

O que é a notação de sintaxe abstrata um (ASN.1)?

8 de maio de 2024

[Glossário](#) » [A](#) » O que é a notação de sintaxe abstrata um (ASN.1)?

Abstract Syntax Notation One (ASN.1) é uma notação padronizada usada para definir estruturas de dados que podem ser serializadas e desserializadas em diferentes plataformas. Ele fornece uma estrutura para descrever dados de forma independente de plataforma, tornando-o crucial para protocolos de comunicação.



O que é ASN.1?

Abstract Syntax Notation One (ASN.1) é uma notação padronizada desenvolvida pela União Internacional de Telecomunicações (ITU-T) para especificar estruturas de dados de maneira independente de plataforma. Desempenha um papel crítico em telecomunicações, redes e protocolos de segurança, oferecendo uma linguagem formal para descrever protocolos complexos. [estruturas de dados](#). ASN.1 separa a sintaxe dos dados da codificação, permitindo que os dados sejam transmitidos e processados perfeitamente em vários sistemas.

Suas regras de codificação, como Regras Básicas de Codificação (BER), Regras de Codificação Distintas (DER) e Regras de Codificação Embaladas (PER), garantem que os dados serializados sejam transmitidos de forma eficiente, mantendo a fidelidade estrutural. ASN.1 define não apenas os tipos e valores dos dados, mas também como essas estruturas de dados se relacionam entre si, tornando-o indispensável em áreas onde é necessária uma troca de dados precisa e interoperável. É especialmente útil em segurança [aplicações](#), certificados digitais e padrões de telecomunicações onde a consistência no tratamento de dados é crucial.

Exemplo de sintaxe ASN.1

Aqui está um exemplo de sintaxe ASN.1 que demonstra uma estrutura de mensagem simples para uma pessoa com nome e idade:

```
Person DEFINITIONS ::= BEGIN
PersonRecord ::= SEQUENCE {
name UTF8String,
age INTEGER
}
END
```

Neste exemplo, o módulo **Pessoa** contém uma estrutura chamada **Registro de Pessoa** que é definido como um **SEQÜÊNCIA**. Esta

sequência inclui dois campos:

1. **nome**, que é um **String UTF8**.
2. **idade**, que é um **INTEGER**.

Exemplo de codificação ASN.1

Suponha que queiramos definir uma estrutura para uma pessoa com dois campos: **nome** (uma string) e **idade** (um número inteiro). Isso seria escrito em ASN.1 da seguinte forma:

```
Person DEFINITIONS ::= BEGIN PersonRecord ::= SEQUENCE { name UTF8String, age INTEGER } END
```

Exemplo de dados

Suponha que queremos representar uma pessoa chamada "Alice", que tem 30 anos.

Exemplo de codificação

Se usarmos BER (Regras Básicas de Codificação) para codificação, os dados de "Alice, 30" seriam serializados da seguinte forma:

1. **Identificador de Sequência: 0x30** (indicando uma SEQUÊNCIA)
2. **Comprimento: 0x0C** (12 bytes para toda a sequência)
3. **Identificador UTF8String: 0x0C** (indicando UTF8String)
4. **Comprimento do nome: 0x05** (5 bytes para o nome "Alice")
5. **Valor do nome: 0x41 0x6C 0x69 0x63 0x65** ("Alice" codificado em UTF-8)
6. **Identificador INTEIRO: 0x02** (indicando INTEIRO)
7. **Comprimento da idade: 0x01** (1 byte para o número inteiro)
8. **Valor da idade: 0x1E** (30 em hexadecimal)

Assim, os dados codificados em BER ficariam assim:

```
30 0C 0C 05 41 6C 69 63 65 02 01 1E
```

Esta representação hexadecimal corresponde à estrutura de dados definida em ASN.1, mostrando como ela pode ser utilizada para trocar dados entre sistemas de forma consistente.

Tipos e valores ASN.1

ASN.1 (Abstract Syntax Notation One) fornece um conjunto abrangente de tipos de dados e valores para representar várias estruturas de forma padronizada. Esses tipos de dados garantem uma comunicação eficaz em diferentes plataformas. Aqui está uma análise de alguns tipos e valores comuns de ASN.1:

INTEGER

Um tipo INTEGER representa valores inteiros (positivos e negativos). Pode ser de qualquer tamanho e é frequentemente usado para identificadores numéricos ou contagens.

Exemplo:

```
age INTEGER ::= 30
```

BOOLEAN

O Tipo **BOOLEAN** representa valores lógicos verdadeiros ou falsos. Normalmente é usado para sinalizadores e decisões binárias.

Exemplo:

```
isActive BOOLEAN ::= TRUE
```

CADEIA DE OCTETOS

Uma OCTET STRING contém uma sequência de octetos ([bytes](#)), comumente usado para dados binários como imagens, [arquivos](#) ou chaves criptográficas.

Exemplo:

```
rawData OCTET STRING ::= 'E04FD020EA3A6910A2D808002B30309D'H
```

NULL

O tipo NULL significa a ausência de um valor. Frequentemente usado como espaço reservado em campos opcionais.

Exemplo:

```
nothing NULL ::= NULL
```

IDENTIFICADOR DE OBJETO

Um IDENTIFICADOR DE OBJETO (OID) é um identificador globalmente exclusivo usado para fazer referência a padrões, protocolos ou objetos específicos.

Exemplo:

```
myObjectIdentifier OBJECT IDENTIFIER ::= { iso member-body us(1) ansi-x9-57(100) }
```

ENUMERADO

É semelhante a um INTEGER, mas restrito a uma lista predefinida de valores. O tipo ENUMERADO é útil para representar opções ou estados.

Exemplo:

```
status ENUMERATED { active(0), inactive(1), pending(2) }
```

String UTF8

Este é um tipo de string codificado em UTF-8. É ideal para representar dados textuais contendo caracteres especiais ou alfabetos não latinos.

Exemplo:

```
fullName UTF8String ::= "Alice Smith"
```

SEQUÊNCIA

Uma SEQUÊNCIA é uma coleção de diferentes campos de dados, agindo como uma estrutura ou registro na programação. Os campos podem ser obrigatórios ou opcionais.

Exemplo:

```
PersonRecord ::= SEQUENCE { name UTF8String, age INTEGER, address UTF8String OPTIONAL }
```

SEQUÊNCIA DE

Uma SEQUÊNCIA DE é uma coleção de elementos do mesmo tipo, semelhante a um array.

Exemplo:

```
PhoneNumbers ::= SEQUENCE OF UTF8String
```

SET

É como uma SEQUÊNCIA, mas seus campos não precisam aparecer em uma ordem específica.

Exemplo:

```
Identity SET { firstName UTF8String, lastName UTF8String }
```

Macros ASN.1

No ASN.1, foram introduzidas macros para simplificar a especificação de certas estruturas, definindo modelos reutilizáveis para estruturas de dados. Embora as macros tenham sido amplamente substituídas por outros mecanismos em revisões posteriores do ASN.1, compreender o seu uso histórico é importante para trabalhar com especificações mais antigas.

As macros desempenharam uma função semelhante às funções ou modelos em [linguagens de programação](#). Eles permitiram que os usuários definissem um padrão reutilizável que poderia ser expandido com detalhes específicos sempre que necessário, reduzindo assim a redundância e simplificando definições complexas.

Sintaxe macro

Uma macro é definida usando a seguinte estrutura geral:

```
MyMacro MACRO ::= BEGIN -- Macro definition content goes here END
```

Exemplo de uma macro

Um exemplo simples poderia envolver a definição de uma macro que cria um **TaggedType**:

```
TaggedType MACRO ::= BEGIN TYPE NOTATION ::= Type VALUE NOTATION ::= value INTEGER END
```

Aqui, **NOTAÇÃO DE TIPO** e a **NOTAÇÃO DE VALOR** são espaços reservados que são preenchidos quando a macro é usada.

Com a evolução do ASN.1, as macros foram substituídas por outras mais poderosas e flexconstruções acessíveis, como tipos parametrizados e classes de objetos de informação. Os tipos parametrizados fornecem uma maneira de especificar modelos que podem ser reutilizados com diferentes tipos de dados, enquanto as classes de objetos de informação permitem que restrições e relacionamentos sejam especificados de forma mais explícita.

Regras de codificação ASN.1

As regras de codificação ASN.1 definem como as estruturas de dados descritas em ASN.1 devem ser serializadas e desserializadas em um formato binário adequado para transmissão ou armazenamento. Cada regra de codificação fornece diferentes níveis de eficiência e aplicabilidade com base em requisitos específicos, como tamanho, velocidade de processamento ou rigor.

Aqui está uma lista e explicação de algumas regras de codificação comuns em ASN.1:

- **Regras básicas de codificação (BER).** BER é um flexregra de codificação compatível que usa um formato TLV (type-length-value) para codificação. Cada elemento começa com um identificador para indicar seu tipo, seguido do comprimento do valor e, por fim, do próprio valor. BER permite codificação de comprimento indefinido, o que fornece flexibilidade, mas pode resultar em dados codificados maiores em comparação com outras regras.
- **Regras de codificação distintas (DER).** DER é um subconjunto de BER que garante codificação consistente, aplicando regras específicas, como usar a codificação de menor comprimento possível. Ele fornece representações binárias inequívocas de estruturas de dados. O DER é amplamente utilizado em protocolos de segurança e certificados digitais porque garante que a mesma estrutura ASN.1 codifique sempre de forma idêntica.
- **Regras de codificação canônica (CER).** CER também é um subconjunto de BER projetado para codificação inequívoca. É semelhante ao DER, mas permite codificação de comprimento indefinido de alguns tipos, o que é útil para grandes conjuntos de dados. O CER é ideal para aplicações de streaming onde as estruturas de dados podem não caber em um comprimento predefinido.
- **Regras de codificação compactada (PER).** PER foi projetado para ser altamente eficiente, removendo metadados desnecessários. Ele otimiza a codificação usando o número mínimo de bits necessários para cada campo, geralmente com base em intervalos ou restrições predefinidas. PER produz tamanhos de dados significativamente menores em comparação com BER, DER e CER, tornando-o adequado para aplicações sensíveis à largura de banda, como comunicações móveis.
- **Regras de codificação XML (XER).** XER mapeia estruturas ASN.1 para uma representação XML, tornando-a legível por humanos. O objetivo é melhorar a interoperabilidade aproveitando a popularidade do XML para troca de dados. O XER é particularmente útil quando os dados precisam ser revisados ou modificados por humanos ou integrados em fluxos de trabalho existentes baseados em XML.

Regras de codificação JSON (JER). JER mapeia estruturas ASN.1 para uma representação JSON, tornando-as legíveis e utilizáveis em [Aplicativos da web](#). Ele aproveita a popularidade do JSON em serviços da web. JER é útil para sistemas modernos baseados na Web que dependem de JSON para intercâmbio e armazenamento de dados.



Anastasia Spasojevic

Anastazija é uma redatora de conteúdo experiente, com conhecimento e paixão por cloud computação, tecnologia da informação e segurança online. No phoenixNAP, ela se concentra em responder a questões candentes sobre como garantir a robustez e a segurança dos dados para todos os participantes do cenário digital.

[< O que é automação de liberação de aplicativos \(ARA\)?](#)

[O que é uma chave de API? >](#)

Artigos Relacionados

[Segredos do Kubernetes – Como criar, usar e acessar segredos](#)

[Instalando FFmpeg no Mac](#)

[Tipos de dados do Cassandra: integrados, de coleção e definidos pelo usuário](#)

[Comando de arquivo do Linux: como determinar o tipo de arquivo no Linux](#)

[Comando curl do Linux explicado com exemplos](#)

[Como criar um banco de dados no PostgreSQL](#)

BATE-PAPO AO VIVO SAIBA MAIS

SUPORTE 1.855.330.1509 VENDAS 1.877.588.5918