Les clés principales sont détaillées dans le tableau suivant :

Clé	Description
name	Le nom du module.
description	La description du module.
package	Le package dans lequel se trouve le module. Le package permet de regrouper des modules ayant le même objectif, sur la page **Modules**.
version	La version du module.
core	La version de Drupal avec laquelle le module est compatible.
datestamp	La date de dernière modification du module sous forme de timestamp.
dependencies	Les dépendances nécessaires à l'exécution du module.
files	La liste des fichiers composant le module.
configure	Le chemin d'accès à la page de configuration du module.

Exemple de fichier exemple.info du module "exemple" :

```
name = Exemple de module
description = Ceci est un module d'exemple pour le livre Drupal 7
package = Addvista
version = 1.0
dependencies[] = field
dependencies[] = node
core = 7.x
```

1.2 Le fichier .module

Le fichier *.module* est le cœur du module : c'est le fichier qui contient tout le code PHP du module et qui contient toutes les opérations à réaliser. Bien que son extension ne l'exprime pas, il s'agit d'un fichier PHP.

© Editions ENI - All rights reserved

Le fichier *.module* contient un ensemble de fonctions utilitaires permettant d'agir sur le système. On appelle cela des *hooks* (crochets en français). Le système de hooks est décrit plus loin dans ce chapitre.

Chaque module doit comporter un fichier *.module*, dont le nom doit être rigoureusement identique au nom du module concerné, de la même façon que le fichier *.info*.

Par exemple, un module nommé *exemple* doit être composé d'un fichier nommé *exemple.module*.

Exemple de fichier .module :

```
<?php

function exemple_block_info(){

  // CONTENU DE LA FONCTION exemple_block_info()

}//exemple_block_info()

function exemple_search_info() {

  // CONTENU DE LA FONCTION exemple_search_info()

}//exemple_search_info()

function exemple_menu() {

  // CONTENU DE LA FONCTION exemple_menu()

}//exemple_menu()
```

Remarque

Toutes les fonctions des modules activés étant chargées au moment de l'affichage d'une page, il est possible qu'il y ait des conflits entre les noms des fonctions utilisées. Pour parer à cela, il est d'usage de faire précéder les noms des fonctions par le nom du module dans lequel elles se trouvent.

1.3 Le fichier .install

Le fichier d'extension *.install* est un fichier PHP qui permet de définir les actions à réaliser à l'installation du module. Ce fichier doit respecter la norme de nommage de Drupal, à savoir que son nom doit correspondre au nom du module avec l'extension *.install*. En règle générale, ce fichier contient a minima trois fonctions :

– Le hook `hook_install()`

– Le hook `hook_uninstall()`

– Le hook `hook_schema()`

Ces fonctions permettent entre autres de générer un schéma de base supplémentaire (en clair, de nouvelles tables) spécifique au module, de créer de nouvelles variables Drupal, etc.

Par exemple, pour un module nommé *exemple,* le fichier *exemple.install* pourrait ressembler au code suivant :

```
<?php

function exemple_install() {

  // CODE A EXECUTER A L'INSTALLATION DU MODULE.

}//exemple_install()

function exemple_uninstall() {

  // CODE A EXECUTER A LA DESINSTALLATION DU MODULE.

}//exemple_uninstall()

function exemple_schema() {

  // CODE PERMETTANT DE DEFINIR LA STRUCTURE DES TABLES A CREER.

}//exemple_schema()
```

© Editions ENI - All rights reserved

2. Les hooks

2.1 Le principe de surcharge

De façon générale, on parle du principe de surcharge pour décrire le fait de modifier le comportement d'une fonctionnalité (souvent modélisée par une fonction) sans en modifier son code source d'origine.

On parle par exemple de surcharge dans les langages orientés objet dans le cas de méthodes polymorphes (qui peuvent prendre plusieurs formes). Ce principe s'applique dans beaucoup de CMS, notamment Drupal.

L'utilisation de ce procédé sur la plateforme Drupal est née du besoin de pouvoir rajouter un ensemble d'instructions au moment de l'exécution de certaines fonctionnalités intégrées au moteur (enregistrement ou affichage d'un nœud, affichage de blocs, etc.) sans modifier le code interne du CMS. En effet, ce type d'amélioration empêcherait toute montée en version du système sans perte des modifications apportées. L'utilisation de ce procédé sur la plateforme Drupal passe par l'utilisation de *hook*.

Un hook est une fonction dont la notation spécifique prend la forme `nomdumoduleoudutheme_nomduhook()`, et qui sera appelée automatiquement par Drupal au moment opportun.

Par exemple, pour effectuer un ensemble d'instructions à la création d'un nœud, il suffit de créer un module nommé *test* et d'écrire dans le fichier *test.module* la fonction `test_node_insert()`, le hook `hook_node_insert()` étant celui permettant de gérer les opérations sur les nœuds au moment de leur création. Toutes les instructions placées à l'intérieur de cette fonction seront par conséquent exécutées.

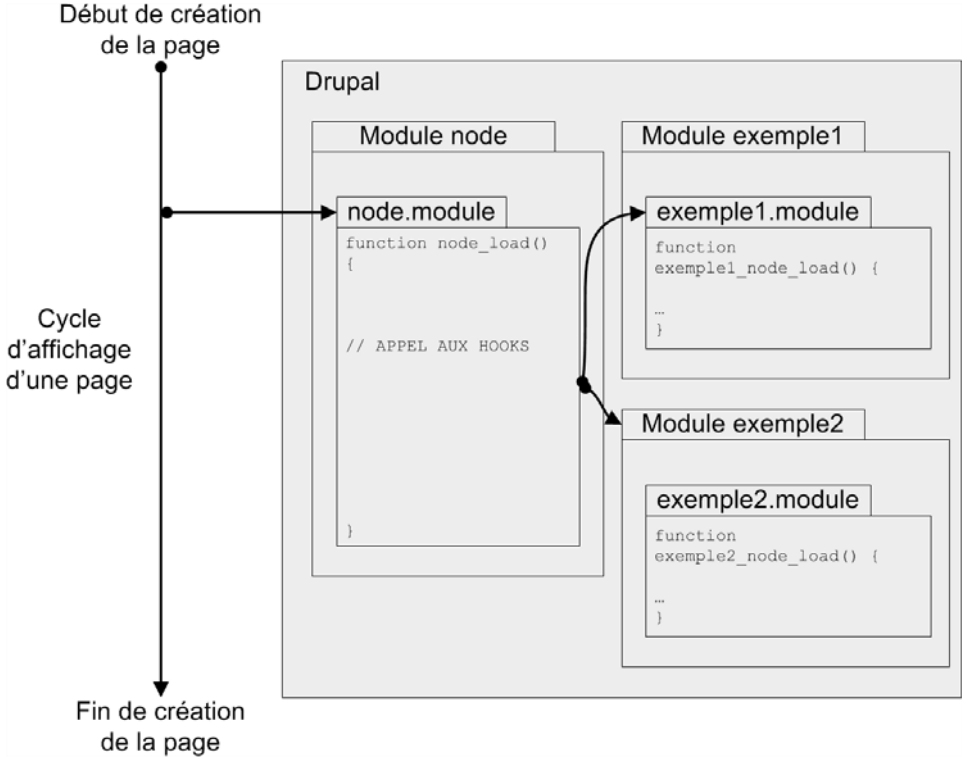
2.2 Le fonctionnement

En réalité, à chaque hook exécuté sur la plateforme, Drupal exécute ce même hook sur tous les modules installés et activés.

Au cours de la génération d'une page, le hook du module standard de Drupal est exécuté et fait appel à son tour à tous les hooks, ici le `hook_node_load()`, sur l'ensemble des modules activés.

Par exemple, au cours du cycle de création d'une page, au moment où le hook `hook_node_load()` est appelé pour rendre l'affichage d'un nœud, Drupal appelle automatiquement tous les hooks `hook_node_load()` des autres modules actifs.

Cet exemple est illustré par le schéma suivant :



© Editions ENI - All rights reserved

2.3 L'extension du système de hook

Il est possible lors du développement d'un module de le rendre évolutif en proposant de nouveaux hooks qui seront appelés à l'exécution des fonctionnalités de ce module.

Drupal utilise la fonction suivante au sein de ses modules pour parvenir à cela :

```
function module_invoke_all('nom_du_hook', 'suite', 'des',
'paramètres');
```

En effet, Drupal et ses modules standard, à certains moments clés (enregistrement d'un nœud, visualisation d'un bloc, etc.), exécutent la fonction `module_invoke_all()` en lui passant comme premier paramètre le nom du hook à exécuter. Tous les autres paramètres seront automatiquement passés au hook appelé.

Il est possible d'utiliser cette fonction au sein d'un module pour créer de nouveaux hooks, spécifiques au module courant.

Par exemple, nous considérons un module permettant de gérer un ensemble de citations. Une citation peut être exportée dans un format échangeable tel que le format CSV. La fonction suivante est un exemple de fonction du module *exemple* permettant l'implémentation du système de hooks :

```
function exemple_export($node) {

  // INSTRUCTIONS D'EXPORT AU FORMAT CSV

  ..

  module_invoke_all('export', $node);

}//exemple_export()
```

Ici, la fonction demande à Drupal de parcourir tous les modules activés et d'exécuter une fonction portant le nom du module suivi du mot-clé export. À chacune de ces fonctions sera passé un paramètre correspondant au nœud de la citation.