RAPPORTS

Service
SG/SPSSI

Sous-service
CP2I/DO Ouest

Date 06/09/2010

Etude d'architecture du plugin « Pipeline pour autoriser »

C. Imberti – 06/09/2010

Ressources, territoires, habitats et logement
Développement durable
Énergie et climat
Infrastructures, transports et mer
Prévention des risques

Présent pour l'avenir

Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

Ministère de l'Écologie, du Développement durable, des Transports et du Logement

Ministère de l'Ecologie,
du Développement durable,
des Transports et du Logement

www-developpement-durable.gouv.fr

Historique des versions du document

Version	Date	Commentaires
1	06/09/2010	

Auteur du document

Christophe IMBERTI **-** SG/SPSSI/CP2I/DO Ouest

Sommaire

1. L'API « autoriser » de SPIP 2

1.1 Principe

Depuis sa version 1.9.2, SPIP propose toutefois une API (interface de programmation) qui centralise tous les contrôles d'autorisations.

Par exemple, dans l'espace privé, lorsque l'on sélectionne un article, SPIP va tout d'abord vérifier si on a l'autorisation de voir cet article (autoriser('voir', 'article', $id_article)).

Puis il va adapter la page en fonctions de nos autorisations :

- est-on autorisé à publier dans la rubrique de cet article (autoriser('publierdans', 'rubrique', $id_rubrique)) ?
- est-on autorisé à modifier cet article (autoriser('modifier', 'article', $id_article)) ?
- est-on autorisé à modérer le forum de cet article (autoriser('modererforum', 'article', $id_article)) ?

1.2 La fonction « autoriser »

La fonction autoriser(($faire, $type, $id, $qui, $opt) de SPIP vérifie, comme son nom l'indique, les autorisations. Le premier paramètre correspond à l'action demandée, le second au type d'objet, le troisième à l'identifiant de l'objet. Par exemple : autoriser('modifier', 'article', $id_article).

Cette fonction a pour objectif de centraliser les autorisations. En pratique, elle sert d'aiguillage, car elle va déléguer à d'autres fonctions les vérifications de chaque cas.

Reprenons notre exemple consistant à vérifier si on est autorisé à modifier un article précis : autoriser('modifier', 'article', $id_article).

La fonction « autoriser » de SPIP 2 va d'abord rechercher s'il existe, dans l'ordre une fonction nommée :

- « autoriser_article_modifier »
- « autoriser_article_modifier_dist »
- « autoriser_article »
- « autoriser_article_dist »
- « autoriser_modifier »
- « autoriser_modifier_dist »
- « autoriser_defaut »
- « autoriser_defaut_dist »

La fonction qui va être trouvée est la fonction « **autoriser_article_modifier_dist** » de SPIP 2 :

```
function autoriser_article_modifier_dist($faire, $type, $id, $qui, $opt) {
	$r	=	sql_fetsel("id_rubrique,statut",	"spip_articles",
"id_article=".sql_quote($id));

	include_spip('inc/auth'); // pour auteurs_article si espace public

	return
		autoriser('publierdans', 'rubrique', $r['id_rubrique'], $qui, $opt)
		OR (
			in_array($qui['statut'], array('0minirezo', '1comite'))
			AND in_array($r['statut'], array('prop','prepa', 'poubelle'))
			AND auteurs_article($id, "id_auteur=".$qui['id_auteur'])
		);
}
```

Un plugin peut donc ajouter une fonction « autoriser_article_modifier » qui sera prise en compte à la place de la fonction « autoriser_article_modifier_dist » de SPIP 2.

Prenons un autre exemple consistant à vérifier si on est autorisé à voir un article précis : autoriser('voir', 'article', $id_article).

La fonction « autoriser » de SPIP 2 va d'abord rechercher s'il existe, dans l'ordre une fonction nommée :

- « autoriser_article_voir »
- « autoriser_article_voir_dist »
- « autoriser_article »
- « autoriser_article_dist »
- « autoriser_voir »
- « autoriser_voir_dist »
- « autoriser_defaut »
- « autoriser_defaut_dist »

La fonction qui va être trouvée est la fonction « **autoriser_voir_dist** » de SPIP 2 :

```
function autoriser_voir_dist($faire, $type, $id, $qui, $opt) {
	if ($type == 'document')
		return autoriser_document_voir_dist($faire, $type, $id, $qui, $opt);
	if ($qui['statut'] == '0minirezo') return true;
	if ($type == 'auteur') return false;
	if ($type == 'groupemots') {
		$acces = sql_fetsel("comite,forum", "spip_groupes_mots",
"id_groupe=".intval($id));
		if ($qui['statut']=='1comite' AND ($acces['comite'] == 'oui' OR
$acces['forum'] == 'oui'))
			return true;
		if ($qui['statut']=='6forum' AND $acces['forum'] == 'oui')
			return true;
		return false;
	}
	if ($type != 'article') return true;
	if (!$id) return false;

	// un article 'prepa' ou 'poubelle' dont on n'est pas auteur : interdit
	$r = sql_getfetsel("statut", "spip_articles", "id_article=".sql_quote($id));
	include_spip('inc/auth'); // pour auteurs_article si espace public
	return
		in_array($r, array('prop', 'publie'))
		OR auteurs_article($id, "id_auteur=".$qui['id_auteur']);
}
```

En effet, SPIP 2 ne fournit pas de fonction « autoriser_article_voir_dist ».

Un plugin peut donc ajouter une fonction « autoriser_article_voir_**dist** » qui sera prise en compte à la place de la fonction « autoriser_voir_dist » de SPIP 2. Ou bien, il peut ajouter une fonction « autoriser_voir » qui sera prise en compte à la place de la fonction « autoriser_voir_dist » de SPIP 2.

SPIP 2 permet ainsi de surcharger des autorisations et même d'en créer de nouvelles.

1.3 Le pipeline « autoriser »

SPIP offre le pipeline « autoriser ». Toutefois son fonctionnement diffère de celui des autres pipelines. En effet, aucun paramètre n'est passé à ce pipeline et ce dernier ne renvoi rien.

En fait, il sert uniquement à permettre aux plugins (et aux squelettes) de charger leurs fonctions d'autorisations (nouvelles ou surcharges) au tout premier appel de la fonction « autoriser ».

2. Le problème rencontré

2.1 Un exemple concret

Si un plugin doit intervenir sur les autorisations de voir les rubriques, il peut charger, via le pipeline « autoriser », sa propre fonction « autoriser_rubrique_voir ».

Si un second plugin doit également intervenir, de manière différente, sur les autorisations de voir les rubriques, il peut charger, via le pipeline « autoriser », sa propre fonction « autoriser_rubrique_voir ».

Toutefois, la fonction « autoriser_rubrique_voir » figurera dans les deux plugins et PHP considère comme erreur fatale le fait de déclarer deux fois la même fonction.

On peut vérifier l'existence de la fonction préalablement, mais dans ce cas une seule fonction « autoriser_rubrique_voir » pourra être utilisée et l'un des plugin restera sans effet sur les autorisations de voir les rubriques.

2.2 Le fond du problème

Le fond du problème est que le pipeline « autoriser » n'offre pas les mêmes possibilités que les autres pipelines.

3. Architecture d'une solution

3.1 Principe

Le principe est de disposer d'un véritable pipeline à qui l'on passe un tableau des autorisations et qui retourne ce tableau des autorisations complété.

Au final, il s'agit de calculer le résultat de l'ensemble des autorisations contenues dans le tableau.

L'idée est d'avoir l'équivalent d'une combinaison de clauses « AND » et « OR » et de tenir compte des fonctions dist.

3.2 Surcharger une seule fois les autorisations

Pour éviter le problème rencontré, il convient de surcharger une seule fois une fonction d'autorisation.

Dans un premier temps, on se limitera aux fonctions suivantes (il sera possible d'étendre la liste) :

- autoriser_rubrique_publierdans
- autoriser_rubrique_voir
- autoriser_article_voir
- autoriser_breve_voir
- autoriser_site_voir
- autoriser_rubrique_modifier
- autoriser_article_modifier
- autoriser_breve_modifier
- autoriser_site_modifier

Chacune de ces fonctions fait appel à une fonction de centralisation nommée « ciautoriser_pipeline ».

Exemple :

```
function autoriser_rubrique_voir($faire, $type, $id, $qui, $opt) {
	return ciautoriser_pipeline($faire, $type, $id, $qui, $opt);
}
```

3.3 Offrir un véritable pipeline

La fonction de centralisation nommée « ciautoriser_pipeline », fait appel au pipeline « ciautoriser ».

Dans les paramètres passés à ce pipeline, on ajoute un tableau des autorisations.

Ensuite, la fonction fait appel au calcul du résultat de la combinaison de clauses.

```
function ciautoriser_pipeline($faire, $type, $id, $qui, $opt) {
   $param = array('faire'=>$faire,'type'=>$type,'id'=>$id,'qui'=>$qui,'opt'=>$opt);

   // Ajout d'un tableau des autorisations
   $param['autorisations'][] = array();

   // Appel du pipeline
   $param = pipeline('ciautoriser', $param);

   // Appel au calcul du résultat de la combinaison de clauses
   return ciautoriser_ciresultat($param);
}
```

Exemple d'utilisation du pipeline « ciautoriser » par le plugin redacteur restreint (extrait) :

```
function cirr_ciautoriser($param) {
	$faire = $param['faire'];
	$type = $param['type'];
	$id = $param['id'];
	$qui = $param['qui'];
	$opt = $param['opt'];
	$cifonction = $type.'_'.$faire;



	// Autoriser a voir la rubrique
	if ($cifonction=='rubrique_voir') {
		// avec l'operateur 'AND' mettre true par defaut
		$autoriser = true;

		if (!$id) {
			// mettre imperativement true
			// sinon on n'a plus le bouton pour creer une rubrique, etc.
			$autoriser = true;
		} else {
			// les redacteurs restreints (et admin restreints)
			// ne doivent voir que leurs rubriques
			if (in_array($qui['statut'], array('0minirezo', '1comite'))
				AND ($qui['restreint'] AND $id
					AND !in_array($id, $qui['restreint'])))
					  $autoriser = false;
		}

		// utilisation l'operateur 'AND' pour retrecir ce droit
		$param['autorisations'][] = array('autoriser' => $autoriser,
						'operateur' => 'AND');
	}

	return $param;
}
```

Le paramètre 'autoriser' vaut true ou false et l'opérateur est 'OR' ou 'AND'

3.4 Tenir compte des fonctions dist de SPIP

Pour éviter de devoir reproduire le comportement des fonctions d'autorisation « dist » de SPIP, il convient de prendre en compte ces dernières dans le tableau des autorisations. L'opérateur ne sera ni AND, ni OR, mais « dist ».

La version complète de la fonction de centralisation nommée « ciautoriser_pipeline » est alors la suivante :

```
function ciautoriser_pipeline($faire, $type, $id, $qui, $opt) {
   $param = array('faire'=>$faire,'type'=>$type,'id'=>$id,'qui'=>$qui,'opt'=>$opt);

       // Chercher une fonction d'autorisation "dist"
       // Dans l'ordre on va chercher
        // autoriser_type_faire_dist, autoriser_type_dist,
       // autoriser_faire_dist, autoriser_defaut_dist
       $fonctions = $type
              ? array (
                     'autoriser_'.$type.'_'.$faire.'_dist',
                     'autoriser_'.$type.'_dist',
                     'autoriser_'.$faire.'_dist',
                     'autoriser_defaut_dist'
              )
              : array (
                     'autoriser_'.$faire.'_dist',
                     'autoriser_defaut_dist'
              );

       foreach ($fonctions as $f) {
              if (function_exists($f)) {
                     $autoriser = $f($faire,$type,$id,$qui,$opt);
                     $param['autorisations'][] = array('autoriser' => $autoriser,
                                                  'operateur' => 'dist');
                     break;
              }
       }

       // Appel du pipeline
       $param = pipeline('ciautoriser', $param);

      // Appel au calcul du résultat de la combinaison de clauses
       return ciautoriser_ciresultat($param);
}
```

3.5 Calculer le résultat de la combinaison de clauses

L'idée est d'avoir l'équivalent d'une combinaison de clauses « AND » et « OR » et de tenir compte des fonctions dist.

Aussi le calcul du résultat d'un ensemble d'autorisations est effectué de manière à obtenir la combinaison de clauses suivantes : (dist OR cumul_des_OR) AND cumul_des_AND

Les possibilités sont moindre que lorsque l'on maîtrise l'ordre de l'écriture d'une combinaison de clauses, mais en exploitant judicieusement le tableau on peut répondre au problème

Bien évidemment, si un plugin intervient sur les droits d'accès aux rubriques, sans utiliser ce nouveau pipeline, le risque de conflit demeure

```
function ciautoriser_ciresultat($param) {
	$dist = true;
	$cumul_des_AND = true;
	$cumul_des_OR = false;

	if (isset($param['autorisations'])) {
		if (is_array($param['autorisations'])) {
			foreach($param['autorisations'] as $key=>$val){
				if ($val['operateur']=='dist') {
					if ($dist)
						$dist = $val['autoriser'];
				} elseif ($val['operateur']=='AND') {
					if ($cumul_des_AND)
						$cumul_des_AND = $val['autoriser'];
				} elseif ($val['operateur']=='OR') {
					if (!$cumul_des_OR)
						$cumul_des_OR = $val['autoriser'];
				}
			}
		}
	}

	return ($dist OR $cumul_des_OR) AND $cumul_des_AND;
}
```

3.6 Illustration

A titre d'illustration, le plugin "cirv : rédacteur valideur", qui offre la possibilité d'autoriser certains rédacteurs à publier leurs propres articles, et le plugin « cirv : rédacteur restreint », qui étend aux rédacteurs la notion d'administrateurs restreints, utilisent tous les deux le plugin 'ciautoriser'.

Ils sont téléchargeables à l'adresse suivante (ils incluent chacun le plugin ciautoriser) :
http://www.spip-contrib.net/cirv-plugin-redacteur-valideur
http://www.spip-contrib.net/cirr-plugin-redacteur-restreint