5 可加载模块

5.1 概述

可加载模块提供了一种关于Zabbix性能扩展的选项。

可以通过以下方式扩展Zabbix功能：

- user parameters（代理指标）
- external checks（无代理监控）
- `system.run[]` Zabbix agent item.

它们能运行的很好，但有一个主要的缺点，即fork□□□Zabbix必须在每次处理用户指标时分配一个新进程，这对于性能不利。这一般来讲不是严重的问题，但是在监控嵌入式系统，具有大量监控参数或具有复杂逻辑或启动时间很长的复杂脚本的情况下，这也许是一个严重的问题。

支持可加载模块提供了扩展Zabbix代理、服务器、代理服务器的方法，同时不牺牲性能。

可加载模块基本上是Zabbix守护程序使用的共享库，并在启动时加载。该库应包含某些功能，以便Zabbix进程可能会检测到该文件确实是可以加载和使用的模块。

可加载模块具有许多优点。卓越的性能和可实现任何逻辑的能力是非常重要的，但更最重要的优势是使用和共享Zabbix模块的开发能力。它有助于可靠的维护，并有助于更轻松、更独立于Zabbix核心代码库提供新功能。

5.2 模块API

为了将共享库按照Zabbix模块进行处理，它应该执行和导出多个功能□Zabbix模块API目前有六个功能，其中只有一个是强制性的，另外五个是可选的。

5.2.1 强制接口

唯一的强制功能是 **zbx_module_api_version()**:

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

此函数应该返回此模块的API版本，为了模块加载此版本必须匹配Zabbix支持的模块API版本□ Zabbix支持的模块API版本为ZBX_MODULE_API_VERSION□ 所以这个函数应该返回一个常数。用于此目的的旧常数ZBX_MODULE_API_VERSION_ONE现在被定义为等于ZBX_MODULE_API_VERSION以保持兼容性，但不推荐使用。

5.2.2 可选接口

可选的接口是 **zbx_module_init()**, **zbx_module_item_list()**, **zbx_module_item_timeout()**, **zbx_module_history_write_cbs()** 和 **zbx_module_uninit()**:

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

该功能应该对模块进行必要的初始化（如果有的话）。如果成功，应该返回ZBX_MODULE_OK，否则，应该返回ZBX_MODULE_FAIL，在后一种情况下，Zabbix将不会启动。

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

此功能应返回模块支持的监控项列表。每个监控项都是在ZBX_METRIC结构中定义的，有关详细信息，请参阅下面的部分。列表由ZBX_METRIC结构终止，“key”字段为NULL。

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

如果模块导出**zbx_module_item_list()**，那么Zabbix使用该函数来指定Zabbix配置文件中由模块实现的监控项检查应该遵循的超时设置。这里，“timeout”参数是以秒为单位。

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

该函数应该返回回调函数Zabbix服务器或代理服务器将用于导出不同数据类型的历史记录。回调函数作为ZBX_HISTORY_WRITE_CBS结构的字段提供，如果模块对某种类型的历史不感兴趣，则字段可以为NULL。

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

此功能应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。

模块加载时，Zabbix启动时，所有功能除了zbx_module_uninit()都将被调用一次。在卸载模块时，Zabbix关闭时会调用一次。

5.2.3 定义监控项

每个监控项都在ZBX_METRIC结构中定义：

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

这里□key是监控项Key□例如□“dummy.random”□□标志是CF_HAVEPARAMS或0（取决于监控项是否接受参数□□function是实现该项目的C函数（例如□“zbx_module_dummy_random” □□test_param是使用“-p”标志启动Zabbix代理时使用的参数列表（例如，“1,1000”，可以为NULL□□ 示例定义如下所示：

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

实现一个监控项的每个函数应该接受两个指针参数，第一个是AGENT_REQUEST类型，第二个是AGENT_RESULT类型的参数：

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

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

    return SYSINFO_RET_OK;
}
```

如果监控项的值成功获取，这些函数应返回SYSINFO_RET_OK□否则，它们应该返回SYSINFO_RET_FAIL□有关如何从AGENT_REQUEST获取信息以及如何在AGENT_RESULT中设置信息的详细信息，请参阅下面的示例“dummy”模块。

5.2.4 提供历史记录导出回调

模块可以指定按类型导出历史数据的函数□Numeric□float□□Numeric□unsigned□□Character□Text和Log□

```
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;
```

每个人都应该以“history_num”元素的“history”数组为参数。 根据要导出的历史数据类型“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;
```

数据写入Zabbix数据库并保存在值缓存中之后，Zabbix服务器或代理服务器历史记录进程将在历史记录同步过程结束时使用回调。

只有原始值可用于通过代理模块导出。（自定义乘法器将不能应用，三角形将不被计算等）

5.2.5 构建模块

模块是要在Zabbix源代码树中构建的，因为模块API依赖于在Zabbix头文件中定义的一些数据结构。

可加载模块最重要的头是**include/module.h**，它定义了这些数据结构。另一个有用的头是**include/sysinc.h**，它执行包含必要的系统头，这本身就有助于include/module.h正常工作。

为了include/module.h和include/sysinc.h被包括在内，应该首先在.bbbix源代码树的根目录中运行**./configure**命令（不带参数）。这将创建**include/config.h**文件，其中包括/sysinc.h依赖。（如果你获得Zabbix源代码作为Subversion存储库检出，则./configure脚本不存在，应首先运行./bootstrap.sh命令生成它。）

在上述情况都考虑到之后，一切都为建立模块准备好了。该模块应包括**sysinc.h**和**module.h**，构建脚本应确保这两个文件位于包含路径中。有关详细信息，请参见下面的“dummy”模块示例。

另一个有用的头是**include/log.h**，它定义了zabbix_log()函数，可用于记录和调试目的。

5.3 配置参数

Zabbix代理、服务器和代理服务器支持两种模块参数:

- LoadModulePath – 可加载模块位置的全路径
- LoadModule – 模块在启动时加载。模块必须位于LoadModulePath指定的目录中。允许包含多个LoadModule参数。

例如，要扩展Zabbix代理，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
```

代理启动后，将从/usr/local/lib/zabbix/agent目录加载mariadb.so、apache.so、kernel.so和dummy.so模块。如果模块丢失、权限不匹配或共享库不是Zabbix模块，则会失败。

5.4 Web前端的配置

Zabbix代理、服务器和代理服务器支持可加载模块。 因此，Zabbix Web前端中的监控项类型取决于模块的加载位置。如果模块加载到代理中，则监控项类型应为“Zabbix代理”或“Zabbix代理（活动）”。 如果模块加载到服务器或代理服务器，则监控项类型应为“简单检查”。

通过Zabbix模块的历史导出不需要任何前端配置。如果模块由服务器或代理服务器成功加载，并提供返回至少一个非NULL回调函数的**zbx_module_history_write_cbs()**函数，那么历史导出将自动启用。

5.5 Dummy模块

Zabbix包含用C语言编写的示例模块。该模块位于src/modules/dummy下：

```
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
```

该模块有详细的文档，它可以用作你自己的模块的模板。

在./configure已经在Zabbix源代码树的根目录中运行，如上所述，只需运行**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;
}
```

该模块导出三个新监控项:

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

5.6 限制

仅针对Unix平台实现可加载模块的支持。这意味着它不适用于Windows代理。

在某些情况下，模块可能需要从*zabbix_agentd.conf*读取相关的配置参数。如果你需要使用模块来使用某些配置参数，那么你应该可以实现对特定于模块的配置文件的解析。