5 可加载模块

概览

可加载模块提供了一个侧重性能的选项，来扩展Zabbix的功能。

目前已经有以下的功能来扩展Zabbix功能：

- 用户自定义变量 (Agent指标)
- 扩展检查（无Agent监控）
- `system.run[]` Zabbix Agent监控项.

这些功能工作的十分优秀，但是存在一个重要的缺陷，名字叫fork()□在每次处理用户指标的时候都必须创建一个新的子进程，这样不会有优秀的性能表现。通常这并不是个大问题，然而这会在监控嵌入式系统、拥有大量监控参数或运行具有逻辑繁多或启动时间长的脚本的情况下成为一个严重的问题。

可加载模块提供了在不额外消耗性能的情况下，扩展Zabbix Agent□Server和Proxy□

一个可加载模块是基于一个在Zabbix守护进程启动时加载的共享库。这个库包含了一些功能，以便Zabbix可以检测到该文件确实事一个可以被加载和使用的模块。

可加载模块具有许多有点。出众的性能和实现任何逻辑的能力非常重要，但最重要的能力是开发、使用和分享的Zabbi模块。可加载模块有助于实现无故障维护，有助于更轻松的提供新功能并且不依赖于Zabbix核心代码库。

二进制形式的模块的授权和分发应在GPL许可证的许可下管理（模块运行时连接到Zabbix并且使用Zabbix的头文件；目前ZABBIX的代码根据GPL许可证进行授权□□ZABBIX 不保证二进制兼容性。

在一个ZABBIX LTS(长期支持)版本支持周期内保证API模块的稳定性□ZABBIX API的稳定性无法保证（从技术上讲，可以从模块调用ZABBIX内部函数，但不能保证这些模块可以工作）。

模块 API

为了将共享库视作ZABBIX模块，它应该实现并导出一些函数。目前□ZABBIX 模块 API 中由六个函数，其中一个是强制性的，另外五个是可选的。

强制接口

唯一的强制函数是**zbx_module_api_version()**:

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

此函数应该返回实现这个模块以来的API版本，并且为了模块能被加载，这个版本必须与 ZABBIX 支持的模块API版本匹配□Zabbux支持的模块API的版本为ZBX_MODULE_API_VERSION□座椅这个函数应该返回这个常量。用于此目的的旧常量ZBX_MODULE_API_VERSION_ONE□现在被定义为等于ZBX_MODULE_API_VERSION以保持源兼容性，但不建议使用它。

1 可选接口

可选的函数是**zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** and **zbx_module_uninit()**:

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

这个函数应该对模块的执行进行必要的初始化（如果有的话）。如果成功，则返回ZBX_MODULE_OK，否则它应该返回 ZBX_MODULE_FAIL ，若为后一种情况，ZABBIX 将无法启动。

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

此函数应当返回一个支持的监控项的列表。每个监控项目被定义为ZBX_METRIC的结构下，详细信息请见后文。这个列表应以“key”字段为NULL作为ZBX_METRIC结构的终止。

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

如果模块输出**zbx_module_item_list()**，那么基于这个模块的监控项会遵守这个函数，而不是遵照ZABBIX配置文件中的超时设置。这边“timeout”参数以秒为单位。

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

这个函数应当返回ZABBIX服务器将用于导出不同数据类型历史记录的回调函数。回调函数应以 ZBX_HISTORY_WRITE_CBS 结构的字段提供，如果模块对于某种类型的历史纪录不感兴趣，则字段可以为 NULL 。

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

这个函数应当执行必要的反初始化（如果有的话），如释放分配的资源、关闭文件描述符等。

所有的函数会在ZABBIX启动的时候加载模块时，除了zbx_module_uninit()都将被调用一次。在卸载磨块时，zbx_module_uninit()会被ZABBIX调用一次。

定义监控项

每个监控项都应当被定义在 ZBX_METRIC 结构中：

```
typedef struct
{
```

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

这里的**key**指的时监控项的key□例如□“ dummy.random ”□□**flags**可以是 CF_HAVEPARAMS 或 0（取决于监控项是否接受参数）， **function** 是实现该监控项的 C 函数（例如□“zbx_module_dummy_random”□□最后 **test_param** 是使用 -P 标志启动 ZABBIX Agent 时使用的参数里列表（例如：“1,1000”， 可以是 NULL□□下面是一个具体示例：

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

每个实现一个监控项的函数应该接受俩哥哥指针参数函数，第一个是一种AGENT_REQUEST类型，第二个是一种AGENT_RESULT类型：

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

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

    return SYSINFO_RET_OK;
}
```

如果这个监控项的值被成功获取，这些函数应当返回 SYSINFO_RET_OK □否则，应当返回 SYSINFO_RET_FAIL□关于如何从 AGENT_REQUEST 获取信息以及如何设定 AGENT_RESULT 的详情，请参阅示例“dummy”模块。

提供历史记录输出的回调

从ZABBIX 4.0.0开始，不再支持通过ZABBIX Proxy 经模块输出历史记录

模块可以按来行指定输出历史数据的函数：数字（浮点）、数字（无符号）、字符串、文本和日志：

```
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;
```

每个输出历史纪录的函数都应当把 “history_num”元素 作为 “history” 数组的参数。依据需要输出的历史记录类型，“history” 分别是以下结构的数组：

```
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;
```

回调会在ZABBIX server 的历史记录同步进程完成历史记录同步操作，数据被写入ZABBIX 数据库并将值保存在值缓存中后执行。

构建模块

目前，模块应当再ZABBIX源代码树中构建，因为模块API依赖于一些ZABBIX头文件中定义的一些数据结构。

对可加载模块来说，最重要的头是**include/module.h**，它定义了这些住居结构。另一个很有用的头文件**include/sysinc.h** ，它的执行会包含必要的系统头文件，这有助于 include/module.h 的正常工作。

为了include/module.h 和 include/sysinc.h被导入，应在ZABBIX 源代码树的根目录下执行**./configure**命令。这将创建 **include/config.h** 文件，其中包含了 include/sysinc.h 依赖。（如果你获得的ZABBIX源代码来自子版本存储库，则 ./configure 脚本尚不存在，应首先运行 **./bootstrap.sh** 脚本来生成它。）

记住这些信息，一切都准备好了去构建模块。该模块应包含 **sysinc.h** 和 **module.h** ，构建脚本应确保这两个文件包含于路径中。有关详细信息，参见下文“dummy” 模块。

其它有用的头文件**include/log.h**，它定义了**zabbix_log()**函数，可用于记录和调试目的。

配置参数

ZABBIX Agent, Server和Proxy支持两个 参数来处理模块：

- LoadModulePath – 可加载模块所在的完整路径
- LoadModule – 启动时加载的模块。这些模块必须位于 LoadModulePath 制定的目录中。允许包含多个 LoadModule 参数

举个例子：要扩展ZABBIX Agent 我们可以添加以下参数：

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

在启动Agent时，它将从/usr/local/lib/zabbix/agent/目录加载mariadb.so, apache.so, kernel.so and

dummy.so 模块。如果发生缺少模块、权限错误或该共享库文件不是ZABBIX模块，那么Agent的启动将失败。

前端配置

ZABBIX Agent□Server和Proxy支持可加载模块。因此ZABBIX前端中的监控项类型依据模块在哪里被加载。如果模块在Agent端被加载那么监控项类型应当设置为“Agent检查”或“Agent检查（主动）”。如果在Server端或Proxy端被加载，那么响应的类型应当为“简单检查”。

通过ZABBIX模块历史记录输出不需要进行前端配置。如果模块成功加载并提供**zbx_module_history_write_cbs()**函数且该函数应至少返回一个非NULL回调方法，则将自动启动历史记录输出。

Dummy模块

ZABBIX包含一个用C语言编写的示例模块。该模块位于 `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
```

这个模块由详细的文档，可以作为您编写自己的模块的模板。

如上所述，在ZABBIX源代码根目录下运行./configure命令后，至于要运行 **make** 即可构建 **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;
}
```

这个模块导出三个新的监控项类型：

- dummy.ping - 总是返回 '1'
- dummy.echo[param1] - 总是返回第一个参数，例如 dummy.echo[ABC] 将返回 ABC
- dummy.random[param1, param2] - 返回param1与param2范围内的随机数，例如, dummy.random[1,1000000]

限制

仅对类Unix平台实现了可加载模块的支持。这意味着它不适用于Windows 平台的Agent。

某些情况下，模块可能要从*zabbix_agentd.conf*读取与模块相关的配置参数。目前不支持这么操作。如果您需要模块使用某些配置参数，则应该实现特定与模块的配置文件解析。