



Beyond symfony 1.2

Fabien Potencier

Work on symfony 2
started 2 years ago
with a brand new code base

symfony 2 goals

Learn from our mistakes

Take user feedback into account

But then, I realized that

Starting from scratch is a lot of work!

We loose all the debugging we have already done

I didn't want symfony 2 to be the next Perl 6 ;)

But I learned a lot with this new code

I used it to test new ideas...

...that were later incorporated into symfony 1

Event dispatcher

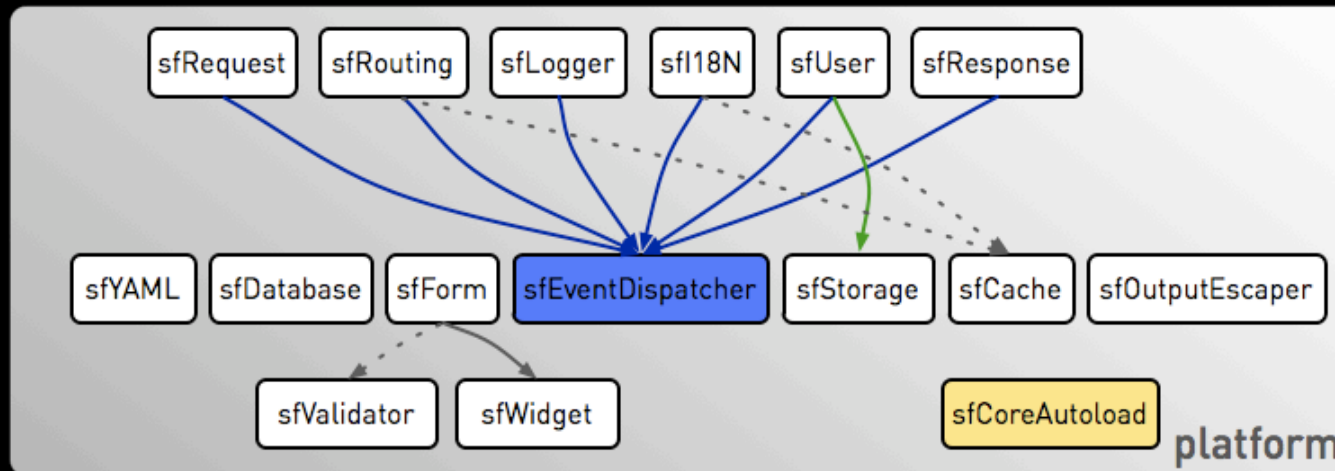
Form framework

symfony 2

- symfony 2 will be an evolution of symfony 1, not a revolution
- A lot of changes in symfony 1.1 and 1.2 are done to support symfony 2
- symfony 1.1 decoupling was to support symfony 2
- There will be a symfony 1.3

symfony 2

- Based on the same symfony platform



- New libraries / sub-frameworks
 - Dependency injection container
 - Real template layer
 - New controller layer

Dependency Injection

The world of programming
is evolving fast... very fast

But the ultimate goal of these
changes is the same

Build better tools to avoid reinventing the wheel

Build on existing code / work / libraries

Specialize for your specific needs

That's why Open-Source wins

Better tools are possible because
we base our work on existing work

We learn from other projects

We can adopt their ideas

We can make them better

This is the process of evolution

symfony

- PHP
 - Mojavi : Controller
 - Propel : ORM
 - Doctrine : ORM
- Ruby
 - Rails : Helpers, routing
- Java
 - Spring : Dependency injection container
- Python
 - Django : request / response paradigm, controller philosophy
- ... and much more

Read code to learn

This is why Open-Source wins

Open-Source drives

Creativity Innovation

Create a website

- Assembly code
- C code
- Operating system
- MySQL, Apache
- PHP
- ...
- Your code

Create a website

- You really need a framework nowadays
- But you want it to be flexible and customizable
- The framework needs to find the sweet spot
 - Features / Complexity
 - Easyness / Extensibility
- Each version of symfony tries to be closer to this sweet spot
- I hope symfony 2 will be the one
- A framework is all about reusability

Reusability

Customization / Extension
Configuration

Documentation / Support

Dependency Injection

A symfony example

In symfony, you manipulate a User object

- To manage the user culture
- To authenticate the user
- To manage her credentials
- ...

- `setCulture()`, `getCulture()`
- `isAuthenticated()`
- `addCredential()`
- ...

The User information need to be persisted
between HTTP requests

We use the PHP session for the Storage

```
<?php
```

```
class User
```

```
{
```

```
    protected $storage;
```

```
    function __construct()
```

```
    {
```

```
        $this->storage = new SessionStorage();
```

```
    }
```

```
}
```

```
$user = new User();
```

I want to change the session cookie name

```
<?php
```

```
class User
```

```
{
```

```
    protected $storage;
```

```
    function __construct()
```

```
    {
```

```
        $this->storage = new SessionStorage();
```

```
    }
```

```
}
```

```
sfConfig::set('storage_session_name', 'SESSION_ID');
```

```
$user = new User();
```

Add a global
Configuration?

```
<?php
```

```
class User  
{
```

```
    protected $storage;
```

```
    function __construct($sessionName)
```

```
    {
```

```
        $this->storage = new SessionStorage($sessionName);
```

```
    }
```

```
}
```

```
$user = new User('SESSION_ID');
```



Configure via
User?

```
<?php
```

```
class User
```

```
{  
    protected $storage;
```

```
    function __construct($options)
```

```
    {  
        $this->storage = new SessionStorage($options);  
    }
```

```
}
```

```
$user = new User(  
    array('session_name' => 'SESSION_ID')
```

```
);
```

Configure with
an array

I want to change the session storage engine

Filesystem

MySQL

SQLite

...

```
<?php
```

```
class User  
{
```

```
    protected $storage;
```

```
    function __construct()  
    {
```

```
        $this->storage = Context::get('session_storage');  
    }
```

```
}
```

```
$user = new User();
```



Use a global
context object?

The User now depends on the Context

Instead of hardcoding
the Storage dependency
in the User class

Inject the storage dependency
in the User object

```
<?php
```

```
class User
```

```
{
```

```
    protected $storage;
```

```
    function __construct(StorageInterface $storage)
```

```
    {
```

```
        $this->storage = $storage;
```

```
    }
```

```
}
```

```
$storage = new Storage(
```

```
    array('session_name' => 'SESSION_ID')
```

```
);
```

```
$user = new User($storage);
```

Use built-in Storage strategies
Configuration becomes natural
Wrap third-party classes (Adapter)
Mock the Storage object (for test)

Easy without changing the User class

That's Dependency Injection

Nothing more

« Dependency Injection is where components are given their dependencies through their constructors, methods, or directly into fields. »

<http://www.picocontainer.org/injection.html>

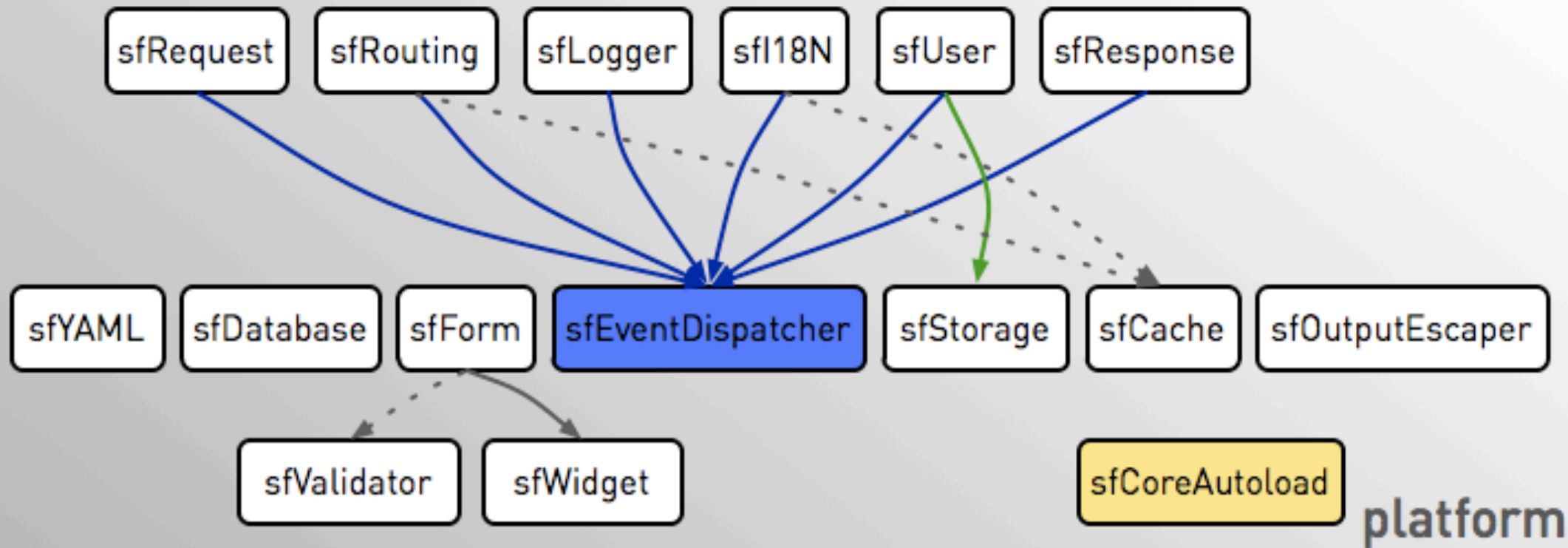
Configurable

Extensible

Decoupled

Reusable

symfony platform



```
<?php
```

```
$dispatcher = new sfEventDispatcher();
```

```
$request = new sfWebRequest($dispatcher);
```

```
$response = new sfWebResponse($dispatcher);
```

```
$storage = new sfSessionStorage(  
    array('session_name' => 'symfony')  
);
```

```
$user = new sfUser($dispatcher, $storage);
```

```
$cache = new sfFileCache(  
    array('dir' => dirname(__FILE__).'/cache')  
);
```

```
$routing = new sfPatternRouting($dispatcher, $cache);
```

```
<?php
```

```
class Application
```

```
{
```

```
function __construct()
```

```
{
```

```
    $this->dispatcher = new sfEventDispatcher();
```

```
    $this->request = new sfWebRequest($this->dispatcher);
```

```
    $this->response = new sfWebResponse($this->dispatcher);
```

```
    $this->storage = new sfSessionStorage(
```

```
        array('session_name' => 'symfony')
```

```
    );
```

```
    $this->user = new sfUser($this->dispatcher, $this->storage);
```

```
    $this->cache = new sfFileCache(
```

```
        array('dir' => dirname(__FILE__).'/cache')
```

```
    );
```

```
    $this->routing = new sfPatternRouting($this->dispatcher,  
    $this->cache);
```

```
    }
```

```
}
```

```
$application = new Application();
```

Back to square 1

We need a Container

Describe objects
and their relationships

Configure them

Instantiate them

Dependency Injection in symfony 2

sfServiceContainer

```
<?php
```

```
$container = new sfServiceContainer(array(  
    'dispatcher' => new sfServiceDefinition('sfEventDispatcher'),  
));
```

```
$dispatcher = $container->getService('dispatcher');
```



It is a just a
Service Locator?

```
<?php
```

```
$dispatcherDef = new sfServiceDefinition('sfEventDispatcher');
```

```
$requestDef = new sfServiceDefinition('sfWebRequest',  
    array(new sfServiceReference('dispatcher'))  
);
```

```
$container = new sfServiceContainer(array(  
    'dispatcher' => $dispatcherDef,  
    'request'    => $requestDef,  
));
```

```
$request = $container->getService('request');
```

```
$request = $container->getService( 'request' );
```

Get the configuration for the request service

The sfWebRequest constructor must be given a dispatcher service

Get the dispatcher object from the container

Create a sfWebRequest object by passing the constructor arguments

```
$request = $container->getService( 'request' );
```

is roughly equivalent to

```
$dispatcher = new sfEventDispatcher();  
$request = new sfWebRequest($dispatcher);
```

```
<?php
```

```
$container = new sfServiceContainer(array(  
    'dispatcher' => new sfServiceDefinition('sfEventDispatcher'),  
    'request'    => new sfServiceDefinition('sfWebRequest',  
        array(new sfServiceReference('dispatcher'))  
    ),  
));
```

PHP



```
<service id="dispatcher" class="sfEventDispatcher" />
```

```
<service id="request" class="sfWebRequest">  
    <constructor_arg type="service" id="dispatcher" />  
</service>
```

XML



```
dispatcher:  
    class: sfEventDispatcher
```

```
request:  
    class: sfWebRequest  
    constructor_args: [@dispatcher]
```

YAML

```
<?php
```

```
$container = new sfServiceContainer(array(  
    'dispatcher' => new sfServiceDefinition('sfEventDispatcher'),  
));
```

```
<parameter key="dispatcher.class">sfEventDispatcher</parameter>
```

```
<service id="dispatcher" class="%dispatcher.class%" />
```

```
parameters:
```

```
    dispatcher.class: sfEventDispatcher
```

```
services:
```

```
    dispatcher:
```

```
        class: %dispatcher.class%
```

```
<parameter key="core.with_logging">true</parameter>
<parameter key="core.charset">UTF-8</parameter>
<parameter key="response.class">sfWebResponse</parameter>

<service id="response" class="%response.class%" global="false">
  <constructor_arg type="service" id="event_dispatcher" />
  <constructor_arg type="collection">
    <arg key="logging">%core.with_logging%</arg>
    <arg key="charset">%core.charset%</arg>
  </constructor_arg>
</service>
```

```
parameters:
  core.with_logging: true
  core.charset:      UTF-8
  response.class:    sfWebResponse
```

```
services:
  response:
    class: %response.class%
    global: false
    constructor_args:
      - @event_dispatcher
      - [logging: %core.with_logging%, charset: %core.charset%]
```

```
// YML
```

```
$loader = new sfServiceLoaderYml(array('/paths/to/config/dir'));  
list($definitions, $parameters) = $loader->load('/paths/to/yml');
```

```
// XML
```

```
$loader = new sfServiceLoaderXml(array('/paths/to/config/dir'));  
list($definitions, $parameters) = $loader->load('/paths/to/xml');
```

```
// Create the container
```

```
$container = new sfServiceContainer($definitions, $parameters);
```

```
// Access parameters
```

```
$withLogging = $container['core.with_logging'];
```

```
// Access services
```

```
$response = $container->response;
```

```
<import resource="core.xml" />  
  
<parameter key="dispatcher.class">sfEventDispatcher</parameter>  
  
<service id="dispatcher" class="%disaptcher.class%" />  
  
<import resource="app_dev.xml" />
```

```
imports_1: core.xml
```

```
parameters:  
  dispatcher.class: sfEventDispatcher
```

```
services:  
  dispatcher:  
    class: %dispatcher.class%
```

```
imports_2: app_dev.xml
```

sfServiceContainer replaces several symfony 1.X concepts

-sfContext

-sfConfiguration

-sfConfig

-factories.yml

-settings.yml / logging.yml / i18n.yml

Into one integrated system

```
services:  
  response:  
    class: %response.class%  
    global: false
```

Each time you ask for a response object,
you get a new one

```
services:  
  user:  
    class: %user.class%  
    lazy: true
```

By default, the services are created on startup except if lazy is true

Does it scale?

`sfServiceContainerApplication`
caches the service definition

Questions?

Sensio S.A.

92-98, boulevard Victor Hugo
92 115 Clichy Cedex
FRANCE

Tél. : +33 1 40 99 80 80

Contact

Fabien Potencier
fabien.potencier@sensio.com

<http://www.sensiolabs.com/>

<http://www.symfony-project.org/>