1. **Probe机制**
2. Probe是一个探测类，用来数据收集，其原理是类的内部含有一个TraceSink函数，用来绑定到需要收集的类对象的Trace变量。并且设置了通过对象绑定和通过路劲绑定两种绑定方式。
3. Probe的子类（Uinteger16Probe……）内部还设置了一个新的trace变量，当TraceSink函数被调用，会更新这个trace变量到一个新的值，或者调用这个trace变量，即trace变量有可能是一个回调。
4. Trace机制：
5. ns3中的很多类中含有TracedCallBack回调，当对应的事件发生变化时，就会调用这个回调，所以我们的任务是绑定一个回调给这个TracedCallBack变量。
6. 自定义类时，如果需要trace某个值，可以将该值设置位TracedValue类，并将该值注册就行。
7. 绑定回调有几种方式：

* Config::Connect()函数

Config::Connect(string path, string context, CallBack);

Config::ConnectWithoutContext(string path, string context, CallBack);

* ObjectBase.Connect()函数

Obj.TraceConnect(string tracesource, string context, CallBack);

Obj.TraceConnectWithoutContext(string tracesource, CallBack);

* 通过Probe机制

Probe.ConnectByObject(string tracesource, Ptr<object> obj); 这个会调用obj.TraceConnectWithoutContext(..)函数。

Probe.ConnectByPath(string path);这个会调用Config::ConnectWithoutContext(..)函数。

3 name机制

1. Names::Add(string name, Ptr<object> obj); 将obj添加到名字name中去
2. Names::Add(string path, string name, Ptr<object> obj);将obj添加到name中去，

4 FileAggregator类用来生产数据到文件中， FileHelper是对FileAggregator帮助类，集成了Probe机制和FileAggregator机制，即将Probe采集的数据写到文件中去。

5 PlotAggregator类是用来将数据生成可以画图的文件，PlotHelper是帮助类，连接了Probe机制和PlotAggregator机制，即用Probe来收集数据，然后写进文件，用来画图。

6. Plot是画图类，PlotHelper是其帮助类。

7. Queue机制：

* **QueueSize**类：表示却列的大小，成员：大小的单位（包、比特），大小的值

Uint32\_t GetValue()：获得大小的值。

* **QueueItem**类：表示队列的成员类型，成员是一个Packet指针。

Ptr<Packet> GetPacket()：获得Packet对象指针。

Uint32\_t GetSize()：获得packet的大小。

void Print(std::ostream &os)：打印packet。

* **QueueDiscItem**类（抽象且继承自QueueSize类）：表示队列的成员，内容为一个Packet指针，Pakcet的目的MAC地址，网络层协议
* **QueueLimits**抽象基类：表示队列的长度
  + void Completed(uint32\_t count) : 记录已经完成的字节数并重新计算长度
  + void Queued(uint32\_t count):可以发送给NetDevice的字节数。
* **QueueBase**抽象基类：表示一个Packet的队列，如下属性即成员：
  + 队列中的字节数，也是个trace变量‘
  + 队列中的Packet数，也是个trace变量
  + 总的收到的packet数
  + 总的收到的字节数
  + 总的丢弃的包数
  + 总的丢弃的字节数
  + 入队前总的丢弃的字节数
  + 入队后总的丢弃的字节数
  + 入队前总的丢弃的包数
  + 入队后总的丢弃的包数。
  + 队列的大小。
  + 入队前被丢弃是因为队列放不下这个包了，只能丢弃

方法：

AppendItemTypeIfNotPresent(string &typeId, const string &itemType):参数分别数队列的类型，队列元素的类型。这个方法可以被链路模型的helper类调用，如

PointToPointHelper::SetQueue(“ns3::DropTailQueue”);

* **Queue**模板类（继承自QueueBase）：成员：
  + 一个队列元素的链表
  + 队列元素的const和非const迭代器
  + 包入队回调
  + 包出队回调
  + 包丢弃的回调
  + 包入队前被丢弃的回调
  + 包出队后被丢弃的回调

方法:

1. Void Enqueue(Ptr<Item> item) = 0 : 将元素入队，每个自类自定义入队位置，并计算入队的个数
2. Ptr<Item> Dequeue() = 0; 出队，每个自类自定义出队位置，并计算出队的个数
3. Ptr<Item> Remove() = 0; 移除一个元素，自类自定义位置，并计算丢弃的个数。
4. Ptr<Item> Peek() = 0; 得到某个元素的拷贝，不移除，自类自定义位置
5. Void Flush(); 刷新队列
6. 两种迭代器对应的begin（）和end（）方法
7. Bool DoEnqueue(ConstIterator pos, Ptr<Item> item);将包入队，并更新成员值，调用对应的回调。
8. Ptr<Item> DoDequeue(ConstIterator pos);将pos位置的元素出队。
9. Ptr<Item> DoRemove（ConstITerator pos）；将pos位置的元素移除，调用出队和丢包两个回调。
10. Ptr<Item> DoPeek(pos):返回pos位置的元素指针
11. Void DropBeforeEnqueue(Ptr<Item> item):将元素在入队前丢弃，更新相关值，调用回调
12. Void DropAfterDequeue(Ptr<Item> item);将元素出队后丢弃，更新相关值，调用回调。

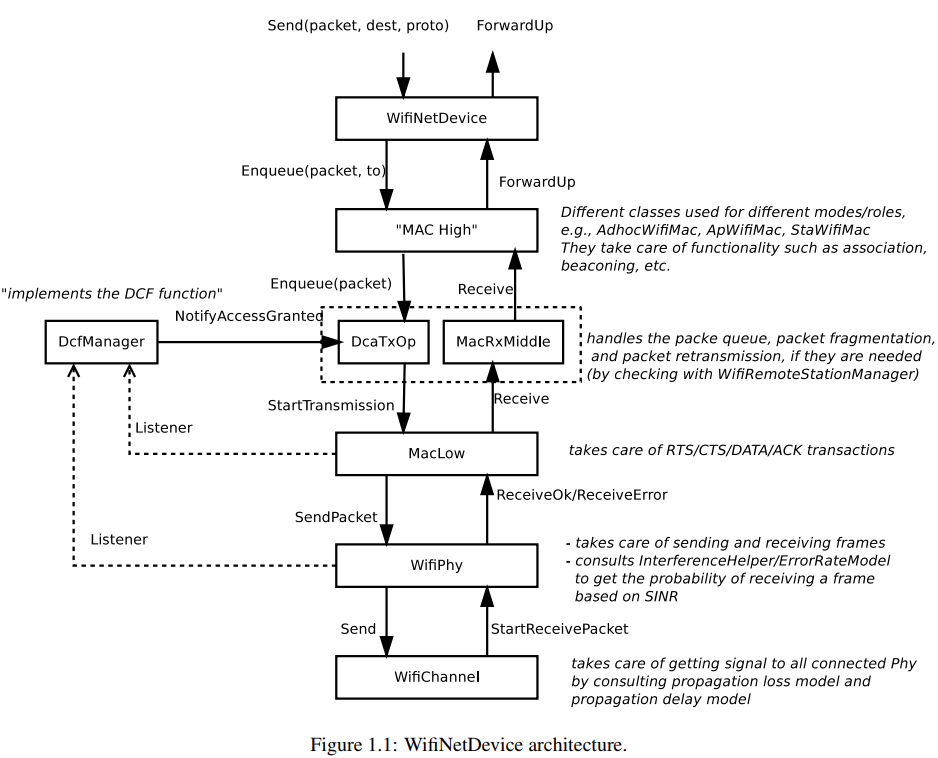
**Wifi模块**

RegularWifiMac的单个属性：QosSupported, HtSupported, VhtSupported,对应802.11e,802.11n,802.11ac.

MacLow:负责处理RTS/CTS/DATA/ACK传输

MacMiddle：负责处理packet的queue，和packet重传。

MacHigh:负责beacon/association帧的功能。



WifiPhy:

* 存在多个对应不同调制方式和标准的错误模型。
* 存在一个持续跟踪所有接受信号的对象，因此能正确计算每个包的干扰能量
* 接收到的每个数据包都要进行概率评估（查找错误模型），以确定接收是否成功。这种可能性取决于调制、分组的信噪比(和干扰比)以及物理层的状态(例如，在传输或休眠时，接收是不可能的)

WIfiPhy:

给每个连接的WifiPhy发送信号，通过查询传播损耗模型和传播时延模型。

从WifiPhy发过老的包带有一个信号能量，然后根据传播模型发送给每个关联的WifiPhy，其信号能量是计算后的能量。