打开PostCrudController，我们来覆盖父类的`configureActions()`方法。`public function configureActions()`，我们可以对Actions对象进行修改。这里删除一下，我们使用$actions对象的update()方法来更新页面上的按钮。

update()方法有三个参数，第一个参数是页面的名称，第二个参数是按钮的名称，第三个参数是回调方法。页面名称我们使用Crud类的常量`PAGE_INDEX`，按钮的名称，我们使用Action类的常量，这里选择EasyAdminBundle的Action类，我们先来调整一下`EDIT`按钮。第三个参数是个回调方法，输入function，回调方法有个参数是Action对象，对应的是当前的按钮对象。

我们来换行，在回调方法中，我们可以设置$action对象是否显示。`return $action->displayIf()`，在`displayIf()`方法中再次使用一个回调，这次我们使用箭头方法。fn参数是当前的文章对象`Entity $entity`，箭头，`$this->isGranted(PostVoter::POST_OWNER_EDIT)`。第二个参数传入$entity，如果当前的用户是文章的作者，那么我们就显示编辑按钮。调整一下代码格式，同样的我们使用update()方法来更新一下删除按钮，command+d复制。这里修改为`DELETE`，这里修改为`POST_OWNER_DELETE`。

```
#src/Controller/Admin/PostCrudController.php

class PostCrudController extends AbstractCrudController
{
    // ...

    public function configureActions(Actions $actions): Actions
    {
        return $actions->update(Crud::PAGE_INDEX, Action::EDIT,
            function (Action $action) {
                return $action->displayIf(fn($entity) => $this-
>isGranted(PostVoter::POST_OWNER_EDIT, $entity));
            })->update(Crud::PAGE_INDEX, Action::DELETE,
            function (Action $action) {
                return $action->displayIf(fn($entity) => $this-
>isGranted(PostVoter::POST_OWNER_DELETE, $entity));
            });
    }
}
```

回到浏览器刷新，我们看到不是editor用户的文章，它们的按钮都消失了，我们来切换一下用户，`_switch_user=simple_admin`。这次我们看到只有`simple_admin`用户的文章显示了编辑和删除按钮。我们的`simple_admin`用户是个管理员用户。通常管理员用户拥有较高的权限，这里我们设置要让普通管理员及以上权限的用户可以编辑和删除所有的文章。

我们要修改PostVoter类，回到项目。打开PostVoter，在对文章的作者进行判断之前，我们需要添加一个判断。我们判断当前的用户是不`ROLE_ADMIN`角色。我们可以在构造方法中通过依赖注入的方式注入Security对象，初始化$security属性。添加一个条件判断，`if( $this->security->isGranted('ROLE_ADMIN') )`。如果当前用户是管理员用户，直接返回true，调整一下代码格式。

```
#src/Security/Voter/PostVoter.php

class PostVoter extends Voter
```

```
{
    const POST_OWNER_EDIT = 'post_owner_edit';
    const POST_OWNER_DELETE = 'post_owner_delete';
    /**
     * @var Security
     */
    private Security $security;

    public function __construct(Security $security)
    {
        $this->security = $security;
    }

    protected function supports(string $attribute, $subject): bool
    {
        // replace with your own logic
        // https://symfony.com/doc/current/security/voters.html
        return in_array($attribute, [self::POST_OWNER_DELETE,
self::POST_OWNER_EDIT])
            && $subject instanceof \App\Entity\Post;
    }
    protected function voteOnAttribute(string $attribute, $subject,
TokenInterface $token): bool
    {
        $user = $token->getUser();
        // if the user is anonymous, do not grant access
        if (!$user instanceof UserInterface) {
            return false;
        }
        // ... (check conditions and return true to grant permission) ...
        switch ($attribute) {
            case self::POST_OWNER_EDIT:
                // logic to determine if the user can EDIT
                // return true or false
//                break;
            case self::POST_OWNER_DELETE:
                // logic to determine if the user can VIEW
                // return true or false
                if ($subject->getAuthor() == $user){
                if ($this->security->isGranted('ROLE_ADMIN')) {
                    return true;
                }
                if ($subject->getAuthor() == $user) {
                    return true;
                }
                break;
        }
        return false;
    }
}
```

回到浏览器刷新，现在我们的普通管理员用户就可以编辑所有的文章了。我们编辑一下editor用户的文章，可以进行编辑。这就是自定义Voter的使用方法。

Symfony的Voter有几种使用的策略，在下节课我们来更多的了解一下Voter。