Transmission分析

\_\_\_\_transmission-daemon

目录

[一、Transmission整体框架 1](#_Toc442191518)

[二、 rpc框架分析 2](#_Toc442191519)

[2.1 注册rpc请求的函数调用顺序 2](#_Toc442191520)

[2.2 处理rpc请求的函数调用顺序 3](#_Toc442191521)

[2.3 对rpc请求方法的判断 3](#_Toc442191522)

[2.3.1 判断rpc请求的方法 3](#_Toc442191523)

[2.3.2 调用处理rpc请求的方法 4](#_Toc442191524)

[三、DHT框架分析 6](#_Toc442191525)

[3.1 注册dht事件的函数调用顺序 7](#_Toc442191526)

[3.2 transmission中DHT收发包的实现 8](#_Toc442191527)

[3.2.1对DHT/UDPtracker/utp包的处理函数 8](#_Toc442191528)

[3.2.2 transmission对DHT包的解析 9](#_Toc442191529)

[四、添加BT/磁力链下载过程分析 11](#_Toc442191530)

[五、Transmission DHT数据结构和算法 14](#_Toc442191531)

[5.1 DHT 网络中的四种包类型 14](#_Toc442191532)

[5.2 DHT网络中的数据结构分析 17](#_Toc442191533)

[5.2.1 基于Kademlia算法的DHT网络数据结构 17](#_Toc442191534)

[5.2.2 Transmission中的数据结构 19](#_Toc442191535)

[5.3 DHT网络距离的计算 20](#_Toc442191536)

[5.3.1 磁力链的infohash和种子文件infohash的区别 20](#_Toc442191537)

[5.3.2 如何通过DHT网络下载文件 20](#_Toc442191538)

# 一、Transmission整体框架

启动transmission并初始化配置

注册dht事件

注册rpc事件

监听事件

处理事件

结束

# 二、 rpc框架分析

Transmission对rpc请求的处理是基于libevent框架。首先通过调用evhttp\_set\_gencb()注册rpc事件的监听，然后等待rpc事件的发生，当有rpc

请求来后，首先解析出数据包的方法，然后就调用相应的回调函数进行数据的处理。

## 2.1 注册rpc请求的函数调用顺序

Main(Daemon.c)

->tr\_sessionInit();

->tr\_runInEventThread(session, tr\_sessionInitImpl, &data);

->tr\_sessionInitImpl()

->tr\_sessionSet()

->tr\_runInEventThread (session, sessionSetImpl, &data);

-> sessionSetImpl ()

->tr\_rpcInit ();

-> tr\_runInEventThread (session, startServer, s)

->startServer ()

-> evhttp\_set\_gencb (server->httpd, handle\_request, server);

完成http rpc请求的回调函数注册

## 2.2 处理rpc请求的函数调用顺序

Ps:只针对url为<http://127.0.0.1:9091/transmission/rpc>的rpc请求处理

-> handle\_request()

-> handle\_rpc (req, server);

->handle\_rpc\_from\_json()

->r\_rpc\_request\_exec\_json (..rpc\_response\_func, …);

->request\_exec() //处理rpc事件

->rpc\_response\_func() //完成rpc请求的回应

## 2.3 对rpc请求方法的判断

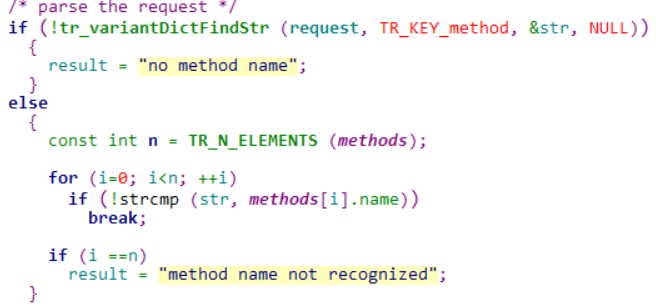
Ps:因为所有的rpc请求最后都会调用到request\_exec()函数，所以这里的分析就针 对此函数内的调用进行分析

### 2.3.1 判断rpc请求的方法

Rpc的方法列表是一个全局的结构体数组(图2-1)，在transmission得到rpc 请求后，会对比结构体数组里面的方法字符串(图2-2)，来调用对用的rpc请求处理 函数。



(图2-1，rpc请求方法列表)



比较方法的字符串

(图2-2，判断rpc请求方法)

### 2.3.2 调用处理rpc请求的方法

调用结构体数组里面对应的回调函数(图2-3)进行数据的处理，处理完后 调用rpc\_response\_func()回调函数返回结果给请求者(图2-4)**。**

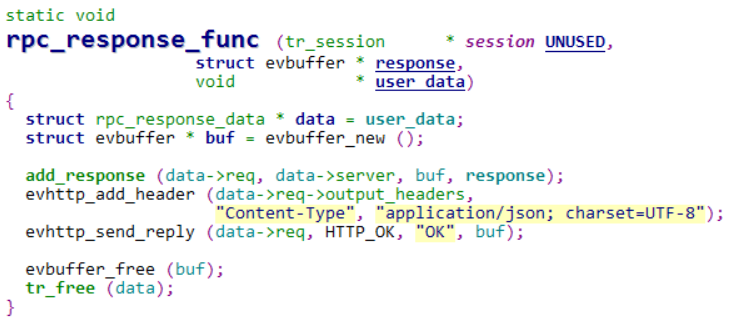
****

调用结数组里面的回调函数

调用结数组里面的回调函数

判断调用的方法是否需要快速返回

(图2-3)



回应rpc请求

(图2-4)

# 三、DHT框架分析

注册dht事件

监听事件

判断事件类型

处理UDP tracker包

处理DHT

处理UTP包

## 3.1 注册dht事件的函数调用顺序

Main(Daemon.c)

->tr\_sessionInit();

**->** tr\_runInEventThread (session, tr\_sessionInitImpl, &data);

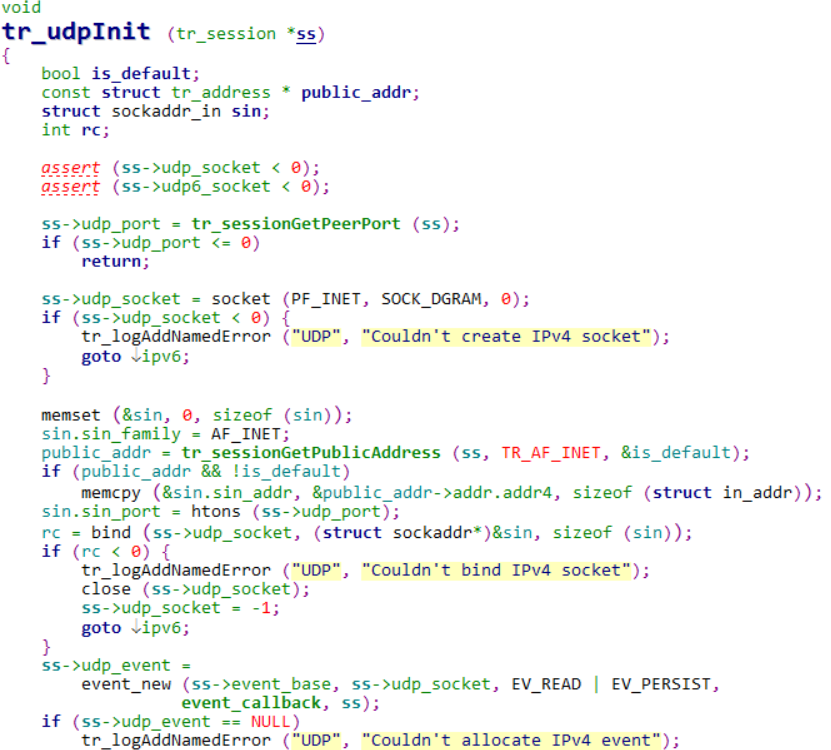
-> tr\_sessionInitImpl()

->tr\_udpInit()

-> event\_new (ss->event\_base, ss->udp\_socket, EV\_READ | EV\_PERSIST, event\_callback, ss)

-> event\_callback()//DHT事件的处理函数

关键源代码：



注册DHT事件的回调函数

## 3.2 transmission中DHT收发包的实现

## 3.2.1对DHT/UDPtracker/utp包的处理函数

Transmission支持的包格式有DHT、UDP tracker、utp。如果收到的buff里面的第一个字节是d,就进入DHT包处理函数；如果收到的包前4个字节是0~3的数，则进入UDPtracker包的处理流程，其他的就进入UTP包的流程(utp协议的支持可以通过配置文件设置)。

Transmission中通过libevent框架实现对DHT/UDPtracker/utp包的收发处理，在event\_callback函数中对包进行判断和处理(图3-1)。



处理DHT包

处理UDP tracker包

处理UTP包

(图3-1)

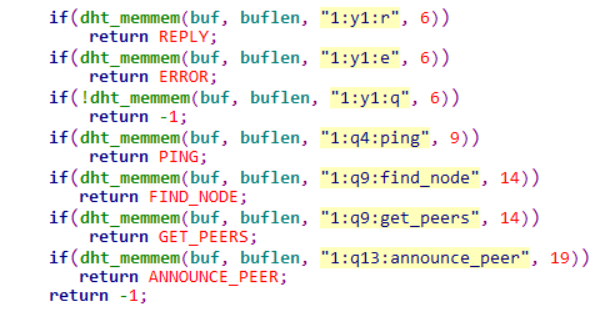
### 3.2.2 transmission对DHT包的解析

当触发了tr\_dhtCallback()回调函数后，就会调用dht\_periodic()函数进行DHT包解析并得到对应的参数和包类型。



解析DHT包

dht\_periodic()函数里面通过对B编码DHT包的判断得到对应的DHT包类型：

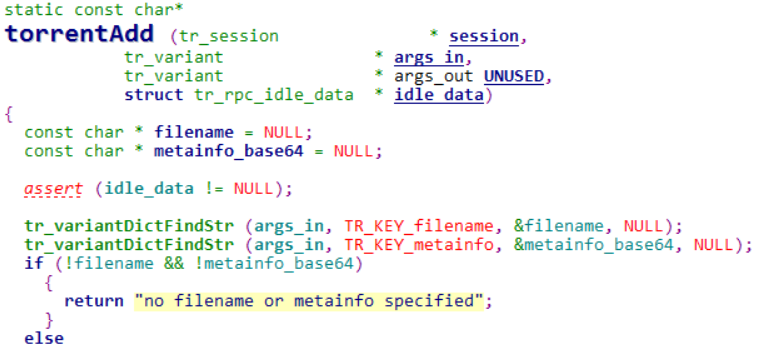


通过比较字符串得到对应的DHT包类型

# 四、添加BT/磁力链下载过程分析

当通过rpc请求，把需要下载的磁力链和torrent添加进transmission时，会触发rpc框架中的rpc事件回调函数--torrentADD()，这里就解析此函数的关键代码部分。

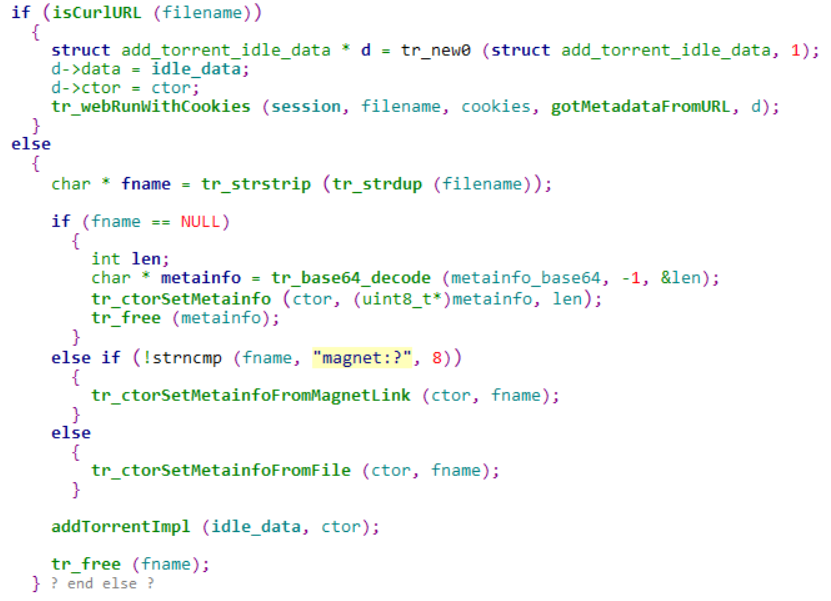
1. **判断参数是否正确**



解析出rpc请求数据中的文件名和metainfo

判断是否为http/https/ftp

**2.判断下载的文件类型**

****

参数为磁力链，设置torrent结构体参数

为BT下载时，对base64编码的torrent文件进行解码并设置torrent结构体参数

**3．添加新任务进下载链表中**



添加任务到下载的队列中

新建一个tr\_torrent结构的对象,并通过传入的文件hash值判断下载任务是否重复

# 五、Transmission DHT数据结构和算法

当一个已经加入DHT网络P2P客户端通过种子文件开始下载资源时，首先在网络中查询该资源的peer列表，这个过程通过get\_peer完成。当某个节点从get\_peer返回peer信息时，查询者便开始下载，然后通过announce\_peer告诉返回这个peer的节点。

Transmission中对DHT这四种包的实现放在Dht.c文件中。

## 5.1 DHT 网络中的四种包类型

所有的请求都包含一个关键字id 代表请求节点的在DHT网络中的ID。所有的回复也包含关键字id代表回复节点在DHT网络中的ID。

Ping：

最基础的请求就是ping。这时KPRC协议中的“q”=“ping”。Ping请求包含一个参数id，它是一个20字节的字符串包含了发送者网络字节序的nodeID。对应的ping回复也包含一个参数id，包含了回复者的nodeID。

参数: {"id" : "<querying nodes id>"}

回复:{"id" : "<queried nodes id>"}

报文包例子

ping请求={"t":"aa", "y":"q","q":"ping", "a":{"id":"abcdefghij0123456789"}}

B编码=d1:ad2:id20:abcdefghij0123456789e1:q4:ping1:t2:aa1:y1:qe

回复={"t":"aa", "y":"r", "r":{"id":"mnopqrstuvwxyz123456"}}

B编码=d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

Find\_node

Findnode被用来查找给定ID的node的联系信息。这时KPRC协议中的q=“find\_node”。find\_node请求包含2个参数，第一个参数是id，包含了请求node的nodeID。第二个参数是target，包含了请求者正在查找的node的nodeID。当一个node接收到了find\_node的请求，他应该给出对应的回复，回复中包含2个关键字id和nodes，nodes是一个字符串类型，包含了被请求节点的路由表中最接近目标node的K(8)个最接近的nodes的联系信息。

参数: {"id" : "<querying nodes id>","target" : "<id of target node>"}

回复:{"id" : "<queried nodes id>","nodes" : "<compact node info>"}

报文包例子

find\_node请求={"t":"aa", "y":"q","q":"find\_node", "a":{"id":"abcdefghij0123456789","target":"mnopqrstuvwxyz123456"}}

B编码 =d1:ad2:id20:abcdefghij01234567896:target20:mnopqrstuvwxyz123456e1:q9:find\_node1:t2: aa1:y1:qe

回复={"t":"aa", "y":"r", "r":{"id":"0123456789abcdefghij", "nodes":"def456..."}}

B编码=d1:rd2:id20:0123456789abcdefghij5:nodes9:def456...e1:t2:aa1:y1:re

Get\_peers

Getpeers与torrent文件的info\_hash有关。这时KPRC协议中的”q”=”get\_peers”。get\_peers请求包含2个参数。第一个参数是id，包含了请求node的nodeID。第二个参数是info\_hash，它代表torrent文件的infohash。如果被请求的节点有对应info\_hash的peers，他将返回一个关键字values,这是一个列表类型的字符串。每一个字符串包含了"CompactIP-address/portinfo"格式的peers信息。如果被请求的节点没有这个infohash的peers，那么他将返回关键字nodes，这个关键字包含了被请求节点的路由表中离info\_hash最近的K个nodes，使用"Compactnodeinfo"格式回复。在这两种情况下，关键字token都将被返回。token关键字在今后的annouce\_peer请求中必须要携带。Token是一个短的二进制字符串。

参数: {"id" : "<querying nodes id>","info\_hash" : "<20-byte infohash of targettorrent>"}

回复:{"id" : "<queried nodes id>","token" :"<opaque write token>","values" : ["<peer 1 info string>","<peer 2 info string>"]}

or:{"id" : "<queried nodes id>","token" :"<opaque write token>","nodes" : "<compact node info>"}

报文包例子

get\_peers请求={"t":"aa", "y":"q","q":"get\_peers", "a":{"id":"abcdefghij0123456789","info\_hash":"mnopqrstuvwxyz123456"}}

B编码 =d1:ad2:id20:abcdefghij01234567899:info\_hash20:mnopqrstuvwxyz123456e1:q9:get\_peers1 :t2:aa1:y1:qe

回复peers ={"t":"aa", "y":"r", "r":{"id":"abcdefghij0123456789", "token":"aoeusnth","values": ["axje.u", "idhtnm"]}}

B编码=d1:rd2:id20:abcdefghij01234567895:token8:aoeusnth6:valuesl6:axje.u6:idhtnmee1:t2:aa1:y1:re

回复最接近的nodes= {"t":"aa", "y":"r", "r":{"id":"abcdefghij0123456789", "token":"aoeusnth","nodes": "def456..."}}B编码 =d1:rd2:id20:abcdefghij01234567895:nodes9:def456...5:token8:aoeusnthe1:t2:aa1:y1:re

Announce\_peer

这个请求用来表明发出announce\_peer请求的node，正在某个端口下载torrent文件。announce\_peer包含4个参数。第一个参数是id，包含了请求node的nodeID；第二个参数是info\_hash，包含了torrent文件的infohash；第三个参数是port包含了整型的端口号，表明peer在哪个端口下载；第四个参数数是token，这是在之前的get\_peers请求中收到的回复中包含的。收到announce\_peer请求的node必须检查这个token与之前我们回复给这个节点get\_peers的token是否相同。如果相同，那么被请求的节点将记录发送announce\_peer节点的IP和请求中包含的port端口号在peer联系信息中对应的infohash下。

参数:

{"id": "<querying nodes id>", "info\_hash" :"<20-byte infohash of target torrent>", "port": <port number>, "token" : "<opaque token>"}

回复:

{"id": "<queried nodes id>"}

**报文包例子**

announce\_peers请求={"t":"aa", "y":"q","q":"announce\_peer", "a":{"id":"abcdefghij0123456789","info\_hash":"mnopqrstuvwxyz123456", "port":6881, "token": "aoeusnth"}}

B编码

=d1:ad2:id20:abcdefghij01234567899:info\_hash20:<br />

mnopqrstuvwxyz1234564:porti6881e5:token8:aoeusnthe1:q13:announce\_peer1:t2:aa1:y1:qe

回复={"t":"aa", "y":"r", "r":{"id":"mnopqrstuvwxyz123456"}}

B编码=d1:rd2:id20:mnopqrstuvwxyz123456e1:t2:aa1:y1:re

## 5.2 DHT网络中的数据结构分析

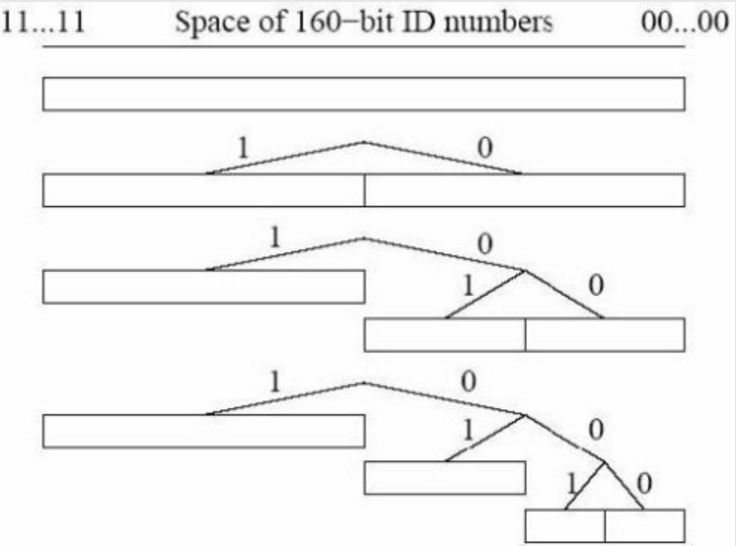
### 5.2.1 基于Kademlia算法的DHT网络数据结构

基于Kademlia算法的DHT网络的数据结构用类似于一种前缀树的结构的二叉树来组织路由表(DHT网络中节点的信息)。

最开始时的路由表只有一个K桶，当学习到新的节点信息后，会尝试把新节点的信息，根据其前缀值插入到对应的K桶中(图4-1)。

如果该K桶没有满，则新节点直接插入到这个K桶中；

如果该K桶已经满了：  
 ⑴如果该K桶覆盖范围包含了新节点的ID，则把该K桶分裂为两个大小相同的 新K桶，并对原K桶内的节点信息按照新的K桶前缀值进行重新分配  
 ⑵如果该K桶覆盖范围没有包含新节点的ID，则直接丢弃该新节点信息

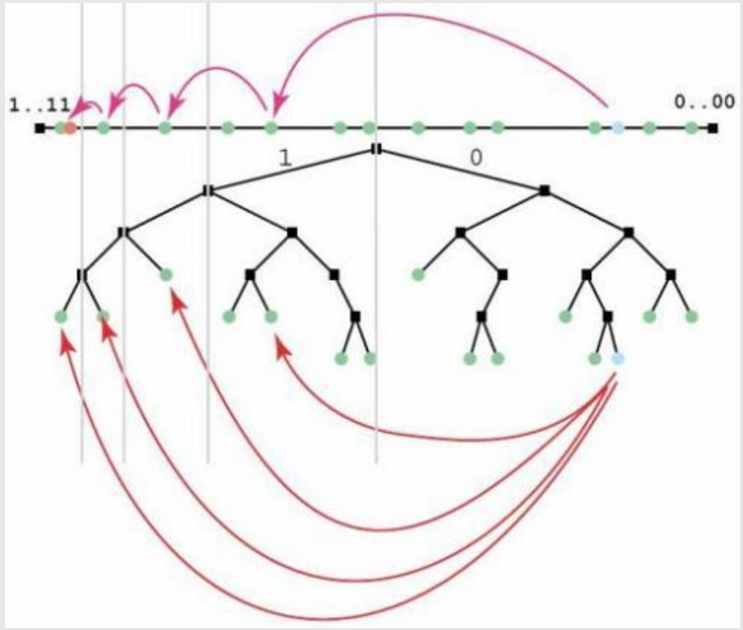
****

最开始时的K桶只有一个

当有相同前缀节点id加入并且K桶已满时，则分裂为两个K桶

图(4-1)

DHT网络中的节点



前缀树路由表，树的中间节点不保存信号，只在叶子节点上保存信息，此叶子节点通常被称为K桶，K桶保存的是具有相同id前缀的节点信息，此前缀就是节点在二叉树中的位置

### 5.2.2 Transmission中的数据结构

Transmission中对桶的组织是一个单向链表结构如下图：

一个桶所存储的节点范围是:b->first 到 b->next->first,插入新节点

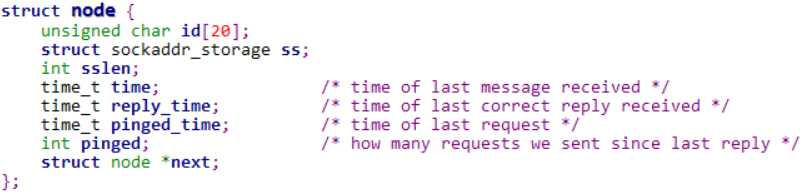
进入桶的时候，如果当前桶已经满了则会分裂成两个新桶。

Transmission中对桶的实现在dht.c文件中,其中的new\_node函数

负责新节点的加入和桶的分裂。

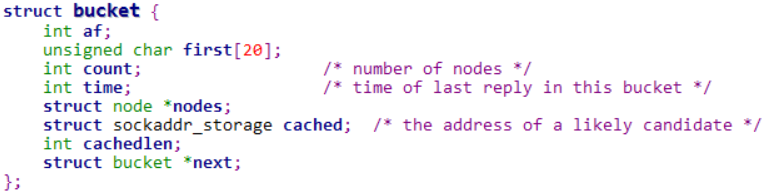
**bucket:**

**node:**



DHT网络中的节点id

一个桶中节点开始值



## 5.3 DHT网络距离的计算

### 5.3.1 磁力链的infohash和种子文件infohash的区别

种子文件的infohash是通过种子文件的文件描述信息动态计算得到。一个种子文件包含了对应资源的描述信息，例如文件名、文件大小等。

磁力链中包含的infohash一般为40字节或32字节，它只是资源infohash（20字节）的一种编码形式，编码方式为SHA1(安全哈希算法)。

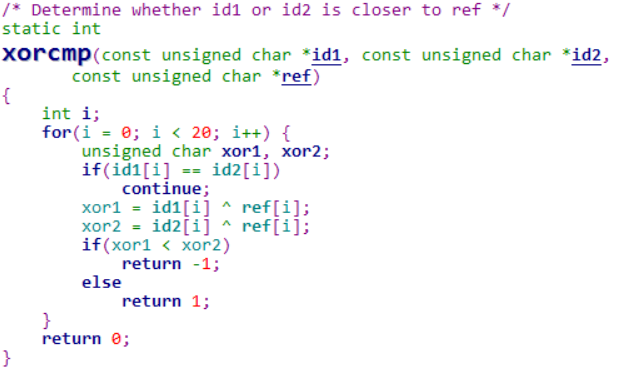
Transmission中对磁力链的解析函数为tr\_metainfoParseImpl。

### 5.3.2 如何通过DHT网络下载文件

在dht网络中，将路由表中的节点ID和infohash进行异或取值，此值越小说明距离越近。在异或的运算中，高位不同的值越多则以后出来的值越大。

当节点要为种子寻找peer时，它将自己路由表中的具有最大长度的相同前缀的节点ID和infohash进行异或。然后向路由表中离infohash最近的节点发送get\_peer请求,获取到正在下载此资源的peer信息。

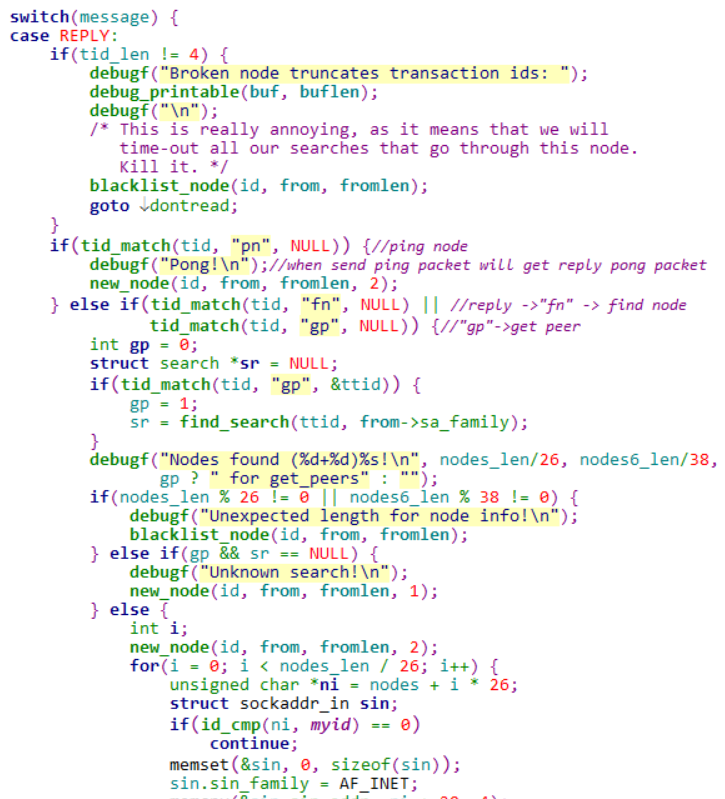
Transmission中Xorcmp函数来比较距离参数ref就是infohash



把节点id和infohash进行

异或运算

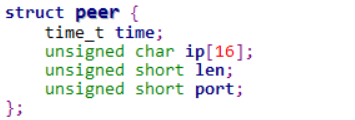
**Transmission中对get\_peer的实现在dht\_periodic函数中**

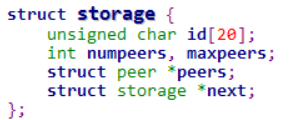
****

当从event事件中，得到了get\_peer请求的回应后，将回应参数里面的peer信息保存进全局的链表中。

**Transmission中对peer信息的保存**

用静态全局结构体链表保存正在下载资源的peer列表。

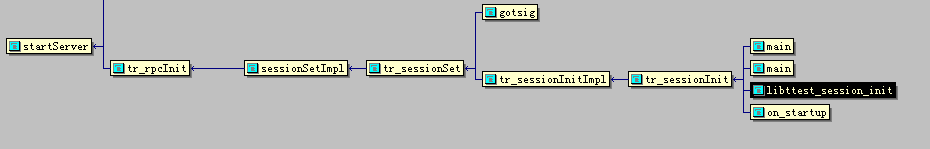


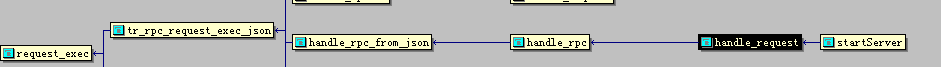
****

Torrent文件的infohash

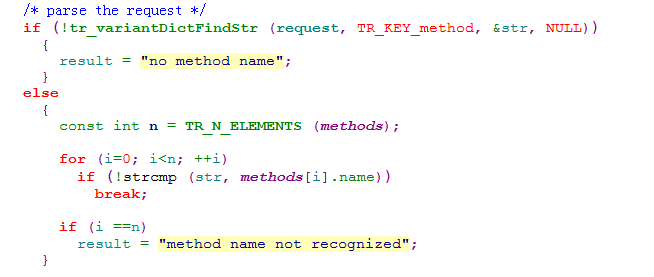
**Transmission源码分析：**

**1，rpc请求：**





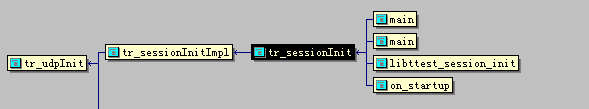
Request\_exec：

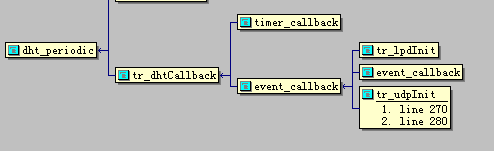






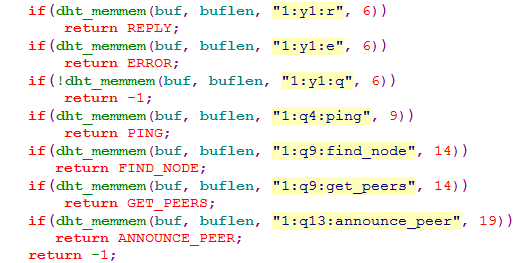
2，dht请求：





dht\_periodic:

parse\_message:



REPLY

PING

FIND\_NODE

GET\_PEERS

ANNOUNCE\_PEER