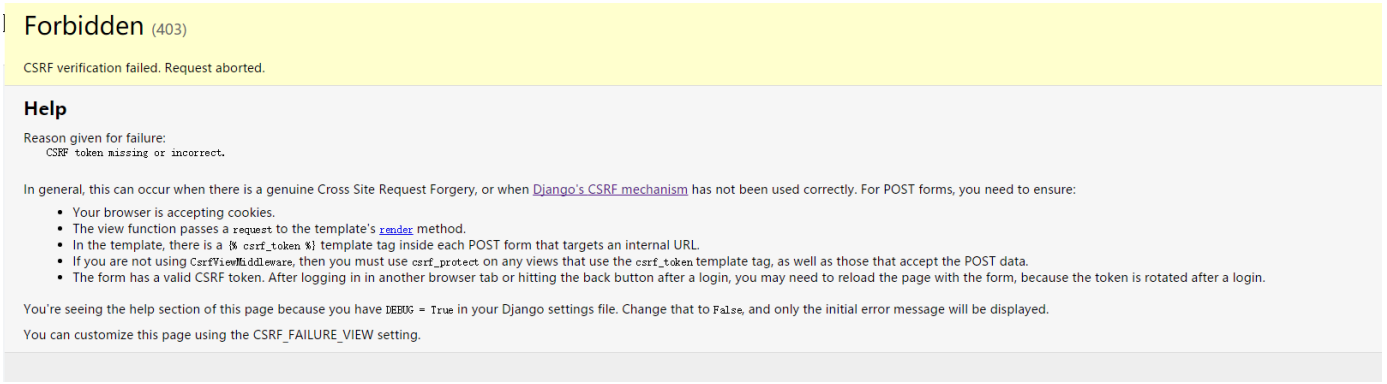
在python项目中提交数据发现报403错误：



Forbidden (403)

CSRF verification failed. Request aborted.

Help

Reason given for failure:
CSRF token missing or incorrect.

In general, this can occur when there is a genuine Cross Site Request Forgery, or when Django's CSRF mechanism has not been used correctly. For POST forms, you need to ensure:

- Your browser is accepting cookies.
- The view function passes a request to the template's render method.
- In the template, there is a {% csrf_token %} template tag inside each POST form that targets an internal URL.
- If you are not using CsrfViewMiddleware, then you must use csrf_protect on any views that use the csrf_token template tag, as well as those that accept the POST data.
- The form has a valid CSRF token. After logging in in another browser tab or hitting the back button after a login, you may need to reload the page with the form, because the token is rotated after a login.

You're seeing the help section of this page because you have DEBUG = True in your Django settings file. Change that to False, and only the initial error message will be displayed.

You can customize this page using the CSRF_FAILURE_VIEW setting.

https://docs.djangoproject.com/en/1.10/ref/csrf/

perfectionists with deadlines.

OVERVIEW DOWNLOAD

Documentation

Cross Site Request Forgery protection

The CSRF middleware and template tag provides easy-to-use protection against Cross Site Request Forgeries. This type of attack occurs when a malicious website contains a link, a form button or some JavaScript that is intended to perform some action on your website, using the credentials of a logged-in user who visits the malicious site in their browser. A related type of attack, 'login CSRF', where an attacking site tricks a user's browser into logging into a site with someone else's credentials, is also covered.

The first defense against CSRF attacks is to ensure that GET requests (and other 'safe' methods, as defined by **RFC 7231#section-4.2.1**) are side effect free. Requests via 'unsafe' methods, such as POST, PUT, and DELETE, can then be protected by following the steps below.

How to use it

To take advantage of CSRF protection in your views, follow these steps:

1. The CSRF middleware is activated by default in the **MIDDLEWARE** setting. If you override that setting, remember that **'django.middleware.csrf.CsrfViewMiddleware'** should come before any view middleware that assume that CSRF attacks have been dealt with.

   If you disabled it, which is not recommended, you can use **csrf_protect()** on particular views you want to protect (see below).

2. In any template that uses a POST form, use the **csrf_token** tag inside the **<form>** element if the form is for an internal URL, e.g.:

   ```
   <form action="" method="post">{% csrf_token %}
   ```

   This should not be done for POST forms that target external URLs, since that would cause the CSRF token to be leaked, leading to a vulnerability.

3. In the corresponding view functions, ensure that **RequestContext** is used to render the response so that **{% csrf_token %}** will work properly. If you're using the **render()** function, generic views, or contrib apps, you are covered already since these all use **RequestContext**.

AJAX

While the above method can be used for AJAX POST requests, it has some inconveniences: you have to remember to pass the CSRF token

```
1.  <form action="http://127.0.0.1:8000/app2/db/select" method="post">{% csrf_token %}
```

这样就OK

用 django 有多久，我跟 csrf 这个概念打交道就有久了。

每次初始化一个项目时都能看到 django.middleware.csrf.CsrfViewMiddleware 这个中间件
每次在模板里写 form 时都知道要加一个 {% csrf_token %} tag
每次发 ajax POST 请求，都需要加一个 X_CSRFTOKEN 的 header
但是一直我都是知其然而不知其所以然，没有把 csrf 的机制弄清楚。昨天稍微研究了一下，总结如下。

什么是 CSRF
CSRF, Cross Site Request Forgery, 跨站点伪造请求。举例来讲，某个恶意的网站上有一个指向你的网站的链接，如果

某个用户已经登录到你的网站上了，那么当这个用户点击这个恶意网站上的那个链接时，就会向你的网站发来一个请求，

你的网站会以为这个请求是用户自己发来的，其实呢，这个请求是那个恶意网站伪造的。

具体的细节及其危害见 wikipedia

Django 提供的 CSRF 防护机制
django 第一次响应来自某个客户端的请求时，会在服务器端随机生成一个 token，把这个 token 放在 cookie 里。然后每次 POST 请求都会带上这个 token，

这样就能避免被 CSRF 攻击。

在返回的 HTTP 响应的 cookie 里，django 会为你添加一个 csrftoken 字段，其值为一个自动生成的 token
在所有的 POST 表单时，必须包含一个 csrfmiddlewaretoken 字段（只需要在模板里加一个 tag，django 就会自动帮你生成，见下面）
在处理 POST 请求之前，django 会验证这个请求的 cookie 里的 csrftoken 字段的值和提交的表单里的 csrfmiddlewaretoken 字段的值是否一样。如果一样，则表明这是一个合法的请求，否则，这个请求可能是来自于别人的 csrf 攻击，返回 403 Forbidden.
在所有 ajax POST 请求里，添加一个 X-CSRFTOKEN header，其值为 cookie 里的 csrftoken 的值
Django 里如何使用 CSRF 防护
首先，最基本的原则是：GET 请求不要用有副作用。也就是说任何处理 GET 请求的代码对资源的访问都一定要是“只读“的。
要启用 django.middleware.csrf.CsrfViewMiddleware 这个中间件
再次，在所有的 POST 表单元素时，需要加上一个 {% csrf_token %} tag
在渲染模块时，使用 RequestContext。RequestContext 会处理 csrf_token 这个 tag，从而自动为表单添加一个名为 csrfmiddlewaretoken 的 input