5 Modules chargeables

1 Aperçu

Les modules chargeables offrent une option orientée vers la performance pour étendre les fonctionnalités de Zabbix.

Il existe déjà des moyens d'étendre les fonctionnalités de Zabbix par :

- Les paramètres utilisateurs (métriques de l'agent)
- Les vérifications externes (supervision sans agent)
- L'élément de l'agent Zabbix : `system.run[]`.

Ils fonctionnent très bien, mais ont un inconvénient majeur, à savoir fork(). Zabbix doit forker un nouveau processus chaque fois qu'il gère une métrique utilisateur, ce qui n'est pas bon pour les performances. Ce n'est pas un gros problème normalement, cependant cela pourrait être un problème sérieux lors de la supervision de systèmes embarqués, ayant un grand nombre de paramètres surveillés ou de gros scripts avec une logique complexe ou un long temps de démarrage.

Le support des modules chargeables permet d'étendre l'agent Zabbix, le serveur et le proxy sans sacrifier les performances.

Un module chargeable est essentiellement une bibliothèque partagée utilisée par le démon Zabbix et chargée au démarrage. La bibliothèque doit contenir certaines fonctions, de sorte qu'un processus Zabbix puisse détecter que le fichier est bien un module qu'il peut charger et avec lequel il peut travailler.

Les modules chargeables ont un certain nombre d'avantages. De grandes performances et la capacité à implémenter n'importe quelle logique sont très importantes, mais peut-être l'avantage le plus important est la capacité à développer, utiliser et partager des modules Zabbix. Il contribue à une maintenance facilité et aide à fournir de nouvelles fonctionnalités plus facilement et indépendamment du code de base de Zabbix.

La licence et la distribution des modules sous forme binaire sont régies par la licence GPL (les modules sont en liaison avec Zabbix en exécution et utilisent des en-têtes Zabbix, actuellement l'intégralité du code Zabbix est sous licence GPL). La compatibilité binaire n'est pas garantie par Zabbix.

La stabilité de l'API est garantie au cours d'un cycle d'une release LTS (Long Term Support) de Zabbix. La stabilité de l'API Zabbix n'est pas garantie (techniquement, il est possible d'appeler les fonctions internes de Zabbix à partir d'un module, mais rien ne garanti que de tels modules fonctionneront).

2 Module d'API

Pour qu'une bibliothèque partagée soit traitée comme un module Zabbix, elle doit implémenter et exporter plusieurs fonctions. Il y a actuellement six fonctions dans l'API Zabbix, dont une seule est obligatoire et les cinq autres sont facultatives.

2.1 Interface obligatoire

La seule fonction obligatoire est **zbx_module_api_version()** :

```
int zbx_module_api_version(void);
```

Cette fonction doit retourner la version du module de l'API implémentée par ce module et pour que le module puisse charger cette version doit correspondre à la version du module d'API supporté par Zabbix. La version du module d'API supportée par Zabbix est ZBX_MODULE_API_VERSION. Donc, cette fonction doit retourner cette constante. L'ancienne constante ZBX_MODULES_API_VERSION_ONE utilisée dans ce but est maintenant définie égale à ZBX_MODULE_API_VERSION pour préserver la compatibilité des sources, mais il s'agit d'une utilisation non recommandées.

2.2 Interface optionnelle

Les fonctions optionnelles sont **zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** et **zbx_module_uninit()**:

```
int zbx_module_init(void);
```

Cette fonction doit effectuer l'initialisation nécessaire pour le module (si nécessaire). En cas de succès, il devrait retourner ZBX_MODULE_OK. Sinon, il devrait retourner ZBX_MODULE_FAIL. Dans ce dernier cas, Zabbix ne démarrera pas.

```
ZBX_METRIC  *zbx_module_item_list(void);
```

Cette fonction devrait renvoyer une liste d'éléments supportés par le module. Chaque élément est défini dans une structure ZBX_METRIC, voir la section ci-dessous pour plus de détails. La liste se termine par une structure ZBX_METRIC avec le champ “clé” de NULL.

```
void    zbx_module_item_timeout(int timeout);
```

Si le module exporte **zbx_module_item_list()** alors cette fonction est utilisée par Zabbix pour spécifier les paramètres de timeout dans le fichier de configuration Zabbix que les vérifications d'éléments implémentées par le module doivent obéir. Ici, le paramètre “timeout” est en secondes.

```
ZBX_HISTORY_WRITE_CBS   zbx_module_history_write_cbs(void);
```

Cette fonction doit renvoyer les fonctions de rappel que le serveur Zabbix utilisera pour exporter l'historique de différents types de données. Les fonctions de rappel sont fournies sous forme de

champs de structure ZBX_HISTORY_WRITE_CBS, les champs peuvent être NULL si le module n'est pas intéressé par l'historique de certain type.

```
int zbx_module_uninit(void);
```

Cette fonction doit effectuer la désinitialisation nécessaire (si nécessaire), comme la libération des ressources allouées, la fermeture des descripteurs de fichiers, etc.

Toutes les fonctions sont appelées une fois au démarrage de Zabbix lorsque le module est chargé, à l'exception de zbx_module_uninit(), qui est appelée lors de l'arrêt de Zabbix lorsque le module est déchargé.

2.3 Définir des éléments

Chaque élément est défini dans une structure ZBX_METRIC :

```
typedef struct
{
    char        *key;
    unsigned    flags;
    int        (*function)();
    char        *test_param;
}
ZBX_METRIC;
```

Ici, **key** est la clé de l'élément (par exemple, "dummy.random"), **flags** est soit CF_HAVEPARAMS soit 0 (selon le fait que l'élément accepte ou non les paramètres), **function** est une fonction C qui implémente l'élément (par exemple, "zbx_module_dummy_random"), et **test_param** est la liste des paramètres à utiliser lorsque l'agent Zabbix est démarré avec l'option "-p" (par exemple, "1,1000", peut être NULL). Un exemple de définition peut ressembler à ceci :

```
static ZBX_METRIC keys[] =
{
    { "dummy.random", CF_HAVEPARAMS, zbx_module_dummy_random, "1,1000" },
    { NULL }
}
```

Chaque fonction qui implémente un élément doit accepter deux paramètres de pointeur, le premier de type AGENT_REQUEST et le second de type AGENT_RESULT :

```
int zbx_module_dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    ...
```

```
    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}
```

Ces fonctions doivent renvoyer SYSINFO_RET_OK, si la valeur de l'élément a été obtenue avec succès. Sinon, ils doivent renvoyer SYSINFO_RET_FAIL. Voir l'exemple de module “factice” ci-dessous pour plus de détails sur la façon d'obtenir des informations à partir de AGENT_REQUEST et comment définir des informations dans AGENT_RESULT.

2.4 Fournir des rappels d'export d'historique

L'export d'historique via un module n'est plus supporté par le proxy depuis Zabbix 4.0.0.

Le module peut spécifier des fonctions pour exporter les données d'historique par type : numérique (float), numérique (non signé), caractère, texte et journal :

```
typedef struct
{
    void    (*history_float_cb)(const ZBX_HISTORY_FLOAT *history, int
history_num);
    void    (*history_integer_cb)(const ZBX_HISTORY_INTEGER *history, int
history_num);
    void    (*history_string_cb)(const ZBX_HISTORY_STRING *history, int
history_num);
    void    (*history_text_cb)(const ZBX_HISTORY_TEXT *history, int
history_num);
    void    (*history_log_cb)(const ZBX_HISTORY_LOG *history, int
history_num);
}
ZBX_HISTORY_WRITE_CBS;
```

Chacun d'eux devrait prendre le tableau “history” des éléments “history_num” comme arguments. Selon le type de données d'historique à exporter, “history” est un tableau de structures suivantes, respectivement :

```
typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    double        value;
}
ZBX_HISTORY_FLOAT;

typedef struct
```

```
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    zbx_uint64_t    value;
}
ZBX_HISTORY_INTEGER;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
}
ZBX_HISTORY_STRING;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
}
ZBX_HISTORY_TEXT;

typedef struct
{
    zbx_uint64_t    itemid;
    int        clock;
    int        ns;
    const char    *value;
    const char    *source;
    int        timestamp;
    int        logeventid;
    int        severity;
}
ZBX_HISTORY_LOG;
```

Les rappels seront utilisés par les processus de synchronisation de l'historique du serveur Zabbix à la fin de la procédure de synchronisation de l'historique après que les données aient été écrites dans la base de données Zabbix et enregistrées dans le cache des valeurs.

2.5 Construction de modules

Les modules sont actuellement destinés à être construits dans l'arborescence source de Zabbix, car le module d'API dépend de certaines structures de données définies dans les en-têtes Zabbix.

L'en-tête le plus important pour les modules chargeables est **include/module.h**, qui définit ces

structures de données. Un autre en-tête utile est **include/sysinc.h**, qui exécute l'inclusion des en-têtes système nécessaires, ce qui permet à include/module.h de fonctionner correctement.

Pour que include/module.h et include/sysinc.h soient inclus, la commande **./configure** (sans arguments) doit d'abord être exécutée à la racine de l'arborescence source de Zabbix. Cela créera le fichier **include/config.h**, lequel repose sur **include/sysinc.h**. (Si vous avez obtenu le code source de Zabbix d'un dépôt SVN, le script ./configure n'existe pas encore et la commande **./bootstrap.sh** doit d'abord être exécutée pour le générer.)

Avec cette information à l'esprit, tout est prêt pour la construction du module. Le module doit inclure **sysinc.h** et **module.h**, et le script de construction doit s'assurer que ces deux fichiers se trouvent dans le chemin d'inclusion. Voir l'exemple "module factice" ci-dessous pour plus de détails.

Un autre en-tête utile est **include/log.h**, qui définit la fonction **zabbix_log()**, qui peut être utilisée à des fins de journalisation et de débogage.

3 Paramètres de configuration

L'agent, le serveur et le proxy Zabbix supportent deux paramètres pour gérer les modules :

- LoadModulePath - chemin d'accès complet à l'emplacement des modules chargeables
- LoadModule - module(s) à charger au démarrage. Les modules doivent être situés dans un répertoire spécifié par LoadModulePath ou (depuis 4.0.9) le chemin doit précéder le nom du module. Si le chemin précédent est absolu (commençant par '/'), LoadModulePath est ignoré. Il est autorisé d'inclure plusieurs paramètres LoadModule.

Par exemple, pour étendre l'agent Zabbix, nous pourrions ajouter les paramètres suivants :

```
LoadModulePath=/usr/local/lib/zabbix/agent/
LoadModule=mariadb.so
LoadModule=apache.so
LoadModule=kernel.so
LoadModule=/usr/local/lib/zabbix/dummy.so
```

Au démarrage de l'agent, il chargera les modules mariadb.so, apache.so et kernel.so et dummy.so du répertoire /usr/local/lib/zabbix/agent tandis que dummy.so sera chargé à partir de /usr/local/lib/zabbix. Il échouera si un module est manquant, en cas de mauvaises permissions ou si une bibliothèque partagée n'est pas un module Zabbix.

4 Configuration de l'interface Web

Les modules chargeables sont supportés par l'agent Zabbix, le serveur et le proxy. Par conséquent, le type d'élément dans l'interface Zabbix dépend de l'endroit où le module est chargé. Si le module est chargé dans l'agent, le type d'élément doit être "Agent Zabbix" ou "Agent Zabbix (actif)". Si le module est chargé dans le serveur ou le proxy, le type d'élément doit être "Vérification simple".

L'exportation d'historique via les modules Zabbix n'a pas besoin de configuration frontale. Si le module est chargé correctement par le serveur et fournit la fonction

zbx_module_history_write_cbs() qui renvoie au moins une fonction de rappel non NULL, l'export de l'historique sera automatiquement activé.

5 Module factice

Zabbix inclut un exemple de module écrit en langage C. Le module se trouve sous src/modules/dummy :

```
alex@alex:~trunk/src/modules/dummy$ ls -l
-rw-rw-r-- 1 alex alex 9019 Apr 24 17:54 dummy.c
-rw-rw-r-- 1 alex alex   67 Apr 24 17:54 Makefile
-rw-rw-r-- 1 alex alex  245 Apr 24 17:54 README
```

Le module est bien documenté, il peut être utilisé comme modèle pour vos propres modules.

Après avoir exécuté la commande ./configure dans la racine de l'arborescence source de Zabbix comme décrit ci-dessus, lancez make pour construire **dummy.so**.

```
/*
** Zabbix
** Copyright (C) 2001-2016 Zabbix SIA
**
** This program is free software; you can redistribute it and/or modify
** it under the terms of the GNU General Public License as published by
** the Free Software Foundation; either version 2 of the License, or
** (at your option) any later version.
**
** This program is distributed in the hope that it will be useful,
** but WITHOUT ANY WARRANTY; without even the implied warranty of
** MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
** GNU General Public License for more details.
**
** You should have received a copy of the GNU General Public License
** along with this program; if not, write to the Free Software
** Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301, USA.
**/

#include "sysinc.h"
#include "module.h"

/* the variable keeps timeout setting for item processing */
static int  item_timeout = 0;

/* module SHOULD define internal functions as static and use a naming
pattern different from Zabbix internal */
/* symbols (zbx_*) and loadable module API functions (zbx_module_*) to avoid
conflicts                      */
static int  dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result);
```

```
static int  dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result);
static int  dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result);

static ZBX_METRIC keys[] =
/*  KEY             FLAG        FUNCTION    TEST PARAMETERS */
{
    {"dummy.ping",        0,        dummy_ping,    NULL},
    {"dummy.echo",        CF_HAVEPARAMS,    dummy_echo,    "a message"},
    {"dummy.random",    CF_HAVEPARAMS,    dummy_random,    "1,1000"},
    {NULL}
};

/******************************************************************************
***
 *
*
 * Function: zbx_module_api_version
*
 *
*
 * Purpose: returns version number of the module interface
*
 *
*
 * Return value: ZBX_MODULE_API_VERSION - version of module.h module is
*
 *               compiled with, in order to load module successfully Zabbix
*
 *               MUST be compiled with the same version of this header file
*
 *
*
******************************************************************************
**/
int zbx_module_api_version(void)
{
    return ZBX_MODULE_API_VERSION;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_item_timeout
*
 *
*
 * Purpose: set timeout value for processing of items
*
 *
```

```
*
 * Parameters: timeout - timeout in seconds, 0 - no timeout set
*
 *
*
******************************************************************************
**/
void    zbx_module_item_timeout(int timeout)
{
    item_timeout = timeout;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_item_list
*
 *
*
 * Purpose: returns list of item keys supported by the module
*
 *
*
 * Return value: list of item keys
*
 *
*
******************************************************************************
**/
ZBX_METRIC  *zbx_module_item_list(void)
{
    return keys;
}

static int  dummy_ping(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    SET_UI64_RESULT(result, 1);

    return SYSINFO_RET_OK;
}

static int  dummy_echo(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param;

    if (1 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
```

```
    }

    param = get_rparam(request, 0);

    SET_STR_RESULT(result, strdup(param));

    return SYSINFO_RET_OK;
}

/******************************************************************************
***
 *
*
 * Function: dummy_random
*
 *
*
 * Purpose: a main entry point for processing of an item
*
 *
*
 * Parameters: request - structure that contains item key and parameters
*
 *              request->key - item key without parameters
*
 *              request->nparam - number of parameters
*
 *              request->timeout - processing should not take longer than
*
 *                                 this number of seconds
*
 *              request->params[N-1] - pointers to item key parameters
*
 *
*
 *             result - structure that will contain result
*
 *
*
 * Return value: SYSINFO_RET_FAIL - function failed, item will be marked
*
 *                                  as not supported by zabbix
*
 *               SYSINFO_RET_OK - success
*
 *
*
 * Comment: get_rparam(request, N-1) can be used to get a pointer to the Nth
*
 *          parameter starting from 0 (first parameter). Make sure it exists
```

```
*
 *          by checking value of request->nparam.
*
 *
*
******************************************************************************
**/
static int  dummy_random(AGENT_REQUEST *request, AGENT_RESULT *result)
{
    char    *param1, *param2;
    int    from, to;

    if (2 != request->nparam)
    {
        /* set optional error message */
        SET_MSG_RESULT(result, strdup("Invalid number of parameters."));
        return SYSINFO_RET_FAIL;
    }

    param1 = get_rparam(request, 0);
    param2 = get_rparam(request, 1);

    /* there is no strict validation of parameters for simplicity sake */
    from = atoi(param1);
    to = atoi(param2);

    if (from > to)
    {
        SET_MSG_RESULT(result, strdup("Invalid range specified."));
        return SYSINFO_RET_FAIL;
    }

    SET_UI64_RESULT(result, from + rand() % (to - from + 1));

    return SYSINFO_RET_OK;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_init
*
 *
*
 * Purpose: the function is called on agent startup
*
 *          It should be used to call any initialization routines
*
 *
*
```

```
 * Return value: ZBX_MODULE_OK - success
*
 *               ZBX_MODULE_FAIL - module initialization failed
*
 *
*
 * Comment: the module won't be loaded in case of ZBX_MODULE_FAIL
*
 *
*
******************************************************************************
**/
int zbx_module_init(void)
{
    /* initialization for dummy.random */
    srand(time(NULL));

    return ZBX_MODULE_OK;
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_uninit
*
 *
*
 * Purpose: the function is called on agent shutdown
*
 *          It should be used to cleanup used resources if there are any
*
 *
*
 * Return value: ZBX_MODULE_OK - success
*
 *               ZBX_MODULE_FAIL - function failed
*
 *
*
******************************************************************************
**/
int zbx_module_uninit(void)
{
    return ZBX_MODULE_OK;
}

/******************************************************************************
***
 *
```

```
*
 * Functions: dummy_history_float_cb
*
 *            dummy_history_integer_cb
*
 *            dummy_history_string_cb
*
 *            dummy_history_text_cb
*
 *            dummy_history_log_cb
*
 *
*
 * Purpose: callback functions for storing historical data of types float,
*
 *          integer, string, text and log respectively in external storage
*
 *
*
 * Parameters: history     - array of historical data
*
 *             history_num - number of elements in history array
*
 *
*
******************************************************************************
**/
static void dummy_history_float_cb(const ZBX_HISTORY_FLOAT *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_integer_cb(const ZBX_HISTORY_INTEGER *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}
```

```
static void dummy_history_string_cb(const ZBX_HISTORY_STRING *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_text_cb(const ZBX_HISTORY_TEXT *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

static void dummy_history_log_cb(const ZBX_HISTORY_LOG *history, int
history_num)
{
    int    i;

    for (i = 0; i < history_num; i++)
    {
        /* do something with history[i].itemid, history[i].clock,
history[i].ns, history[i].value, ... */
    }
}

/******************************************************************************
***
 *
*
 * Function: zbx_module_history_write_cbs
*
 *
*
 * Purpose: returns a set of module functions Zabbix will call to export
*
 *          different types of historical data
*
 *
*
```

```
 * Return value: structure with callback function pointers (can be NULL if
*
 *               module is not interested in data of certain types)
*
 *
*
******************************************************************************
**/
ZBX_HISTORY_WRITE_CBS   zbx_module_history_write_cbs(void)
{
    static ZBX_HISTORY_WRITE_CBS    dummy_callbacks =
    {
        dummy_history_float_cb,
        dummy_history_integer_cb,
        dummy_history_string_cb,
        dummy_history_text_cb,
        dummy_history_log_cb,
    };

    return dummy_callbacks;
}
```

Le module exporte trois nouveaux éléments :

- `dummy.ping` - retourne toujours '1'
- `dummy.echo[param1]` - retourne le premier paramètre tel qu'il est, par exemple, `dummy.echo[ABC]` retournera ABC
- `dummy.random[param1, param2]` - renvoie un nombre aléatoire compris dans la plage de param1-param2, par exemple, `dummy.random[1,1000000]`

6 Limitations

La prise en charge des modules chargeables est implémentée uniquement pour la plate-forme Unix. Cela signifie que cela ne fonctionne pas pour les agents Windows.

Dans certains cas, un module peut avoir besoin de lire les paramètres de configuration liés au module à partir de *zabbix_agentd.conf*. Ce n'est pas pris en charge actuellement. Si vous avez besoin de votre module pour utiliser certains paramètres de configuration, vous devez probablement implémenter l'analyse d'un fichier de configuration spécifique au module.