



Symfony

# Trabalhando com Formulários no Symfony2

Há 3 maneiras!

## 1 DIRETAMENTE NO CONTROLADOR ← Simples e Fácil

Traduz os dados enviados pelo usuário de volta as propriedades de um objeto

### Classe de Dados

#### Entidade

src/Acme/TaskBundle/Entity/Task.php

```

namespace Acme\TaskBundle\Entity;

class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }
    public function setTask($task)
    {
        $this->task = $task;
    }

    public function getDueDate()
    {
        return $this->dueDate;
    }
    public function setDueDate($dueDate)
    {
        $this->dueDate = $dueDate;
    }
}

```

### Criar / Manipular a Submissão

#### Controlador

src/Acme/TaskBundle/Controller/DefaultController.php

```

namespace Acme\TaskBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Acme\TaskBundle\Entity\Task;
use Symfony\Component\HttpFoundation\Request;

class DefaultController extends Controller
{
    public function newAction(Request $request)
    {
        // just setup a fresh $task object
        $task = new Task();

        $form = $this->createFormBuilder($task)
            ->add('task', 'text')
            ->add('dueDate', 'date')
            ->getForm();

        $request = $this->getRequest();

        if ($request->getMethod() == 'POST') {
            $form->bindRequest($request);

            if ($form->isValid()) {
                // ...
                return $this->redirect($this->generateUrl('task_success'));
            }
            // evita o reenvio dos dados
        }

        return $this->render('AcmeTaskBundle:Default:new.html.twig',
            array(
                'form' => $form->createView(),
            ));
    }
}

```

cria o formulário

tipo do campo determina quais tags HTML de formulário serão renderizadas para o campo

chama a validação

evita o reenvio dos dados

renderiza o formulário (visão)

### Renderizar

#### Visão

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

```

TWIG
<form action="{{ path('task_new') }}" method="post"
    {{ form_enctype(form) }}>
    {{ form_widget(form) }}

    <input type="submit" />
</form>

OU

src/Acme/TaskBundle/Resources/views/Default/new.html.php
PHP
<form action="<?php echo $view['router']->generate('task_new')?>"
    method="post" <?php echo $view['form']->enctype($form) ?>"
    <?php echo $view['form']->widget($form) ?>

    <input type="submit" />
</form>

```

Task

Due date Jul 24, 2011

Você pode renderizar todo o formulário com apenas uma linha (como acima) ou personalizar cada parte de seu formulário.

Tudo pode ser personalizado (Usando fragmentos e temas)

Para personalizar a forma como o formulário será renderizado:

**Twig:** sobrescreva o bloco apropriado

**PHP:** sobrescreva o template existente criando um novo

### Validar

A validação é feita adicionando um conjunto de regras (chamadas constraints) à uma classe.

A chamada `$form->isValid()` é um atalho que pergunta ao objeto `$task` se ele tem ou não dados válidos.

src/Acme/TaskBundle/Entity/Task.php

```

Annotations
use Symfony\Component\Validator\Constraints as Assert;

class Task
{
    /**
     * @Assert\NotBlank()
     */
    public $task;

    /**
     * @Assert\NotBlank()
     * @Assert\Type("\DateTime")
     */
    protected $dueDate;
}

```

OU

src/Acme/TaskBundle/Resources/config/validation.yml

```

YAML
Acme\TaskBundle\Entity\Task:
    properties:
        task:
            - NotBlank: -
        dueDate:
            - NotBlank: -
            - Type: \DateTime

```

Você pode usar **annotations**, **YAML**, **XML** ou **PHP**.  
Veja <http://symfony.com/doc/current/book/forms.html>



Symfony

# Trabalhando com Formulários no Symfony2

Há 3 maneiras!

## 2 CRIANDO CLASSES DE FORMULÁRIOS (classes PHP independentes)

← Melhor forma (melhor prática, reusável)

↙ Não é responsabilidade do formulário

### Criar

#### Classe de Formulário

src/Acme/TaskBundle/Form/Type/TaskType.php

```

namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilder;

class TaskType extends AbstractType
{
    public function buildForm(FormBuilder $builder, array $options)
    {
        $builder->add('task');
        $builder->add('dueDate', null, array('widget' => 'single_text'));
    }

    public function getDefaultOptions(array $options)
    {
        return array(
            'data_class' => 'Acme\TaskBundle\Entity\Task',
        );
    }

    public function getName()
    {
        return 'task';
    }
}

```

data\_class

nome da classe que contém os dados implícitos  
- para formulários embutidos  
- permite ao formulário ver as definições de tipo da classe

Sempre definir! (melhor prática)

### Manipular a Submissão

#### Controlador

src/Acme/TaskBundle/Controller/DefaultController.php

```

// add this new use statement at the top of the class
use Acme\TaskBundle\Form\Type\TaskType;

public function newAction()
{
    $task = // ...
    $form = $this->createForm(new TaskType(), $task);

    // ...
}

```

### Classe de Dados

#### Entidade

src/Acme/TaskBundle/Entity/Task.php

```

namespace Acme\TaskBundle\Entity;

class Task
{
    protected $task;
    protected $dueDate;

    public function getTask()
    {
        return $this->task;
    }

    public function setTask($task)
    {
        $this->task = $task;
    }

    // ...
}

```

### Renderizar

#### Visão

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

```

TWIG

<form action="{{ path('task_new') }}" method="post"
      {{ form_enctype(form) }}>
    {{ form_widget(form) }}

    <input type="submit" />
</form>

```

OU

src/Acme/TaskBundle/Resources/views/Default/new.html.php

```

PHP

<form action="<?php echo $view['router']->generate('task_new')?>"
      method="post" <?php echo $view['form']->enctype($form) ?> >
    <?php echo $view['form']->widget($form) ?>

    <input type="submit" />
</form>

```

Task

Due date Jul ▾ 24 ▾, 2011 ▾

Você pode renderizar todo o formulário com apenas uma linha (como acima) ou personalizar cada parte de seu formulário.

Tudo pode ser personalizado (Usando fragmentos e temas)

Para personalizar a forma como o formulário será renderizado:

Twig: sobrescreva o bloco apropriado

PHP: sobrescreva o template existente criando um novo

### Validar

A validação é feita adicionando um conjunto de regras (chamadas constraints) a uma classe.

A chamada `$form->isValid()` é um atalho que pergunta ao objeto `$task` se ele tem ou não dados válidos.

src/Acme/TaskBundle/Entity/Task.php

#### Annotations

use Symfony\Component\Validator\Constraints as Assert;

```

class Task
{
    /**
     * @Assert\NotBlank()
     */
    public $task;

    /**
     * @Assert\NotBlank()
     * @Assert\Type("\DateTime")
     */
    protected $dueDate;
}

```

OU

src/Acme/TaskBundle/Resources/config/validation.yml

#### YAML

```

Acme\TaskBundle\Entity\Task:
    properties:
        task:
            - NotBlank: -
        dueDate:
            - NotBlank: -
            - Type: \DateTime

```

Você pode usar [annotations](#), [YAML](#), [XML](#) ou [PHP](#).

Veja <http://symfony.com/doc/current/book/forms.html>



Symfony

# Trabalhando com Formulários no Symfony2

Há 3 maneiras!

## 3

### USANDO UM FORMULÁRIO SEM CLASSE DE DADOS



Para Formulários Simples (retorna um array dos dados submetidos)

#### Criar / Manipular a Submissão

##### Controlador

src/Acme/ContactBundle/Controller/ContactController.php

```
// make sure you've imported the Request namespace above the class
use Symfony\Component\HttpFoundation\Request
// ...

public function contactAction(Request $request)
{
    $defaultData = array('message' => 'Type your message here');
    $form = $this->createFormBuilder($defaultData)
        ->add('name', 'text')
        ->add('email', 'email')
        ->add('message', 'textarea')
        ->getForm();

    if ($request->getMethod() == 'POST') {
        $form->bindRequest($request);

        // data is an array with "name", "email", and "message" keys
        $data = $form->getData();
    }

    // ... render the form
}
```

Por padrão, `$form->getData()` retorna um array ao invés de um objeto. Há duas formas de alterar esse comportamento e amarrar o formulário a um objeto:

1. Passar um objeto ao criar o formulário (como o primeiro argumento para o `createFormBuilder` ou o segundo argumento para o `createForm`);
2. Declarar a opção `data_class` no seu formulário.

#### Validar

A validação é feita adicionando você mesmo as constraints e passando-as ao seu formulário.

##### Sem Classe de Formulário

```
// import the namespaces above your controller class
use Symfony\Component\Validator\Constraints\Email;
use Symfony\Component\Validator\Constraints\MinLength;
use Symfony\Component\Validator\Constraints\Collection;

$collectionConstraint = new Collection(array(
    'name' => new MinLength(5),
    'email' => new Email(array(
        'message' => 'Invalid email address'
    ));
});

// create a form, no default values, pass in the constraint option
$form = $this->createFormBuilder(null, array(
    'validation_constraint' => $collectionConstraint,
))->add('email', 'email');
// ...
```

##### Com Classe de Formulário

```
namespace Acme\TaskBundle\Form\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilder;
use Symfony\Component\Validator\Constraints\Email;
use Symfony\Component\Validator\Constraints\MinLength;
use Symfony\Component\Validator\Constraints\Collection;

class ContactType extends AbstractType
{
    // ...

    public function getDefaultOptions(array $options)
    {
        $collectionConstraint = new Collection(array(
            'name' => new MinLength(5),
            'email' => new Email(array(
                'message' => 'Invalid email address'
            ));
        ));

        return array('validation_constraint' => $collectionConstraint);
    }
}
```

Se você estiver usando uma classe de formulário, sobrescreva o método `getDefaultOptions` para especificar a opção

Quando você chamar `$form->isValid()`, a configuração das constraints que você definiu será executada em relação aos dados do seu formulário.

#### Acessando os valores do POST diretamente através do objeto do pedido

```
$this->get('request')->request->get('name');
```

No entanto, na maioria dos casos, usar o método `getData()` é uma escolha melhor, já que ele retorna os dados (geralmente um objeto) depois que foi transformado pelo framework de formulário.

#### Renderizar

##### Visão

src/Acme/TaskBundle/Resources/views/Default/new.html.twig

```
TWIG
<form action="{{ path('task_new') }}" method="post"
    {{ form_enctype(form) }}>
    {{ form_widget(form) }}

    <input type="submit" />
</form>
```

OU

src/Acme/TaskBundle/Resources/views/Default/new.html.php

```
PHP
<form action="<?php echo $view['router']->generate('task_new')?>"
    method="post" <?php echo $view['form']->enctype($form) ?> >
    <?php echo $view['form']->widget($form) ?>

    <input type="submit" />
</form>
```



Task

Duedate Jul 24 2011

Você pode renderizar todo o formulário com apenas uma linha (como acima) ou personalizar cada parte de seu formulário.



#### Tudo pode ser personalizado (Usando fragmentos e temas)

Para personalizar a forma como o formulário será renderizado:  
Twig: sobrescreva o bloco apropriado  
PHP: sobrescreva o template existente criando um novo