active and is_authenticated 为 False
- is_anonymous 为 True
- get_id() 返回 None

如果您对匿名用户有自定义的需求(例如, 他们需要一个permissions字段), 那么您可以提供一个可调用的(类或工厂函数), 用它为LoginManager创建匿名用户:

```python
login_manager.anonymous_user = MyAnonymousUser
```

Remember Me

默认情况下, 当用户关闭浏览器时, Flask会话将被删除, 用户将注销. "Remember Me"可以防止用户在关闭浏览器时意外注销. 这并不意味着在用户注销后, 在登录表单中记住或预先填写用户的用户名或密码.
"Remember Me"功能可能很难实现. 但是, Flask-Login使它几乎是透明的,只需将remember=True传递给login_user调用.cookie将保存在用户的计算机上,如果cookie不在会话中,Flask-Login将自动恢复该cookie的用户

ID,cookie过期前的时间可以通过`memorber_cookie_duration`配置设置,也可以传递给`login_user`.cookie是防篡改的,所以如果用户篡改它(即插入别人的用户ID代替他们自己的),cookie只会被拒绝, 就好像它不存在一样。

该级别的功能是自动处理的. 但是, 您可以(并且应该, 如果您的应用程序处理任何类型的敏感数据)提供额外的基础设施来提高您的记忆cookie的安全性

选择令牌

使用用户ID作为remember令牌的值意味着您必须更改用户ID以使其登录会话无效, 改善这种情况的一种方法是使用一个替代的用户id, 而不是用户的id

```
@login_manager.user_loader
def load_user(user_id):
    return User.query.filter_by(alternative_id=user_id).first()
```

然后用户类的`get_id`方法将返回替代id, 而不是用户的主id

```
def get_id(self):
    return unicode(self.alternative_id)
```

通过这种方式, 当用户更改密码时, 您可以自由地将用户的备用id更改为随机生成的新值, 这将确保他们的旧身份验证会话不再有效. 请注意, 替代id仍然必须惟一地标识用户.....可以将它看作第二个用户ID

Fresh Logins

当用户登录时, 他们的会话被标记为`"fresh"`, 这表明他们实际上在该会话上进行了身份验证.当他们的会话被销毁, 并使用`"remember me"`cookie重新登录时,它被标记为`"non-fresh"`. `login_required`并不区分freshness, 对大部分页面都一样. 然而, 像更改个人信息这样的敏感行为需要重新登录像更改密码这样的操作应该总是需要重新输入密码).

`fresh_login_required`除了验证用户已登录外, 还将确保他们的登录是新的,如果没有,它将把他们发送到一个页面,在那里他们可以重新输入他们的凭证.你可以像定制`login_required`一样定制它的行为,通过设置`LoginManager.refresh_view`, `needs_refresh_message`, 和`needs_refresh_message_category`:

```
login_manager.refresh_view = "accounts.reauthenticate"
login_manager.needs_refresh_message = (u"To protect your account, please reauthenticate to access this page.")
login_manager.needs_refresh_message_category = "info"
```

或通过提供自己的回调来处理刷新

```
@login_manager.needs_refresh_handler
```

```
def refresh():
    # do stuff
    return a_response
```

要再次将会话标记为新会话, 请调用`confirm_login`函数

Cookie设置

可以在应用程序设置中定制cookie的详细信息

变量	解释
`REMEMBER_COOKIE_NAME`	存储`"remember me"`信息的cookie的名称. 默认:`remember_token`
`REMEMBER_COOKIE_DURATION`	通知cookie过期前的时间, 如`datetime.timedelta`, 对象或整数秒. 默认:`365天`
`REMEMBER_COOKIE_DOMAIN`	如果`"remember me"` cookie 应该跨域, 在这里设置域值(例如..example.com将允许该cookie在example.com的所有子域上使用), 默认:`None`
`REMEMBER_COOKIE_PATH`	这将`"remember me"` cookie限制在特定的路径上. 默认值:`/`
`REMEMBER_COOKIE_SECURE`	限制`"remember me"` cookie的范围来保护通道(通常是HTTPS). 默认:`None`
`REMEMBER_COOKIE_HTTPONLY`	防止客户端脚本访问`"Remember Me"` cookie, 默认:`False`
`REMEMBER_COOKIE_REFRESH_EACH_REQUEST`	如果设置为`True`, cookie在每次请求时都会刷新, 这会影响生存期. 类似于Flask的`SESSION_REFRESH_EACH_REQUEST`. 默认值:`False`

会话的保护

虽然上述功能有助于从cookie窃贼手中保护您的`"Remember Me"`令牌, 但会话cookie仍然是脆弱的. Flask-Login包含了会话保护, 可以帮助防止用户的会话被窃取.

您可以在`LoginManager`和应用程序的配置中配置会话保护,如果启用它,它可以在`basic`模式或`strong`模式下运行. 要在`LoginManager`上设置它,请将`session_protection`属性设置为`"basic"`或`"strong"`:

```
login_manager.session_protection = "strong"
```

或者, 禁用它

```
login_manager.session_protection = None
```

默认情况下, 它是在"basic"模式下激活的. 通过将SESSION_PROTECTION设置为None、"basic"或"strong", 可以在应用程序的配置中禁用它.

当会话保护激活时, 每个请求都会为用户的计算机生成一个标识符(基本上是IP地址和用户代理的安全散列),如果会话没有相关联的标识符,则将存储所生成的标识符.如果它有一个标识符,并且它与生成的标识符相匹配, 那么请求就是OK的

如果标识符在basic模式中不匹配, 或者当会话是永久的, 则会话将被标记为非新的, 任何需要重新登录的操作都将强制用户重新进行身份验证. Of course, you must be already using fresh logins where appropriate for this to have an effect.

如果标识符在非常任会话的strong模式下不匹配, 则删除整个会话(以及存在的记忆令牌)

为api禁用会话Cookie

在对api进行身份验证时, 您可能希望禁用对Flask会话cookie的设置. 为此, 使用一个自定义会话接口, 该接口根据您在请求上设置的标志跳过保存会话. 例如:

```
from flask import g
from flask.sessions import SecureCookieSessionInterface
from flask_login import user_loaded_from_header


class CustomSessionInterface(SecureCookieSessionInterface):
    """Prevent creating session from API requests."""
    def save_session(self, *args, **kwargs):
        if g.get('login_via_header'):
            return
    return super(CustomSessionInterface, self).save_session(*args, **kwargs)

app.session_interface = CustomSessionInterface()

@user_loaded_from_header.connect
def user_loaded_from_header(self, user=None):
    g.login_via_header = True
```

这可以防止在用户使用header_loader进行身份验证时设置Flask会话cookie

本地化

默认情况下, 当用户需要登录时, LoginManager使用flash显示消息,这些信息是英文的.如果需要本地化,可以

将LoginManager的localize_callback属性设置为在消息发送到flash(例如gettext)之前使用这些消息调用的函数. 这个函数将与消息一起被调用, 它的返回值将被发送到flash中

API文档

这个文档是由Flask-Login的源代码自动生成的

`class flask_login.LoginManager(app=None, add_context_processor=True)`
此对象用于保存用于登录的设置, LoginManager的实例并不绑定到特定的应用程序, 因此您可以在代码主体中创建一个实例, 然后将其绑定到工厂函数中的应用程序

`setup_app(app, add_context_processor=True)`
这种方法已被废弃. 请使用LoginManager.init_app()

`unauthorized()`
当用户需要登录时调用此函数, 如果您向LoginManager.unauthorized_handler()注册一个回调, 那么它将被调用, 否则, 它将采取以下操作

- 1.向user发送LoginManager.login_message信息
- 2.如果应用程序正在使用蓝图, 请使用blueprint_login_views查找当前蓝图的登录视图. 如果应用程序没有使用蓝图或没有指定当前蓝图的登录视图, 请使用login_view的值.
- 3.将用户重定向到login视图, 他们试图访问的页面将被传递到下一个查询字符串变量中, 因此如果存在, 您可以重定向到那里, 而不是主页, 或者, 如果设置了USE_SESSION_FOR_NEXT, 它将作为下一个添加到会话

如果LoginManager.login_view没有定义, 那么它只会引发一个HTTP 401(未授权)错误

这应该从视图或before/after_request函数中返回, 否则重定向将不起作用.

`needs_refresh()`

这在用户登录时调用, 但是他们需要重新验证, 因为他们的会话已经过期
如果用needs_refresh_handler注册回调, 那么将调用它. 否则, 将采取以下行动:

- 1.向User发送 LoginManager.needs_refresh_message信息
- 2.将用户重定向到 LoginManager.refresh_view, 他们试图访问的页面将被传递到下一个查询字符串变量中, 因此如果存在, 您可以重定向到那里, 而不是主页

如果没有定义LoginManager.refresh_view, 然后它只会引发一个HTTP 401(未授权)错误.

这应该从视图或before/after_request函数中返回, 否则重定向将不起作用

常规配置

`user_loader(callback)`

功能: 这设置了从会话中重新加载用户的回调. 您设置的函数应该接受一个用户ID(一个unicode)并返回一个user对象, 如果用户不存在, 则返回None
参数:

- callback (callable):用于检索用户对象的回调

`header_loader(callback)`

这个功能已经被废弃了. 请使用`LoginManager.request_loader()`
功能:这将设置从标题值加载用户的回调. 您设置的函数应该接受一个身份验证令牌并返回一个user对象, 如果用户不存在, 则返回`None`
参数:

- `callback (callable)`:用于检索用户对象的回调

`anonymous_user`

生成匿名用户的类或工厂函数, 在没有人登录时使用

未经授权的配置

`login_view`:用户需要登录时重定向到的视图的名称.(如果您的身份验证机制是应用程序外部的, 那么它也可以是一个绝对URL.)
`login_message`:当用户被重定向到登录页面时, 他向`flash`发送消息.

`unauthorized_handler(callback)`

功能:这将为未经授权的方法设置回调, `login_required`使用的方法之一就是这个方法, 它不接受参数, 并且应该返回一个要发送给用户的响应, 而不是普通的视图
参数:

- `callback (callable)`:未授权用户的回调

`needs_refresh`配置

`refresh_view`:在用户需要重新验证时重定向到的视图的名称
`needs_refresh_message`:当用户被重定向到重新身份验证页面时, 发送给`flash`的消息

`needs_refresh_handler(callback)`

功能:这将为`needs_refresh`方法设置回调, 其中`fresh_login_required`使用了这个方法, 它不接受参数, 并且应该返回一个要发送给用户的响应, 而不是普通的视图.
参数:

- `callback (callable)`: 未授权用户的回调

登录机制

`flask_login.current_user`
当前用户的代理

`flask_login.login_fresh()`
如果当前登录是新的, 则返回`True`

`flask_login.login_user(user, remember=False, duration=None, force=False, fresh=True)`

登录用户. 您应该将实际的`user`对象传递给它, 如果用户的`is_active`属性为`False`,则除非`force`为`True`,否则不会登录.如果登录成功,则返回`True`;如果失败,则返回`False`(例如,因为用户处于非活动状态)

参数:

- `user`(object):要登录的用户对象
- `remember`(bool):是否在会话过期后记住用户. 默认值为`False`
- `duration`(datetime.timedelta):记住cookie过期之前的时间. 如果没有, 则使用设置中设置的值. 默认为`None`.
- `force`(bool):如果用户处于非活动状态, 则将其设置为True将登录用户. 默认值为`False`
- `fresh`(bool):将此设置为`False`, 用户将登录一个标记为`"not fresh"`的会话. 默认值为`True`

`flask_login.logout_user()`

注销用户(您不需要传递实际的用户)如果`"Remrmber Me"` cookie存在的话, 这也会清除它.

`flask_login.confirm_login()`

这将当前会话设置为`fresh`. 当从cookie重新加载会话时, 它们就会变得陈旧.

保护视图

`flask_login.login_required(func)`

如果用它修饰视图, 它将确保在调用实际视图之前当前用户已登录并通过了身份验证(如果不是, 则调用`LoginManager.unauthorized`回调函数), 例如

```
@app.route('/post')
@login_required
def post():
    pass
```

如果你只需要在特定时间要求你的用户登录, 你可以这样做:

```
if not current_user.is_authenticated:
    return current_app.login_manager.unauthorized()
```

也就是这个函数添加到视图中的代码

在进行单元测试时, 可以方便地全局地关闭身份验证. 要启用此功能, 如果将应用程序配置变量`LOGIN_DISABLED`设置为`True`, 则将忽略此装饰器.

参数:

- `func`(function):要修饰的视图函数

`lask_login.fresh_login_required(func)`

如果你用这个装饰一个视图, 它将确保当前用户的登录是新鲜的——即他们的会话没有从一个`"Remember Me"`cookie恢复. 一些敏感的操作, 比如更改密码或电子邮件, 应该用这个来保护, 以阻止cookie窃贼的行动

如果用户没有经过身份验证，则正常调用 `LoginManager.unauthorized()`, 如果它们通过了身份验证, 但是它们的会话不是新的, 那么它将调用`LoginManager.needs_refresh()`, 在这种情况下, 需要提供`LoginManager.refresh_view`

就配置变量而言, 其行为与`login_required()`装饰器相同

参数:
`func`(function):要修饰的视图函数

用户对象助手

`class flask_login.UserMixin`

这为`Flask-Login`期望用户对象拥有的方法提供了默认实现.

`class flask_login.AnonymousUserMixin`

这是表示匿名用户的默认对象。

Utilities

`flask_login.login_url(login_view, next_url=None, next_field='next')`

功能:创建重定向到登录页面的URL.如果只提供`login_view`, 这将只返回它的URL, 但是, 如果提供了`next_url`, 这将向查询字符串附加一个`next=URL`参数, 以便login视图可以重定向回该URL, Flask-Login的默认未授权处理程序在重定向到您的登录url时使用此函数. 要强制使用主机名, 请设置`FORCE_HOST_FOR_REDIRECTS`到主机, 这可以防止在请求头主机或`x - forwarding`出现时重定向到外部站点.

参数:

- `login_view`(str):login视图的名称. (或者, 登录视图的实际URL)

- `next_url`(str):提供用于重定向的登录视图的URL
- `next_field`(str):存储下一个URL的字段.(默认为next)

信号

有关如何在代码中使用这些信号的信息, 请参阅有关信号的Flask文档(http://flask.pocoo.org/docs/signals/)

`flask_login.user_logged_in`

当用户登录时发送. 除了应用程序(即发送方)之外, 它还传递用户(即登录的用户)

`flask_login.user_logged_out`

当用户注销时发送. 除了应用程序(即发送方)之外, 它还传递用户(即注销的用户)

`flask_login.user_login_confirmed`

当用户的登录被确认时发送, 标记为新鲜(这不是一个正常的登录调用)除了应用程序之外, 它没有收到其他参数

`flask_login.user_unauthorized`

在 LoginManager上调用未授权的方法时发送. 除了应用程序之外, 它没有收到其他参数

`flask_login.user_needs_refresh`

在LoginManager上调用needs_refresh方法时发送. 除了应用程序之外, 它没有收到其他参数.

`flask_login.session_protected`

当会话保护生效时发送, 并且一个会话被标记为不新鲜或已删除. 除了应用程序之外, 它没有收到其他参数.