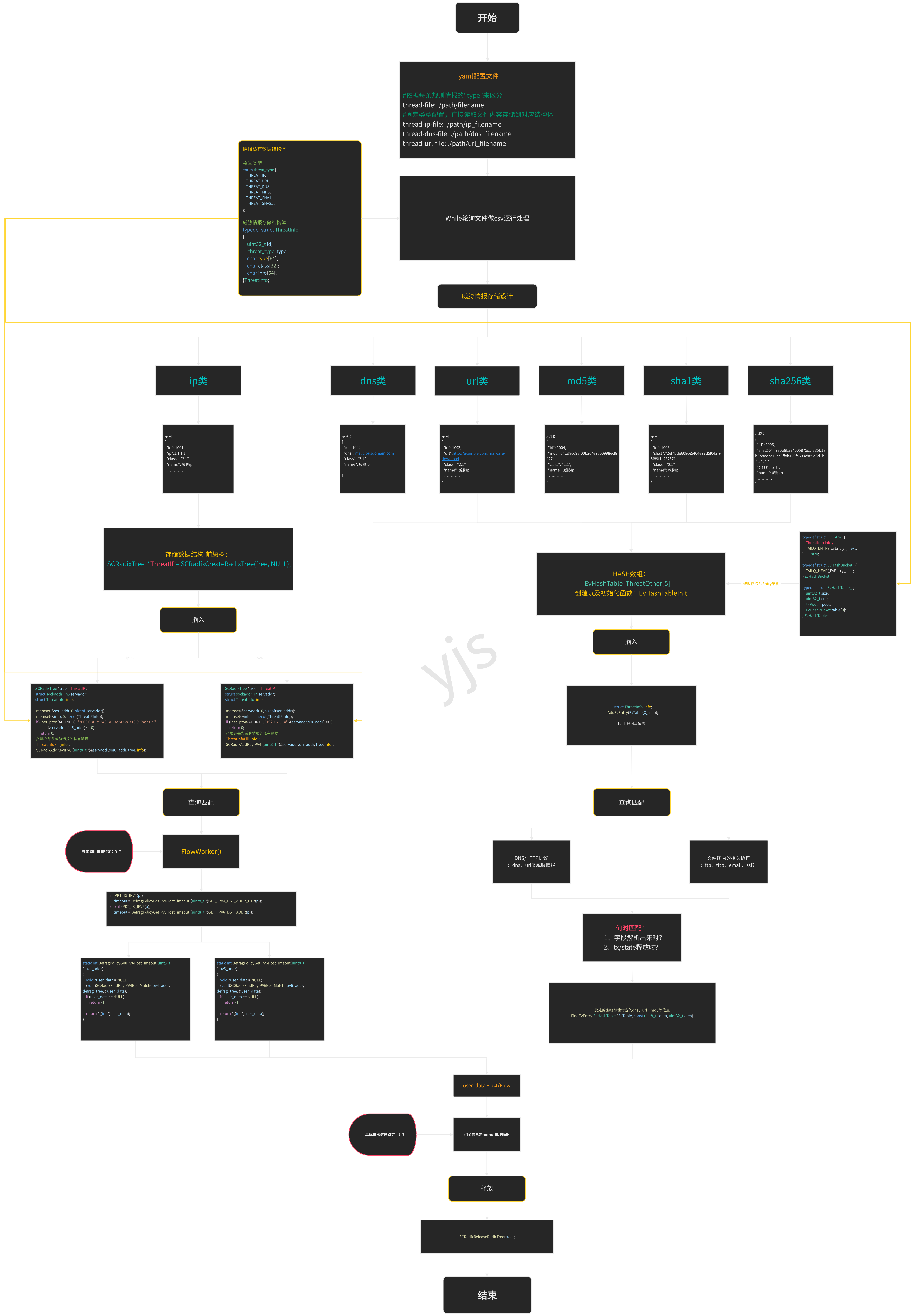

yaml配置文件

```
#依据每条规则情报的"type"来区分
thread-file: ./path/filename
#固定类型配置，直接读取文件内容存储到对应结构体
thread-ip-file: ./path/ip_filename
thread-dns-file: ./path/dns_filename
thread-url-file: ./path/url_filename
```

情报私有数据结构体

枚举类型

```
enum threat_type {
  THREAT_IP,
  THREAT_URL,
  THREAT_DNS,
  THREAT_MD5,
  THREAT_SHA1,
  THREAT_SHA256
};
```

威胁情报存储结构体

```
typedef struct ThreatInfo_
{
   uint32_t id;
    threat_type  type;
   char type[64];
   char class[32];
   char info[64];
}ThreatInfo;
```

While轮询文件做csv逐行处理

示例：

```
{
 "id": 1001,
 "ip":1.1.1.1
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

示例：

```
{
 "id": 1002,
 "dns": maliciousdomain.com
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

示例：

```
{
 "id": 1003,
 "url":http://example.com/malware/download
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

示例：

```
{
 "id": 1004,
 "md5":d41d8cd98f00b204e9800998ecf8427e
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

示例：

```
{
 "id": 1005,
 "sha1":"2ef7bde608ce5404e97d5f042f95f89f1c232871 "
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

示例：

```
{
 "id": 1006,
 "sha256":"9a0b8b3a4605875d5f385b18b8b8ed7c15ac8ff8b420fa599cb85d3d1b7fa4c4 "
 "class": "2.1",
 "name": 威胁ip
  ...........
}
```

存储数据结构-前缀树：

```
SCRadixTree  *ThreatIP= SCRadixCreateRadixTree(free, NULL);
```

```
SCRadixTree *tree = ThreatIP;
struct sockaddr_in6 servaddr;
struct ThreatInfo  info;

 memset(&servaddr, 0, sizeof(servaddr));
 memset(&info, 0, sizeof(ThreatIPInfo));
 if (inet_pton(AF_INET6, "2003:0BF1:5346:BDEA:7422:8713:9124:2315",
          &servaddr.sin6_addr) <= 0)
   return 0;
 // 填充每条威胁情报的私有数据
 ThreatInfoFill(info);
 SCRadixAddKeyIPV6((uint8_t *)&servaddr.sin6_addr, tree, info);
```

```
SCRadixTree *tree = ThreatIP;
struct sockaddr_in servaddr;
struct ThreatInfo  info;

 memset(&servaddr, 0, sizeof(servaddr));
 memset(&info, 0, sizeof(ThreatIPInfo));
 if (inet_pton(AF_INET, "192.167.1.4", &servaddr.sin_addr) <= 0)
   return 0;
 // 填充每条威胁情报的私有数据
 ThreatInfoFill(info);
 SCRadixAddKeyIPV4((uint8_t *)&servaddr.sin_addr, tree, info);
```

```
if (PKT_IS_IPV4(p))
   timeout = DefragPolicyGetIPv4HostTimeout((uint8_t *)GET_IPV4_DST_ADDR_PTR(p));
else if (PKT_IS_IPV6(p))
   timeout = DefragPolicyGetIPv6HostTimeout((uint8_t *)GET_IPV6_DST_ADDR(p));
```

```
static int DefragPolicyGetIPv4HostTimeout(uint8_t
*ipv4_addr)
{
   void *user_data = NULL;
   (void)SCRadixFindKeyIPV4BestMatch(ipv4_addr,
defrag_tree, &user_data);
   if (user_data == NULL)
      return -1;

   return *((int *)user_data);
}
```

```
static int DefragPolicyGetIPv6HostTimeout(uint8_t
*ipv6_addr)
{
   void *user_data = NULL;
   (void)SCRadixFindKeyIPV6BestMatch(ipv6_addr,
defrag_tree, &user_data);
   if (user_data == NULL)
      return -1;

   return *((int *)user_data);
}
```

HASH数组：

```
EvHashTable  ThreatOther[5];
```

创建以及初始化函数：EvHashTableInit

```
typedef struct EvEntry_ {
   ThreatInfo info;
   TAILQ_ENTRY(EvEntry_) next;
} EvEntry;

typedef struct EvHashBucket_ {
   TAILQ_HEAD(,EvEntry_) list;
} EvHashBucket;

typedef struct EvHashTable_ {
   uint32_t size;
   uint32_t cnt;
   YFPool   *pool;
   EvHashBucket table[0];
} EvHashTable;
```

```
struct ThreatInfo  info;
AddEvEntry(EvTable[X], info);
```

hash根据具体的

DNS/HTTP协议
：dns、url类威胁情报

文件还原的相关协议
：ftp、tftp、email、ssl?

何时匹配：
1、字段解析出来时?
2、tx/state释放时?

此处的data即使对应的dns、url、md5等信息

```
FindEvEntry(EvHashTable *EvTable, const uint8_t *data, uint32_t dlen)
```

```
SCRadixReleaseRadixTree(tree);
```

yjs