Zookeeper使用ACL进行访问权限控制

ZooKeeper使用ACL来控制访问其znode（ZooKeeper的数据树的数据节点）。ACL的实现方式非常类似于UNIX文件的访问权限：它采用访问权限位 允许/禁止
对节点的各种操作以及能进行操作的范围。不同于UNIX权限的是，ZooKeeper的节点不局限于 用户（文件的拥有者），组和其他人（其它）这三个标准范围。ZooKeeper不具有znode的拥有者的概念。相反，ACL指定id集以及与之对应的权限。

需要注意的是：
一条ACL仅针对于一个特定的节点，并不会把权限应用于子节点。例如，如果/app
只对IP：172.16.16.1可读
而/app/status是对任何人可读的，那么任何人都对/app/status拥有读权限，ACL不是递归的。

ZooKeeper支持可插拔的身份验证方案。 id使用如下形式 scheme:id，其中 scheme 是 id 所对应一个认证方案。例如，IP：172.16.16.1，id为主机的地址 172.16.16.1。

当客户端连接到ZooKeeper验证自己时，ZooKeeper将有关该客户端的所有Id与客户连接关联。客户端试图访问一个节点时，这些ID与该znodes的ACL验证。
ACL是由（scheme:expression, perms）对构成。其中expression的格式指定为scheme。例如，（IP：19.22.0.0/16，READ）表示对所有起始IP为19.22的客户端具有读权限。

ACL 权限

ZooKeeper 支持以下权限：

CREATE: 能创建子节点
READ：能获取节点数据和列出其子节点
WRITE: 能设置节点数据
DELETE: 能删除子节点
ADMIN: 能设置权限

CREATE权限和DELETE权限从WRITE权限中分离出来，是为了获得更好的访问控制。使用CREATE和DELETE权限的场景是：你想让A用户能够设置节点数据，但不允许创建或删除子节点。

有CREATE但无DELETE权限：客户端发出一个在父目录下创建节点的请求。你想让所有客户端能够添加，但是只有创建者能够删除。（这类似于文件的APPEND权限）。

内置的 ACL schemes

ZooKeeper有如下内置的schemes

world 有个唯一的id, anyone ，代表所有人。
auth 不使用任何id，代表任何已认证的用户。
digest 用 username:password 字符串来产生一个MD5串，然后该串被用来作为ACL ID。认证是通过明文发送username:password 来进行的，当用在ACL时，表达式为username:base64 ，base64是password的SHA1摘要的编码。
ip 使用客户端的主机IP作为ACL ID 。这个ACL表达式的格式为addr/bits ，此时addr中的有效位与客户端addr中的有效位进行比对。

ZooKeeper C 客户端 API

如下的常量是由ZooKeeper C语言库中提供的：

```
const int ZOO_PERM_READ;  //能读节点的值及列出其子节点
const int ZOO_PERM_WRITE; //能设置节点的值
const int ZOO_PERM_CREATE;  //能创建子节点
const int ZOO_PERM_DELETE; // 能删除子节点
const int ZOO_PERM_ADMIN;  //能执行set_acl()
const int ZOO_PERM_ALL; // 上面所有值的OR
```

下面是标准的ACL IDs

```
struct Id ZOO_ANYONE_ID_UNSAFE; //('world','anyone')
struct Id ZOO_AUTH_IDS;// ('auth','')
```

ZOO_AUTH_IDS 为空时，应被理解成“创建者的Id”

ZooKeeper 客户端有三种标准的ACL：

```
struct ACL_vector ZOO_OPEN_ACL_UNSAFE; //(ZOO_PERM_ALL,ZOO_ANYONE_ID_UNSAFE)
struct ACL_vector ZOO_READ_ACL_UNSAFE;// (ZOO_PERM_READ, ZOO_ANYONE_ID_UNSAFE)
struct ACL_vector ZOO_CREATOR_ALL_ACL; //(ZOO_PERM_ALL,ZOO_AUTH_IDS)
```

ZOO_OPEN_ACL_UNSAFE使所有ACL都“开放”了：任何应用程序在节点上可进行任何操作，能创建、列出和删除它的子节点。对任何应用程序，ZOO_READ_ACL_UNSAFE是只读的。CREATE_ALL_ACL赋予了节点的创建者所有的权限，在创建者采用此ACL创建节点之前，已经被服务器所认证（例如，采用“ digest”方案）。

以下ZooKeeper方法处理ACL:

```
//应用程序使用zoo_add_auth方法来向服务器认证自己，如果想用
//不同的方案来认证，这个方法可以被调用多次。
int zoo_add_auth (zhandle_t *zh,const char* scheme,const char* cert,
        int certLen, void_completion_t completion,
        const void *data);

//zoo_create(...)方法创建一个新节点。acl 参数是一个与这个节点关联的ACL
//列表，父节点权限项的CREATE位已被设(set,即由权限)。
int zoo_create (zhandle_t *zh, const char *path, const char *value,
        int valuelen, const struct ACL_vector *acl, int flags,
        char *realpath, int max_realpath_len);

//这个方法返回这个节点的ACL信息
int zoo_get_acl (zhandle_t *zh, const char *path,
        struct ACL_vector *acl, struct Stat *stat);


//这个方法用新的节点的ACL列表替换老的，
//这个节点的ADMIN位必须被设置（set,即具有ADMIN权限）。
int zoo_set_acl (zhandle_t *zh, const char *path,
        int version,const struct ACL_vector *acl);
```

这有一个使用上面API的例子，采用”foo”方案认证，创建一个“/xyz”的暂态节点，设置其为”只创建“权限。这是一个非常简单的例子，它的目的是展示如何与ZooKeeper ACL交换。C客户端的实现，参考.../trunk/src/c/src/cli.c:

```
#include <string.h>
#include <errno.h>

#include "zookeeper.h"

static zhandle_t *zh;

/**
 * In this example this method gets the cert for your
```

```
 *   environment -- you must provide
 */
char *foo_get_cert_once(char* id) { return 0; }

/** Watcher function -- empty for this example, not something you should
 * do in real code */
void watcher(zhandle_t *zzh, int type, int state, const char *path,
             void *watcherCtx) {}

int main(int argc, char argv) {
  char buffer[512];
  char p[2048];
  char *cert=0;
  char appId[64];

  strcpy(appId, "example.foo_test");
  cert = foo_get_cert_once(appId);
  if(cert!=0) {
    fprintf(stderr,
            "Certificate for appid [%s] is [%s]₩n",appId,cert);
    strncpy(p,cert, sizeof(p)-1);
    free(cert);
  } else {
    fprintf(stderr, "Certificate for appid [%s] not found₩n",appId);
    strcpy(p, "dummy");
  }

  zoo_set_debug_level(ZOO_LOG_LEVEL_DEBUG);

  zh = zookeeper_init("localhost:3181", watcher, 10000, 0, 0, 0);
  if (!zh) {
    return errno;
  }
  if(zoo_add_auth(zh,"foo",p,strlen(p),0,0)!=ZOK)
    return 2;

  struct ACL CREATE_ONLY_ACL[] = {{ZOO_PERM_CREATE, ZOO_AUTH_IDS}};
  struct ACL_vector CREATE_ONLY = {1, CREATE_ONLY_ACL};
  int rc = zoo_create(zh,"/xyz","value", 5, &CREATE_ONLY, ZOO_EPHEMERAL,
                      buffer, sizeof(buffer)-1);

  /** this operation will fail with a ZNOAUTH error */
  int buflen= sizeof(buffer);
  struct Stat stat;
  rc = zoo_get(zh, "/xyz", 0, buffer, &buflen, &stat);
  if (rc) {
```

```
    fprintf(stderr, "Error %d for %s₩n", rc, __LINE__);
  }

  zookeeper_close(zh);
  return 0;
}
```

本文的翻译自：http://zookeeper.apache.org/doc/trunk/zookeeperProgrammers.html#sc_ZooKeeperAccessControl