La notion de hook

01 Définition de la notion de hook.
Le système de modules de Drupal est basé sur le concept de «hook» (qui signifie crochet en anglais).
Un hook est une fonction PHP qui est nommée MODULE_HOOK(), où «MODULE» est le nom du module auquel elle appartient et «HOOK» est le nom d'un hook.

Pour étendre Drupal, un module doit implémenter des hooks.

Lorsqu'un certain **événement** correspondant à un **hook** se présente, Drupal recherche les modules activés qui implémentent ce hook et appelle à chaque fois la **fonction** correspondante.

Exemple :
Evénement : Drupal veut (re)mettre en cache toutes les URLs de toutes les pages du site.
Action : Drupal appelle toutes les fonctions dont le nom est du type **MODULE_menu** dans tous les modules activés. Il récupère ainsi les chemins de toutes les pages définies dans les modules avec à chaque fois le nom de la fonction qui la génère. Ces chemins et les informations associées sont ainsi enregistrés dans la table menu_router et deviennent utilisable dans le site Drupal.

Liste de hooks :
Les hooks permettant de modifier le fonctionnement de Drupal sont listés sur la page :
http://api.drupal.org/api/drupal/includes--module.inc/group/hooks/7

Expérimentation pédagogique :
Nous allons réaliser un module **module3** qui montre comment utiliser certains des hooks les plus communs : init, permission, menu et cron.

Ce module gère un block (construction puis affichage) qui montre les dernières pages visitées par l'utilisateur.

Le nombre de pages affichées est variable et peut être configuré en cliquant dans le haut du block.
Dans configuration>module3 nous pourrons voir le maximum d'items qui peuvent être affichés dans un tel bloc.

Fichiers à utiliser :
Dans votre site, rajoutez puis activez le module module3 dont la première version contient les fichiers suivants :

module3.info

```
name = module3
description = Mémorise une liste des pages visitées précédemment.
package = Cours Drupal DF
core = 7.x
configure = admin/config/module3
files[] = module3.module
files[] = module3.admin.inc
dependencies[] = block
```

module3.module (première version)

```
<?php
/**
 * @file
 * Enregistre et affiche la liste des pages visitées précédemment
 */
/**
 * Implements hook_init().
 */
function module3_init() {
  return;
}

/**
 * Implements hook_permission().
 */
function module3_permission() {
  return;
}

/**
 * Implements hook_menu().
 */
function module3_menu() {
  return;
}

/**
 * Implements hook_cron().
 */
function module3_cron() {
  return;
}
```

02 Invocation d'un hook avec module_invoke_all

Pour comprendre le fonctionnement de Drupal nous allons utiliser EclipsePHP et le debugger Xdebug.

Votre interpréteur PHP doit être correctement configuré.
L'option (Xdebug) : Remote debug doit être activée :
- Wampserver>PHP>Configuration PHP, (Xdebug) : Remote debug coché,
- ou xdebug.remote_enable = On dans le fichier php.ini)

- Lancez Eclipse. Le workspace choisi doit être accessible localement en écriture.
- Créez un nouveau projet Php. Sélectionnez l'option « Create project at existing location ».
  Cliquez sur browse pour rechercher le répertoire de votre site Drupal. Le chemin de ce répertoire apparaitra dans la zone "directory". Donnez au projet le même nom que celui du répertoire.
- Vérifiez que Eclipse est configuré pour lancer les projets dans un navigateur externe
  Choisissez **Fenêtre** de la barre de menu, puis sélectionnez **Préférences**. Développez la sous-arborescence **Général**, puis cliquez sur N**avigateur Web** . Sélectionnez le bouton radio **Utiliser un navigateur externe** et cliquez sur **Appliquer**.
- Eclipse doit être configuré afin de reconnaître les fichiers .module comme des fichiers PHP.
  Window>Preferences>Général>Content Types>Text>Php Content Type : Add.. *.module.
  Pour les Php Content Type mettre aussi Default encoding à UTF-8 et cliquez sur Update.

Configuration et lancement du débuggage :
Cliquez sur la flèche à droite du debug button  et sélectionnez Debug Configurations...
Double-cliquez sur PHP Web Page pour créer une nouvelle configuration que vous nommerez Drupal.

- Dans l'onglet « Server », champ File, cliquez sur browse pour sélectionner le fichier index.php de votre site Drupal (index.php est le seul fichier directement exécutable de votre site Drupal. C'est lui qui charge tous les autres fichiers).
- Dans l'onglet « Debugger » choisissez Xdebug au lieu de Zend Debugger.
- Dans l'onglet « Common », encoding other UTF-8.

Le module3 doit être **activé** pour que la suite fonctionne.
Ouvrez le fichier module3.module.
Mettez un point d'arrêt dans la fonction module3_init(). Pour cela, double-cliquez dans la marge à gauche du numéro de ligne, en face de l'instruction return. Un point bleu apparaît.

Lancez le debuggage et mettez vous dans la perspective Php DEBUG. Normalement Eclipse vous le propose.
Xdebug s'arrête une première fois à la ligne 17 du fichier index.php.
Cliquez sur le bouton « Continuer ». Le programme s'arrête une seconde fois dans module3_init.

Etude de la pile des appels (Call Stack) représentée ici de bas en haut.

- index.php, ligne 21
- dans bootstrap.inc, ligne 1980, drupal_boostrap
- dans common.inc, drupal_bootstrap_full, ligne 4844, module_invoke_all('init');
- dans module.inc, ligne 768, module_invoke_all()
  boucle sur le résultat de module_implements($hook)
  et appel à chaque fois de de call_user_func_array()
- dans module.inc, ligne 0, call_user_func_array

En cliquant sur une ligne on peut aller dans le code.
Exercice : Cliquer successivement sur les premières lignes et expliquer.

Etude du code de la fonction module_invoke_all

```
function module_invoke_all() {
  $args = func_get_args();
  $hook = $args[0];
  unset($args[0]);
  $return = array();
  foreach (module_implements($hook) as $module) {
    $function = $module . '_' . $hook;
    if (function_exists($function)) {
      $result = call_user_func_array($function, $args);
      if (isset($result) && is_array($result)) {
        $return = array_merge_recursive($return, $result);
      }
      elseif (isset($result)) {
        $return[] = $result;
      }
    }
  }
  return $return;
}
```

Cette fonction nécessite une implémentation performante de la fonction module_implements() qui renvoie la liste de tous les modules implémentant le hook passé en paramètre.
La boucle permet d'appeler l'une après l'autre toutes les fonctions correspondant à ce hook.
Pour chaque exécution d'une fonction_hook, le résultat est récupéré dans le tableau $result puis rajouté dans le tableau $return.

Au final, $return contient la fusion de tous les résultats de toutes les fonctions_hook.

03 Invocation d'un hook avec module_invoke

Mettez un point d'arrêt sur module3_permission()
Dans votre navigateur, entrez l'URL : localhost/*votre_site_Drupal*/admin/people/permissions

Etude du Call Stack (Pile des appels)
dans user.admin.inc, ligne 656, définition de user_admin_permissions(), fonction qui crée un formulaire, après avoir récupéré les noms des modules renvoyés par module_implements('permission'), boucle sur les noms et, ligne 685, appelle module_invoke($module, 'permission')
- dans module.inc, ligne 790, définition de la fonction module_invoke()

Code de la fonction module_invoke() :

```
function module_invoke() {
  $args = func_get_args();
  $module = $args[0];
  $hook = $args[1];
  unset($args[0], $args[1]);
  if (module_hook($module, $hook)) {
    return call_user_func_array($module . '_' . $hook, $args);
  }
}
```

Exercice : Pourquoi ne pas utiliser module_invoke_all ? (1 : rapport avec $args, 2 : boucle englobante)

Mettez un point d'arrêt dans module3_cron()
Lancez admin/reports/status report, puis cliquez sur "run cron manually"

Etude du Call Stack :
dans common.inc, ligne 5083, définition de la fonction drupal_cron_run(),
ligne 5121, module_invoke($module, 'cron');

04 Invocation d'un hook par des fonctions personnalisées

Mettez un point d'arret dans module3_menu. Dans l'interface d'administration lancez : configuration/development/performance et cliquer sur "Clear all caches"
Pour garantir les performances, drupal doit exécuter tous les hook menu pour les enregistrer dans la BD.

Etude du Call Stack
- dans menu.inc, ligne 2724, définition de menu_router_build(), boucle sur les modules implémentant 'menu' dans laquelle, ligne 2729, on exécute : $router_items = call_user_func($module . '_menu');

05 Résumé de l'implémentation des hooks
Dans votre rapport, indiquez les trois patterns qui peuvent être utilisés pour invoquer les hooks

06 Executer du code sur chaque page en utilisant hook_init

Dans le fichier module3.module, remplacer le code de la fonction module3_init par :

```
function module3_init() {
  $trail = variable_get('module3_history', array());
  // Grab the trail history from a variable

  $trail[] = array(                          // Add current page to trail.
    'title' => strip_tags(drupal_get_title()),
    'path' => $_GET['q'],
    'timestamp' => REQUEST_TIME,
  );

  variable_set('module3_history', $trail);   // Save the trail as a variable
}
```

1) Expliquez ce que fait cette fonction (3 actions principales)

2) Sachant que tous les hook_init sont exécutés avant le traitement de toutes les pages, expliquez quel est le contenu de la variable $trail à la fin de cette fonction.

07 Visualisation des changements dans la BD.

Le Module3 étant activé, visitez 2 pages de l'interface d'administration.

Examinez ensuite le contenu de la table variable en utilisant PhpMyAdmin. Elle comporte 2 colonnes: name et value.

Le champ value de l'enregistrement de name module3_history contient un tableau sérialisé.
Expliquez cette valeur (si le blob est masqué, cliquez sur le lien + options et cochez la case "Montrer le contenu BLOB").

08 Implémentation du hook permission

Dans le fichier module3.module, remplacer le code de la fonction module3_ permission par :

```
function module3_permission() {
  return array(
    'administer trails' => array(
      'title' => t('Administrer le module module3'),
      'description' => t('Effectue les tâches d\'administration du module3.'),
    ),
    'access trails blocks' => array(
      'title' => t('Accès aux blocs du module3'),
      'description' => t('Affiche les blocs générés par module3.'),
    ),
  );
}
```

Ce hook permet de rajouter 2 permissions utilisateur à Drupal : 'administer trails' et 'access trails blocks'.
Test : Allez dans l'interface d'administration, modules, trouver le module module3, cliquer sur le lien Droits (Permissions) pour afficher le tableau des permissions. Expliquez ce qui se passe.

09 Ajouter une page de configuration spécifique pour le module

Dans le fichier module3.module, remplacer le code de la fonction module3_ menu par :

```
function module3_menu() {
  // Module settings.
  $items['admin/config/module3'] = array(
    'title' => 'Cheminement dans le site',
    'description' => 'Configuration historique des pages',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('module3_admin_settings'),
    'access arguments' => array('administer trails'),
    'file' => 'module3.admin.inc',
    'file path' => drupal_get_path('module', 'module3'),
  );
  return $items;
}
```

Etudiez le code de cette fonction. Elle déclare un chemin vers une page de configuration spécifique qui sera accessible grâce au lien "Configurer" contenu dans le tableau des modules.
Pour créez cette page, ajoutez un fichier PHP intitulé module3.admin.inc contenant :

```
function module3_admin_settings() {
  $form['module3_block_max'] = array(
    '#type' => 'select',
    '#title' => t('Nombre Maximum d\'items à afficher'),
    '#options' => drupal_map_assoc(range(1, 200)),
    '#default_value' => variable_get('module3_block_max', ''),
    '#description' => t('Définit le nb max. d\'items dans un historique'),
    '#required' => TRUE,
  );
  return system_settings_form($form);
}
```

Remarque :
L'étiquette de l'item de menu et le nom de la variable doivent être identiques (ici les deux ont pour valeur 'module3_block_max'). N'oubliez pas la balise <?php au début du fichier !

Exercice :
Expliquez ce code
Dans Drupal effacez le cache (Clear all caches)
Dans le tableau des modules de l'interface d'administration, cliquez sur "Configurer" sur la ligne module3
Expliquez comment à été générée la page sur laquelle vous êtes.

10 Programmer des actions régulières avec le hook cron.

Nous voulons réduire régulièrement la taille du tableau module3_history à 5 éléments. Pour cela nous devons régulièrement récupérer ce tableau dans la BD, le réduire puis le ré-enregistrer.
Dans le fichier module3.module, remplacer le code de la fonction module3_ cron par :

```
function module3_cron() {
  $trail = variable_get('module3_history', array());
  $count_minus_5 = count($trail) - 5;
  $short_trail = array_slice($trail, $count_minus_5);
  variable_set('module3_history', $short_trail);
}
```

Expliquez ligne par ligne ce que fait cette fonction.
Sachant que les hook_cron sont régulièrement déclenchés par Drupal, expliquez pourquoi nous avons atteint notre objectif.

11 Créer un nouveau block grâce à notre module.

Si, dans notre module, nous voulons créer un nouveau bloc, nous devons d'abord le déclarer au moyen du hook_block_info().
Cette fonction renvoie un tableau dont les éléments sont les infos des blocs créés par le module.
Pour chaque bloc déclaré, la clé est le nom du block et la valeur est un tableau imbriqué contenant les informations de configuration du block.
Dans le fichier module3.module, rajoutez le code suivant pour déclarer un bloc de nom history :

```
/**
 * Implements hook_block_info().
 */
function module3_block_info() {
  $blocks['history'] = array(
    'info' => t('History'),
    'cache' => DRUPAL_NO_CACHE,
  );
  return $blocks;
}
```

Test : Allez sur admin/structure/block. Notre bloc history apparaît dans la liste des blocs désactivés.
Dans cette page nous pouvons positionner notre block dans une région du thème (exemple : sidebar).

12 Configurer un block

Un module doit fournir un formulaire de configuration de ses blocs au moyen du **hook_block_configure**.
Un module doit également fournir une fonction de sauvegarde des configurations au moyen du **hook_block_save**.

Dans le fichier module3.module ajoutez le code suivant :

```
/**
 * Implements hook_block_configure().
 */
function module3_block_configure($delta = '') {
  // Get the maximum allowed value from the configuration form.
  $max_to_display = variable_get('module3_block_max', 50);

  // Add a select box of numbers from 1 to $max_to_display.
  $form['module3_block_num'] = array(
    '#type' => 'select',
    '#title' => t('Nombre d\'items à montrer'),
    '#default_value' => variable_get('module3_block_num', '5'),
    '#options' => drupal_map_assoc(range(1, $max_to_display)),
  );
  return $form;
}

/**
 * Implements hook_block_save().
 */
function module3_block_save($delta = '', $edit = array()) {
  variable_set('module3_block_num', $edit['module3_block_num']);
}
```

Remarque : Ici la variable $delta ne sert pas car le module ne génère qu'un seul bloc. Dans le cas contraire il faut tester lavaleur de$delta dans une instruction if ou switch ... case.

Les hooks hook_block_configure et hook_block_save sont activés lorsque, dans la page admin/structure/block l'utilisateur clique, à droite d'un block créé par notre module, sur le lien configure dans la colonne OPERATIONS.

Le formulaire de configuration qui apparaît alors (contenant par défaut un item Block title et Region Settings) est complété par le hook_block_configure.
L'enregistrement de cette configuration est réalisé par le hook_block_save.

13 Afficher un block

Un module doit aussi fournir le code HTML de ses blocs.
Quand un block doit être affiché, c'est le **hook_block_view** du module contenant la construction du code HTML du block, qui est appelé.
Dans le fichier module3.module ajoutez le code suivant puis testez le :

```
/**
 * Implements hook_block_view().
 */
function module3_block_view($delta = '') {
  if (user_access('access trails blocks')) { //test permission
    $block = array();
    switch ($delta) {
      case 'history': // Create list of previous paths.
        $trail = variable_get('module3_history', array());
        $reverse_trail = array_reverse($trail); //newest pages first.
        $num_items = variable_get('module3_block_num', '5');

        // Output the latest items as a list
        $output = '';
        for ($i = 0; $i < $num_items; $i++) {
          if ($item = $reverse_trail[$i]) {
            $output .= '<li>' . l($item['title'], $item['path']) . ' - '
                    . format_interval(REQUEST_TIME - $item['timestamp'])
                    . ' ' . t('ago') . '</li>';
          }
        }
        if ($output != '') {
          $output = '<p>'
            . t('Ci-dessous les !num dernières pages que vous avez visitées.',
                array('!num' => $num_items))
            . '</p> <ul>' . $output . '</ul>';
        }

        // Prepare the $block variable, subject=title, content=output.
        $block['subject'] = 'History';
        $block['content'] = $output;
        break;
    }
    return $block;
  }
}
```

Remarques :
- La fonction l() génère le code html d'un lien.
- La fonction t() traduit la chaîne de caractère donnée en premier argument. Cette chaîne peut contenir des « placeholder » dont les valeurs sont dans le tableau donné en second argument.

Exercices :
1) Expliquez précisément ce que fait cette fonction. Vous pouvez mettre un point d'arrêt sur la première ligne de la fonction et utiliser le debugger pour comprendre, pas à pas son exécution.
2) Le block n'affiche qu'un certain nombre de lignes mais peut-il y avoir plus d'items dans la variable module3_history ? Si oui jusqu'à quand ? Quand a lieu le lancement du cron sur votre site ?
3) Rappelez les 4 hooks permettant à un module de créer et de gérer des blocs en précisant le rôle de chaque hook.