03. NS3 对象模型架构

通过前几天的学习，我真有点学不下去了

1. 对象模型是程序的切入点（Object Mode）

```
//对象模型的三大基类 (Object，ObjectBase，SimpleRefCount)

一个模拟场景的产生，有多个网络元素构成。如：节点、节点协议栈、分组、连接节点的信道等。每一个元素通过C++基类去表示：
    基类定义了网络元素的基本行为，在模拟中真正运行的是基类的子类对象。
    比如NetDevice作为父类，既是PointToPointNetDevice的继承，又是CsmaNetDevice的继承。
```

```
//ns3对象模型的作用在于此：
1.单一类的管理
    每个网络元素的类都不用，但是却有相同的特征： 如动态内存管理和属性配置。
    在每一个类中实现这些相同的需求是没有必要的，也是不现实的。
    因此需要一些顶层的基类来统一管理共性特征，而子类只需要关注自身的属性特征就好。
2. 对于多个类的管理
    任何一个单一的类都是无法完成网络的模拟的，这些类需要被关联起来。
    如何进行类之间类灵活高效的关联，是对象模型解决的问题之一。
```

```
对象模型的三个基本类： SimpleRefCount，ObjectBase，Object
几乎所有的网络元素都是这三个基类的子类

// SimpleRefCount 是解决单个类的动态内存管理问题。 SimpleRefCount定义了一个引用计数器，相当于C++的智能指针，去掌管类对象在堆中的分配。(第4章讲了为什么不用C++自带的智能指针)

// ObjectBase 类解决了属性配置和trace变量。ObjectBase定义了对于这些变量的配置方法和存储的数据结构。子类需要定义自己的属性和trace变量。

// Object类就是把多个类动态的关联起来。动态关联是通过对象的聚合。比如Node类中去定义网络元素的指针变量，然后通过调用函数进行聚合。但是这些对象的指针是放在动态指针数组中规划起来的。
```

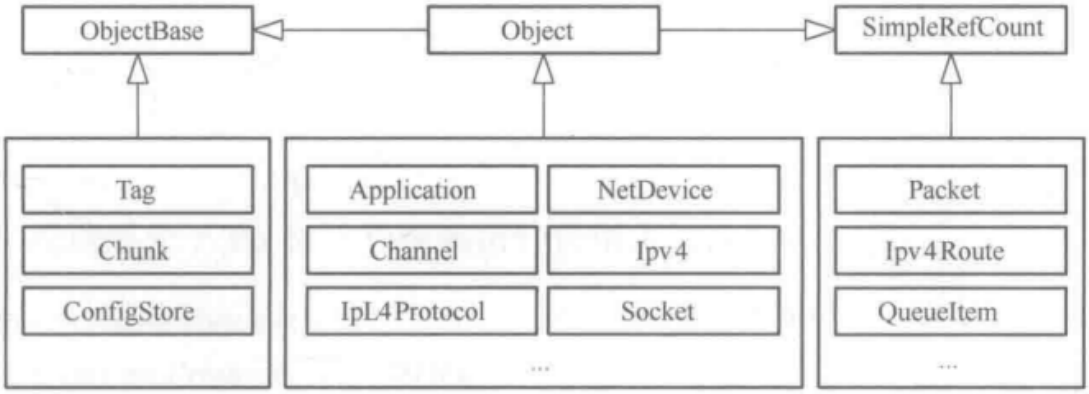


图 3-1　对象模型的基类继承关系

解释一下上面的图：
有些类并不要设置属性，比如Packet类，路由表Ipv4Route类，或者队列元素QueueItem类等等。
分组标签Tag类，分组头尾的Chunk类，和属性相关的ConfigStore类等等。

还有很多的类都不属于对象模型C++ 类，比如表示Ip地址的Ipv4Address，Ipv6Address，存储Node对象的NodeContainer等等，他们对网络起辅助作用。所有的助手类也不是对象模型的范畴。

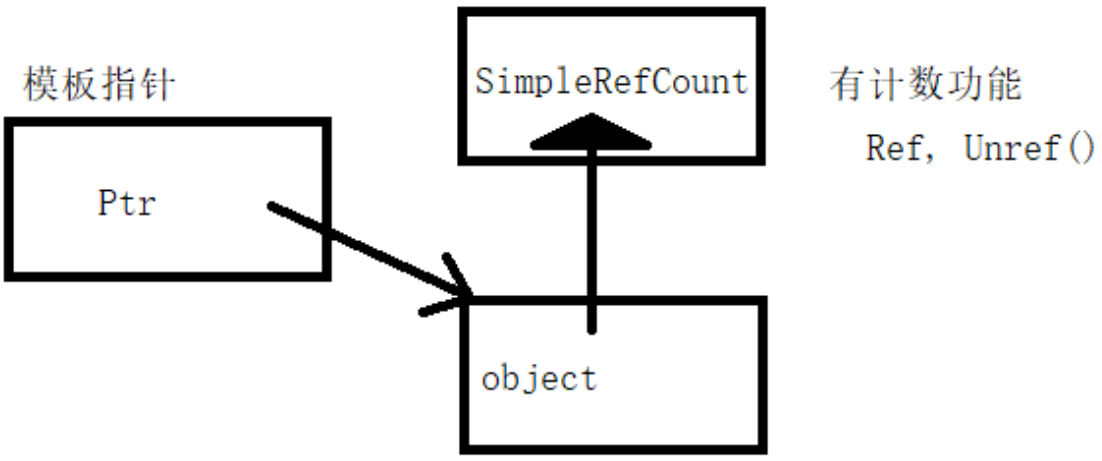
//在我看来，只有能实体化的才叫对象模型，暂时可以如此理解。

02. 智能指针ptr和SimpleRefCount

SimpleRefCount是ns3内部智能指针的实现，只要一个子类继承了SimpleRefCount类，那么就有了智能指针的功能。

4.1 设计原理

```
//原始的指针，开辟一个1000B大小的分组Packet对象。
Packet* p=new Packet(1000);
//Ptr的指针
Ptr<Packet> p=Create<Packet>(1000);

//可以把这两句话看成等同，要想这么写，那就必须是SimpleRefCount的子类，因此所有的Object类和子类都可以使用Ptr指针。
```



4.2 使用实例

```
1. 初始化
Packet* p=NULL;
Ptr<Packet> ptr; //不给ptr初始化就是NULL

2. 创建对象
Packet* p=new Packet();
Ptr<Packet> ptr=Create<Packet>();

3. 赋值操作
Packet* p=new Packet();
Ptr<Packet> ptr_1=p;  //等号右边既可以是原始指针
Ptr<Packet> ptr_2=ptr; //也可以是Ptr指针

4.指针的运算
ptr->getuid(); //和原始指针完全相同

5.比较运算(既可以比较原始指针和ptr，也可以比较两个ptr)
    if(ptr==ptr_1){};
    if(ptr!=ptr_1){};
    if(ptr==p){};
    if(ptr!=p){};

6.流插入
    std::cout<<"address:"<<ptr<<std::end;

7.拷贝(浅拷贝)
    Packet* p=new Packet();
    Ptr<Packet> ptr(p);
    Ptr<Packet> ptr_1(ptr); //每次拷贝对象的计数器+1

8. 对象拷贝: 深拷贝(Copy函数)
    Ptr<Packet> ptr=Create<Packet>();
    Ptr<Packet> ptr_1=Copy(ptr);
    创造一个新的对象，并用ptr指向。这种在ns3中使用的较少。

9.类型转换
```

```
    //提供了三个转化的指针:DynamicCast(), StaticConst(), ConstCast();
    //分别对应C++标准库中的dynamic_cast, static_cast, const_cast函数
    //上述三个ptr强制转换函数，主要用于对象模型那些非Object子类。对于Object子类来说，类型转换推荐使用GetObject()函数 (第六章)

    Ptr<NetDevice> pDev=Create<LoopbackNetDevice>(); //父类指子类
    Ptr<LoopbackNetDevice> pLoopDev=DynamicCast<LoopbackNetDevice>(pDev); //指针强制类型转换
    if(!pLoopDev){ //如果转化不成功
        NS_LOG_UNCOND("DynamicCast failture");
    }

10. 获取原始指针
    1. PeekPointer(); //对象计数器不变
    2. GetPointer();  //创建一个新的指针指向对象，然后对象的计数器+1，需要用户手动Unref;
    Ptr<Packet> ptr=Create<Packet>();
    Packet* p=Peekpointer(ptr);
```

4.3 适用范围

```
Ptr的局限:
1. 对象的构造函数参数少于等于7个类。如果超过7个，则需要用户实现新的Create()函数。
2. 必须是SimpleRefCount的子类。像有些只继承于ObjectBase的，只能用C++标准库里的运算符，例如dynamic_cast, new 和 delete。
```

03. 对象模型的基石： 元信息，ObjectBase，Object

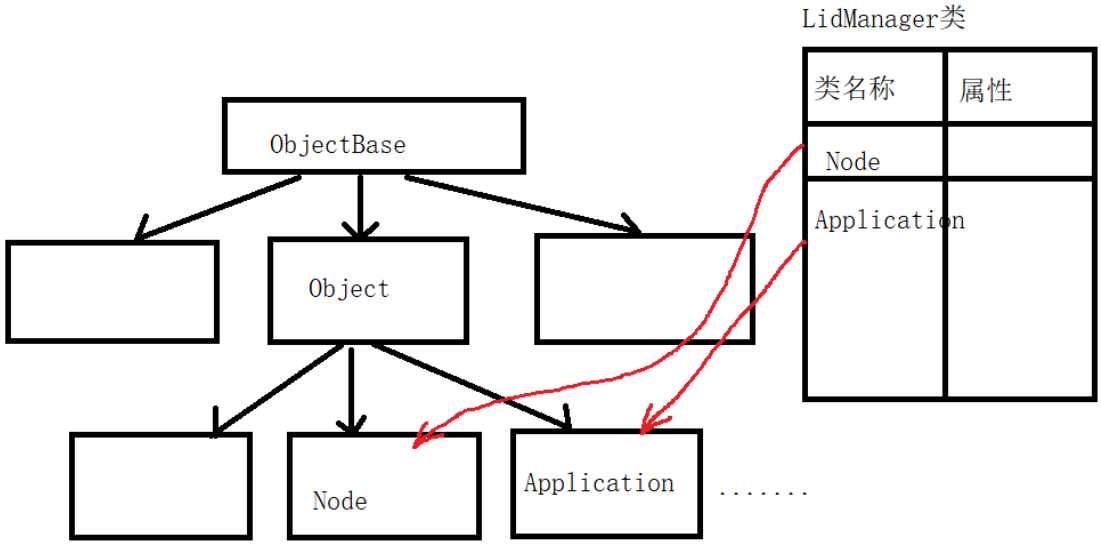
上面提到过Object 用于多个类之间的动态关联，ObjectBase用于单个类的属性和trace变量配置。那么这两个基类有一个共同实现的基础：元信息

3.1 什么是元信息

```
//元信息 (以类为单位):
 元信息就是ObjectBase及其子类的辅助信息。每一个子类都有且只有唯一一组属于自己的元信息，就像每一个类的编号一样。子类的元信息被集中存储在一个数据结构中，并以类名字作为唯一的标识符。

//元信息的种类:
 类名字: C++类的名字，一个类名有且只能有一组元信息
 类的构造函数: 对用户屏蔽类的构造函数的具体细节
 父类TypeID: 用于类聚合中的类查找，即Object::GetObject()函数。
 属性类型: 该类的所有属性的辅助信息
 trace信息: 该类的所有trace变量的辅助信息。

// 元信息是实现对象聚合、属性、trace变量配置的基础。对象聚合数组利用元信息中的类名称和父类TypeID查找对象。
```

3.2 元信息存储：lidManager 类

```
//我们需要一个完美的容器，把ObjectBase类和其所有子类的元信息存储起来，并且以类的名字去查找关键字。lidManager类就是这样的一个类，用于管理vector容器。在ns-3模拟过程中有且仅有一个实例（单例模式）。因为一个类的元信息有且只有一组。

lidManager管理的向量容器，查找速度通过建立索引把时间复杂度从O(n)降到O(1).

lidManager对象一个静态的变量，在编译期间已经被初始化，并在整个模拟运行期间是不会被改变的。向量容器中的元信息条目=ObjectBase子类总数量+1 （1是Objective本身）
```



3.3 元信息管理接口： TypeId类

```
//lidManager为每一个objectBase类做了一个索引，方便进行查找

这个索引值并不是以成员变量的形式存储在ObjectBase中，而是保存在TypeId的类中。TypeId是元信息管理的接口。因为除了保存索引，还定义了ObjectBase和LidManager之间的一些交互函数接口，比如：元信息查找函数等等。
```

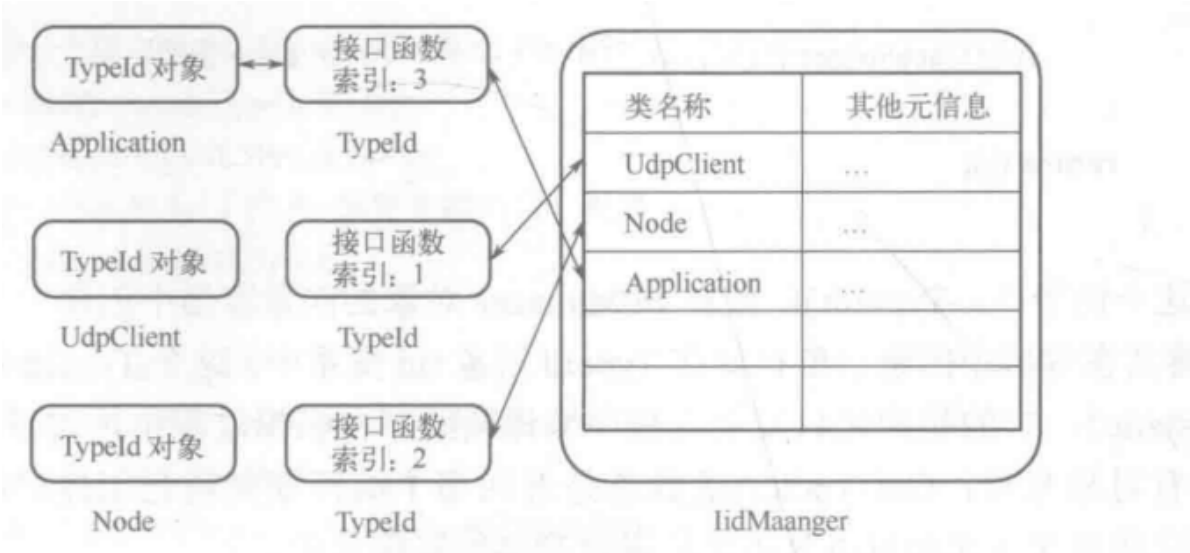


图 5-2　基于 TypeId 的元信息存储结构

Notes：怎么说呢，这个设计就非常的玄幻了。
对于每一个类，都有一个静态的TypeId对象。也就是说这个类的所有对象，都共用这一个对象，TypeId可以用于和lidManager交互，增加元信息，或者增加trace源。

3.4 TypeId的使用

3.4.1 如何为一个类创建一个TypeID

比如说，已经有Application类，然后我想写一个新的基于Application的子类。
首先，写一个UdpEchoClient去继承Application

```
class UdpEchoClient:public Application
{
 public:
    static TypeId GetTypeID(void); //其中有一个GetTypeID() 函数，目的就是创造一个关于UdpEchoClient的元信息，然后将这个元信息插入到LidManager类中。这个实在编译的时候就执行的。
}

//让我们看看具体的UdpEchoClient
TypeId
UdpEchoClient::GetTypeId(void){  //返回的是TypeID
    //这几个函数其实是连在一起的，但是为了加注释我就分割开了!!
    //如果没有的话就创建，如果有的话表示获取一个类的TypeId对象。
    static TypeId tid=TypeId("ns3::UdpEchoClient")  //创建一个tid然后放到里的Manager类的向量容器中，然后将容器的索引值存储在typeID的tid变量中。
        .SetParent<Application>() //添加信息:父类Application类
        .SetGroupName("Application") //添加信息: Application类
        .AddConstructor<UdpEchoClient>() //添加信息:添加UdpEchoClient的构造函数
        .AddAttribute("MaxPacket",.....) //添加信息: 添加属性，最大包是多少
        .AddTraceSource("Tx",.....); //添加信息: 添加trace
    return tid;
}
```

3.4.2 TypeID的运算符

```
TypeID对象内部封装了索引值，因此TypeID重载了一些运算符来模拟索引值的运算。
1. 赋值运算符：将索引值由右侧对象赋值给左侧对象。
2. 比较运算符：比较索引值的大小。索引值仅代表一个元信息在IidManager中向量容器存储的位置，单纯比较大小其实意义并不大。
3. 流插入<<输出当前索引值指向的元信息的类名。
```

3.4.3 获取TypeID

```
ObjectBase::GetTypeId(); //获取一个ObjectBase子类的TypeId对象，上面也说了，这样就是获取Node类的TypeId对象。GetTypeID函数是一个静态成员函数，被一个类的所有对象共享，因此既可以通过Node调用，也可以通过类名调用。

node->GetTypeId();
Node::GetTypeId();

//获取索引值
node->GetTypeId().GetUid();

//获取类名
node->GetTypeId().GetName();

//通过类名去查找TypeId
TypeId tid=TypeId::LookupByName("ns3::Node");
```

3.4.4 GetInstanceTypeID() 和 GetTypeId()

```
//getInstance() 表示的是对象创建时候的TypeId，并且这个对象在创建完成后不会改变。
//getTypeId() 是会改变的

比如：
Ptr<Ipv4L3Protocol> ipv4L3Prot= CreateObject<Ipv4L3Protocol>();
Ptr<Ipv4> ipv4=ipv4L3Prot; //接口的对接：父类指针指向子类的对象

所以我调用
ipv4->GetTypeId(); //打印ns3::Ipv4类的TypeID
ipv4->GetInstanceTypeId(); //打印的是ns3::Ipv4L3Protocol的TypeID
```

04. Object类：对象聚类

Object是大部分元素类的基类，它继承了SimpleRefCount类和ObjectBase类，实现了对象聚合功能。我们介绍Object类的集大成者：Node类。

这个时候我们该介绍一下Node类

Node类是网络节点的实现载体，从应用层到物理层的所有算法都必须依托Node类才能实现。同样的，单纯的Node对象是无法处理和传递分组的，必须关联到各个网络层协议才能称为一个可以使用的通信节点。

一个ns3-Node可以采用TCP/IP参考模型去搭建：

1. 物理层:Channel
2. 数据链路层: NetDevice
3. 网络层: Ipv4 和 Ipv6
4. 传输层: IpL4Protocol
5. 应用层: Application

//1. 获取节点的一个应用层对象: Node::GetApplication()函数

在Node对象中，存储的应用层对象是存储在vector中的

```
Ptr<Application> app=node->GetApplication(i); //表示获取node节点中的第i个对象
```

如果想调取每一个对象就要用到循环:

```
for(uint32_t i=0;i<node->GetNApplications();i++){
    Ptr<Application> app=node->GetApplication(i);
}
```

//2. 获取node的传输层对象（在聚合数组中）

传输层对象也存在一个数组中，数组元素的获取函数是GetObject()。调用的时候只需要指定类名即可。

```
Ptr<TcpL4Protocol> tcp=node->GetObject<TcpL4Protocol>(); //获取到node的传输层对象
```

这个数组在ns-3中叫做对象聚合数组，也就是说，这些存储的都是对象(一个节点中只能出现一次的，比如，一个Node中只能有一个TCP对象或者UDP对象)。但是一个node中可以有很多的Application。因此Application不能用数据聚合数组存储。

//3. 网络层对象的获取（在聚合数组中）

```
Ptr<Ipv4> ipv4=node->GetObject<Ipv4>();
Ptr<Ipv6> ipv6=node->GetObject<Ipv6>();
```

Notes:

这个里提供了获取IP地址的方法，一个设备的Ipv4地址保存在其对应接口的Ipv4Interface对象中。（一个Node有很多接口，每个接口又有很多的IP地址）

```
for(uist32_t ifNum=0; ifNum<ipv4->GetNInterfaces();ifNum++){ //有几个接口
    for(uint32_t addrNum=0;addrNum<ipv4->GetNAddress(ifNum);addrNum++){
        Ipb4InterfaceAddress address=ipv4->GetAddress(ifNum,addrNum);
    }
}
```

通过GetObject()方式获取的对象还有很多，比如路由协议、移动节点的移动模型等等。这些都是唯一的，并且存储在聚合数组中。

//4. 获取链路层对象

一个节点可以有多个NetDevice，就和Application一样，Node使用向量容器存储NetDevice对象。

```
for(uint32_t i=0;i<node->GetNDevices();i++){
    Ptr<NetDevice> dev=node->GetDevice(i);
    Address addr=dev->GetAddress(); //获取MAC地址
}
```

//5.获取物理层对象

可以通过网络层设备去获取channel对象:

```
Ptr<Channel>  channel=dev->GetChannel();
```

```
//6. 各个协议层的对象去获得Node对象，GetNode();
   Ptr<Node> ptr=dev->GetNode();
   脚本每个节点都有唯一的ID，可以通过NodeGetID()函数去获取。
```

4.1 对象聚合

我们知道Node类中，有定义其他协议层核心对象的指针。对象聚合的作用就是统一定义这些对象指针的存储、配置和提取方式：

```
传输层和网络层对象统一以Object基类指针的形式存储在Node对象的一个指针数组中。这个数组就是对象聚合所使用的指针数组，也较聚合数组。Node安装的大部分协议对象指针都存储在这个聚合数组中。当然也有例外，比如应用层(Application)和链路层(NetDevice)对象就分别存储在两个指针向量容器(std::vector)中。
  那为什么Application 和 NetDevice 也是 Object的子类，为什么不能存储在聚合数组中呢？
```

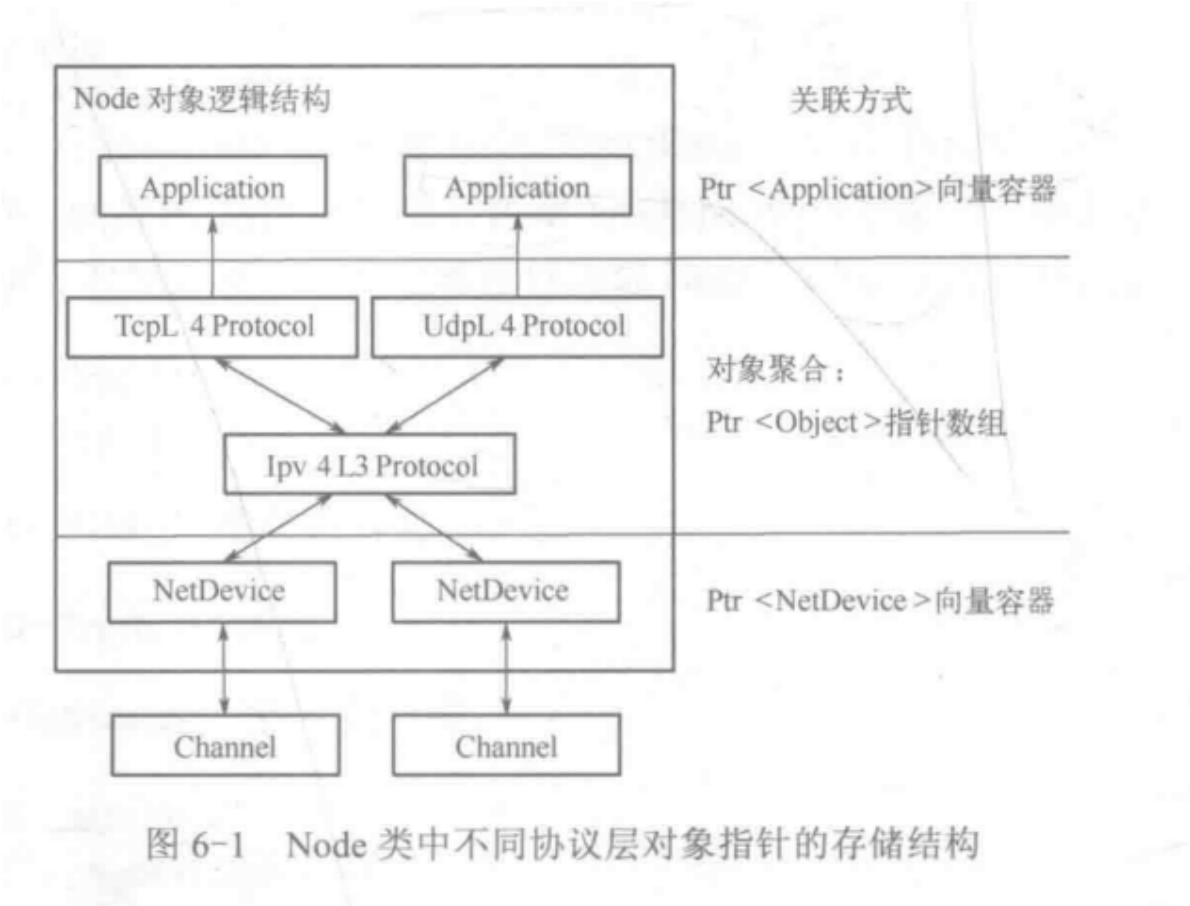


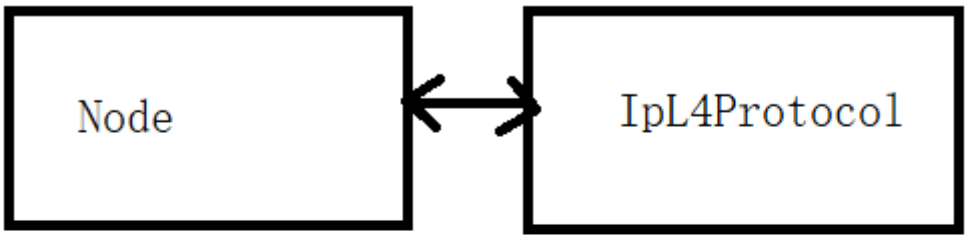
图 6-1　Node 类中不同协议层对象指针的存储结构

4.1.1. 对象聚合的技术原理

对于ns3用户来说，对象聚合在脚本中最直接的体现就是简化了关联对象的获取。例如： 获取一个Node的传输层协议的对象，可以直接用传输层协议名字去获取：

```
Ptr<IpL4Protocol> ipL4Proto=node->GetObject<IpL4Protocol>();
Ptr<Node> node=ipL4Proto->GetObject<Node>();
```

4.1.2 这个技术是如何做到的呢？？



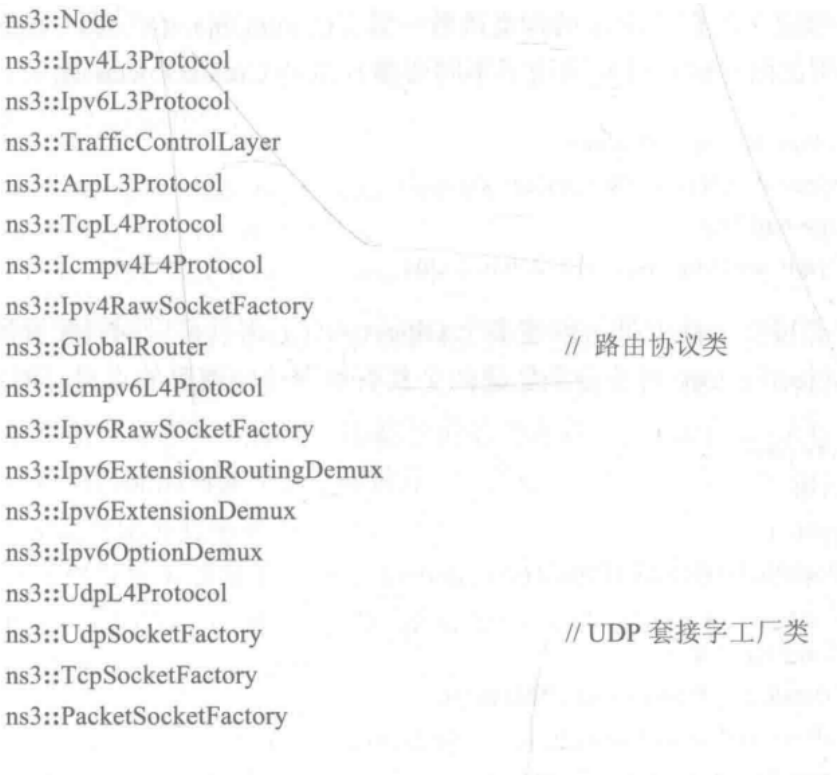
传统的聚合方法就是把IpL4Protocol的指针放在Node类中

这样做的结果是啥呢？就是太多的指针需要管理，而且这些指针都要写set的get函数，就很麻烦。

对象聚合是如何做的呢？

```
class Object:public SimpleRefCount<Object,ObjectBase,ObjectDeleter>{
public:
    void AggregateObject(Ptr<Object> other); //set函数
    template<typename T>
    inline Ptr<T> GetObject(void) const; //get函数

    private:
      struct Aggregates{
        uint32_t n;    //聚合数组的元素数量
        Object* buffer[1];  //聚合函数的起始指针
      };
      struct Aggregates* m_aggregate; //聚合数组
}
//总是就是在每个Object中放一个Object*组成的指针数组，这样就可以把所有的Object指针类型存储在聚合数组中。
//并且通过遍历聚合数组去寻找具体的Object，提取方法GetObject() 函数。这就证明了一个类在聚合数组中只能有一个，因为类名称是聚合数组元素的唯一标识。
//AggregateObject() 函数就是把Object子类对象放到Object的聚合数组中，并且还会进行聚合的同步。[我猜测这里是个递归]。往往都是通过调用Helper实现聚合同步
例如： InternetStackHelper::Install()函数就实现了所有协议栈对象的聚合

//我们知道聚合数组是一个Object指针组成的数组，那么如何访问这个数组的元素呢？
    Object::AggregateIterator iter=
        nodes.Get(0)->GetAggregateIterator();
    while(iter.HasNext()){
        Ptr<const Object> obj=iter.Next();
        NS_LOG_UNCOND(obj->GetInstanceTypeId().GetName());
    }
```

我们发现除了网络层和传输层的协议，还关联了路由协议、套接字等多个网络元素。



Notes: 聚合的使用范围

1.只能用于一个Node的一个对象实例，像Application 和 NetDevice就不行，就要用特有的容器去进行聚合。

4.2 object的创建与获取

Object对象的创建方法有两种：CreateObject()函数 和 ObjectFactory 类。

```
//CreateObject() 函数用来创造一个Object类的对象,最多可以7个参数
Ptr<PointToPointNetDevice> ppp=CreateObject<PointToPointNetDevice>();
```

```
//ObjectFactory类可以一次批量创建多个Object对象，而且可以同时配置属性。

ObjectFactory pppF;
//配置类名称
pppF.SetTypeId(
    PointToPointNetDevice::GetTypeId()
);

//配置属性并创建对象1
pppF.Set("DataRate",StringValue("5Mbps"));
Ptr<PointToPointNetDevice> ppp1=
    pppF.Create<PointToPointNetDevice>();

//配置属性并创建对象2
pppF.Set("DataRate",StringValue("2.5Mbps"));
Ptr<PointToPointNetDevice> ppp2=
    pppF.Create<PointToPointNetDevice>();
```

```
Notes: 第一步，创建一个ObjectFactory对象
       第二步，通过SetTypeId() 和 Set() 成员函数设置所创建对象的类名称和属性值。
       第三步，通过ObjectFactor::Create()成员函数创建对象，多次调用，多次创建。
```

2. 获取Object

获取Object是从一个对象的聚合数组中获取指定对象的指针，使用的函数是Object::GetObject()。

1. 使用场景

```
Ptr<Ipv4> ipv4=node->GetObject<Ipv4>(); //获取Ipv4的指针
Ptr<Ipv4L3Protocol> ipv4=node->GetObject<Ipv4L3Protocol>(); //上下行转换，从基类到子类的转换。
//如果类查找或者上下行转换失败，则GetObject()就会返回一个nullptr
//GetObject是上下行转化最便捷和安全的方式。也可以使用Dynamic_Cast()函数
```