Guia Rápido de Utilização do ns-3

Renê Cardozo
rene.cardozo@usp.br

O primeiro passo para qualquer simulação no ns-3 é a definição de nós de conexão, armazenando-os em uma instância da classe NodeContainer. Esta instância pode gerar nós automaticamente através da função Create(n_nos) a qual cria um número n de nós. Outra função possível é passar dois ou mais objetos NodeContainer para a função Create, gerando um container concatenado destes.

Por serem objetos básicos, as instâncias da classe Node possuem características muito ilustrativas do modelo de software adotado no simulador.

1 Objetos

Objetos no ns-3 podem ser criados tal qual em C++, contudo, para um melhor aproveitamento da memória e adição de funcionalidades, o ns-3 implementa seus próprios métodos de criação de objectos.

Há três classes principais no ns-3, das quais derivam grande parte dos objetos encontrados no simulador:

- Object
- ObjectBase

- SimpleRefCount

Apesar de não obrigatório, a herança destas classes dá propriedades especiais aos objetos. As classes que derivam de Object possuem acesso ao sistema de atributos e tipos do ns-3, bem como o sisistema de agregação de objetos e contagem de referência por smart-pointers.

1.1 Contagem de Referência

A contagem de referência é utilizada para facilitar o gerenciamento de memória. Todas as vezes que um ponteiro é criado, uma chamada é realizada para a função Ref() e a contagem de referência do objeto é incrementada. O usuário deve então aplicar a função Unref() ao ponteiro. Assim, quando a contagem de referência chegar a zero, o ponteiro pode ser eliminado.

A classe Ptr<> age de maneira similar a classe intrusive_ptr da biblioteca Boost, assumindo que a classe utilizada em seu template implementa os métodos Ref() e Unref().

Para a criação dos nós a classe utiliza a função CreateObject<>(), a qual é tambem é central para o ns-3. Objetos derivados da classe Object() nunca devem ser criados com a função new do C++. É necessário utilizar a função CreateObject para realizar a contagem de referências de maneira correta.

Se o objeto não derivar de nenhum objeto do ns-3, mas é de desejo que suas referências sejam contadas, utiliza-se a função Create<>(), como por exemplo, para a criação de pacotes.

1.2 Sistema de Agregação

No ns-2, muitos protocolos eram estendidos utilizando herança e polimorfismo, tal como diferentes versões do TCP. Para tornar este tipo de

utilização mais fácil, o ns-3 usa um design pattern que evita eventuais problemas.

A classe Node é um bom exemplo de utilização do sistema de agregação. Não existem classes derivadas da classe Node, apesar de sua utilização em centenas de combinações de protocolos. Os protocolos devem ser agregados ao nó. Por exemplo:

```
static void
AddIpv4Stack(Ptr<Node> node) {
    Ptr<Ipv4L3Protocol> ipv4 = CreateObject<Ipv4L3Protocol>();
    ipv4->SetNode(node);
    node->AggregateObject(ipv4);
    Ptr<Ipv4Impl> ipv4Impl = CreateObject<Ipv4Impl>();
    ipv4Impl->SetIpv4(ipv4);
    node->AggregateObject(ipv4Impl);
}
```

Apenas um objeto de cada tipo pode ser agregado a uma instância.

As partes do protocolo são agregadas ao objeto. Assim, se uma aplicação necessitar da implementação deste protocolo, pode requerir em runtime através do método GetObject<>();. Se o objeto não possuir o atributo especificado no template de GetObject<>(); a função retornará null.

1.3 Fábricas de Objetos

Uma maneira comum de criar diversos objetos iguais é utilizando a classe ObjectFactory, a qual permite a configuração prévia destes por meio de:

```
void SetTypeId(TypeId tid);
void Set(std::string name, const AttributeValue &value);
Ptr<T> Create (void) const;
```

O primeiro método especifica o tipo do objeto criado, o segundo

especifica atributos para os objetos criados, e o terceiro permite a criações dos objetos em si. Por exemplo:

```
ObjectFactory factory;
factory.SetTypeId("ns3::FriisPropagationLossModel");
factory.Set("SystemLoss", DoubleValue(2.0));
Ptr<Object> object = factory.Crate();
factory.Set("SystemLoss", DoubleValue(3.0));
Ptr<Object> object = factory.Create();
```

2 Configuração

No ns-3 temos dois principais aspectos de configuração: a topologia da simulação e como os objetos estão conectados; e os valores utilizados pelos modelos nesta topologia.

Muitos objetos do ns-3 derivam da classe Object, a qual possui propriedades de organização do sistema e melhora no gerenciamento de memória.

Dentro da classe são armazenados metadados sobre: a classe base, os construtores da classe, os atributos da classee se estes são read-only ou read-write, e ainda os valores permitidos para cada atributo.

Objetos que herdam a classe Object deve ser alocados na heap utilizando o método CreateObject<>(). Alguns helpers podem derivar de ObjectBase(), os quais são alocados na stack.

2.1 TypeId

As classes derivadas de Object podem possuir um TypeId que é utilizado para o sistema de agregação e gerenciamento de componentes. Esse identificador é uma string única identificando a classe, sua classe base, o conjunto de construtores da classe e uma lista dos atributos públicos desta.

O header da classe node inclui uma função estática GetTypeId(). Definida da forma:

```
TypeId Node::GetTypeId(void) {
    statis TypeId tid = TypeId("ns3::Node")
    .SetParent<Object>()
    .SetGroupName("Network")
    .AddConstructor<Node>()
    .AddAttribute("DeviceList",
                  "The list of devices associated to this Node."
                  ObjectVectorValue(),
                  MakeObjectVectorAccessor(&Node::m\_devices),
                  MakeObjectVectorChecker<NetDevice>())
    .AddAttribute("ApplicationList",
                  "The list of applications associated to this N
                  ObjectVectorValue(),
                  MakeObjectVectorAccessor(&Node::m\_application
                  MakeObjectVectorChecker<Application>())
    .AddAttribute("Id",
                  "The id (unique integer) of this Node.",
                  TypeId:ATTR\_GET,
                  UintegerValue(0),
                  MakeUintegerAccessor(&Node::m\_id),
                  MakeUintegerChecker<uint32\_t>());

    return tid;
}
```

A função SetParent<Object>() é utilizada para permitir casting e mecanismos de agragação. Também herda os atributos da classe pai, Object.

O método AddConstructor<Node>() é utilizada é utilizado em conjunto com a fábrica de objetos, permitindo a construção de objetos

mesmo que o usuário não tenha conhecimeto exato sobre o construtor da classe.

Por fim, AddAttribute() associa uma string com um valor fortemente tipado da classe. Deve-se ainda informar uma string de ajuda para a linha de comando. Cada atributo é associado com mecanismos para acessá-lo em uma variável da classe. O Checker é utilizado para validar os valores em questão de tipos e de máximos e mínimos.

Pode-se criar nós através de:

```
Ptr<Node> n = CreateObject<Node>();
```

ou, por meio de:

```
ObjectFactory factory;
const std::string typeId = "ns3::Node";
factory.SetTypeId(typeId);
Ptr<Object> node = factory.Create<Object>();
```

No caso da inicialização por meio da fábrica de objetos, não há conhecimento da classe concreta do nó.

2.2 Atributos

No sistema de atributos do ns-3, se quisermos prover um valor padrão para o atributo de uma classe ou acessar o valor em uma classe já instanciada, temos que consultar o objeto TypeId da classe.

A função AddAttribute() no método GetTypeId() de cada classe realiza diversas ações sobre um atributo. Liga sua variável privada a uma string pública, indica o valor da variável, provê um texto de ajuda com o significado de seu valor, atribui um "Checker"que será utilizado para avaliar a correta tipagem a intervalo da variável.

A especificação destas características pela função AddAttribute em GetTypeId torna os atributos seus valores padrão acessíveis no espaço de nomes dos atributos, o qual é baseado em strings que definem cada um dos atributos.

2.3 Argumentos da Linha de Comando

Pode-se acrescentar a leitura de valores da linha de comando por meio da classe CommandLine e seu método AddValue, o qual recebe uma string que indica o nome o atributo, uma mensagem de ajuda e a variável que armazenará o valor.

Após determinar quais valores devem ser lidos, utiliza-se o método Parse() para que as variáveis sejam devidamente atualizadas.

Hello