ICS 35.100.60
L 79

中华人民共和国国家标准

GB/T 16262.2—2006/ISO/IEC 8824-2:2002

信息技术　抽象语法记法一(ASN.1)第2部分:信息客体规范

Information technology—Abstract Syntax Notation One (ASN.1)—Part 2: Information object specification

(ISO/IEC 8824-2:2002,IDT)

2006-03-14 发布　　2006-07-01 实施

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会　发布

目　次

前言 …… Ⅲ
引言 …… Ⅳ
1　范围 …… 1
2　规范性引用文件 …… 1
3　术语和定义 …… 1
3.1　基本记法规范 …… 1
3.2　约束规范 …… 1
3.3　ASN.1规范的参数化 …… 1
3.4　附加定义 …… 1
4　缩略语 …… 4
5　约定 …… 4
6　记法 …… 4
6.1　赋值 …… 4
6.2　类型 …… 4
6.3　值 …… 4
6.4　元素 …… 4
7　ASN.1词项 …… 4
7.1　信息客体类别引用 …… 5
7.2　信息客体引用 …… 5
7.3　信息客体集合引用 …… 5
7.4　类型字段引用 …… 5
7.5　值字段引用 …… 5
7.6　值集合字段引用 …… 5
7.7　客体字段引用 …… 5
7.8　客体集合字段引用 …… 5
7.9　字 …… 5
7.10　附加关键字 …… 5
8　引用定义 …… 5
9　信息客体类别定义和赋值 …… 6
10　语法表 …… 10
11　信息客体定义和赋值 …… 12
12　信息客体集合定义和赋值 …… 15
13　关联表 …… 16
14　客体类别字段类型记法 …… 17
15　来自客体的信息 …… 19
附录A(规范性附录)　类型标识符信息客体类别 …… 22

附录 B(规范性附录)　抽象语法定义 …… 23
附录 C(规范性附录)　单一实例类型 …… 24
附录 D(资料性附录)　示例 …… 26
附录 E(资料性附录)　ASN.1 客体集合扩展模块的个别指导附录 …… 30
附录 F(资料性附录)　记法综述 …… 31

前　言

GB/T 16262 在《信息技术　抽象语法记法一(ASN.1)》总标题下，目前包括以下 4 个部分：

第 1 部分(即 GB/T 16262.1)：基本记法规范；

第 2 部分(即 GB/T 16262.2)：信息客体规范；

第 3 部分(即 GB/T 16262.3)：约束规范；

第 4 部分(即 GB/T 16262.4)：ASN.1 规范的参数化。

本部分为 GB/T 16262 的第 2 部分，等同采用国际标准 ISO/IEC 8824-2:2002《信息技术　抽象语法记法一(ASN.1)：信息客体规范》(英文版)。与该项国际标准的等同文本是 ITU-T 建议 X.681。

按照 GB/T 1.1—2000 的规定，本部分对 ISO/IEC 8824-2:2002 作了下列编辑性修改：

——“本标准”一词改为“本部分”；

——在引用的标准中，凡已转化成我国标准的各项标准，均用我国的相应标准编号代替。

本部分的附录 A、附录 B 和附录 C 是规范性附录，附录 D、附录 E 和附录 F 是资料性附录。

本部分由中华人民共和国信息产业部提出。

本部分由中国电子技术标准化研究所归口。

本部分起草单位：中国电子技术标准化研究所。

本部分主要起草人：郑洪仁、安金海、徐云弛。

引　言

应用设计者经常需要设计一种协议，该协议用来处理某一类别信息客体中的任何一个实例，其中，类别的实例可由各个不同的其他实体定义，并可以不断增加。这种信息客体类别的示例有远程操作服务(ROS)中的操作和OSI目录的属性。

GB/T 16262的本部分提供一种记法，使得能定义信息客体类别，各个信息客体和信息客体集合，以及给出引用名。

信息客体类别由其实例所占有字段的种类表征。字段可以包括：

——任意类型(类型字段)；或

——规定类型的单个值(固定类型值字段)；或

——已命名类型字段中规定类型的单个值(可变类型值字段)；

——规定类型非空值集合(固定类型值集合字段)；或

——已命名类型字段中规定类型的非空值集合(可变类型值集合字段)；或

——规定信息客体类别中的单个信息客体(客体字段)；

——规定信息客体类别中的信息客体集合(客体集合字段)。

可以选择信息客体类别的固定类型值字段以提供类别中信息客体的唯一标识。这称为类别的标识符字段。如果提供标识符字段的值，它则应在为此类别所定义的信息客体集合中是唯一的。它们可以，但不必，无歧义地标识更广范围内的类别的信息客体，尤其是通过使用客体标识符作为标识符字段的类型。

通过规定下列内容来定义信息客体类别：

——字段的名称；

——每个字段的形式(类型、固定类型值、可变类型值、固定类型值集合、可变类型值集合、客体或客体集合)；

——字段的任选性和默认设置；

——如有必要，则应明确哪个字段是标识符字段，

类别中各个信息客体则对每个字段提供必要的信息予以定义。

本标准定义的记法允许通过引用某个信息客体类别的字段规定ASN.1的类型——客体类别字段类型。在GB/T 16262.3中，通过引用某个具体的信息客体集合使提供这种类型的记法受到限制。

将信息客体类别的定义用来定义基础概念表(关联表)的形式可能是有用的。此表是每个字段一列，整个行定义一个信息客体。此表的形式(由信息客体类别规范确定)确定为完成某个协议规范要收集的和使用的信息。基础概念表为规定此类别信息客体的记法和完成其规范需要此信息的协议之间提供链接。典型地，用来完成特定协议规范的实际信息客体集合应是此协议的参数(见GB/T 16262.4)。

引用特定客体或客体集合(或许是参数)的"InformationFromObjects"记法可用来从概念表的字符元中提取信息。

本部分：

——规定定义信息客体类别和用引用名标识它的记法(见第9章)；

——规定信息客体类别的定义符用以提供此类别信息客体定义用的定义语法的记法；对未定义语法业已定义的类别提供默认记法(见第10章)；

——规定定义信息客体，和将它赋于引用名的记法(见第11章)，并提供客体集合的模拟记法(见第12章)；

——定义一个类别的客体或客体集合的“关联表”(见第 13 章)；

——规定客体类别字段类型及其值的记法(见第 14 章)；

注：这些结构能够使用已命名信息客体类别的已命名字段规定 ASN.1 类型。关于将此类型限于与规定的信息客体集合有关的值的约束在 GB/T 16262.3 中规定。

——规定从客体中提取信息的记法(见第 15 章)。

在定义 ASN.1 规范时，可以部分或全部不知道定义客体集合中所使用的信息客体集合。例如，网络管理中出现的情况：当网络管理器正在执行时，被管客体在改变。本部分规定在客体集合定义中包含扩展标记的规则以向实施者告知设计者关注的，在 ASN.1 规范中没有完全定义客体集合内容。当用扩展标记定义客体集合时，实施者必须提供(可能超过 ASN.1 的范围)对客体集合动态地增加客体和从客体集合中删除先前增加客体的措施。

附录 A(本部分的构成部分)规定其客体类别引用是 TYPE-IDENTFIER 的信息客体类别。这是最简单有用的类别，它只有两个字段，一个是类型客体标识符的标识符字段，一个是定义传送此类别中任一特定客体所有信息的 ASN.1 类型的单一类型字段。本标准定义它，是因为这种形式的信息客体有广泛的使用。

附录 B(本部分的组成部分)用一合适的信息客体的定义规定定义抽象语法的记法。

附录 C(本部分的组成部分)规定单一实例类型的记法(单一记法)。

附录 D(不是本部分的组成部分)提供如何使用本部分所述记法的示例。

附录 E(不是本部分的组成部分)提供 ASN.1 客体集合扩展模型的综述。

附录 F(不是本部分的组成部分)提供本部分定义的记法综述。

信息技术　抽象语法记法一(ASN.1)
第2部分:信息客体规范

1　范围

GB/T 16262的本部分是抽象语法记法一(ASN.1)的一个部分,并提供规定信息客体类别、信息客体和信息客体集合的记法。

2　规范性引用文件

下列文件中的条款通过GB/T 16262的本部分的引用而成为本部分的条款。凡是注日期的引用文件,其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本部分,然而,鼓励根据本部分达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件,其最新版本适用于本部分。

GB/T 16262.1—2006　信息技术　抽象语法记法一(ASN.1)　第1部分:基本记法规范(ISO/IEC 8824-1:2002,IDT)

GB/T 16262.3—2006　信息技术　抽象语法记法一(ASN.1)　第3部分:约束规范(ISO/IEC 8824-3:2002,IDT)

GB/T 16262.4—2006　信息技术　抽象语法记法一(ASN.1)　第4部分:ASN.1规范的参数化(ISO/IEC 8824-4:2002,IDT)

3　术语和定义

下列术语和定义适用于GB/T 16262—2006的本部分。

3.1　基本记法规范

本部分使用GB/T 16262.1—2006中定义的术语。

3.2　约束规范

本部分使用GB/T 16262.3—2006中定义的下列术语:

——表约束　table constraint。

3.3　ASN.1规范的参数化

本部分使用GB/T 16262.4—2006中定义的下列术语:

a)　参数化类型　parameterized type;

b)　参数化值　parameterized value。

3.4　附加定义

3.4.1

关联表　associated tabe

通过由已有的链接字段(见3.4.15)而产生的展开分层结构可从客体或客体集合导出的(某个信息客体或信息客体集合)抽象表。

注:关联表能用来确定某种约束的精确性质(见GB/T 16262.3—2006),此约束是在使用客体集合时施加的。

3.4.2

默认语法　default syntax

对定义者没有提供定义语法的类别,定义这种类别信息客体的记法(见11.10的示例)。

3.4.3

定义语法 defined syntax

由类别的定义者提供的记法，它允许以用户友好的方式定义此类别的信息客体。

注：例如，类别 OPERATION 的定义语法可以允许此类别的实例定义如下：字 ARGUMENT 后接 &ArgumentType，然后，字 RESULT 后接 &ResultType，然后，字 CODE 后接 &OperationCode（见 11.11 的示例）。

3.4.4

可扩展客体集合 extensible object set

具有扩展标记的客体集合或对可扩展的客体集合进行集合运算定义的客体集合。

3.4.5

字段 field

信息客体类别的成分。每个字段是类型字段、固定类型值字段、可变类型值字段、固定类型值集合字段、可变类型值集合字段、信息客体字段或信息客体集合字段。

3.4.6

字段名称 field name

标识某个类别的字段的名称；或是直接规定此字段的类别，在此情况中，此名称是原始字段名称，或是具有与其中此字段实际上已被规定的情况有关的一系列链接字段的类别。

3.4.7

支配（类别） governing(class)

支配者 governor

要求其引用或规定支配类别的信息客体时，影响 ASN.1 语法某部分解释的信息客体类别定义或引用。

3.4.8

标识符字段 identifier field

为提供类别中信息客体的唯一标识，所选用的类别的固定类型值字段。标识符字段值，如果提供，在对此类别定义的任一信息客体集合中应是无歧义的。它们可以，但不需要在更广范围内起到无歧义地标识此类别的信息客体。

注 1：标识符字段具有固定的 ASN.1 类型，此类型的值可包含在协议中以标识此类型中的信息客体。

注 2：标识符为无歧义的范围是信息客体集合的范围。然而，它还能在任一给定的抽象语言中，或完整的应用上下文中是无歧义的，或甚至通过标识符字段使用客体标识符类型能对所有类别是完全无歧义的。

3.4.9

信息客体 information object

某个信息客体类别中的实例，由符合此类别字段规范的字段集合组成。

注：例如，信息客体类别 OPERATION（在 3.4.10 的示例中提及的）一个具体实例可以是 invertMatrix，它具有 &ArgumentType 字段（包含类型 Matrix），&ResultType 字段（也包含类型 Matrix）和 &operationCode 字段（包含值 7）（见 10.13 的示例）。

3.4.10

信息客体类别（类别） information object class(class)

对可能未界定的信息客体（此类别的实例）汇集为定义，形成模板的字段集合。

注：例如，信息客体类别 OPERATION 可以与远程操作服务（ROS）的操作概念相对应予以定义。此外，每个不同命名的字段规范可能与一个操作实例变到另一个操作实例的某个方面相对应。因而，可能有 &ArgumentType、&ResultType 和 &operationCode 字段，前两个规定类型字段，第三个规定值字段。

3.4.11

信息客体字段 information object field

包含某个规定类别信息客体的字段。

3.4.12

信息客体集合 information object set

使用相同信息客体类别引用名定义的所有信息客体的非空集合。

注：例如，类别 OPERATION(在 3.4.10 例中使用的)一个信息客体集合 MatrixOperation 可以包含 invertMatrix (在 3.4.9 中提及过)和其他有关操作(例如，addMatrices、multiplyMatrices 等)。这种客体集合可用来定义对调用作出规定并产生所有这些操作的结果报告的抽象语法(见 12.11 的示例)。

3.4.13

信息客体集合字段 information object set field

包含某个规定类别信息客体集合的字段。

3.4.14

单一实例类型 instance-of type

通过引用与类型相关联的客体标识符的信息客体类别所定义的类型。

3.4.15

链接字段 link field

客体或客体集合字段。

3.4.16

客体类别字段类型 object class field type

通过对信息客体类别中的某个字段的引用所规定的类型。在 GB/T 16262.3 中，提供的记法通过对此类别信息客体集合的引用使此类型受到限制。

3.4.17

原始字段名称 primitive field name

在信息客体类别定义中无须使用链接字段而直接规定的名称。

3.4.18

递归定义(引用名称的) recursive definition(of a reference name)

解析引用名称或引用名称定义的支配者要求解析最初引用名称的引用名称。

注：信息客体类别的递归定义是允许的。11.2 和 12.2 各自禁用信息客体或信息客体集合的递归定义。

3.4.19

递归实例(参数化引用名称的) recursive instantiation of a parameterized reference name

在解析实际参数要求解析最初引用名称时，引用名称的实例。

注：信息客体类别(包括编码结构)的递归实例是允许的。11.2 和 12.2 均禁用信息客体或信息客体集合的递归实例。

3.4.20

类型字段 type field

包含任意类型的字段。

3.4.21

值字段 value field

包含值的字段。这种字段或是固定类型或是可变类型。前者，值的类型是由字段规范确定的；后者，值的类型含在同一信息客体某个(指定)类型字段之中。

3.4.22

值集合字段 value set field

包含某个类型非空值集合的字段。这种字段或是固定类型或是可变类型。前者，值的类型是由字

段规范确定的;后者,值的类型含在同一信息客体某个(指定)类型字段之中。

注:对一信息客体而言,值集合字段中的值集合构成规定类型的子类型。

4 缩略语

本部分使用下列缩略语:

ASN.1 抽象语法记法一

5 约定

本部分采用 GB/T 16262.1—2006 第 5 章定义的记法约定。

6 记法

本章综述本部分定义的记法。

6.1 赋值

本部分定义了下列记法,这些记法能用作"Assignment"(见 GB/T 16262.1—2006 第 12 章)的替代记法:

——ObjectClassAssignment(见 9.1);

——ObjectAssignment(见 11.1);

——ObjectSetAssignment(见 12.1)。

6.2 类型

6.2.1 本部分定义了下列记法,这些记法能用作"BuitinType"(见 GB/T 16262.1—2006 的 16.2)的替代记法:

——ObjectClassFieldType(见 14.1);

——InstanceOfType(见附录 C)。

6.2.2 本部分定义了下列记法,这些记法能用作"ReferencedType"(见 GB/T 16262.1—2006 的 16.3)的替代记法:

——TypeFromObject(见 15 章);

——ValueSet FromObjects (见 15 章)。

6.3 值

6.3.1 本部分定义了下列记法,此记法能用作"Value"(见 GB/T 16262.1—2006 的 16.7)的替代记法:

——ObjectClassFieldValue(见 4.6)。

6.3.2 本部分定义了下列记法,此记法能用作"BuiltinValue"(见 GB/T 16262.1—2006 的 16.9)的替代记法:

——InstanceOfValue(见附录 C)。

6.3.3 本部分定义了下列记法,此记法能用作"ReferencedValue"(见 GB/T 16262.1—2006 的 16.11)的替代记法:

——ValueFromObject(见 15 章)。

6.4 元素

6.4.1 本部分定义了下列记法,此记法能用作"Elements"(见 GB/T 16262.1—2006 的 46.5)的替代记法:

——ObjectSetElements(见 12.10)。

7 ASN.1 词项

除 GB/T 16262.1—2006 第 11 章中规定的词汇之外,本部分中还使用下列各条规定的词项。适用

于这些词项的一般规则是按 GB/T 16262.1—2006 的 11.1 定义的。这些新的词项使用GB/T 16262.1—2006 第 10 章规定的 ASN.1 字符集，以及字符“&”。

注：GB/T 16262.1—2006 的 10.1 中的注也适用于 7.1 到 7.9 中规定的词项。

7.1 信息客体类别引用

词项名称——objectclassreference

“objectclassreference”应由 GB/T 16262.1—2006 的 11.2 对“typereference”规定的字符序列组成，但不应包括小写字母。

7.2 信息客体引用

词项名称——objectreference

“objectreference”应由 GB/T 16262.1—2006 的 11.4 对 “valuereference”规定的字符序列组成。

7.3 信息客体集合引用

词项名称——objectsetreference

“objectsetreference”应由 GB/T 16262.1—2006 的 11.2 对“typereference”规定的字符序列组成。

7.4 类型字段引用

词项名称——typefieldreference

“typefieldreference”应由“&”紧接 GB/T 16262.1—2006 的 11.2 对“typereference”规定的字符序列组成。

7.5 值字段引用

词项名称——valuefieldreference

“valuefieldreference”应由“&”紧接 GB/T 16262.1—2006 的 11.4 对“valuereference”规定的字符序列组成。

7.6 值集合字段引用

词项名称——valuesetfieldreference

“valuesetfieldreference”应由“&”紧接 GB/T 16262.1—2006 的 11.2 对“typereference”规定的字符序列组成。

7.7 客体字段引用

词项名称——objectfieldreference

“objectfieldreference”应由“&”紧接 7.2 对“objectreference”规定的字符序列组成。

7.8 客体集合字段引用

词项名称——objectsetfieldreference

“objectsetfieldreference”应由“&”紧接 7.3 对“objectsetreference”规定的字符序列组成。

7.9 字

词项名称——word

“word”应由 GB/T 16262.1—2006 的 11.2 对“type reference”规定的字符序列组成，但不应包括小写字母或数字。

7.10 附加关键字

名称 CLASS、INSTANCE、SYNTAX 和 UNIQUE 在 GB/T 16262.1—2006 的 11.27 中作为备用字列出。

8 引用定义

8.1 结构：

```
DefinedObjectClass::=
    ExternalObjectClassReference |
```

```
        objectclassreference |
        UsefulObjectClassReference
    DefinedObject::=
        ExternalObjectReference |
        objectreference
    DefinedObjectSet::=
        ExternalObjectSetReference |
        objectsetreference
```

分别用来引用类别，信息客体和信息客体集合定义。

8.2 对信息客体和信息客体集合的引用有一个支配类别。要求所引用的信息客体和所引用的信息客体集合中的信息客体应是支配类别中的客体。对信息客体而言，没有所规定的等效“value mappings”(见 GB/T 16262.1—2006 的附录 B)，所以上句意指信息客体或信息客体集合必须使用与支配者使用的相同信息客体类别引用(或通过简单引用赋值获得的引用)加以定义。就此要求而言，两个等同的信息客体类别记法的实例(但字面上不同)不能视为同样的信息客体类别。

8.3 除 GB/T 16262.1—2006 的 12.15 规定外，“objectclassreference”、“objectreference”和“objectsetreference”的替代记法只能用于将类别或信息客体或信息客体集合赋于此引用的模块(见 9.1、11.1 和 12.1)。

“ExternalObjectClassReference”、“ExternalObjectReference”和“ExternalObjectSetreference”的替代记法定义如下：

```
    ExternalObjectClassReference::=
        modulereference
        "."
        objectclassreference
    ExternalObjectReference::=
        modulereference
        "."
        objectreference
    ExternalObjectSetReference::=
        modulereference
        "."
        objectsetreference
```

不应使用这些替代记法，除非在由相应的“modulereference”标识的模块(与引用模块不同)中已将类别或信息客体或信息客体集合赋于相应的“objectclassreference”、“objectreference”或“objectsetreference”(见 9.1、11.1 和 12.1)。这就是分别被引用的类别或信息客体或信息客体集合。

8.4 “DefinedObjectClass”的替代记法“UsefulObjectclassReference”定义如下：

```
UsefulObjectClassReference::=TYPE-IDENTIFIER| ABSTRACT-SYNTAX
```

其中，第一记法在附录 A 中规定，第二记法在附录 B 中规定。

注：名称 TYPE-IDENTIFIER 和 ABSTRACT-SYNTAX 在 GB/T 16262.1—2006 的 11.27 中作为备用字列出。

9 信息客体类别定义和赋值

9.1 结构“ObjectClassAssignment”用来将信息客体类别赋于引用名(“objectclassreference”)。此结构是 GB/T 16262.1—2006 的 12 章，“Assignment”的替代记法之一，并定义如下：

```
    ObjectClassAssignment::=
```

objectclassreference
"::= "
ObjectClass

9.2 信息客体类别是由结构“ObjectClass”定义的类别：

ObjectClass::=
DefinedObjectClass |
ObjectClassDefn |
ParameterizedObjectClass

如果“object class”是：

a) “DefinedObjectclass”，那么此类别定义与此类别所引用的定义相同；

b) “ObjectClassDefn”，那么此类别按 9.3 中所述定义；

c) ParameterizedObjectClass”，那么此类别按 GB/T 16262.4 的 9.2 中所述定义。

9.3 每个类别归根结底是由“ObjectClassDefn”定义的：

ObjectClassDefn::=
CLASS
"{" FieldSpec "," + "}"
withSyntaxSpec?
WithSyntaxSpec::=WITH SYNTAX SyntaxList

这种记法允许类别的定义者提供命名的字段规范，其中每个规范是 9.4 中定义的一种“FieldSpec”。作为选择，定义者可以提供按 10.5 定义的信息客体定义语法(“SyntaxList”)。类别的定义者还可规定与此类别定义相关的语义。

9.4 每个“FieldSpec”规定并命名字段之一，它们应该或可以与此类别的实例相关联：

FieldSpec::=
TypeFieldSpec |
FixedTypeValueFieldSpec |
VairableTypeValueFieldSpec |
FixedTypeValueSetFieldSpec |
VariableTypeValueSetFieldSpec |
ObjectFieldSpec |
ObjectSetFieldSpec

“FieldSpec”的各种替代记法在以下各条中规定。

9.5 “TypeFieldSpec”规定此字段是类型字段(见 3.4.20)：

TypeFieldSpec::=
typefieldreference
TypeOpetionalitySpec?
TypeOptionalitySpec::=OPTIONAL|DEFAULT Type

此字段的名称是“ typefieldreference”。如果没有“typeOptionalitySpec”，则要求此类别的所有信息客体定义包括此字段的类型规范。如果 OPTIONAL 存在，那么，可以不定义此字段。如果 DEFAULT 存在，那么，在“Type”之后，则为(此字段)提供在定义中省略的默认设置。

9.6 “FixedTypeValueFieldSpec”规定此字段是固定类型值字段(见 3.4.21)：

FixedTypeValueFieldSpec::=
valuefieldreference
Type

UNIQUE?

ValueOpetionalitySpec?

ValueOptionalitySpec::=OPTIONAL|DEFAULT Value

此字段的名称是“valuefieldreference”。此“Type”结构规定此字段中的包含的值的类型。如果ValueOpetionalitySpec存在，它规定在信息客体定义中可以省略此值，或在DEFAULT情况时，它规定省略随后的“Value”(它应是此类型的值)。存在关键字UNIQUE时，它规定此字段是3.4.8(也见GB/T 16262.3—2006的10.20)定义的标识符字段。如果此关键字存在，则“Value OpetionalitySpec”不应是“DEFAULT Value”。

9.7 对标识符字段赋值时，要求此值在任一定义的信息客体集合中是无歧义的。

9.8 “VariableTypeValueFieldSpec”规定此字段是可变类型值字段(见3.4.21)：

VariableTypeValueFieldSpec::=

valuefieldreference

FieldName

ValueOptionalitySpec?

此字段名称是“valuefieldreference”。“FieldName”(见9.14)与要规定的类别相关。它应是类型字段；此类型字段或是作为值字段处于相同的信息客体中，或通过客体字段链(其引用出现在“FieldName”中)链接。此类型字段应包含值的类型。(字段引用出现在“FieldName”中的所有链接字段应是客体字段。)如果存在ValueOpetionalitySpec，它规定在信息客体定义中可以省略此值，或在DEFAULT情况时，它规定省略随后的“Value”。“ValueOptionalitySpec”应是这样：

a) 如果由“FieldName”表示的类型字段有OPTIONAL的“TypeOptionality Spec”，那么，“ValueOptionalitySpec”也应是OPTIONAL；以及

b) 如果“ValueOptionalitySpec”是“DEFAULT Value”，那么，由“FieldName”表示的类型字段应有“DEFAULT Type”的“TypeOptionlitySpec”，并且“Value”应是此类型的值。

9.9 “FixedTypeValueSetFieldSpec”规定此字段是固定类型值集合字段(见3.4.22)：

FixedTypeValueSetFieldSpec::=

valuesetfieldreference

Type

ValueSetOptionalitySpec?

ValueSetOptionalitySpec::=OPTIONAL|DEFAULT ValueSet

注：“ValueSet”在GB/T 16262.1—2006的15.6和15.7中定义，并允许对值集合显式列表(用波浪大括号)，或对“Type”的子类型使用“type reference”。

此字段的名称是“valuesetfieldreference”。“Type”结构规定此字段中所包含的值的类型。如果存在“ValueSetOptionalitySpec”，它规定在信息客体定义中可以不规定此字段，或在DEFAULT情况时，规定省略随后的“ValueSet”，它应是此类型的子类型。

9.10 “VariablTypeValueSetFieldSpec”规定此字段是可变类型值集合字段(见3.4.22)：

VariableTypeValueSetFieldSpec::=

valuesetfieldreference

FieldName

ValueSetOptionalitySpec?

此字段的名称是“valuesetfieldreference”。“Field Name”(见9.14)与要规定的类别有关。它应是类型字段；此类型字段或是作为值集合字段处于相同的信息客体中，或通过客体字段链(其引用出现在“FieldName”)链接。此类型字段应包含值的类型。(字段引用出现在“FieldName”中的所有链接字段应是客体字段。)如果存在“value setOptionalitySpec”，它规定在信息客体定义中可以省略此值集合，或

在 DEFAULT 情况时，它规定省略随后的“Valueset”。“ValueSetOptionalitySpec”应是这样：

a) 如果由“FieldName”表示的类型字段有 OPTIONAL 的“TypeOptionalitySpec”，那么，“ValueSetoptiOnalitySpec”也应是 OPTINAL；以及

b) 如果“ValueSetoptiOnalitySpec”是“DEFAULT ValueSet”，那么，由“FieldName”表示的类型字段应有“DEFAULT Type”的“TypeOptionality Spec”，并且“ValueSet”应是此类型的子类型。

9.11 “ObjectFieldSpec”规定此字段是信息客体字段(见 3.4.11)：

```
ObjectFieldSpec::=
    objectfieldreference
    DefinedObjectClass
    ObjectOptionalitySpec?
ObjectOptionalitySpec::=OPTIONAL|DEFAULT Object
```

此字段的名称是“objectfieldreference”。“DefinedObjectClass”引用此字段(它可以是当前被定义的“ObjectClass”中所包含的客体类别。如果存在“Object OptionalitySpec”，它规定可以在信息客体定义中不规定此字段，或在 DEFAULT 情况时，规定省略随后的“object”(它应是“DefinedObjectClass”的客体)。

9.12 “ObjectSetFieldSpec”规定此字段是信息客体集合字段(见 3.4.13)：

```
ObjectSetFieldSpec::=
    objectsetfieldreference
    DefinedObjectClass
    ObjectSetOptionalitySpec?
ObjectSetOptionlitySpec::=OPTIONAL|DEFAULT ObjectSet
```

此字段的名称是“objectsetfieldreference”。“DefinedObjectClass”引用此字段中包括的客体类别。如果存在“ObjectSetOptionalitySpec”，它规定可以在信息客体定义中省略此字段，或在 DEFAULT 情况时，规定省略随后的“ObjectSet”(见 12.3)。其所有客体应是“DefinedObjectClass”的客体。

9.13 结构“primitiveFieldName”用来标识与包含其规范的类别有关的字段：

```
PrimitiveFieldName::=
    typefieldreference          |
    valuefieldreference         |
    valuesetfieldreference      |
    objectfieldreference        |
    objectsetfieldreference
```

在类别定义中所规定的所有字段名称应是各不相同的。

9.14 结构“FieldName”用来标识与直接包含此字段规范的某个类别有关的字段，或具有一系列链接字段链接到此包含类别的某个类别有关的字段。此链由用句点隔开的“PrimitiveFieldName”列表指明。

```
FieldName::=PrimitiveFieldNam"."+
```

9.15 如果有如下情况的一系列链接字段(见 3.4.15)规范(长度为 1 或更长)：

a) 第一个处于被定义的类别中而不是被定义的字段；和

b) 每个相继的规范是定义前一个规范的类别中的字段；及

c) 最后一个是使用被定义的类别加以定义。

那么，至少有一个字段规范应具有“ObjectOptionalitySpec”或“ObjectSetOptionalitySpec”(视合适而定)。

注：这是为防止递归的信息客体类别定义(一般情况下是允许的)。对此递归类别的信息客体具有无限的表示。

9.16 实例

在3.4.10中作为示例的非正式描述的信息客体类别的展开形式可以定义如下：

```
OPERATION::=CLASS
    {
        &ArgumentType       OPTIONAL,
        &ResultType         OPTIONAL,
        &Errors             ERROR OPTIONAL,
        &Linked             OPERATION OPTIONAL,
        &resultReturned     BOOLEAN DEFAULT TRUE,
        &Code               INTEGER UNIQUE
    }
ERROR::=CLASS
    {
        &ParameterType      OPTIONAL,
        &code               INTEGER UNIQUE
    }
```

注1：此示例是基于远程操作标准的操作和差错概念，但是作为示例，它被简化了。

注2：对此类别规定的字段包括2个类型字段(&ArqumentType 和 &ResultType)，2个客体集合字段(&Errors 和 &Linked)和2个值字段(&result Returned 和 &code)，后者是标识符字段。

注3：由OPRATIONS构成的任何信息客体集合必须是对 &code 字段，此集合中没有2个客体具有相同的值。(这同样适用于ERRORS的客体集合。)

注4：OPERATION 信息客体类别包括一系列9.15所描述的链接字段。本链长度为了并通过 &linked 构成，它借助 OPERATION 加以(递归)规定。然而，这是十分有效的，因为将 OPERATION 赋于此字段(见9.15)。

注5：这些示例中没有一个示例包括“withSyntaxSpec”。但是在10.13中提供这样做的相应示例。

10 语法表

10.1 经常的情况是单独一个规范定义信息客体类别，对多种信息客体而有许多独立的规范分别予以定义。为此类别信息客体定义提供用户友好的记法，这可能对类别的定义者是合适的。

10.2 本章规定信息客体类别的制定者用以对此类别信息客体规范定义具体类别定义语法的记法。

10.3 此记法是语法结构“SyntaxList”，它出现在于语法结构“ObjectClassDefn”(见9.3)之中。

10.4 “SyntaxList”规定被定义类别的一个信息客体定义的语法。在下列各条中，此语法以“DefinedSyntax”表现。

注：此规范的特征是由“SyntaxList”定义的任一语法结构的末端(“DefinedSyntax”的实例)可由下列各条确定：

a) 无 ASN.1 说明；

b) 将字符串值处理成文字符号；

c) 要求一个初始的“{”，嵌入成对的“{”和“}”，并终止于一个未配对的“}”。

10.5 “SyntaxList”规定出现在“DefinedSyntax”(见11.6)中的“DefinedSyntaxToken”的顺序：

```
SytaxList::= "{" TokenOrGroupSpec empty+ "}"
TokenOrGroupSpec::=RequiredToken|OptionalGroup
OptionalGroup::= "["TokenOrGroupSpec empty+ "]"
RequiredToken::=
        Literal     |
        PrimitiveFieldName
```

注1：“SyntaxList”的编写者没有提供 BNF 的全部功能。记法功能差不多等于通常用于规定命令解释程序的命令

行语法的权力。给出可能的“RequiredToken”的表，在于准许他们这样做；将一串或几串符号置于方括号中，使得它们成为任选的。

注2：当分析“SyntxList”时，“[[”(或“]]”)的存在，不能分别解释为GB/T 16262.1—2006的11.19和11.20中规定的词项“[[”(或“]]”)，但可分别解释为“[”和“[”(或“]”和“]”)两个词项。

10.6 用作“Literal”的“word”符号不应是下列之一：

BIT
BOOLEAN
CHARACTER
CHOICE
EMBEDDED
END
ENUMERATED
EXTERNAL
FALSE
INSTANCE
INTEGER
INTERSECTION
MINUS-INFINITY
NULL
OBJECT
OCTET
PLUS-INFINITY
REAL
RELATIVE-OID
SEQUENCE
SET
TRUE
UNION

注：此表仅包含ASN.1备用的，表现为“Type”、“value”、“valueset”、“Object”或“ObjectSet”第一个词项的字以及备用字END。使用ASN.1其他备用字不引起歧义性的，允许使用。若定义语法用于一个“word”还是“typereference”和“objectsetreference”的环境时，首先采用当作“word”的用法。

10.7 “Literal”规定此“Literal”的实际内含，它是定义语法中此位置处的“word”或逗号(“,”)：

```
Literal::=
        Word |
        ","
```

10.8 每个“PrimitiveFieldName”对相应字段规定“Setting”(见11.7)的内含(在新语法中的此位置)。

10.9 信息客体类别的每个“PrimitiveFieldName”只能准确地出现一次。

10.10 在分析过程中，当遇到“OptionalGroup”，以及随后的词项在语法上可作为任选组中的第一个词项时，那么认为此组是存在的。如果它在语法上不能作为任选组的第一个词项，那么，认为此组是不存在的。

注：为避免不希望的影响，设计者通常使任选组中的第一个词项成为“Literal”。

10.11 使用“DefinedSyntax”的实例是无效的，除非它对信息客体类别规定所有强制字段。

10.12 为了保证易于分析新语法和防止误用，对新语法的定义者提出如下附加限制：

a) 要求每个“OptionalGroup”内至少有一个“PrimitiveFieldName”或“OptionalGroup”；

注1：这有助于防止在信息客体任一字段中没被反映的信息明显汇集。

b) "OptionalGroup"的使用应该使得在分析过程中决不能出现"setting",它可能是一个以上"Field Name"的设置;

c) 如果"OptionalGroup"以一个"Literal"开始,那么"OptionalGroup"之后的第一个符号也应是一个"Literal",并且应与紧接在"OptionalGroup"之前所有第一个"Literal"不同;

同时对"DefinedSyntax"的使用者提出如下限制:

d) 在"Optional Group"中出现的"Defined Syntax"内存在一个"Literal",那么此"Optional Group"中"Primitive Field Name"的"setting"也应存在。

注 2:这有助于防止在信息客体任一字段中没被反映的信息明显汇集。

注 3:下列的示例是法定语法,而限制 d)防止使用者书写 LITERAL 时没有在其之后用此任选组之一或两者:[LITERAL [A&field] [B&field2]]

10.13 示例

上面 9.16 类别定义的示例可以装配定义语法,以提供定义这些类别实例的"用户友好"的方法(此定义语法用于 11.11 的示例中):

```
OPERATION::=CLASS
    {
        &ArgumentType                OPTIONAL,
        &ResultType                  OPTIONAL,
        &Errors                      ERROR OPTIONAL,
        &Linked                      OPERATION OPTIONAL,
        &resultReturned              BOOLEAN DEFAULT TRUE,
        &operationCode               INTEGER UNIQUE
    }
    WITH SYNTAX
    {
        [ARGUMENT                    &ArgumentType]
        [RESULT                      &ResultType]
        [RETURN RESULT               &resultReturned]
        [ERRORS                      &Errors]
        [LINKED                      &Linked]
        CODE                         &operationCode
    }
ERROR::=CLASS
    {
        &ParameterType               OPTIONAL,
        &errorCode                   INTEGER UNIQUE
    }
    WITH SYNTAX
    {
        [PARAMETER                   &ParameterType]
        CODE                         &errorCode
    }
```

11 信息客体定义和赋值

11.1 语法结构"ObjectAssignment"用来将规定类别的信息客体赋于引用名("objectreference")。此

结构是 GB/T 16262.1—2006 中第 12 章“Assignment”的替代记法之一，并定义如下：

```
ObjectAssignment::=
    objectreference
    DefinedObjectClass
    "::= "
    Object
```

11.2 不应有“objectreference”递归定义（见 3.4.18），以及不应有“objectreference”递归实例（见 3.4.9）。

11.3 “DefinedObjectClass”引用的类别的信息客体是由结构“Object”定义的：

```
Object::=
    DefinedObject |
    ObjectDefn |
    ObjectFromObject |
    ParameterizedObject
```

如果“Object”是：

a) “DefinedObject”，那么，此客体与所引用的客体相同；

b) “ObjectDefn”，那么，此客体按 11.4 规定；

c) “ObjectFromObject”，那么，此客体按 15 章规定；

d) “ParameterizedObject”，那么，此客体按 GB/T 16262.4—2006 的 9.2 规定予以定义。

11.4 每个信息客体归根结底是由“ObjectDefn”定义的：

```
ObjectDefn::=
    DefaultSyntax |
    DefinedSyntax
```

如果类别定义不包括“withSyntaxSpec”，则“ObjectDefn”应是“DefaultSyntax（见 11.5）”，如果它包括，则“ObjectDefn”应是“DefinedSyntax”（见 11.6）。

11.5 “DefaultSyntax”结构定义如下：

```
DefaultSyntax::= "{" FieldSetting "," * "}"
FieldSetting::=PrimitiveFieldName Setting
```

对不是 OPTIONAL 并且没有 DEFAULT 的类别定义中的每个“FieldSpec”应准确地有一个“FieldSetting”，对其他的每个“FieldSpec”至多有一个“Fieldsetting”。“FieldSetting”可以以任意次序出现。每个“FieldSetting”中的“PrimitiveFieldName”应是相应“FieldSpec”的名称。结构“Setting”在 11.7 中规定。

11.6 “DefinedSyntax”结构定义如下：

```
DefinedSyntax::= "{" DefinedSyntaxToken empty * "}"
DefinedSyntaxToken::=
    Literal |
    Setting
```

“withSyntaxSpec”中的“SyntaxList”（见第 10 章）确定要出现在“DefinedSyntax”中的“DefinedSyntaxToken”的顺序。结构“setting”在 11.7 中规定；每一次的出现规定信息客体某个字段的设置。结构“Literal”在 10.7 中定义；“Literal”是人们可读性的体现。

11.7 “setting”规定所定义信息客体中某个字段的设置：

```
Setting::=
    Type |
```

```
        Value    |
        ValueSet |
        Object   |
        ObjectSet
```

如果此字段是：

a) 类型字段，则应选用“Type”替代记法；

b) 值字段，则应选用“Value”替代记法；

c) 值集合字段，则应选用“ValueSet”替代记法；

d) 信息客体字段，则应选用“Object”替代记法；

e) 信息客体集合字段，则应选用“ObjectSet”替代记法。

注：设置将按 9.5 到 9.12 和 11.8 到 11.9 合适条款所述予以进一步限制。

11.8 可变类型值字段的设置应是由相同或所链接客体的合适类型字段的规定的类型值(即，不采用开放类型的值记法)。

11.9 可变类型值集合字段的设置应是由相同或所链接客体的合适类型字段所规定的类型值集合(即，不采用开放类型的值记法)。

11.10 示例(默认语法)

已知上面 9.16 的信息客体类别定义(不包括“WithSyntaxSpec”)，此类别的实例可使用“DefaultSyntax”予以定义。例如(3.4.9 给出的示例的展开形式)

```
invertMatrix OPERATION::=
    {
        &ArgumentType          Matrix,
        &ResultType            Matrix,
        &Errors                {determinantIsZero},
        &operationCode         7
    }
determinantIsZero ERROR::=
    {
        &errorcode             1
    }
```

11.11 示例(定义语法)

在 10.13 中，示例类别被提供给“WithSyntaxSpec”，因此，此类别实例使用“DefinedSyntax”予以定义。所以，11.10 的示例要写成：

```
invertMatrix OPERATION::=
    {
        ARGUMENT               Matrix
        RESULT                 Matrix
        ERRORS                 {determinantIsZero}
        CODE                   7
    }
determinantIsRero ERROR::=
    {
        CODE                   1
    }
```

12 信息客体集合定义和赋值

12.1 语法结构“ObjectSetAssignment”用来将规定类别的信息客体集合赋于引用名(“objectsetreference”)。此结构是 GB/T 16262.1—2006 第 12 章“Assignment”的替代记法之一,并定义如下:

```
ObjectSetAssignment::=
    objectsetreference
    DefinedObjectClass
    "::= "
    ObjectSet
```

12.2 不应有“objectsetreference”递归定义(见 3.4.18),以及不应有“objectsetreference”递归实例(见 3.4.9)。

12.3 “DefinedObjectClass”引用的类别的信息客体集合是由结构“ObjectSet”定义的:

```
ObjectSet::= "{" ObjectSetSpec " }"
ObjectSetSpec::=
    RootElementSetSpec    |
    RootElementSetSpec "," "…"|
    "..."
"…" "," AdditionalElementSetSpec    |
RootElementSetSpec ", " "..." ","AdditionalElementSetSpec
```

“RootElementSetSpec”和“AdditionalElementSetSpec”在 GB/T 16262.1—2006 中规定,并能利用支配类别的信息客体或集合来规定一个信息客体集合。集合中应至少有一个信息客体,除非规定了“ObjectSetSpec”的第三个替代记法(“…”)。在后一种情况,省略号的存在指明此客体集合最初是空的,但是通过应用程序将有客体动态地加入。

注 1:“ObjectSetSpec”引用的元素是“RootElementSetSpec”和“AdditionalElementSetSpec”所引用元素的并集

注 2:与可扩展类型,例如 SET 或 SEQENCE,或可扩展子类型约束(相对为 ASN.1 规范每种型式而设置的“understood”值集合是静态的)不同,可扩展客体集合在一给定的形式中能够动态地减少和增加。换言之,它可以在使用应用程序的给定实例中扩充和减少,好象它是动态地定义或未定义客体(进一步讨论见附录 E)。

12.4 涉及可扩展信息客体集合的集运算结果在 GB/T 16262.1—2006 第 46 章中规定。

12.5 如果可扩展信息客体集合 A 在另一个客体集合定义中被引用,那么,其扩展标记和其扩展由 B 继承。

12.6 如果使用可扩展信息客体集合定义“ValueSetFromObject”(见第 15 章)则最后产生的值集合不继承此信息客体集合的扩展标记。

12.7 如果类型由表约束所约束(见 GB/T 16262.3—2006 的 10.3 条)并且表约束中所引用的客体集合是可扩展的,则此类型不继承此客体集合的扩展标记。如果此类型预定是可扩展的,那么,应将扩展标记明确地加到其“ElementSetSpec”。

12.8 如果类型由不是可扩展的信息客体集合约束,那么,一致性实现应支持此集合中的所有信息客体,并且不应使用非此集合中的信息客体产生编码。

12.9 如果类型由可扩展信息客体集合约束,那么,一致性实现可根据对所约束类型每个实例的本地决定选择支持是根或是扩展中的一个信息客体。不应使用可扩展信息客体集合的根或扩展中具有 UNIQUE字段值的其他信息客体产生编码,但是,另一方面,可对所要求类别的任一信息客体产生编码。

12.10 “ObjectSetElements”记法如下:

```
ObjectSetElements::=
```

```
        Object                  |
        DefinedObjectSet        |
        ObjectSetFromObjects    |
        ParamenterizedObjectSet
```

此记法规定的元素由采用的记法确定，具体如下：

a) 如果使用“Object”替代记法，则仅规定确已赋值的客体。此客体应是支配类别的客体；

b) 如果使用其余替代记法的任一种，则规定确已赋值的集合的所有客体。这些客体应是支配类别的客体。如果使用“DefinedObjectSet”替代记法，则客体集合是被引用的集合。如果使用“ObjectSetFromObjects”替代记法，那么客体集合按第 15 章规定。如果使用“Parameterized ObjectSet”替代记法，那么客体集合按 GB/T 16262.4—2006 的 9.2 条规定。

12.11 示例

在 3.4.12 注中非正式描述的信息客体集合可规定如下：

```
MatrixOperations OPERATION::=
    {
        invertMatrix    |
        addMatrices     |
        subtractMatrices    |
        MultiplyMatrices
    }
```

13 关联表

13.1 每个信息客体或信息客体集合能够看作一个表：其关联表。关联表的每个字符元对应于信息客体某个字段的设置，或是空的。关联表的列集合由客体或客体所属的类别确定；然而，行集合是由所涉及的客体或几个客体确定。

13.2 已知类别的定义，列集合定义如下：

a) 此类别定义的每个字段规范有一列。每个列用对应的“PrimitiveFieldName”命名；

b) 有一个附加的列集合对应于每个链接字段规范。此列集合是由链接字段的支配类别规则的应用确定的，但其名称以链接字段的“PrimitiveFieldName”和句号(“.”)为前缀的除外。

注：这些规则是递归的，并且是：如果类别是直接或间接自引用的，则列集合不是有限的。这不被禁止。

13.3 已知某个类别的信息客体，关联表是将 13.4 应用于正好包含此客体的客体集合而产生的表。

13.4 已知某个类别的信息客体集合，关联表中的行集合是执行下列递归规程而产生的：

a) 客体集合中的每个客体以一行开始。在每一行中，列中由“PrimitiveFieldName”命名的字符元将对应客体中合适字段的设置，而其他所有字符元是空的；

b) 对出现在行集合中某个行的每个链接字段：

1) 产生链接字段内容的(次级)关联表。

2) 下一步，用一批行代替链接字段所在的行，次级关联表的每个行进行一次替换。此批中的每一行与被取代的行相同，但用次级关联表被选择行的字符元填充，至今是空的对应字符元除外。其“FieldName”以“PrimitiveFieldName”为前缀。

注：这些规则是递归的，并且是：如果信息客体是直接或间接自引用的，则该规程将不会终止。这不被禁止。实际上，为了知道有限长度名命字的字符元的内容才是必要的，为此可以设计一种有界限的规程。

13.5 有效“FieldName”的示例

下列“FieldName”对类别 OPERATON(见 10.13 中定义)信息客体或信息客体集合关联表是有

效的：

&ArgumentType
&Errors.&Parameter
&Errors.&errorCode
&Linked.&ArgumentType
&Linked.&Linked.&operetionCode
&Linked.&Linked.&Linked.&Linked.&Linked.&Errors.&errorCode

因为类别 OPERATION 是自引用的(通过 &Linked 字段)，所以列的数目不是有限的。

14 客体类别字段类型记法

用此记法被引用的类型与字段名称的种类有关。对不同的字段名称种类，在 14.2 到 14.5 中规定了被引用的类型。

14.1 客体类别字段类型(见 3.4.16)的记法应是“ObjectClassFieldType”：

ObjectClassFieldType::=
DefinedObjectClass
"."
FieldName

其中，“FieldName”按 9.14 规定，与“DefinedObjectClass”标识的类别有关。

14.2 对类型字段，此记法定义一种开放类型，即其值集合是能用 ASN.1 规定的所有可能值的全集。使用对应信息客体集合的约束规范(见 GB/T 16262.3—2006)可将此类型限于一特定类型。当“FieldName”引用类型字段时，关于使用这种记法的下列约束适用：

a) 这种记法不应直接或间接用于信息客体类别值或值集合字段的类型定义；

b) 此记法有一个不确定的标记，因此当要求与某个其他类型不同的标记时不能使用这种记法；

注 1：通过显式标记此类型通常可以避免这种限制。

注 2：尽管在 GB/T 16262.1—2006 的 48.7.3 中规定对扩展标志概念上所加的元素有一个与所有已知 ASN.1 类型的标记不同的标记，但是要求开放类型有一个与概念上所加的元素的标记不同的标记时，仍不应使用此开放类型。

c) 不应隐式标记这种记法；

注 3：当将开放类型限于可以是一个选用类型的特定类型时应显式标记。

d) 要求编码规则对赋于定义的成分的值进行编码，使得接收者无须知道此成分的实际类型就能成功地确定与包含此成分的结构所有其他部分相对应的抽象值。

注 4：“Type”结构通常使用信息客体集合和“AtNotation”予以约束(见 GB/T 16262.3—2006 第 10 章)。然而，要告诫 ASN.1 的使用者在没有应用约束时使用这种记法可能导致实现要求的歧义性，通常应予以避免。

14.3 对固定类型值或固定类型值集合字段，此记法表示出现在信息客体类别定义中此字段的规范中的“Type”。

具有客体、标识符或值引用的特定选择名，在这种产生式后的“SimpleTableConstraint”(见 GB/T 16262.3—2006 的 10.3)也能是有效的“SingleValue”(见 GB/T 16262.1—2006 的 47.2)子类型约束。在这种情况下，它应被看作是“SimpleTableConstraint”。

14.4 对可变类型值或可变类型值集合字段，此记法表示开放类型，其使用受 14.2 规定的相同限制所支配。

14.5 如果此字段是客体字段或客体集合字段，则不允许这种记法。

14.6 定义此类型的值的记法应是“ObjectClassFieldValue”，或用于“XMLTypeValue”时，是“XMLObjectClassFieldValue”：

```
ObjectClassFieldValue::=
        OpenTypeFieldVal    |
        FixedTypeFieldVal
OpenTypeFieldVal::=Type ":" Value
FixedTypeFieldVal::=BuiltinValue|ReferencedValue
XMLObjectClassFieldValue::=
      XMLOpenTypeFieldVal    |
      XMLFixedTypeFieldVal
XMLOpenTypeFieldVal::=XMLTypedValue
XMLFixedTypeFieldVal::=XMLBuiltinValue
```

14.7 对由“ObjectClassFieldType”固定类型值或值集合字段，应使用“FixedTypeFieldVal”或“XMLFixedTypeFieldVal”，并应是信息客体类别定义中规定的“Type”值。

14.8 对由“ObjectClassFieldType”定义的类型字段或可变类型值或值集合字段，“OpenTypeFieldVal”应用于任一“Value”。“OpenTypeFieldVal”中的“Type”应是任一 ASN.1 类型，并且“Value”应是此类型的任一值。

14.9 对由“ObjectClassFieldType”定义的类型字段或可变类型值或值集合字段，“XMLOpenTypeFieldVal”应用于任一“XMLValue”。当用于 ASN.1 模块时，由 XMLTypedValue 所标识的类型应是任一的 ASN.1 类型(且见 GB/T 16262.1—2006 的 13.3)，并且“XMLTypeValue”中的“XMLValue”应是此类型的任一值。

注：当按 GB/T16262.4—2006 的 8.3.1 的规定使用此记法时，“XMLOpenTypeFieldVal”中的“XMLTypedValue”的类型由协议(例如，由成分关系式约束)标识，由此求得“XMLTypeValue”中的“NonParameterizedTypeName”，并且“XMLValue”是此类型的值。

14.10 (作为“ObjectClassFieldType”的)XML 值记法(“XMLFixedTypeFieldVal”)的“xmlasnltypename”项中的字符序列应是信息客体类别中规定的“Type”的字符序列。单一序列和单一集合的 XML 值记法(见 GB/T 16262.1—2006 表 5)应由信息客体类别中规定的“Type”确定。

14.11 (作为“ObjectClassFieldType”的)XML 值记法(“XMLOpenTypeFieldVal”)的“xmlasnltypename”项中的字符序列应是“OPEN-TYPE”。单一序列和单一集合的 XML 值记法(见 GB/T 16262.1—2006 的表 5)应是“XMLDelimitedItemList”。

14.12 对于“XMLOpenTypeFieldVal”，如果信息客体(忽略任何标记之后)中规定的“Type”是“typereference”或“ExternalTypeReference”，那么，“NonParameterizedTypeName”应是“typeteference”或“ExternalTypeReference”，否则，应是 GB/T 16262.1—2006 的表 4 规定的“xmlasnltypename”，并与信息客体中规定的内置类型相对应。

14.13 “ObjectClassFieldType”用法示例

下列每个示例基于 10.13 的示例并表明(a)可能的“ObjectClassFieldType”，(b)与示例类型(a)等价的类型(当没有约束地使用时)和(c)此类型值示例的记法。

```
1 (a) OPERATION.&operationCode
  (b) INTEGER
  (c) 7
2 (a) OPERATION.&ArgumentType
  (b) open type
  (c) Matrix:
      {{1, 0, 0, 0},
      {0, 1, 0, 0},
```

{0, 0, 1, 0},
{0, 0, 0, 1}}

3 (a) OPERATION. &Linked. &Linked. &Errors. &errorcode
(b) INTEGER
(c) 1

4 (a) OPERATION. &Linked. &ArgumentType
(b) open type
(c) UniversalString:{planckConstant, "and", hamiltonOperator}

15 来自客体的信息

15.1 客体或客体集合关联表列中的信息可由"InformationFromObject"记法的各种情况予以引用：

```
InformationFromObjects::=
    ValueFromObject       |
    ValueSetFromObjects   |
    TypeFromObject        |
    ObjectFromObject      |
    ObjectSetFromObjects
ValueFromObject::=
    ReferencedObjects
    "."
    FieldName
ValueSetFromObject::=
    ReferencedObjects
    "."
    FieldName
TypeFromObject::=
    ReferencedObjects
    "."
    FieldName
ObjectFromObject::=
    ReferencedObjects
    "."
    FieldName
ObjectFromObject::=
    ReferencedObjects
    "."
    FieldName
ReferncedObject::=
    DefinedObject         |
    ParameterizedObject   |
    DefinedObjectSet      |
    ParameterizedObjectSet
```

注：提供"InformationFromObject"生成式是为了帮助理解和在英文上下文中使用，它不在本部分中其他地方引用。

15.2 本记法引用“ReferencedObject”关联表被引用列的全部内容。

15.3 根据“ReferencedObjects”和“FieldName”的形式，此记法可能表示值、值集合、类型、客体或客体集合。由结构“ValueFromObject”、“ValueSetFromObjects”、“TypeFromObject”、“ObjectFromObject”和“ObjectsSetFromObjects”分别表示这五种情况。每一种结构是“InformationFromObjects”的一种特例。

15.4 “InformationFromObjects”生成式可划分为两部分。第一部分通过删除最后的(或仅有的)“PrimitiveFieldName”，及其前面的句号(.)。如果第一部分表示客体或客体集合，那么，15.5 到 15.12 适用。否则此记法是不合法的。第二部分是最后的(或仅有的)“PrimitiveFieldName”。

注：(指导性的)已知下列定义：

obj. &a. &b. &c. &d

定义中的第一部分是 obj. &a. &b. &c.，第二部分是 &d。

15.5 表 1 的第一列是指 15.4 中定义的第一部分。第二列是指 15.4 中定义的第二部分。第三列是指“InformationFromObjects”5 种情况中适用的(即使有也很少)情况。

表 1 允许的“InformationFromObjects”情况

InformationFromObjects 的第一部分	InformationFromObjects 的第二部分	结 构
客体	固定类型值字段	“ValueFromObject”
	可变类型值字段	“ValueFromObject”
	固定类型值集合字段	“ValueSetFromObject”
	可变类型值集合字段	不被允许
	类型字段	“TypeFromObject”
	客体字段	“ObjectFromObject”
	客体集合字段	“ObjectSetFromObject”
客体集合	固定类型值字段	“ValueSetFromObject”
	可变类型值字段	不被允许
	固定类型值集合字段	“ValueSetFromObject”
	可变类型值集合字段	不被允许
	类型字段	不被允许
	客体字段	“ObjectSetFromObject”
	客体集合字段	“ObjectSetFromObject”

15.6 对“TypeFromObject”和“ValueSetFromObjects”，单一序列和单一集合的 XML 值记法(见 GB/T 16262.1—2006 表 5)应由信息客体中规定的“Type”确定。

15.7 如果第一部分引用一个客体，第二部分引用固定类型值集合字段，那么“ValueSetFromObjects”与具有 SimpleTableConstraint 的类型等效。此类型是“＜ClassName＞.＜FieldName＞”，其中＜ClassName＞是客体的信息客体类别，“＜FieldName＞”是第二部分引用的字段。SimpleTableConstraint 是只包含第一部分引用的客体的客体集合组成。

15.8 如果第一部分引用客体集合，第二部分引用固定类型值字段或固定类型值集合字段，那么“ValueSetFromObjects”与具有“SimpleTableConstraint”的类型等效。此类型是“＜ClassName＞.＜FieldName＞”，其中，“＜ClassName＞”是第一部分引用的客体集合的信息客体类别，“＜FieldName＞”是第二部分引用的字段。“SimpleTableConstraint”是由第一部分引用的客体集合组成。

15.9 能使用起初是空的但可扩展的信息客体集合定义“ValueSetFromObject”。凡是被一种应用使用这种信息客体集合定义的值集合时，这种信息客体集合中应至少有一个客体。

15.10 如果包含客体集合，并且最后的“PrimitiveFieldName”标识一个客体集合字段，那么“Object-

SetFromObjects"是所选客体集合的并集。

15.11 如表1所示，如果包含客体集合，并且最后的"PrimitiveFieldName"标识可变类型值或值集合字段或类型字段，则不允许这种记法。

15.12 如果列中被引用的所有字符元是空的，则不允许使用这种记法，这会使此字段成为空的(或默认的)，但使用这种记法直接定义 OPTIONAL(或 DEFAULT)信息客体的字段则是允许的。

15.13 来自客体的信息的示例

采用 11.10、11.11 和 12.11 示例中的定义，下列结构(左列)是有效的，并且可按与右列表示相等效而加以使用。

"ValueFromObject"

invertMatrix. &operationCode	7
determinantIsZero. &errorCode	1

"TypeFromObject"

invertMatrix. &ArgumentType	矩阵

"ValueSetFromObject"

invertMatrix. &Errors. &errorCode	{1}
MatrixOperations. &operationCode	{7 |和其他}

"ObjectSetFromObject"

invertMatrix. &Errors	{determinantIsZero}
MatrixOperations. &Errors	{determinantIsZero |和其他}

附 录 A
（规范性附录）
类型标识符信息客体类别

A.1 本附录规定一种有用的具有类别引用 TYPE-IDENTIFIER 的信息客体类别。

注：此信息客体类别是最简单的有用的类别，只具有两个字段，类型 OBJECT IDENTIFIER 的标识符字段和类型字段。类型字段定义含有有关此类别中任一特定客体所有信息的 ASN.1 类型。在本部分中予以定义是因为这种形式的信息客体有着广泛使用。

A.2 TYPE-IDENTIFIER 信息客体定义如下：

```
TYPE-IDENTIFIER::=CLASS
    {
        &id OBJECT IDENTIFIER UNIQUE,
        &Type,
    }
    WITH SYNTAX{&Type IDENTIFIER BY &id}
```

A.3 此类别定义为“有用的”信息客体类别，并且无须引入它即可在任一模块中得到。

A.4 示例

报文处理系统(MHS)通信的主体可定义为：

```
MHS-BODY-CLASS::=TYPE-IDENTIFIER
94FaxBody MHS-BODY-CLASS::=
    {BIT STRING IDENTIFIED BY {mhsbody 3}}
```

协议设计者通常定义一个成分以通过规定 C.10 中定义的类型 INSTANCE OF MHS-BODY-CLASS 附有 MHS-BODY-CLASS。

附 录 B
（规范性附录）
抽象语法定义

B.1 本附录规定了对定义抽象语法有用的信息客体类别 ABSTRACT-SYNTAX。

注：推荐凡是在抽象语法定义为一个 ASN.1 类型的值时要定义此信息客体类别的实例。

B.2 ABSTRACT-SYNTAX 信息客体类别定义为：

```
ABSTRACT-SYNTAX::=CLASS
    {
        &id OBJECT IDENTIFIER UNIQUE,
        &Type,
        &property BIT STRING {handles-invalid-encodings(0)} DEFAULT{}
    }
    WITH SYNTAX {
        &Type IDENTIFIED BY &id [HAS PROPERTY &property]
    }
```

当 &Type 字段包含其值构成抽象语法的一个 ASN.1 类型时，每个 ABSTRACT-SYNATAX 的 &id 字段是抽象语法名称。特性 handles-invalid-encodings 表示在解码过程期间，无效的编码不要处理成差错，以及如何处理这种无效编码留给应用去决定。

B.3 此信息客体类别定义为“有用的”，因为它具有通用性并且无须引入它即可在任一模块中得到。

B.4 示例

如果定义 ASN.1 类型叫做 XXX-PDU，那么可通过下列记法规定包含 XXX-PUD 所有值的抽象语法：

```
XXX-Abstract-Syntax ABSTRACT-SYNTAX::=
        {XXX-PDU IDENTIFIED BY{XXX 5}}
```

详细使用 ABSTRACT-SYNTAX 信息客体类别的示例见 GB/T 16262.1—2006 中附录 E 的 E.3。

B.5 经常是利用参数化类型定义抽象语法（如 GB/T 16262.4—2006 中定义的），例如用参数提供协议某些成分的界限。这些参数（受 GB/T 16262.4—2006 的第 10 章规定的限制）可在抽象语法定义时予以确定，或可作为抽象语法的参数予以推进。

附 录 C
（规范性附录）
单一实例类型

C.1 本附录规定单一实例类型（见 3.4.14）的类型和值记法。通过使用信息客体类别赋值（规定信息客体类别引用作此记法的一部分），这引起类型能够传送定义为类别 TYPE-IDENTIFIER 的信息客体类别中的信息客体的任何值。

C.2 在 GB/T 16262.1—2006 的 16.2 中引用了“InstanceOfType”记法，作为生成“Type”的记法之一，并定义如下：

InstanceOfType::=INSTANCE OF DefinedObjectClass

注：在 GB/T 16262.3—2006 第 10 章规定了通过应用“表约束”能约束此类型的方法，将此类型的值限于表示此类别某个特定信息客体集合的值。

C.3 此记法规定按照“DefinedObjectClass”的实例含有 &id 字段（OBJECT IDENTIFIER）和 &Type 值字段的类型。

注：这种结构通常由一个客体集合约束，而此客体集合常常是（但未必是）（GB/T 16262.4—2006 中 8.3～8.11 定义的）模型引用名，其实际客体集合已在别处定义。

C.4 所有单一实例类型具有通用类型的标记，号码 8。

注：这与外部类型的通用标记相同，在使用 ASN.1 基本编码规则时，使用单一实例类型能与外部类型比特兼容。

C.5 单一实例类型有一个相关的序列类型，用此相关的序列类型定义单一实例类型的值和子类型。

注：在此类型受 GB/T 16262.3—2006 约束记法约束时，此相关序列类型亦受到约束。由于对单一实例的约束从而导致对此相关序列类型的约束在 GB/T 16262.3—2006 的附录 A 中予以规定。

C.6 假设相关的序列类型在实施 EXPLCIT TAGS 的环境中予以定义。

C.7 相关的序列类型应是：

```
SEQENCE
    {
        type-id <DefinedObjectClass>. &id
        value [0]<DefinedObjectClass>. &Type
    }
```

式中“〈DefinedObjectClass〉”用于“InstanceOfType”记法中时由特定的“DefinedObjectClass”代替。

C.8 “InstanceOfType”记法的值记法“InstanceOfValue”和“XMLInstanceOfValue”应是相关序列类型的值记法。

InstanceOfValue::=Value

XMLInstanceOfValue::=XMLValue

C.9 单一实列和单一集合（见 GB/T 16262.1—2006 表 5）的 XML 值记法应是“XMLDelimitedItemList”。

C.10 示例

基于 A.4 所给示例的一个示例如下：

此类型：

INSTANCE OF MHS-BODY-CLASS

具有下列的相关序列类型：

```
SEQENCE
    {
```

```
        type-id MHS-BODY-CLASS.&id
        value[0] MHS-BODY-CLASS.&Type
    }
```

表约束应用于此类型的示例可在 GB/T 16262.3—2006 的附录 A 中找到。

附 录 D
（资料性附录）
示 例

D.1 简化的 OPERATION 类别用法举例

设 OPERATION 和 ERROR 信息客体类别的简单定义如下：

```
OPERATION ::= CLASS
    {
        &ArgumentType                   OPTIONAL,
        &ResultType                     OPTIONAL,
        &Errors                         ERROR OPTIONAL,
        &Linked                         OPERATION OPTIONAL,
        &resultReturned                 BOOLEAN DEFAULT TRUE,
        &operationCode                  INTEGER UNIQUE
    }
WITH SYNTAX
    {
        [ARGUMENT                       &ArgumentType]
        [RESULT                         &ResultType]
        [RETURN RESULT                  &resultReturned]
        [ERRORS                         &Errors]
        [LINKED                         &Linked]
        CODE                            &operationCode
    }
ERROR ::= CLASS
    {
        &ParameterType                  OPTIONAL,
        &errorCode                      INTEGER UNIQUE
    }
WITH SYNTAX
    {
        [PARAMETER                      &ParameterType]
        CODE                            &errorCode
    }
```

我们能够定义包含两个 OPERATION 客体的下列客体集合：

```
My-Operations OPERATION ::= { operationA  | operationB }
operationA OPERATION ::= {
        ARGUMENT INTEGER
        ERRORS { { PARAMETER INTEGER CODE 1000 }  | { CODE 1001 } }
```

```
        CODE 1
    }
operationB OPERATION ::= {
        ARGUMENT IA5String
        RESULT BOOLEAN
        ERRORS { { CODE 1002 } | { PARAMETER IA5String CODE 1003 } }
        CODE 2
    }
```

按下式从此客体集合提取 ERROR 客体集合。

```
My-OperationErrors ERROR ::={My-Operation. &Errors}
```

得到的客体集合是：

```
My-OperationErrors ERROR ::= {
        { PARAMETER INTEGER CODE 1000 }  |
        { CODE 1001 } |
        { CODE 1002 } |
        { PARAMETER IA5String CODE 1003 }
    }
```

按下式提取操作差错的差错代码集合：

```
My-OperationErrorCodes INTEGER ::={My-Operation. &Errors. &errorCode}
```

得到的值集合是：

```
My-OperationErrorCodes INTEGER ::={1000 | 1001 | 1002 | 1003}
```

D.2 “ObjectClassFieldType”用法举例

“ObjectClassFieldType”可用于类型规范，例如：

```
--从此类别提取“ObjectClassFieldType”。在此提取中只能使用前 5 个字段。
EXAMPLE-CLASS ::= CLASS {
        &TypeField                          OPTIONAL,
        &fixedTypeValueField                INTEGER OPTIONAL,
        &variableTypeValueField             &TypeField OPTIONAL,
        &FixedTypeValueSetField             INTEGER OPTIONAL,
        &VariableTypeValueSetField          &TypeField OPTIONAL,
        &objectField                        SIMPLE-CLASS OPTIONAL,
        &ObjectSetField                     SIMPLE-CLASS OPTIONAL
    }
WITH SYNTAX {
        [TYPE-FIELD                         &TypeField]
        [FIXED-TYPE-VALUE-FIELD             &fixedTypeValueField]
        [VARIABLE-TYPE-VALUE-FIELD          &variableTypeValueField]
        [FIXED-TYPE-VALUE-SET-FIELD         &FixedTypeValueSetField]
        [VARIABLE-TYPE-VALUE-SET-FIELD      &VariableTypeValueSetField]
        [OBJECT-FIELD                       &objectField]
        [OBJECT-SET-FIELD                   &ObjectSetField]
    }
```

```
    SIMPLE-CLASS ::= CLASS {
            &value INTEGER
        }
    WITH SYNTAX {
            &value
        }
-此类型包含使用"ObjectClassFieldType"记法规定的成分。
-在类型字段和可变类型值和值集合字段情况,得到的成分类型是开放类型。
-在固定类型值和值集合字段情况,得到的成分类型是"INTEGER"。
-- 注:在"ObjectClassFieldType"的所有下列使用中,删去约束;
-- 当引用"ObjectClassFieldType"时,通常你会使用约束。
    ExampleType ::= SEQUENCE {
            openTypeComponent1          EXAMPLE-CLASS.&TypeField,
            integerComponent1           EXAMPLE-CLASS.&fixedTypeValueField,
            openTypeComponent2          EXAMPLE-CLASS.&variableTypeValueField,
            integerComponent2           EXAMPLE-CLASS.&FixedTypeValueSetField,
            openTypeComponent3          EXAMPLE-CLASS.&VariableTypeValueSetField
        }
    exampleValue ExampleType ::= {
            openTypeComponent1          BOOLEAN : TRUE,
            integerComponent1           123,
            openTypeComponent2          IA5String : "abcdef",
            integerComponent2           456,
            openTypeComponent3          BIT STRING : '0101010101'B
        }
```

D.3 举例说明客体和客体集合的用法

下面使用 D.2 中定义的客体类别:

```
    objectA EXAMPLE-CLASS ::= {
            FIXED-TYPE-VALUE-FIELD                  123
            FIXED-TYPE-VALUE-SET-FIELD              { 1 | 2 | 3 }
            OBJECT-FIELD                            { 1 }
            OBJECT-SET-FIELD                        { { 2 } | { 3 } }
        }
    objectB EXAMPLE-CLASS ::= {
            TYPE-FIELD                              IA5String
            FIXED-TYPE-VALUE-FIELD                  456
            VARIABLE-TYPE-VALUE-FIELD               "abc"
            VARIABLE-TYPE-VALUE-SET-FIELD           { "d" | "e" | "f" }
        }
-下列客体集合包含二个定义的客体和一个内插客体。
    ObjectSet EXAMPLE-CLASS ::= {
            objectA |
```

```
    objectB |
    {
        TYPE-FIELD                          INTEGER
        FIXED-TYPE-VALUE-FIELD              789
        VARIABLE-TYPE-VALUE-SET-FIELD       { 4 | 5 | 6 }
    }
}
```

--下列定义从客体和客体集合中提取信息。

```
integerValue INTEGER ::= objectA.&fixedTypeValueField
stringValue IA5String ::= objectB.&variableTypeValueField
IntegerValueSetFromObjectA INTEGER ::= { objectA.&FixedTypeValueSetField }
StringType ::= objectB.&TypeField
objectFromObjectA SIMPLE-CLASS ::= objectA.&objectField
ObjectSetFromObjectA SIMPLE-CLASS ::= { objectA.&ObjectSetField }
SetOfValuesInObjectSet INTEGER ::= { ObjectSet.&fixedTypeValueField }
SetOfValueSetsInObjectSet INTEGER ::= { ObjectSet.&FixedTypeValueSetField }
SetOfObjectsInObjectSet SIMPLE-CLASS ::= { ObjectSet.&objectField }
SetOfObjectSetsInObjectSet SIMPLE-CLASS ::= { ObjectSet.&ObjectSetField }
```

附 录 E
（资料性附录）
ASN.1 客体集合扩展模块的个别指导附录

E.1 ASN.1 规范能够定义信息客体集合，并且通过扩展标记或使用集合运算包含可扩展客体集合使这种客体集合成为可扩展的。客体集合时的扩展标记的使用不同于类型时的使用，在类型中，它规定应用能够动态地增加/去掉通信实例用客体集合中的客体。

E.2 如果约束客体集合是可扩展的，与未被满足的约束有关的表和成分，不被认为自身存在差错。在这些情况中，如果在客体集合中没有找到 UNIQUE 字段的值，这不是差错，但是如果找到，那么，如果没有满足对此引用类型施加的约束则是差错。

附 录 F
（资料性附录）
记 法 综 述

在第7章中定义了下列词项：

objectclassreference
objectreference
objectsetreference
typefieldreference
valuefieldreference
valuesetfieldreference
objectfieldreference
objectsetfieldreference
word
CLASS
INSTANCE
SYNTAX
UNIQUE

在GB/T 16262.1—2006中定义了下列词项并用于本部分：

empty
modulereference
xmlasn1typename
"::="
"{"
"}"
","
"."
"["
"]"
":"
DEFAULT
OF
OPTIONAL
WITH

在GB/T 16262.1—2006中定义了下列生成式并用于本部分：

BuiltinValue
ElementSetSpec
NonParameterizedTypeName
ReferencedValue
Type
Value

ValueSet
XMLBuiltinValue
XMLTypedValue
XMLValue

在 GB/T 16262.4—2006 中定义了下列生成式并用于本部分：

ParameterizedObjectClass
ParameterizedObjectSet
ParameterizedObject

本部分中定义了下列生成式：

DefinedObjectClass ::=
ExternalObjectClassReference | objectclassreference | UsefulObjectClassReference
ExternalObjectClassReference ::= modulereference "." objectclassreference
UsefulObjectClassReference ::=
TYPE-IDENTIFIER |
ABSTRACT-SYNTAX
ObjectClassAssignment ::= objectclassreference "::=" ObjectClass
ObjectClass ::= DefinedObjectClass | ObjectClassDefn | ParameterizedObjectClass
ObjectClassDefn ::= CLASS "{" FieldSpec "," + "}" WithSyntaxSpec?
FieldSpec ::=
TypeFieldSpec |
FixedTypeValueFieldSpec |
VariableTypeValueFieldSpec |
FixedTypeValueSetFieldSpec |
VariableTypeValueSetFieldSpec |
ObjectFieldSpec |
ObjectSetFieldSpec
PrimitiveFieldName ::=
Typefieldreference |
Valuefieldreference |
valuesetfieldreference |
objectfieldreference |
objectsetfieldreference
FieldName ::= PrimitiveFieldName "." +
TypeFieldSpec ::= typefieldreference TypeOptionalitySpec?
TypeOptionalitySpec ::= OPTIONAL | DEFAULT Type
FixedTypeValueFieldSpec ::= valuefieldreference Type UNIQUE ? ValueOptionalitySpec ?
ValueOptionalitySpec ::= OPTIONAL | DEFAULT Value
VariableTypeValueFieldSpec ::= valuefieldreference FieldName ValueOptionalitySpec ?
FixedTypeValueSetFieldSpec ::= valuesetfieldreference Type ValueSetOptionalitySpec ?
ValueSetOptionalitySpec ::= OPTIONAL | DEFAULT ValueSet
VariableTypeValueSetFieldSpec ::= valuesetfieldreference FieldName ValueSetOptionalitySpec?
ObjectFieldSpec ::= objectfieldreference DefinedObjectClass ObjectOptionalitySpec?

```
ObjectOptionalitySpec ::= OPTIONAL | DEFAULT Object
 ObjectSetFieldSpec ::= objectsetfieldreference DefinedObjectClass ObjectSetOptionalitySpec ?
ObjectSetOptionalitySpec ::= OPTIONAL | DEFAULT ObjectSet
WithSyntaxSpec ::= WITH SYNTAX SyntaxList
SyntaxList ::= "{" TokenOrGroupSpec empty + "}"
TokenOrGroupSpec ::= RequiredToken | OptionalGroup
OptionalGroup ::= "[" TokenOrGroupSpec empty + "]"
RequiredToken ::= Literal | PrimitiveFieldName
Literal ::= word | ","
DefinedObject ::= ExternalObjectReference | objectreference
ExternalObjectReference ::= modulereference "." objectreference
ObjectAssignment ::= objectreference DefinedObjectClass "::=" Object
Object ::= DefinedObject | ObjectDefn | ObjectFromObject | ParameterizedObject
ObjectDefn ::= DefaultSyntax | DefinedSyntax
DefaultSyntax ::= "{" FieldSetting "," * "}"
FieldSetting ::= PrimitiveFieldName Setting
DefinedSyntax ::= "{" DefinedSyntaxToken empty * "}"
DefinedSyntaxToken ::= Literal | Setting
Setting ::= Type | Value | ValueSet | Object | ObjectSet
DefinedObjectSet ::= ExternalObjectSetReference | objectsetreference
ExternalObjectSetReference ::= modulereference "." objectsetreference
ObjectSetAssignment ::= objectsetreference DefinedObjectClass "::=" ObjectSet
ObjectSet ::= "{" ObjectSetSpec "}"
ObjectSetSpec ::=
    RootElementSetSpec |
    RootElementSetSpec "," "..." |
    "..." |
    "..." "," AdditionalElementSetSpec |
    RootElementSetSpec "," "..." "," AdditionalElementSetSpec
ObjectSetElements ::=
    Object | DefinedObjectSet | ObjectSetFromObjects | ParameterizedObjectSet
ObjectClassFieldType ::= DefinedObjectClass "." FieldName
ObjectClassFieldValue ::= OpenTypeFieldVal | FixedTypeFieldVal
OpenTypeFieldVal ::= Type ":" Value
FixedTypeFieldVal ::= BuiltinValue | ReferencedValue
XMLObjectClassFieldValue ::=
    XMLOpenTypeFieldVal |
    XMLFixedTypeFieldVal
XMLOpenTypeFieldVal ::= XMLTypedValue
XMLFixedTypeFieldVal ::= XMLBuiltinValue
InformationFromObjects ::= ValueFromObject | ValueSetFromObjects |
    TypeFromObject | ObjectFromObject | ObjectSetFromObjects
```

```
ReferencedObjects ::=
    DefinedObject | ParameterizedObject |
    DefinedObjectSet | ParameterizedObjectSet
ValueFromObject ::= ReferencedObjects "." FieldName
ValueSetFromObjects ::= ReferencedObjects "." FieldName
TypeFromObject ::= ReferencedObjects "." FieldName
ObjectFromObject ::= ReferencedObjects "." FieldName
ObjectSetFromObjects ::= ReferencedObjects "." FieldName
InstanceOfType ::= INSTANCE OF DefinedObjectClass
InstanceOfValue ::= Value
XMLInstanceOfValue ::= XMLValue
```