

# symfony でのform作成の基本

PHP Symfony Symfony2

この記事は最終更新日から3年以上が経過しています。

## はじめに

---

案件で何もわからないままsymfonyを使い始めたけど、日本語でのまとまった情報が少ない。。。

meetupとかあるみたいだけど、最近動きが少ない様に見える・・・(php界隈詳しくないので勘違いかもしれません)

特に情報がまとまっていることが少なかったForm関連のクラスを自動生成されたコードに少し手を加えたものを基本にまとめてみました。

今回使用したsymfonyのversionは2.8です。

## 実際に登録・編集 作ってみる

---

# Entity

- まずはテーブルのEntityクラスを作成します。テーブルに連動したメンバーとgetter,setterを作成。

```
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * ToDo
 *
 * @ORM\Table(name="to_do")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\ToDoRepos")
 */
class ToDo
{

    /**
     * @var int
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
```

```
/**
 * @var string
 *
 * @ORM\Column(name="task", type="integer", length=1)
 */
private $task;

/**
 * @var string
 *
 * @ORM\Column(name="memo", type="string", length=255, nullable=true)
 */
private $memo;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="r_datetime", type="datetime", nullable=true)
 */
private $rDatetime;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="u_datetime", type="datetime", nullable=true)
 */
private $uDatetime;

/**
```

```
* Get id
*
* @return integer
*/
public function getId()
{
    return $this->id;
}

/**
 * Set task
 *
 * @param string $task
 * @return ToDo
 */
public function setTask($task)
{
    $this->task = $task;

    return $this;
}

/**
 * Get task
 *
 * @return string
 */
public function getTask()
{
```

```
        return $this->task;
    }

    /**
     * Set memo
     *
     * @param string $memo
     * @return ToDo
     */
    public function setMemo($memo)
    {
        $this->memo = $memo;

        return $this;
    }

    /**
     * Get memo
     *
     * @return string
     */
    public function getMemo()
    {
        return $this->memo;
    }

    /**
     * Set rDatetime
     *
```

```
* @param \DateTime $rDatetime
* @return ToDo
*/
public function setRDatetime($rDatetime)
{
    $this->rDatetime = $rDatetime;

    return $this;
}

/**
 * Get rDatetime
 *
 * @return \DateTime
 */
public function getRDatetime()
{
    return $this->rDatetime;
}

/**
 * Set uDatetime
 *
 * @param \DateTime $uDatetime
 * @return ToDo
 */
public function setUDatetime($uDatetime)
{
    $this->uDatetime = $uDatetime;
}
```

```
        return $this;
    }

    /**
     * Get uDatetime
     *
     * @return \DateTime
     */
    public function getUDatetime()
    {
        return $this->uDatetime;
    }

    /**
     * @ORM\PrePersist
     */
    public function refreshRDatetime()
    {
        $this->setRDatetime(new \DateTime());
    }

    /**
     * @ORM\PrePersist
     * @ORM\PreUpdate
     */
    public function refreshUDatetime()
    {
        $this->setUDatetime(new \DateTime());
    }
}
```

```
}
```

```
}
```

## FormType

- 次にentityに結びついたFormTypeを作成します。

```
<?php
```

```
namespace AppBundle\Form;
```

```
use Symfony\Component\Form\AbstractType;
```

```
use Symfony\Component\Form\Extension\Core\Type as InputType;
```

```
use Symfony\Component\Form\FormBuilderInterface;
```

```
use Symfony\Component\OptionsResolver\OptionsResolver;
```

```
use Symfony\Component\Validator\Constraints;
```

```
class ToDoType extends AbstractType
```

```
{
```

```
    /**
```

```
     * {@inheritdoc}
```

```
     */
```

```
    public function buildForm(FormBuilderInterface $builder, a
```

```
    {
```



```
$builder
    ->add('task',
        InputType\ChoiceType::class,
        [
            'choices' => [
                '1' => 'work',
                '2' => 'hobby'
            ],

            'constraints' =>
                [
                    new Constraints\NotBlank()
                ]
        ]
    )
    ->add('memo',
        InputType\TextareaType::class,
        [
            'constraints' => [
                new Constraints\NotBlank(),
                new Constraints\Length(['max' => 255])
            ]
        ]
    );

$entity = $builder->getData();
if(!$entity->getId()){
    $builder->add('Create', InputType\SubmitType::clas
        [
            'attr' => [
```

```
        'class' => 'btn btn-primary'
    ]
    ]
    );
}
}

/**
 * {@inheritdoc}
 */
public function configureOptions(OptionsResolver $resolver
{
    $resolver->setDefaults(array(
        'data_class' => 'AppBundle\Entity\ToDo'
    ));
}

/**
 * {@inheritdoc}
 */
public function getBlockPrefix()
```

```
{  
    return 'appbundle_todo';  
}  
  
}
```

- configureOptions
  - `data_class` の部分でこのFormTypeクラスと結びつく Entityのクラス名を指定しています。
- buildForm
  - 基本的にどんなパターンの実装でも使う可能性があるのは、`buildForm`の中で記載されている `add` メソッドです。この部分で実際にformに作成されるinputの種類とvalidationを記載することができます。今回の場合はユーザーが編集できる項目は `task` と `memo` の2つだけなので二つの項目をaddしています。第1引数にメンバーの名前、第2引数にformの種類、第3引数にはoptionを渡せます。

```
->add('task',
    InputType\ChoiceType::class,
    [
        'choices' => [
            '1' => 'work',
            '2' => 'hobby'
        ]
    ],
    [
        'constraints' =>
            [
                new Constraints\NotBlank()
            ]
    ]
)
```

上記の部分ではToDoのEntityのtaskの入力項目を選択式(choiceType)でフォームを生成し、実際のセレクトボックスの選択肢を第三引数に配列でchoicesで渡しています。

## Controller

- Controllerクラスです。自動生成で作成したのでCRUDが一通り実装されていますが、今回は新規と編集のみ説明しま

す。

```
namespace AppBundle\Controller;

use AppBundle\Entity\ToDo;
use AppBundle\Form\ToDoType;
use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Method;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;

/**
 * Todo controller.
 *
 * @Route("todo")
 */
class ToDoController extends Controller
{
    /**
     * Lists all toDo entities.
     *
     * @Route("/", name="todo_index")
     * @Method("GET")
     */
    public function indexAction()
    {
        $em = $this->getDoctrine()->getManager();
    }
}
```

```
$toDos = $em->getRepository('AppBundle:ToDo')->findAll

return $this->render('todo/index.html.twig', array(
    'toDos' => $toDos,
));
}

/**
 * Creates a new toDo entity.
 *
 * @Route("/new", name="todo_new")
 * @Method({"GET", "POST"})
 */
public function newAction(Request $request)
{
    $toDo = new Todo();
    $form = $this->createForm(new ToDoType(), $toDo);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($toDo);
        $em->flush($toDo);

        return $this->redirectToRoute('todo_show', array('
    }

    return $this->render('todo/new.html.twig', array(
        'toDo' => $toDo,
```

```
        'form' => $form->createView(),
    ));
}

/**
 * Finds and displays a todo entity.
 *
 * @Route("/{id}", name="todo_show")
 * @Method("GET")
 */
public function showAction(Todo $todo)
{
    $deleteForm = $this->createDeleteForm($todo);

    return $this->render('todo/show.html.twig', array(
        'todo' => $todo,
        'delete_form' => $deleteForm->createView(),
    ));
}

/**
 * Displays a form to edit an existing todo entity.
 *
 * @Route("/{id}/edit", name="todo_edit")
 * @Method({"GET", "POST"})
 */
public function editAction(Request $request, Todo $todo)
{
    $deleteForm = $this->createDeleteForm($todo);
```

```
$editForm = $this->createForm(new ToDoType(), $todo);
$editForm->handleRequest($request);

if ($editForm->isSubmitted() && $editForm->isValid())
    $this->getDoctrine()->getManager()->flush();

    return $this->redirectToRoute('todo_edit', array('
}

return $this->render('todo/edit.html.twig', array(
    'todo' => $todo,
    'edit_form' => $editForm->createView(),
    'delete_form' => $deleteForm->createView(),
));
}

/**
 * Deletes a todo entity.
 *
 * @Route("/{id}", name="todo_delete")
 * @Method("DELETE")
 */
public function deleteAction(Request $request, ToDo $todo)
{
    $form = $this->createDeleteForm($todo);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
```



```
        $em->remove($todo);
        $em->flush();
    }

    return $this->redirectToRoute('todo_index');
}

/**
 * Creates a form to delete a todo entity.
 *
 * @param Todo $todo The todo entity
 *
 * @return \Symfony\Component\Form\Form The form
 */
private function createDeleteForm(Todo $todo)
{
    return $this->createFormBuilder()
        ->setAction($this->generateUrl('todo_delete', array($todo)))
        ->setMethod('DELETE')
        ->getForm()
    ;
}
}
```

- 新規作成

新規作成は下記のactionで行っています。 \$this->

>createForm(new ToDoType(), \$todo); の部分で空の entityとformTypeを渡し、formのインスタンスを作成し、view側に渡しています。 \$form-

>handleRequest(\$request) の部分でviewから渡された Requestの中からEntityに紐づく値とformの状態を取得します。 値を取得した後 if (\$form->isSubmitted() && \$form->isValid()) の所で、\$formがsubmitされているかつ、formTypeのvalidationが通るかを確認し、通った場合はdoctrineを使ってDBの登録を行います。

```
/**
 * Creates a new todo entity.
 *
 * @Route("/new", name="todo_new")
 * @Method({"GET", "POST"})
 */
public function newAction(Request $request)
{
    $todo = new Todo();
    $form = $this->createForm(new ToDoType(), $todo);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $em = $this->getDoctrine()->getManager();
        $em->persist($todo);
    }
}
```

```
$em->flush($todo);

return $this->redirectToRoute('todo_show', array('
}

return $this->render('todo/new.html.twig', array(
    'todo' => $todo,
    'form' => $form->createView(),
));
}
```

- 編集

次に編集部分の解説です。編集と似た構成ですが、編集の場合は既存のEntityを事前取得するところが異なります。 `@Route("/{id}/edit", name="todo_edit")` で指定した{id}の値を利用し、既存のEntityはParamConverterを介し、自動で取得しています。取得したentityは第二引数に指定しています。取得したtodoのEntityをcreateFormの第二引数に渡すことでformに既存のEntityの値を与えています。残りの部分は新規作成と同様で `$form->handleRequest($request)` の部分でviewから渡されたRequestの中からEntityに紐づく値とformの状態を取得します。値を取得した後 `if ($form->isSubmitted() &&`

`$form->isValid()` の所で、`$form`がsubmitされているかつ、`formType`のvalidationが通るかを確認し、通った場合はdoctrineを使ってDBの登録を行います。

今回はdeleteの部分の説明は割愛します。

```
/**
 * Displays a form to edit an existing todo entity.
 *
 * @Route("/{id}/edit", name="todo_edit")
 * @Method({"GET", "POST"})
 */
public function editAction(Request $request, Todo $todo)
{
    $deleteForm = $this->createDeleteForm($todo);
    $editForm = $this->createForm(new TodoType(), $todo);
    $editForm->handleRequest($request);

    if ($editForm->isSubmitted() && $editForm->isValid()) {
        $this->getDoctrine()->getManager()->flush();

        return $this->redirectToRoute('todo_edit', array('id' => $todo->getId()));
    }

    return $this->render('todo/edit.html.twig', array(
        'todo' => $todo,
        'edit_form' => $editForm->createView(),
    ));
}
```

```
        'delete_form' => $deleteForm->createView(),  
    ));  
}
```

## view

- 新規作成画面

新規作成画面はこれだけです。formを実際に作っている部分は下記のコードでform\_startとform\_endでformの開始、終了位置を指定、form\_widget(form)の部分で渡されたTodoTypeにaddされた入力箇所のhtmlを出力しています。

```
{% extends 'base.html.twig' %}  
  
{% block body %}  
    <h1>Todo creation</h1>  
  
    {{ form_start(form) }}  
        {{ form_widget(form) }}  
    {{ form_end(form) }}  
  
    <ul class="unstyled">  
        <li>  
            <a id="index" class="btn btn-info" href="{{ path(''
```

```
</li>
</ul>
{% endblock %}
```

- 実際の画面

## Todo creation

Task

Memo

Create

Back to the list

```
{{ form_start(form) }}
    {{ form_widget(form) }}
{{ form_end(form) }}
```

- 編集 編集もほぼ同様です。

```
{% extends 'base.html.twig' %}

{% block body %}
    <h1>Todo edit</h1>

    {{ form_start(edit_form) }}
        {{ form_widget(edit_form) }}
    {{ form_end(edit_form) }}

    <ul class="unstyled">
        <li>
            <a class="btn btn-info" href="{{ path('todo_index') }}>
        </li>
        <li>
            {{ form_start(delete_form) }}
                <input id="delete" class="btn btn-danger" type="button" value="Delete" />
            {{ form_end(delete_form) }}
        </li>
    </ul>
{% endblock %}
```

- 実際の画面

## Todo edit

Task

work

Memo

test

Edit

Back to the list

Delete

## 最後に

---

symfonyではEntityにannotationで直接validationを付与して、validationを行うことができます。

しかし、実際の実装において必ずしもentityの項目と画面の項目の実装が一致するとは限りません。

実際のvalidation処理の管理はformTypeクラスに切り分けて、entityクラスは出来るだけ、実態に関連する項目や処理だけを



入れた方がコードとして読みやすい気がします。  
ちゃんと扱えれば、なんでも出来るframeworkだと思います  
が、本当まとまった情報が少ない・・・

## github

---

yutachaos/symfony-form-sample

<https://github.com>

MEN

### 参考

[http://api.symfony.com/2.8/Symfony/Component/Form/FormBuilderInterface.html#method\\_add](http://api.symfony.com/2.8/Symfony/Component/Form/FormBuilderInterface.html#method_add)

<http://symfony.com/doc/current/reference/forms/types.html>