**版本历史**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 描述（修改原因） | 作者（修改者） |
| 0.1 | 2011/05/13 | 初稿 | 道远 |
| 0.2 | 2015/08/10 | 修改路由表数据接口 | 程刚 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 引言

## 编写目的

本说明书在概要设计的基础上，对链表模块的逻辑、功能、接口、子模块等分别给出了实现层面上的要求和说明。

软件开发小组的实现成员应该阅读和参考本说明进行代码的编写、测试。

## 项目背景

软件名称： MPDP

提出部门：网络传输部

开发部门：MPDP项目组

完成后的模块可为MPDP协议栈网络层的路由模块提供路由表的添加删除和查表服务。

## 缩写与术语

路由表：在网络路由协议中，通过目的地址查找下一跳地址的依据。

散列表/哈希表(Hash Table)：是根据[关键码值](http://zh.wikipedia.org/w/index.php?title=%E5%85%B3%E9%94%AE%E7%A0%81%E5%80%BC&action=edit&redlink=1)(Key value)而直接进行访问的[数据结构](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)。也就是说，它通过把关键码值[映射](http://zh.wikipedia.org/wiki/%E6%98%A0%E5%B0%84)到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做[散列函数](http://zh.wikipedia.org/wiki/%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0)，存放记录的数组叫做散列表。

散列值/哈希值冲突：不同的关键码值在通过散列函数变换后可能会得到相同的散列值/哈希值。

链表：将数据链式连接的数据结构，这里用于解决散列值冲突问题。

## 参考资料

互联网上的相关必需资料

# 总体设计

## 需求概述

路由表是共性平台中协议栈网络层路由协议模块的基础支撑模块。路由表需要提供如下操作：

发包查表：发包时，通过目的地址查询路由，表得到路由信息——下一跳地址、有效时间和跳数等参数。

路由信息变更：修改路由表中路由条目的某些数据。

添加路由条目：出现新目的节点时，将其路由信息加入路由表。

删除路由条目：目的节点移走或者失效时，在路由表中删除其路由条目。

## 软件结构

路由表模块由路由条目数据结构，路由散列表结构和路由表操作（访问、插入和删除）三部分组成。



图 2‑1 路由表结构和路由条目数据结构

路由条目的数据结构如图 2‑1所示，其中目标地址数据段作为散列表的关键码值(key)，它通过散列函数映射得到散列值，也就是在散列数组中的位置。

路由表结构为一维散列表，散列数组中存储的对象为链表控制头，连接具有相同散列值的路由条目。

路由表的操作包括路由条目的访问、插入和删除。路由表访问操作通过目的地址得到路由表条目的指针，通过它用户可以访问和修改路由条目。路由表插入操作将一个新路由条目插入路由表中。路由条目删除操作将一个已有路由条目从路由表中移除。

此路由表模块的设计具有查找速度快的优点。

## 模块清单

本模块为单一模块，即通用链表模块。

# 程序描述

## 数据结构与函数

路由条目数据类型定义：

typedef struct route\_entry

{

list\_head\_t list;

uint64\_t nui;

uint16\_t nwk\_addr;

uint16\_t next\_hop;

uint8\_t hop\_num;

uint8\_t live\_time;

} route\_entry\_t;

路由表伪类数据类型定义：

typedef struct route\_table

{

uint8\_t route\_entry\_cnt;

list\_head\_t route\_table\_head[RMT\_MAX\_ENTRY];

route\_entry\_t\* (\*alloc)(void);

route\_table\_error\_t (\*free)(route\_entry\_t \*del\_entry);

route\_table\_error\_t (\*insert)

(struct route\_table \*\_this, route\_entry\_t \*new\_entry);

route\_table\_error\_t (\*remove)

(struct route\_table \*\_this, route\_entry\_t \*old\_entry);

route\_entry\_t\* (\*get)

(struct route\_table \*\_this, addr\_t dest\_id);

} route\_table\_t;

路由表操作出错代码枚举：

typedef enum{

RHT\_SUCCESS,

RHT\_ERROR\_ENTRY\_EXIST,

RHT\_ERROR\_ENTRY\_NOT\_EXIST,  
 RHT\_ERROR\_INVALID\_INPUT,

} route\_table\_error\_t;

## 算法

本模块没有采用自定义算法。

## 程序逻辑



图 3‑1 路由表模块工作逻辑

## 接口

路由表管理模块对外提供五个外部接口，分别为路由表模块初始化、路由表模块反初始化、路由模块更新、添加路由条目和访问路由条目。

### 初始化路由表模块

/\*\*

\* @brief 网络路由模块初始化，输入路由表最大数量以及路由表节点存活周期

\* @param[in] route\_node\_num 路由表的最大容量

\* @param[in] live\_cnt 路由表节点存活最大周期（一次心跳作为一次周期）

\* @return 初始化是否成功

\* - 0 初始化成功

\* - -1 初始化失败

\*/

int8\_t nwk\_route\_module\_init(uint16\_t rote\_node\_num, uint8\_t live\_cnt);

### 路由表反初始化

/\*\*

\* @brief 网络路由模块反初始化，释放申请的资源

\* @return 释放成功或失败

\* - 0 反初始化成功

\* - -1 反初始化失败

\*/

int8\_t nwk\_route\_module\_deinit(void);

### 路由表更新

/\*\*

\* @brief 网络路由管理模块更新，把超过最大存活周期还没有刷新的节点剔除掉，把缓冲区资源释放出来

\*/

void nwk\_route\_module\_update(void);

### 向路由表添加路由条目*insert ( )*

/\*\*

\* @brief 网络路由管理模块节点插入，把节点以及到达节点的下一跳网络地址更新，如果

\* 节点已经存在则更新信息，如果节点不存在则插入一条新信息

\* @param[in] entry 指向路由节点信息结构体的指针

\* @return 更新成功或失败

\* - 0 节点路由链路更新成功

\* - -1 资源不够，节点路由链路无法更新进去

\*/

bool\_t nwk\_route\_modele\_insert(route\_entry\_t \*entry);

### 访问路由表中的某个条目

/\*\*

\* @brief 网络路由管理模块查找到某个地址的下一跳地址

\* @param[in] nwk\_addr 要查找的目的网络短地址

\* @return

\* - NULL 网络地址查找不到对应的路由信息

\* - !NULL 查找到了路由节点信息

\*/

route\_entry\_t \* nwk\_route\_modele\_get(uint16\_t nwk\_addr);

## 测试要点

无。