The HttpKernel Component: the Controller Resolver

You might think that our framework is already pretty solid and you are probably right. But let's see how we can improve it nonetheless.

Right now, all our examples use procedural code, but remember that controllers can be any valid PHP callbacks. Let's convert our controller to a proper class::

```
class LeapYearController
{
    public function indexAction($request)
    {
        if (is_leap_year($request->attributes->get('year'))) {
            return new Response('Yep, this is a leap year!');
        }

        return new Response('Nope, this is not a leap year.');
    }
}
```

Update the route definition accordingly::

```
$routes->add('leap_year', new Routing\Route('/is_leap_year/{year}', array(
    'year' => null,
    '_controller' => array(new LeapYearController(), 'indexAction'),
)));
```

The move is pretty straightforward and makes a lot of sense as soon as you create more pages but you might have noticed a non-desirable side-effect... The `LeapYearController` class is *always* instantiated, even if the requested URL does not match the `leap_year` route. This is bad for one main reason: performance wise, all controllers for all routes must now be instantiated for every request. It would be better if controllers were lazy-loaded so that only the controller associated with the matched route is instantiated.

To solve this issue, and a bunch more, let's install and use the HttpKernel component:

.. code-block:: sh

```
$ php composer.phar require symfony/http-kernel
```

The HttpKernel component has many interesting features, but the one we need right now is the *controller resolver*. A controller resolver knows how to determine the controller to execute and the arguments to pass to it, based on a Request object. All controller resolvers implement the following interface::

```
namespace Symfony\Component\HttpKernel\Controller;

interface ControllerResolverInterface
{
    function getController(Request $request);

    function getArguments(Request $request, $controller);
}
```

The getController() method relies on the same convention as the one we have defined earlier: the _controller request attribute must contain the controller associated with the Request. Besides the built-in PHP callbacks, getController() also supports strings composed of a class name followed by two colons and a method name as a valid callback, like 'class::method'::

```
$routes->add('leap_year', new Routing\Route('/is_leap_year/{year}', array(
    'year' => null,
    '_controller' => 'LeapYearController::indexAction',
)));
```

To make this code work, modify the framework code to use the controller resolver from HttpKernel::

```
use Symfony\Component\HttpKernel;

$resolver = new HttpKernel\Controller\ControllerResolver();

$controller = $resolver->getController($request);
$arguments = $resolver->getArguments($request, $controller);

$response = call_user_func_array($controller, $arguments);
```

.. note::

As an added bonus, the controller resolver properly handles the error
management for you: when you forget to define a ``_controller`` attribute
for a Route for instance.

Now, let's see how the controller arguments are guessed.getArguments() introspects the controller signature to determine which arguments to pass to it by using the native PHP reflection_.

The indexAction() method needs the Request object as an argument. getArguments() knows when to inject it properly if it is type-hinted correctly::

```
public function indexAction(Request $request)

// won't work
public function indexAction($request)
```

More interesting, getArguments() is also able to inject any Request attribute; the argument just needs to have the same name as the corresponding attribute::

```
public function indexAction($year)
```

You can also inject the Request and some attributes at the same time (as the matching is done on the argument name or a type hint, the arguments order does not matter)::

```
public function indexAction(Request $request, $year)

public function indexAction($year, Request $request)
```

Finally, you can also define default values for any argument that matches an optional attribute of the Request::

```
public function indexAction($year = 2012)
```

Let's just inject the $year request attribute for our controller::

```
class LeapYearController
{
    public function indexAction($year)
    {
        if (is_leap_year($year)) {
            return new Response('Yep, this is a leap year!');
        }

        return new Response('Nope, this is not a leap year.');
    }
}
```

The controller resolver also takes care of validating the controller callable and its arguments. In case of a problem, it throws an exception with a nice message explaining the problem (the controller class does not exist, the method is not defined, an argument has no matching attribute, ...).

.. note::

With the great flexibility of the default controller resolver, you might
wonder why someone would want to create another one (why would there be an
interface if not?). Two examples: in Symfony2, ``getController()`` is
enhanced to support `controllers as services`_; and in
`FrameworkExtraBundle`_, ``getArguments()`` is enhanced to support
parameter converters, where request attributes are converted to objects
automatically.

Let's conclude with the new version of our framework::

```
<?php

// example.com/web/front.php

require_once __DIR__.'/../vendor/autoload.php';

use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing;
use Symfony\Component\HttpKernel;

function render_template(Request $request)
{
    extract($request->attributes->all());
    ob_start();
    include sprintf(__DIR__.'/../src/pages/%s.php', $_route);

    return new Response(ob_get_clean());
}

$request = Request::createFromGlobals();
$routes = include __DIR__.'/../src/app.php';

$context = new Routing\RequestContext();
$context->fromRequest($request);
$matcher = new Routing\Matcher\UrlMatcher($routes, $context);
$resolver = new HttpKernel\Controller\ControllerResolver();

try {
    $request->attributes->add($matcher->match($request->getPathInfo()));
    $controller = $resolver->getController($request);
    $arguments = $resolver->getArguments($request, $controller);

    $response = call_user_func_array($controller, $arguments);
} catch (Routing\Exception\ResourceNotFoundException $e) {
    $response = new Response('Not Found', 404);
} catch (Exception $e) {
    $response = new Response('An error occurred', 500);
}

$response->send();
```

Think about it once more: our framework is more robust and more flexible than ever and it still has less than 40 lines of code.

.. _reflection: http://php.net/reflection .. _FrameworkExtraBundle: http://symfony.com/doc/current/bundles/SensioFrameworkExtraBundle/annotations/converters.html .. _controllers as services: http://symfony.com/doc/current/cookbook/controller/service.html