



Découpler votre code pour assurer la réutilisabilité et la maintenabilité

Fabien Potencier

Fabien Potencier

- Créateur de Sensio
 - Agence Web
 - Depuis 1998
 - 50 collaborateurs
 - Spécialistes Open-Source
 - Clients grands comptes / institutionnels
- Créateur et développeur principal de symfony
 - Framework PHP

Couplage ?

Coupling is the degree to which each program module relies on each one of the other modules.

[http://en.wikipedia.org/wiki/Coupling_\(computer_science\)](http://en.wikipedia.org/wiki/Coupling_(computer_science))

Couplage faible

Avec un couplage faible, un changement dans un module ne nécessite pas un changement dans l' **implementation** d'un autre module.

Couplage faible

Un module peut interagir
avec un autre à travers
une interface stable

Pourquoi est-ce important ?

Testabilité

La possibilité d'instancier une classe dans un test prouve le niveau de couplage du code

Readabilité

On passe plus de temps
à lire et à maintenir du code,
qu'à en écrire

Maintainabilité

Un changement ici
ne doit pas causer
un problème là-bas

Extensibilité

Des petits modules indépendants
facilite son extension

Réutilisabilité

Le monde du développement
évolue vite... très vite

Mais le but ultime de toutes ces
évolutions reste le même

Construire de meilleurs outils pour éviter de réinventer la roue

Construire sur du code / bibliothèques existants

Le spécialiser pour ses propres besoins

... C'est pour ces raisons que l'Open-Source est
toujours gagnant

La construction d'outils meilleurs est possible
parce que nous les basons sur des outils
existants

On apprend des projets existants

On adopte leurs idées

On les améliore

C'est le processus naturel de l'évolution

Réutilisabilité

Configuration

Personnalisation / Extension

Documentation / Support

Injection de Dépendances

Un exemple Web concret

La plupart des applications doivent stocker les préférences des utilisateurs

- Langue de l'utilisateur
- S'il est authentifié ou non
- Ses droits
- ...

C'est possible avec un objet User

- `setLanguage()`, `getLanguage()`
- `setAuthenticated()`, `isAuthenticated()`
- `addCredential()`, `hasCredential()`
- ...

La classe User doit trouver un moyen de stocker ces données entre les requêtes HTTP

En PHP, on utilise des sessions

```
class SessionStorage
{
    function __construct($cookieName = 'PHP_SESS_ID')
    {
        session_name($cookieName);
        session_start();
    }

    function set($key, $value)
    {
        $_SESSION[$key] = $value;
    }

    // ...
}
```

```
class User
{
    protected $storage;

    function __construct()
    {
        $this->storage = new SessionStorage();
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);
    }

    // ...
}
```

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

Je veux changer le nom du cookie

```
class User
{
    protected $storage;

    function __construct()
    {
        $this->storage = new SessionStorage('SESSION_ID');
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);
    }

    // ...
}

$user = new User();
```

Directement
dans la classe
User

```
class User
{
    protected $storage;

    function __construct()
    {
        $this->storage = new SessionStorage(STORAGE_SESSION_NAME);
    }
}
```

Configuration
globale ?

```
define('STORAGE_SESSION_NAME', 'SESSION_ID');
```

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

```
class User
{
    protected $storage;

    function __construct($sessionName)
    {
        $this->storage = new SessionStorage($sessionName);
    }
}
```

Configuration
via le
constructeur

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

Configuration
avec un
tableau

```
class User
{
    protected $storage;

    function __construct($storageOptions)
    {
        $this->storage = new
        SessionStorage($storageOptions['session_name']);
    }
}
```

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

Je veux changer le support de stockage des sessions

Filesystem

MySQL

SQLite

...

```
class User
{
    protected $storage;

    function __construct()
    {
        $this->storage = Registry::get('session_storage');
    }
}
```

Utilisation d'un
registre global ?

```
$storage = new SessionStorage();
Registry::set('session_storage', $storage);
$user = new User();
```

La classe User dépend maintenant
de la classe Registry

Plutôt que de créer l'instance
Storage dans la classe User

On injecte la dépendance
Storage dans l'objet User

```
class User
{
    protected $storage;

    function __construct($storage)
    {
        $this->storage = $storage;
    }
}
```

Argument du
constructeur ?

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

Utilisation de différent support de stockage

```
class User
{
    protected $storage;

    function __construct($storage)
    {
        $this->storage = $storage;
    }
}
```



Support de
stockage
différent

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

La configuration devient naturelle

```
class User
{
    protected $storage;

    function __construct($storage)
    {
        $this->storage = $storage;
    }
}
```

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



Configuration
naturelle

Utilisation de classes tierces (Interface / Adapter)

```
class User
{
    protected $storage;

    function __construct(ISessionStorage $storage)
    {
        $this->storage = $storage;
    }
}

interface ISessionStorage
{
    function get($key);

    function set($key, $value);
}
```



Ajout d'une
interface

Remplacer l'objet Storage pour les tests (Mock)

```
class User
{
    protected $storage;

    function __construct(ISessionStorage $storage)
    {
        $this->storage = $storage;
    }
}
```

```
class SessionStorageForTests implements ISessionStorage
{
    protected $data
    function set($key, $value)
    {
        self::$data[$key] = $value;
    }
}
```



Mock de la
session

Utilisation de différent supports de stockage

Configuration naturelle

Intégration de classes tierces

Simplicité des tests

Facile sans changement
de la classe User

C'est l'Injection de
Dépendance

Rien de plus

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

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

```
$storage = new SessionStorage();
```

```
// constructor injection
```

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

```
// setter injection
```

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

```
$user->setStorage($storage);
```

```
// property injection
```

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

```
$user->storage = $storage;
```

Un exemple
un peu plus complexe

```
$request = new WebRequest();  
$response = new WebResponse();  
  
$storage = new  
    FileSessionStorage('SESSION_ID');  
$user = new User($storage);  
  
$cache = new FileCache(  
    array('dir' => dirname(__FILE__) . '/cache')  
);  
$routing = new Routing($cache);
```

```
class Application
{
    function __construct()
    {
        $this->request = new WebRequest();
        $this->response = new WebResponse();

        $storage = new FileSessionStorage('SESSION_ID');
        $this->user = new User($storage);

        $cache = new FileCache(
            array('dir' => dirname(__FILE__).' /cache')
        );
        $this->routing = new Routing($cache);
    }
}
```

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

Retour à la case départ

```
class Application
{
    function __construct()
    {
        $request = new WebRequest();
        $response = new WebResponse();

        $storage = new FileSessionStorage('SESSION_ID');
        $user = new User($storage);

        $cache = new FileCache(
            array('dir' => dirname(__FILE__) . '/cache')
        );
        $routing = new Routing($cache);
    }
}
```

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

On a besoin d'un Container

Description des objets et des relations

Configuration


Création

Description d'un service

```
$storageDef = new ServiceDefinition(  
    'FileSessionStorage'  
);
```


Nom de la classe

```
$storageDef = new ServiceDefinition(  
    'FileSessionStorage',  
    array('SESSION_ID')  
);
```



Arguments à
passer au
constructeur

```
$container = new ServiceContainer(array(  
    'storage' => $storageDef,  
));
```



Chaque objet a un
identifiant unique

```
$userDef = new ServiceDefinition('User',  
    array(new ServiceReference('storage'))  
);
```



Référence à un
autre object

```
$storageDef = new ServiceDefinition(
    'FileSessionStorage',
    array('SESSION_ID')
);

$userDef = new ServiceDefinition('User',
    array(new ServiceReference('storage'))
);

$container = new ServiceContainer(array(
    'storage' => $storageDef,
    'user'    => $userDef,
));
```

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

Récupération de la configuration de l'objet user

Le user nécessite un objet storage pour son constructeur

Récupération de la configuration de l'objet storage

Création de l'objet user en injectant le storage

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

Est à peu près équivalent à

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

Un container peu gérer
n'importe quelle classe (POPO)

Les objets ne savent pas
qu'ils sont gérés par un container

Astuces d'implémentation

```
public function getService ($id)
{
    $def = $this->definitions[$id];

    $r = new ReflectionClass($def->getClass());

    if (is_null($r->getConstructor()))
    {
        return $r->newInstance();
    }

    $args = $def->getArguments();
    $args = $this->evaluateArguments($args);

    return $r->newInstanceArgs($args);
}
```

```
public function evaluateArguments($arg)
{
    if (is_array($arg))
    {
        return array_map(
            array($this, 'evaluateArguments'), $arg
        );
    }

    if (
        is_object($arg)
        && $arg instanceof ServiceReference
    )
    {
        return $this->getService($arg);
    }

    return $arg;
}
```

Vers une vraie implémentation

Scope

Quand on récupère un service :

On veut un nouvel objet à chaque fois ?

où

Où le même ?

```
$userDef->setGlobal(true);
```

```
$feedReaderDef->setGlobal(false);
```

Définition des services

```
$container = new ServiceContainer(array(                                     PHP
    'storage' => new ServiceDefinition('FileSessionStorage'),
    'user'    => new ServiceDefinition('User',
        array(new ServiceReference('storage'))
    ),
));
```

```
<service id="storage" class="FileSessionStorage" />           XML
```

```
<service id="user" class="User">
    <constructor_arg type="service" id="storage" />
</service>
```

Configuration des services

```
<parameter key="storage.class">
  SQLiteSessionStorage
</parameter>
<parameter key="storage.session_name">
  SESSION_ID
</parameter>
```

La configuration
est découplée
de la définition

```
<service id="storage" class="%storage.class%">
  <constructor_arg>
    %storage.session_name%
  </constructor_arg>
</service>
```

Configuration des services

```
[storage]  
  class = SQLiteSessionStorage  
  session_name = SESSION_ID
```

```
<service id="storage" class="%storage.class">  
  <constructor_arg>  
    %storage.session_name%  
  </constructor_arg>  
</service>
```

```
// Access parameters
```

```
$sessionName = $container['storage.session_name'];
```

```
// Access services
```

```
$user = $container->user;
```

Implémentations en PHP

- Crafty: <http://phpcrafty.sourceforge.net/>
- Garden: <http://garden.tigris.org/>
spring
- Stubbles: <http://www.stubbles.net/wiki/Docs/IOC>
Google Guice
- symfony 2 will have its dependency injection framework
spring

Rappelez-vous que la plupart du temps, l'utilisation de l'Injecteur de Dépendances ne nécessite pas l'utilisation d'un Container

Vous pouvez commencer
l'utilisation de l'Injection de
Dépendance aujourd'hui

En l'implémentant dans vos
projets

Où en utilisant des bibliothèques
externes qui utilisent déjà ce
mécanisme sans Container

symfony
Zend Framework
ezComponents

Doctrine
Swift Mailer

...

Ajoutons un peu d'i18n

```
class User
{
    protected $storage, $i18n;

    function __construct($storage, $i18n)
    {
        $this->storage = $storage;
        $this->i18n = $i18n;
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);

        $this->i18n->setLanguage($language);
    }

    // ...
}
```

- La dépendance entre User et Storage et logique
 - On ne peut pas utiliser le User sans Storage
- Mais le lien entre User et I18n est moins évident
 - On doit pouvoir utiliser la classe User sans I18n

```
class User
{
    protected $storage, $i18n;

    function __construct($storage, $i18n = null)
    {
        $this->storage = $storage;
        $this->i18n = $i18n;
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);

        if (!is_null($this->i18n))
        {
            $this->i18n->setLanguage($language);
        }
    }
}
```

```
class User
{
    protected $storage;

    function __construct($storage)
    {
        $this->storage = $storage;
    }

    function setI18n($i18n)
    {
        $this->i18n = $i18n;
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);

        if (!is_null($this->i18n))
        {
            $this->i18n->setLanguage($language);
        }
    }
}
```

L'Observateur

- Le motif Observateur permet qu'un changement dans un objet permettent une mise à jour pour d'autres
- C'est un moyen efficace de permettre un lien entre des classes sans avoir à les modifier
- Mots-clés : hook, listener, event, subject, ...

- Utilisé par un nombre important de logiciels:
 - Plugins (Wordpress, ...)
 - Bridges between applications
 - As a master application:
 - Gallery2 (90% events / hooks for user / group create / update / delete, logout, login, ..missing: configuration changes)
 - Xaraya (100% events / hooks for user / group create / update / delete, configuration changes, logout, login, ..)
 - Wordpress (90% events / hooks for user / update / delete, rewrites, logout, login, ..)
 - Drupal (80% maybe?)
 - Joomla (100% joomla 1.5; login, logout, create, update, delete user, block, activation, system before and after start)Typo3 (50% maybe?, e.g. no unified create user event / hook)
 - As a slave application:
 - Gallery 2
 - Phorum.org

Implémentations en PHP

- PEAR_Dispatcher
 - http://pear.php.net/package/Event_Dispatcher
- symfony implementation
 - <http://svn.symfony-project.com/branches/1.1/lib/event/>
 - http://www.symfony-project.org/book/1_1/17-Extending-Symfony
 - Basé sur le Cocoa notification center
 - Simple et puissant
 - Decouplé desymfony
 - Simple à utiliser

```
$i18n = new I18n();  
$dispatcher = new sfEventDispatcher();  
$listener = array($i18n, 'listenToChangeCultureEvent');  
$dispatcher->connect('user.change_language', $listener);
```

```
$storage = new FileSessionStorage();  
$user = new User($dispatcher, $storage);
```

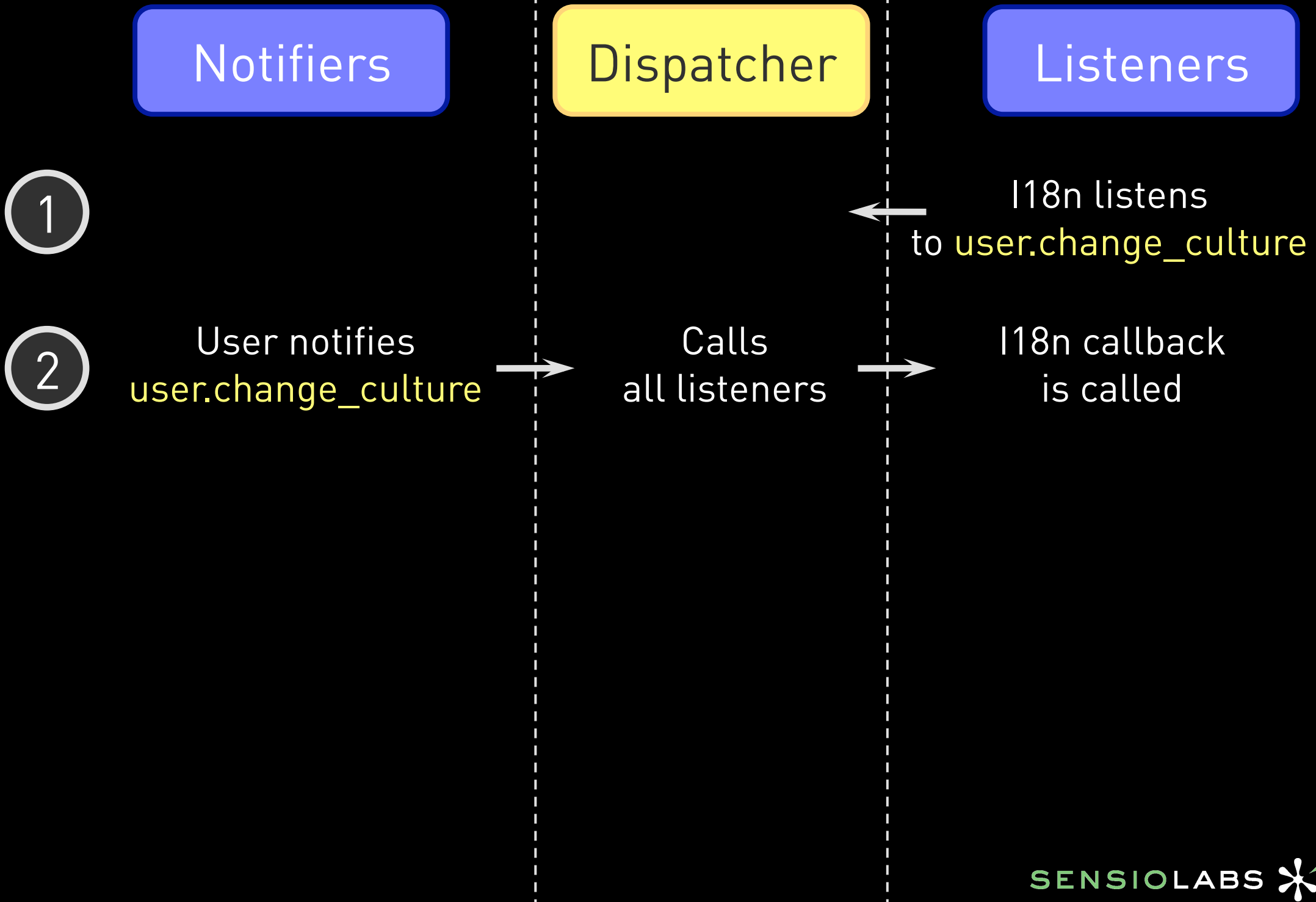
```
class User
{
    protected $storage, $dispatcher;

    function __construct($dispatcher, $storage)
    {
        $this->dispatcher = $dispatcher;
        $this->storage = $storage;
    }

    function setLanguage($language)
    {
        $this->storage->set('language', $language);

        $event = new sfEvent(
            $this,
            'user.change_language',
            array('language' => $language)
        );

        $this->dispatcher->notify($event);
    }
}
```



L'objet User ne connaît rien de l'objet I18n

L'objet I18n ne connaît rien de l'objet User

Il communique grâce à l'objet Dispatcher

N'importe quelle classe peut écouter l'événement
'user.change_culture' et agir en conséquence

Notifiers

Dispatcher

Listeners

1

118n listens
to `user.change_culture`

Your class listens
to `user.change_culture`

2

User notifies
`user.change_culture`

Calls
all listeners

118n callback
is called

Your class callback
is called

Notifiers

Dispatcher

User notifies
user.change_culture



Calls
nothing

Peu d'impact
de
performance

Dans cet exemple, l'implémentation est simple

Pas d'interface à implémenter

Pas besoin de créer une classe pour chaque événement

Un événement est « juste » :

un identifiant unique (user.change_culture)

quelques conventions (noms des paramètres)

Avantages

Simple à l'utilisation

Facile d'ajouter des paramètres supplémentaires

Très rapide

Très facile d'ajouter un nouvel événement en notifiant un nouvel identifiant

Inconvénients

Le contrat est assez lâche entre les participants à l'événement

Implémentation

```
function connect($name, $listener)
{
    if (!isset($this->listeners[$name]))
    {
        $this->listeners[$name] = array();
    }

    $this->listeners[$name][] = $listener;
}

function notify(sfEvent $event)
{
    if (!isset($this->listeners[$name]))
    {
        foreach ($this->listeners[$event->getName()] as $listener)
        {
            call_user_func($listener, $event);
        }
    }
}
```

- 3 types de notification

- Notify : tous les listeners sont appelés les uns après les autres (sans feedback possible)
 - Logging, ...
- Notify Until : tous les listeners sont appelés jusqu'à ce que l'un d'entre eux déclare avoir géré l'événement. Le listener peut alors retourner quelque chose
 - Exceptions, Method not found in `__call()`, ...
- Filter : chaque listener filtre une valeur. La valeur filtrée est alors retournée
 - HTML content, parameters, ...

Questions?

Contact

Fabien Potencier
fabien.potencier@sensio.com

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

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