5 Подгружаемые модули

1 Обзор

Подгружаемые модули предлагают производительную опцию для расширения функциональности Zabbix.

Уже имеются возможности для расширения функциональности Zabbix при помощи:

- пользовательских параметров (метрики агента)
- внешние проверки (мониторинг без агента)
- `system.run[]` элемент данных Zabbix агента.

Они работают очень хорошо, но имеют главный недостаток, называемый форком (fork()). Zabbix должен создавать новый ответвленный процесс каждый раз для сбора пользовательских проверок, что не очень хорошо сказывается на производительности. Обычно это не самая большая проблема, но тем не менее это может быть серьезной проблемой для мониторинга встроенных систем, имеющих большое количество наблюдаемых параметров или тяжелых скриптов со сложной логикой или длительным временем запуска.

Поддержка подгружаемых модулей предлагает пути расширения Zabbix агента, сервера и прокси без ущерба производительности.

Подгружаемый модуль - в своей основе разделяемая библиотека используемая Zabbix демоном и загружаемая при старте демона. Библиотека должна содержать определенные функции такие, чтобы Zabbix процесс мог определить что файл на самом деле модуль и он может загрузить его и работать с ним.

Подгружаемые модули имеют много преимуществ. Отличная производительность и возможность внедрения в любую логику что очень важно, но возможно наиболее важное преимущество - возможность развития, использования и распространения Zabbix модулей. Это поспособствует безпроблемному обслуживанию и поможет вносить новую функциональность легче и независимо от кода ядра Zabbix.

Лицензирование и распространение модулей в бинарной форме регламентируется GPL лицензией (модули линкуются с Zabbix во время выполнения и используют заголовки Zabbix; в настоящее время весь код Zabbix лицензируется под GPL лицензией). Бинарная совместимость не гарантируется Zabbix'ом.

Постоянство API модулей гарантируется в пределах одного цикла Zabbix LTS (Долгосрочная поддержка) выпуска. Постоянство Zabbix API не гаратируется (технически имеется возможность вызова внутренних функций Zabbix из модуля, но гарантии, что такие модули будут работать нет).

2 API модулей

Для того чтобы разделяемая библиотека обрабатывалась как Zabbix модуль, она должна реализовывать и экспортировать несколько функций. На данный момент имеется шесть функций в API модулей Zabbix, только одна из которых обязательны, а остальные пять -

опциональны.

2.1 Обязательный интерфейс

Единственная обязательная функции - **zbx_module_api_version()**:

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

Эта функция должна возвращать API версию реализованную в модуле и, чтобы модуль загрузился, версия должна совпадать с версией API модулей поддерживаемой Zabbix. Версией API модулей поддерживаемой Zabbix является ZBX_MODULE_API_VERSION. Таким образом эта функция должна возвращать эту константу. Старая константа ZBX_MODULE_API_VERSION_ONE, которая ранее использовалась для этих целей, теперь определена равной ZBX_MODULE_API_VERSION для сохранения совместимости исходного кода, но её использование не рекомендуется.

2.2 Опциональный интерфейс

Опциональными функциями являются следующие функции - **zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** и **zbx_module_uninit()**:

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

Эта функция должна выполнять необходимую инициализацию для модуля (если таковые имеются). В случае успеха, функция должна вернуть ZBX_MODULE_OK. В противном случае, она должна вернуть ZBX_MODULE_FAIL. В последнем случае Zabbix не запустится.

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

Эта функция должна возвращать список элементов данных, поддерживаемых модулем. Каждый элемент данных указывается в структуре ZBX_METRIC, смотрите раздел ниже для подробностей. Список завершается при помощи структуры ZBX_METRIC с полем “key” равным NULL.

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

Если модуль экспортирует **zbx_module_item_list()**, тогда эта функция используется Zabbix, чтобы задать опцию времени ожидания в файле конфигурации Zabbix, которой проверки элементов данных реализованных в модуле должны подчиняться. Здесь, параметр “время ожидания” задается в секундах.

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

Эта функция должна возвращать функции обратного вызова (callback), которые будут использоваться Zabbix сервером или прокси для экспорта истории различных типов данных. Функции обратного вызова представляют собой поля ZBX_HISTORY_WRITE_CBS структуры, поля могут быть NULL, если модуль не заинтересован в истории некоторого типа.

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

Эта функция должна выполнять необходимые деинициализации (если таковые имеются), такие как освобождение выделенных ресурсов, закрытие файловых дескрипторов и так далее.

Все фунции вызываются один раз при запуске Zabbix, когда модуль загружен, за исключением zbx_module_uninit(), которая вызывается один раз при завершении работы Zabbix, когда модуль выгружен.

2.3 Определение элементов данных

Каждый элемент данных определяется в структуре ZBX_METRIC:

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

Здесь, **key** - ключ элемента данных (например, “dummy.random”), **flags** - либо CF_HAVEPARAMS, либо 0 (в зависимости от того, принимает ли элемент данных параметры или нет), **function** - C функция, которая обрабатывает элемент данных (например, “zbx_module_dummy_random”), и **test_param** - список параметров, которые используется когда Zabbix агент запускается с флагом “-p” (например, “1,1000”, может быть NULL). Пример определения может выглядеть наподобие этого:

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

Каждая функция, которая обрабатывает элемент данных должна принимать два указателя параметра, первый с типом AGENT_REQUEST и второй с типом 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;
}
```

Эти функции должны возвращать SYSINFO_RET_OK, если значение элемента данных получено успешно. В противном случае, функции должны возвращать SYSINFO_RET_FAIL. Смотрите пример “dummy” модуля ниже для получения деталей как получать информацию от AGENT_REQUEST и как указывать информацию в AGENT_RESULT.

2.4 Предоставление обратного вызова экспорту истории

Модуль может задавать функции для экспорта данных истории по типам: Числовой (с плавающей точкой), Числовой (целое положительное), Символ, Текст и Журнал (лог):

```
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;
```

Каждый из них должен принимать массив “history” из элементов “history_num” в виде аргументов. В зависимости от экспортируемого типа данных истории, “history” является массивом из следующих структур, соответственно:

```
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;
```

Обратные вызовы будут использоваться Zabbix сервером или прокси процессами синхронизации истории в конце процедуры синхронизации истории после того, как данные записаны в базу данных Zabbix и сохранены в кэш значений.

Для экспорта через модули прокси доступны только сырые значения. (Пользовательские множители не будут применяться, дельта не будет вычисляться и т.д.)

2.5 Сборка модулей

В настоящее время подразумевается, что модули должны быть собраны внутри дерева исходных кодов Zabbix, так как API модулей зависит от некоторых структур данных, которые определены в заголовках Zabbix.

Наиболее важный заголовок для подгружаемых модулей - **include/module.h**, который определяет перечисленные выше структуры данных. Другой полезный заголовок - **include/sysinc.h**, который выполняет включение необходимых системных заголовков, который сам по себе помогает include/module.h работать должным образом.

Для того чтобы include/module.h и include/sysinc.h были включены, необходимо сначала выполнить команду **./configure** (без аргументов) из корня дерева исходных кодов Zabbix. Команда создаст **include/config.h** файл, который основывается на include/sysinc.h. (Если вы получили исходные коды Zabbix из хранилища Subversion, скрипт ./configure не будет существовать, сначала выполните команду **./bootstrap.sh**, чтобы скрипт сгенерировался.)

Зная эту информацию, имейте в виду, что все готово для сборки модуля. Модуль должен включать **sysinc.h** и **module.h**, и скрипт сборки должен понимать, что эти два файла указаны в include. Смотрите пример модуля “dummy” ниже для получения деталей.

Другим полезный заголовком является **include/log.h**, который определяет функцию **zabbix_log()**, которую можно использовать для журналирования и отладки.

3 Параметры конфигурации

Zabbix агент, сервер и прокси поддерживают два параметра для работы с модулями:

- LoadModulePath – полный путь до расположения подгружаемых модулей
- LoadModule – модуль(и) для загрузки при старте. Модули должны находиться в папке указанной в параметре LoadModulePath. Допускается добавлять несколько параметров LoadModule.

Например, для расширения возможностей Zabbix агента мы можем добавить следующие параметры:

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

После запуска агента будут загружены модули mariadb.so, apache.so, kernel.so и dummy.so из папки /usr/local/lib/zabbix/agent. Агент не запустится, если модуль отсутствует, либо в случае некорректных прав доступа или разделяемая библиотека не является модулем Zabbix.

4 Настройка веб-интерфейса

Подгружаемые модули подерживаются Zabbix агентом, сервером и прокси. Следовательно, тип элемента данных в Zabbix веб-интерфейсе зависит от того где этот модуль загружен. Если модуль загружен на стороне агента, тогда тип элемента данных должен быть “Zabbix агент” или “Zabbix агент (активный)”. Если модуль загружен на стороне сервера или прокси, тогда тип элемента данных должен быть “Простая проверка”.

Экспорт истории через модули Zabbix не требует какой-либо настройки через веб-интерфейс. Если модуль успешно загружен сервером или прокси и предоставляет функцию **zbx_module_history_write_cbs()**, которая возвращает по крайней мере одну не-NULL функцию обратной связи, тогда экспорт истории будет включен автоматически.

5 Пустой модуль

Zabbix включает простой модуль, написанный на языке C. Модуль находиться в 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
```

Модуль хорошо документирован, он можно использовать как шаблон для ваших собственных модулей.

После выполнения ./configure в корне дерева исходных кодов Zabbix, как описано ранее, просто выполните **make** для сборки **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;
}
```

Модуль экпортирует 3 новых элемента данных:

- dummy.ping - всегда возвращает '1'
- dummy.echo[param1] - возвращает первый параметр как есть, например, dummy.echo[ABC] вернет ABC
- dummy.random[param1, param2] - возвращает случайное число из диапазона param1-param2, например, dummy.random[1,1000000]

6 Ограничения

Поддержка подгружаемых модулей реализована только на платформах Unix. Это означает, что подгружаемый модуль не работает на Zabbix агентах под Windows.

В некоторых случаях модулю может потребоваться прочитать параметры имеющие отношения к модулю из *zabbix_agentd.conf*. В настоящее время этот функционал не поддерживается. Если вам необходимо в вашем модуле использовать некоторые параметры конфигурации вам,

возможно, следует осуществлять анализ отдельного файла конфигурации модуля.