# htp\_audit\_plugin

# 设计说明文档

**文档版本变更记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 日期 | 版本 | 说明 | 变更人 |
| 1 | 2019/1/10 | 1.0 | 初始文档 | 霍伟元 |
| 2 | 2019/1/11 | 1.1 | 1. 细化系统要求  2. 较少已修改函数体介绍  3. 增加数据结构/函数说明  4. 修正序列图 | 霍伟元 |

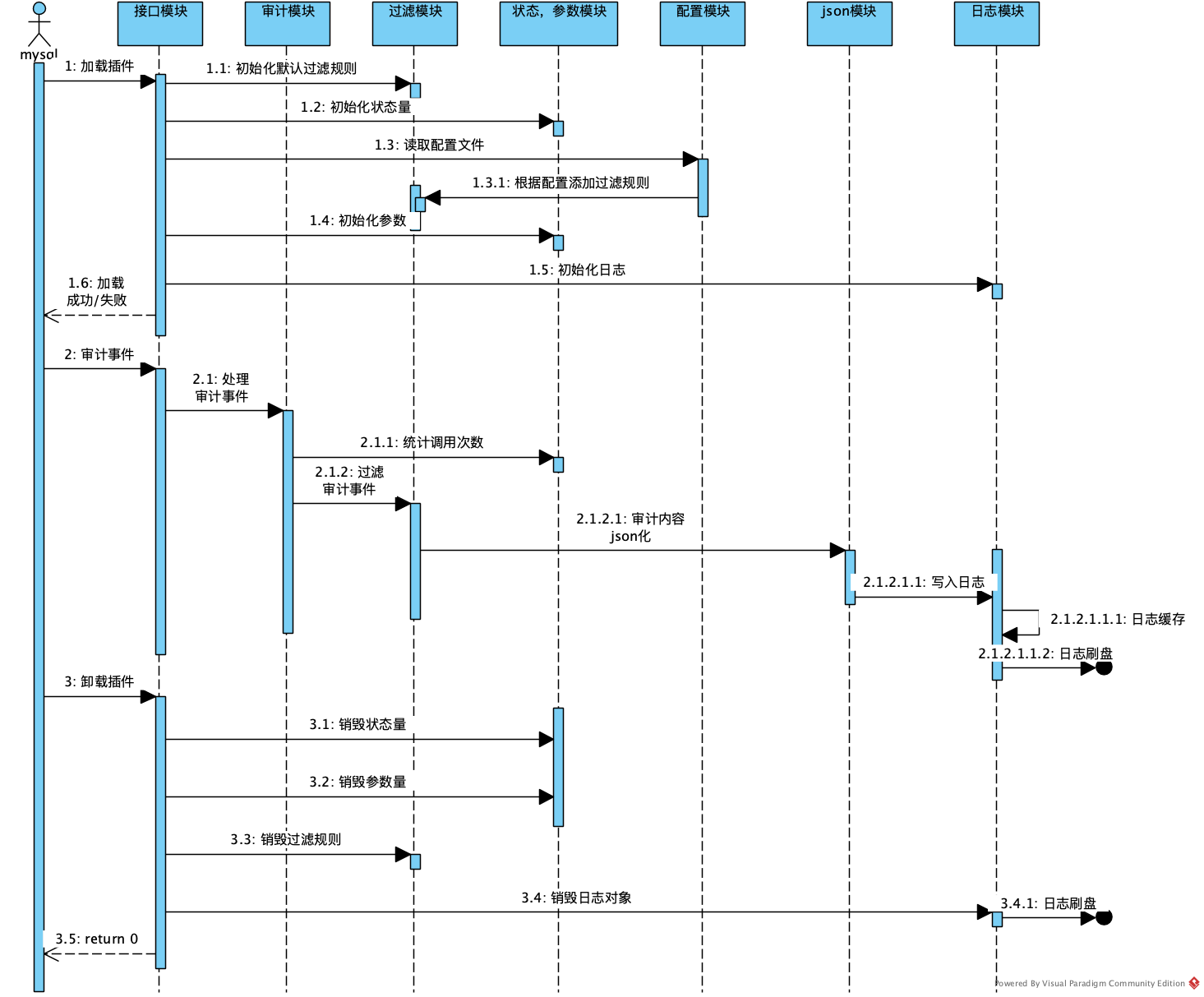
## 系统要求

|  |  |
| --- | --- |
| 组件 | 已验证适用版本号 |
| 操作系统版本 | CentOS 7.x |
| MySQL | 5.7.23 |
| 5.7.24 |
| … |

## 模块说明

|  |  |  |
| --- | --- | --- |
| **模块名称** | **源文件对应关系** | **说明** |
| 接口模块 | htp\_audit\_main.cc | 注册/卸载插件，调用其他模块进行审计工作 |
| 日志模块 | htp\_audit\_filter.h、logger.cc | 将审计内容记录到文件，且可实现日志缓存。 |
| 审计模块 | htp\_audit\_class.cc | 处理审计事件 |
| 过滤模块 | audit\_item.cc | 过滤审计内容，校验读入的过滤格式 |
| 配置模块 | config.c | 读取配置文件 |
| 状态，参数模块 | htp\_audit\_vars.cc | 状态计算和参数设置实现 |
| Json模块 | cJSON.c | 将审计内容转变成json格式 |

## 模块依赖关系



## 源文件说明

|  |  |
| --- | --- |
| **源文件名称** | **说明** |
| audit\_item.cc | 对htp\_audit\_filter.h声明的函数进行实现。 |
| cJSON.c | 开源代码，对写入到日志的信息格式化成json。 |
| config.c | 1. 读取配置文件，将文件内容格式化存储到内存中。 2. 销毁内存中配置。 |
| htp\_audit\_class.cc | 实现对审计事件处理，计算事件类型出现次数，调用htp\_audit\_filter.h声明的函数过滤事件。 |
| htp\_audit\_filter.cc | 1. 写MySQL error日志信息； 2. 定义配置文件关键字； 3. 声明日志处理类； 4. 审计内容的初始化、校验和处理 |
| htp\_audit\_main.cc | 1. 定义插件注册描述符以及相关的初始化、销毁和审计事件的调用接口； 2. 对config读取的配置进行正确性校验并读入到结构体。 |
| htp\_audit\_vars.cc | 1. 相关调用次数统计； 2. 初始化或销毁状态量和参数变量； 3. 动态修改参数变量值； |
| logger.cc | 1. 实现日志类和日志缓存。 |

## 审计接口

### 加载插件

加载插件过程中MySQL调用审计插件注册的初始化函数：htp\_audit\_plugin\_init。在此接口中将会添加锁保护，初始化状态量，初始化参数变量，审计规则记录，以及初始化日志信息。htp\_audit\_plugin\_init位于htp\_audit\_main.cc，函数头是：

static int htp\_audit\_plugin\_init(void \*arg \_\_attribute\_\_((unused)))

### 卸载插件

通过描述符注册，审计插件的卸载是调用的逆初始化接口：htp\_audit\_plugin\_deinit，同样位于htp\_audit\_main.cc文件中。在此接口中将会释放锁，注销参数变量，审计规则记录，以及刷新日志信息。函数头如下：

static int htp\_audit\_plugin\_deinit(void \*arg \_\_attribute\_\_((unused)))

### 审计记录

MySQL会将审计插件注册的关心的审计事件信息传递给审计插件注册的接口: **htp\_audit\_notify**，然后通过mysql\_event\_class\_t枚举类型参数确定事件类型分发到各个处理接口。

表格 一

|  |  |
| --- | --- |
| **类型值** | **调用函数** |
| MYSQL\_AUDIT\_GENERAL\_CLASS | htp\_audit\_process\_general\_event |
| MYSQL\_AUDIT\_CONNECTION\_CLASS | htp\_audit\_process\_connection\_event |
| MYSQL\_AUDIT\_PARSE\_CLASS | htp\_audit\_process\_parse\_event |
| MYSQL\_AUDIT\_AUTHORIZATION\_CLASS | htp\_audit\_process\_auth\_event |
| MYSQL\_AUDIT\_TABLE\_ACCESS\_CLASS | htp\_audit\_process\_table\_access\_event |
| MYSQL\_AUDIT\_GLOBAL\_VARIABLE\_CLASS | htp\_audit\_process\_variable\_event |
| MYSQL\_AUDIT\_SERVER\_STARTUP\_CLASS | htp\_audit\_process\_startup\_event |
| MYSQL\_AUDIT\_SERVER\_SHUTDOWN\_CLASS | htp\_audit\_process\_shutdown\_event |
| MYSQL\_AUDIT\_COMMAND\_CLASS | htp\_audit\_process\_command\_event |
| MYSQL\_AUDIT\_QUERY\_CLASS | htp\_audit\_process\_query\_event |
| MYSQL\_AUDIT\_STORED\_PROGRAM\_CLASS | htp\_audit\_process\_stored\_program\_event |

### 状态量和参数量处理

审计插件的状态量和参数量都通过一般描述符注册到MySQL。而后通过初始化接口，进行初始化。

在注册过程中，参数变量需要绑定对SET 语法对应的接口。此绑定会应用到MySQL的一系列宏定义，例如：

其中， MYSQL\_SYSVAR宏定义为：

struct st\_mysql\_sys\_var \*htp\_audit\_sys\_var[] = {

MYSQL\_SYSVAR(log\_file), MYSQL\_SYSVAR(error\_log\_file), MYSQL\_SYSVAR(rules), MYSQL\_SYSVAR(add\_rule),

MYSQL\_SYSVAR(remove\_rule), MYSQL\_SYSVAR(enable\_buffer), MYSQL\_SYSVAR(flush\_log), MYSQL\_SYSVAR(buffer\_size),

MYSQL\_SYSVAR(version), 0

};

另外还需要用到绑定接口的宏定义，例如：  
当然还涉及到其他的宏定义，这里不列出。这一系列的宏定义共同组成了一个st\_mysql\_sys\_var结构体的数组被注册到MySQL当中。

static MYSQL\_SYSVAR\_BOOL(enable\_buffer, enable\_buffer

, PLUGIN\_VAR\_NOCMDOPT

| PLUGIN\_VAR\_NOCMDARG

, "set whether use buffer to store audit record"

, NULL

, htp\_audit\_set\_enable\_buffer\_update

, FALSE);

#define MYSQL\_SYSVAR\_NAME(name) mysql\_sysvar\_ ## name

#define MYSQL\_SYSVAR(name) \

((struct st\_mysql\_sys\_var \*)&(MYSQL\_SYSVAR\_NAME(name)))

## 主要数据结构/函数说明

### htp\_audit\_main.cc

htp\_audit\_main.cc用config初始化filter\_items

#### 一般描述符mysql\_declare\_plugin

MySQL会通过该描述符知道插件类型、安装插件的名称、插件状态量、参数，安装插件和卸载插件的操作接口。

mysql\_declare\_plugin(htp\_audit)

{

MYSQL\_AUDIT\_PLUGIN, /\* type \*/

&htp\_audit\_descriptor, /\* descriptor \*/

"HTP\_AUDIT", /\* name, install plugin's plugin\_name \*/

"HotPU Corp Jiangyx & Haogq", /\* author \*/

"Htp audit plugin", /\* description \*/

PLUGIN\_LICENSE\_GPL,

htp\_audit\_plugin\_init, /\* init function (when loaded) \*/

htp\_audit\_plugin\_deinit, /\* deinit function (when unloaded) \*/

0x0001, /\* version \*/

htp\_audit\_status, /\* status variables \*/

htp\_audit\_sys\_var, /\* system variables \*/

NULL,

0,

}mysql\_declare\_plugin\_end;

#### 特殊描述符

通过一般插件描述符注册的具备st\_mysql\_audit结构的htp\_audit\_descriptor。

static struct st\_mysql\_audit htp\_audit\_descriptor =

{

MYSQL\_AUDIT\_INTERFACE\_VERSION, /\* interface version \*/

NULL, /\* release\_thd function \*/

htp\_audit\_notify, /\* notify function \*/

{(unsigned long) MYSQL\_AUDIT\_GENERAL\_ALL,

(unsigned long) MYSQL\_AUDIT\_CONNECTION\_ALL,

(unsigned long) MYSQL\_AUDIT\_PARSE\_ALL,

0, /\* This event class is currently not supported. \*/

(unsigned long) MYSQL\_AUDIT\_TABLE\_ACCESS\_ALL,

(unsigned long) MYSQL\_AUDIT\_GLOBAL\_VARIABLE\_ALL,

(unsigned long) MYSQL\_AUDIT\_SERVER\_STARTUP\_ALL,

(unsigned long) MYSQL\_AUDIT\_SERVER\_SHUTDOWN\_ALL,

(unsigned long) MYSQL\_AUDIT\_COMMAND\_ALL,

(unsigned long) MYSQL\_AUDIT\_QUERY\_ALL,

(unsigned long) MYSQL\_AUDIT\_STORED\_PROGRAM\_ALL}

};

#### 审计事件接口（接口函数）

每次发生了审计事件，MySQL都会调用函数htp\_audit\_notify，这个函数是由特殊描述符注册的。函数头如下：

static int htp\_audit\_notify(MYSQL\_THD thd,

mysql\_event\_class\_t event\_class,

const void \*event)

htp\_audit\_notify除了记录调用审计插件次数外，其他操作都交给了htp\_audit\_process\_event，这个函数将会根据审计类型调用对应的函数进行审计（表格 一）。函数头如下：

void htp\_audit\_process\_event(MYSQL\_THD thd \_\_attribute\_\_((unused)),

unsigned int event\_class,

const void \*event)

### htp\_audit\_vars.cc

#### 状态量结构体

状态量结构体是一个st\_mysql\_show\_var类型的结构体数组，形如：

struct st\_mysql\_show\_var htp\_audit\_status[] =

{

{"Htp\_audit\_called",

(char \*) &number\_of\_calls,

SHOW\_LONGLONG, SHOW\_SCOPE\_GLOBAL},

...

};

该结构体定义了注册到MySQL的状态量。在MySQL通过show status like 出来的结果就是在此定义。

#### 参数量结构体

参数量结构体是一个st\_mysql\_sys\_var类型的结构体数组，形如：

struct st\_mysql\_sys\_var \*htp\_audit\_sys\_var[] = {

MYSQL\_SYSVAR(log\_file),

… ,0

};

该结构体定义了注册到MySQL的参数变量，以及设置参数变量调用的接口，具体实现在上文有介绍。

#### 主要函数说明

static int rules2str\_buffer\_init(rules2str\_buffer\_t \*buffer)

初始化buffer结构体，其实就是缓存字符串

static void rules2str\_buffer\_deinit(rules2str\_buffer\_t \*buffer)

释放buffer中buffer所占内存

static void htp\_audit\_rule\_2\_str(

filter\_item\_t \*item, char \*buffer, int size)

将过滤项中内容严格按照配置文件格式严格转化为字符串

static void rules2str\_buffer\_write(const char \*rule, rules2str\_buffer\_t \*buffer)

将字符串规则写入buffer中

static void htp\_audit\_rules\_2\_str(rules2str\_buffer\_t \*buffer)

将所有过滤规则写入到buffer中

void number\_of\_calls\_incr()

更新状态量函数的实现

void htp\_audit\_init\_status()

为状态量初始化为0

static void htp\_audit\_add\_rule\_update(

THD \*thd, /\*!< in: thread handle \*/

struct st\_mysql\_sys\_var \*var, /\*!< in: pointer to

              system variable \*/

void \*var\_ptr, /\*!< out: where the

              formal string goes \*/

const void \*save); /\*!< in: immediate result

              from check function \*/

系统变量更新函数

static int htp\_audit\_add\_rule\_validate(

/\*=============================\*/

THD \*thd, /\*!< in: thread handle \*/

struct st\_mysql\_sys\_var \*var, /\*!< in: pointer to system

variable \*/

void \*save, /\*!< out: immediate result

for update function \*/

struct st\_mysql\_value \*value); /\*!< in: incoming string \*/

系统变量校验函数，，

在

static MYSQL\_SYSVAR\_STR(add\_rule, add\_rule

, PLUGIN\_VAR\_NOCMDOPT

| PLUGIN\_VAR\_NOCMDARG

, "Htp audit add new rule"

, htp\_audit\_add\_rule\_validate

, htp\_audit\_add\_rule\_update

, "");

中调用。

校验中调用到htp\_audit\_filter.cc中函数，因为需要解析添加规则字符串放入filter\_item中；

问题：在validate和update中两次调用htp\_audit\_parse\_filter(dup\_str, &item);

是不是没必要，能不能提高下效率？

### htp\_audit\_filter.cc

#### 判断审计规则

在进行事件过滤之前，需要根据用户配置确定感兴趣的事件，因此需要做一些过滤判断。htp\_audit\_filter\_event函数就是在做这件事情，其存在两个重载：

在通过这个函数的校验之后才会真正进入到审计的记录。

filter\_result\_enum htp\_audit\_filter\_event(event\_info\_t \*info, filter\_item\_t \*item, unsigned int event\_class)

{

}

filter\_result\_enum htp\_audit\_filter\_event(event\_info\_t \*info, unsigned int event\_class)

{

}

#### 主要结构体

enum filter\_result\_enum

{

AUDIT\_EVENT, NOT\_AUDIT\_EVENT

};

是否审计标志

Logger类定义

struct filter\_item\_struct

{

bool name\_setted;

char name[MAX\_FILTER\_NAME\_BUFFER\_SIZE];

bool host\_setted;

char host[MAX\_FILTER\_HOST\_BUFFER\_SIZE];

int host\_length;

bool user\_setted;

char user[MAX\_FILTER\_USER\_BUFFER\_SIZE];

int user\_length;

//事件(event)相关

...

过滤项结构体：对应配置文件中过滤rule；

struct event\_info\_struct

{

const char \*host;

const char \*ip;

const char \*user;

int main\_class;

int sub\_class;

const char \*command;

const char \*query;

const char \*name;

const char \*sql\_command;

const char \*database;

const char \*table;

};

事件信息结构体

struct remove\_parse\_struct

{

int count;

char removes[MAX\_REMOVE\_ITEM][MAX\_FILTER\_NAME\_BUFFER\_SIZE];

};

保存需要删除的过滤项名称

list<int> filters;

filter\_item\_t filter\_items[MAX\_FILTER\_ITEMS];

static char filter\_using\_map[MAX\_FILTER\_ITEMS];

其中filters存储正在占用的filter\_items下标

Filter\_using\_maping存储对应的filter\_items是否正在被使用

#### 主要函数说明

static void htp\_audit\_fill\_event(

filter\_item\_t \*item, int main\_class, int sub\_class)

添加事件类型到filter\_item

int htp\_audit\_parse\_event(const char \*event, int event\_len, filter\_item\_t \*item)

解析被审计事件，设置filter\_item

int htp\_audit\_parse\_filter(const char \*filter\_str, filter\_item\_t \*item)

解析字符串设置filter\_item

filter\_result\_enum

htp\_audit\_filter\_event(event\_info\_t \*info, filter\_item\_t \*item, unsigned int event\_class)

最终根据event\_info\_t和filtr\_item\_t 判断是不是需要审计事件

filter\_result\_enum htp\_audit\_filter\_event(event\_info\_t \*info, unsigned int event\_class)

遍历所有的filter\_item判断是不是要审计

### config.cc

#### 6.4.1 .主要结构体说明

|  |
| --- |
| struct config\_err\_struct {int err\_no; //出现错误时的错误号，0表示无错误int line\_no; //出现错误时的行号}; |
| struct config\_item\_struct {  char key[KEY\_BUFFER\_SIZE];  int key\_len;  char value[VALUE\_BUFFER\_SIZE];  int value\_len;  void \*next;  }; |

存储配置键值对和指向下一键值对的指针

|  |
| --- |
| struct config\_group\_struct {  char name[GROUP\_NAME\_BUFFER\_SIZE];  int name\_len;  int number; /\*0-based。配置文件中，该group的位置。第一个配置组的值为0\*/  config\_item\_t \*items;  void \*next;  }; |

存储配置组信息，每个配置组包括名字、位置、和多个配置项，还有指向下一配置组的指针

|  |
| --- |
| struct config\_struct {  config\_group\_t anonymous; /\*在没有指定任何group的配置项，存放在这里\*/  int group\_amount; /\*配置的文件中的group数目\*/  config\_group\_t \*groups;  }; |

存储整个配置信息，包括没有任何分组的配置信息和配置组的数目和配置组指针链表

|  |
| --- |
| struct config\_parser\_struct {  config\_group\_t \*current\_group;  config\_item\_t \*current\_item;  }; |

配置解析器：存储解析的当前组和当前配置项指针

#### 6.4.2.主要函数说明

文件中主要有对config的初始化和设置，parser解析器的初始化等函数，主要是从配置文件中读取信息放入config之中

### Audit\_item.cc

#### 主要功能描述

文件中主要是对审计事件的具体处理，即将审计信息以cjson的格式输出到审计文件中

### htp\_audit\_class.cc

#### 主要功能描述

根据filter\_items判断是否应该审计，如该审计，则调用audit\_item.cc中的审计输出函数进行输出

#### 主要函数说明

static int htp\_audit\_process\_general\_event(

MYSQL\_THD thd \_\_attribute\_\_((unused)), unsigned int event\_class, const void \*event)

这是general事件的判断，其中event转化为mysql\_event\_general \* 类型，而这个类型在plugin\_audit.h中有定义。

此函数中主要是根据event信息判断是不是需要审计，以此进行状态变量的更新和审计事件的具体处理（如需审计，则审计输出，不需则返回）

void htp\_audit\_process\_event(MYSQL\_THD thd \_\_attribute\_\_((unused)),

unsigned int event\_class,

const void \*event)

所有事件处理的接口函数，被htp\_audit\_main中htp\_audit\_notify注册接口函数调用；