ICS 25.040
N 10

中华人民共和国国家标准

GB/T 33863.3—2017/IEC 62541-3:2010

OPC 统一架构　第 3 部分:地址空间模型

OPC unified architecture—Part 3:Address space model

(IEC 62541-3:2010,IDT)

2017-07-12 发布　　2018-02-01 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

GB/T 33863《OPC 统一架构》由以下各部分组成：

——第 1 部分：概述和概念；

——第 2 部分：安全模型；

——第 3 部分：地址空间模型；

——第 4 部分：服务；

——第 5 部分：信息模型；

——第 6 部分：映射；

第 7 部分：规约；

——第 8 部分：数据访问；

第 9 部分：报警和条件；

第 10 部分：程序；

——第 11 部分：历史访问；

——第 12 部分：发现；

——第 13 部分：聚合。

本部分是 GB/T 33863 的第 3 部分。

本部分按照 GB/T 1.1—2009 给出的规则起草。

本部分使用翻译法等同采用 IEC 62541-3:2010《OPC 统一架构　第 3 部分：地址空间模型》。

本部分由中国机械工业联合会提出。

本部分由全国工业过程测量控制和自动化标准化技术委员会(SAC/TC 124)归口。

本部分起草单位：机械工业仪器仪表综合技术经济研究所、北京三维力控科技有限公司、上海自动化仪表有限公司、重庆川仪自动化股份有限公司、西南大学、中国工程物理研究院动力部。

本部分主要起草人：王麟琨、王春喜、李云、丁露、王玉敏、丁研、张庆军、姚杰、刘枫、郑秋平。

引　言

本部分为OPC统一架构应用开发者提供了规范。本标准给出了为开发标准接口而进行分析和设计的过程，该标准接口可加快由多个供应商完成的应用开发，并实现内部操作的无缝连接。

OPC 统一架构　第 3 部分:地址空间模型

1　范围

GB/T 33863 的本部分描述了 OPC 统一架构(OPC UA)地址空间及其对象。本部分的内容是 OPC UA 元模型,它是 OPC UA 信息模型的基础。

本部分用于指导开发 OPC UA 客户端或服务器应用。

2　规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件,仅注日期的版本适用于本文件。凡是不注日期的引用文件,其最新版本(包括所有的修改单)适用于本文件。

ISO 639(所有部分)　语种名称代码(Codes for the representation of names of languages)

ISO 3166(所有部分)　世界各国和地区名称代码(Codes for the representation of names of countries and their subdivisions)

ISO/IEC 10918-1　信息技术　连续色调静态图像的数字压缩及编码　第 1 部分:要求和指南(Information technology　Digital compression and coding of continuous-tone still images　Part 1: Requirements and guidelines)

ISO/IEC 15948 信息技术　计算机图表处理　便携式网络图形(PNG):功能规范[Information technology　Computer graphics and image processing　Portable Network Graphics (PNG): Functional specification]

IEC/TR 62541-1　OPC 统一架构　第 1 部分:概述和概念(OPC unified architecture—Part 1: Overview and concepts)

IEC 62541-4　OPC 统一架构　第 4 部分:服务(OPC unified architecture—Part 4: Services)

IEC 62541-5　OPC 统一架构　第 5 部分:信息模型(OPC unified architecture　Part 5: Information model)

IEC 62541 6　OPC 统一架构　第 6 部分:映射(OPC unified architecture—Part 6: Mappings)

IEC 62541-8　OPC 统一架构　第 8 部分:数据访问(OPC unified architecture—Part 8: Data access)

IEEE 754-1985　IEEE 二进制浮点数算法标准(IEEE standard for floating-point arithmetic),http://ieeexplore.ieee.org/servlet/ipac? punumber—2355

IETF RFC 3066　语言识别标签(Tags for the identification of languages),http://tools.ietf.org/html/rfc3.66

XML Schema　第 1 部分,http://www.w3.org/TR/xmlschema-1/

XML Schema　第 2 部分,http://www.w3.org/TR/xmlschema-2/

XPATH,http://www.w3.org/TR/xpath/

3　术语、定义、缩略语和约定

3.1　术语和定义

IEC/TR 62541 1 界定的以及下列术语和定义适用于本文件。

3.1.1

数据类型 DataType

数据类型节点的实例，与 ValueRank 属性一起用来定义变量的数据类型。

3.1.2

数据变量 DataVariable

表示对象值的变量，或直接或间接用于复杂变量，这些变量总是 HasComponent 引用的目标节点。

3.1.3

事件类型 EventType

表示事件的类型定义的对象类型节点。

3.1.4

层次结构引用 Hierarchical Reference

地址空间中用于组成层次结构的引用。

注：所有层次结构引用类型都来源于层次结构引用。

3.1.5

实例声明 InstanceDeclaration

由复杂的类型定义节点所使用的用于表示其复杂结构的节点，它是类型定义使用的实例。

3.1.6

建模规则 ModelingRule

实例声明的元数据，它定义了实例声明如何用于实例，也定义了用于实例声明的子类型规则。

3.1.7

特性 Property

用于 HasProperty 引用的目标节点的变量，它描述了节点的特征。

3.1.8

源节点 SourceNode

具有到另一个节点的引用的节点。例如：在引用“A 包含 B”中，A 是源节点。

3.1.9

目标节点 TargetNode

被另一个节点引用的节点。例如：在引用“A 包含 B”中，B 是目标节点。

3.1.10

类型定义节点 TypeDefinitionNode

用于定义另一个节点的类型的节点，对象类型节点和变量类型节点都是类型定义节点。

3.1.11

变量类型 VariableType

表示变量的类型定义的节点。

3.2 缩略语

下列缩略语适用于本文件。

UA：统一架构(Unified Architecture)

UML：统一建模语言(Unified Modeling Language)

URI：统一资源标识符(Uniform Resource Identifier)

W3C：万维网联盟(World Wide Web Consortium)

XML：可扩展标记语言(Extensible Markup Language)

3.3 约定

3.3.1 地址空间图的约定

使用图来描述各个节点及其引用。图 1 描述了在这些图中使用的约定。

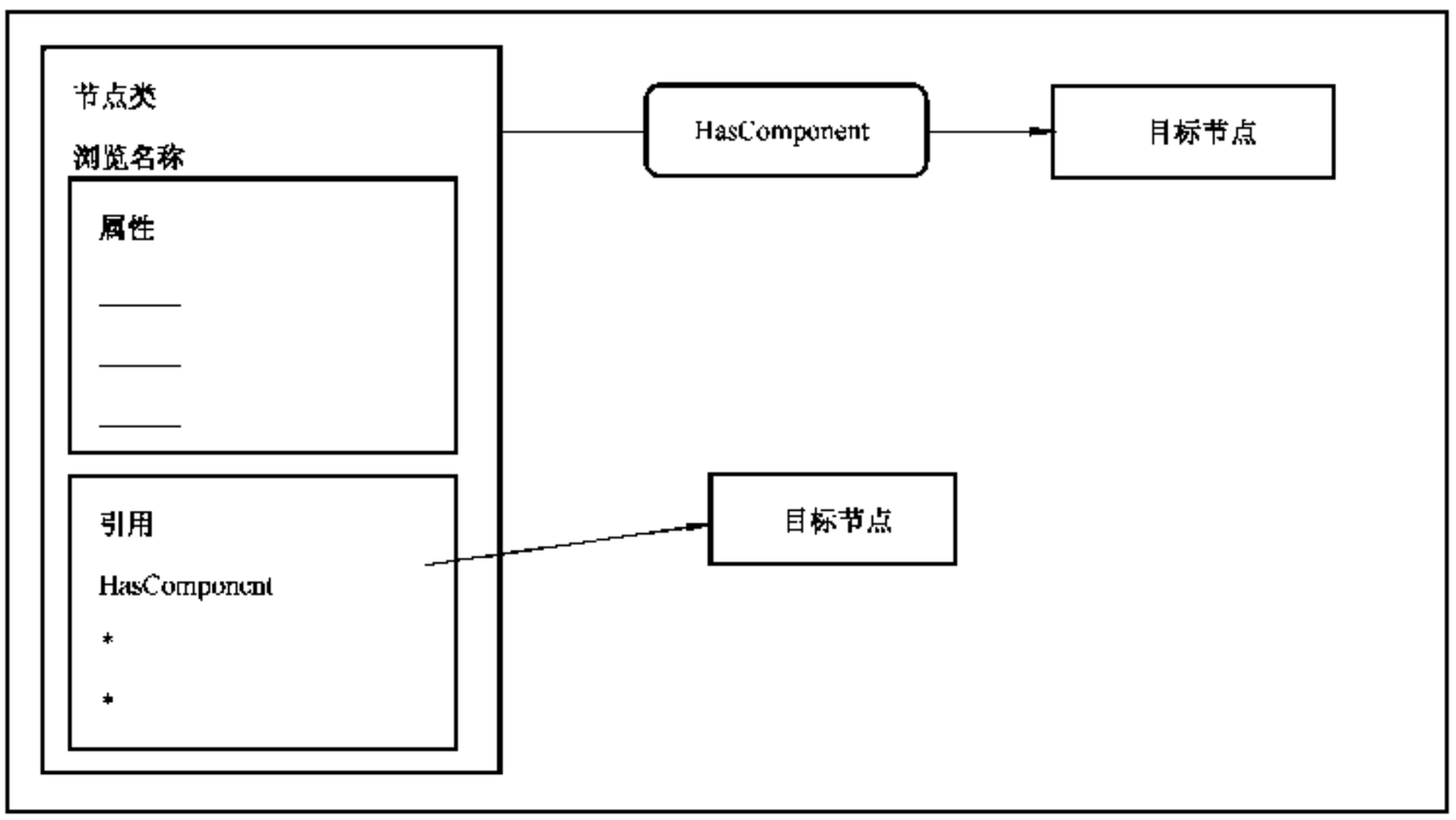


图 1 地址空间节点示图

在这些图中,矩形表示节点。节点矩形可以用一或两行文本说明。当用二行时,第一行文本表示节点类,第二行包含浏览名称。当使用一行文本时,包含浏览名称。

节点矩形中包含用于定义其属性和引用的方框。这些方框中特定的名称表示了特定的属性和引用。

圆角矩形和通过它们的箭头代表引用。箭头始于源节点止于目标节点。引用也可以通过画一个箭头来表示,该箭头从"引用"方框里的引用名称开始,在目标节点停止。

3.3.2 定义节点类的约定

第五章定义了地址空间的节点类。表 1 描述了用于定义节点类的表格式。

表 1 节点类表约定

名称	用法	数据类型	描述
属性			
"属性名称"	"M"或"O"	属性的数据类型	定义属性
引用			
"引用名称"	"1","0..1"或"0..*"	未使用	通过节点类描述引用的使用
标准的特性			
"特性名称"	"M"或"O"	特性的数据类型	定义特性

名称列包含属性的名称，用于创建引用的引用类型的名称，或者使用 HasProperty 引用来引用的特性的名称。

用法列定义了属性或特性是必备(M)的还是可选(O)的。如果是必备的，则属性或特性应存在于节点类的每个节点。对于引用，规定了所用的基数。可以应用以下的值：

——"0..*"表示没有限制，也就是说，不一定要提供引用，但是如果要提供的话，不会限制使用的次数；

——"0..1"表示引用最多提供一次；

——"1"表示引用应只提供一次。

数据类型列包含了属性或特性的数据类型的名称。此项不用于引用。

描述列包含了属性、引用或特性的描述。

仅本部分可以定义属性。因此，表中规定了所有节点类的属性并且本标准的其他部分仅可以对其进行扩展。

本部分也定义了引用类型，但是引用类型也可以通过服务器或客户端使用 IEC 62541-4 中规定的 NodeManagement 服务来定义。因此，本部分中的节点类表中可包含调用引用标识的基本引用类型，该节点类可以使用任何引用类型，包括系统规定的引用类型。节点类表只规定了节点类如何可被用作引用的源节点，而不是目标节点。如果节点类表允许其节点类的某个引用类型用作源节点，这对于引用类型的子类型也是适用的。但是，引用类型的子类可以限制其源节点。

本部分定义了特性，但是特性可以由其他标准组织或供应商定义，并且节点可以有非标准化的特性。本部分中定义的特性由它们的名称定义，这些名称映射到浏览名称，浏览名称包含了命名空间索引 0，该索引表示 OPC UA 的命名空间。

"用法"列(可选或必备)并不表示特性中具体的建模规则。不同的服务器的实施可以选择适用于它们的建模规则。

4 地址空间概念

4.1 概述

以下章节定义了地址空间的概念。第 5 章定义了代表地址空间概念的地址空间的节点类。第 6 章定义了用于对象类型和变量类型的类型模型的细节。第 7 章～第 9 章定义了标准的引用类型、数据类型和事件类型。

资料性附录 A 描述了如何使用地址空间模型的一般考虑，资料性附录 B 提供了地址空间模型的 UML 模型。规范性附录 C 将 OPC 二进制类型描述系统定义为规定数据类型结构的格式，规范性附录 D 定义了用于 OPC UA 数据的图形符号。

4.2 对象模型

OPC UA 地址空间的主要目标是给服务器提供标准方式，以向客户端表示对象。设计 OPC UA 对象模型来满足这一目标。依据变量和方法的对象。也允许表达与其他对象的关系，模型见图 2。

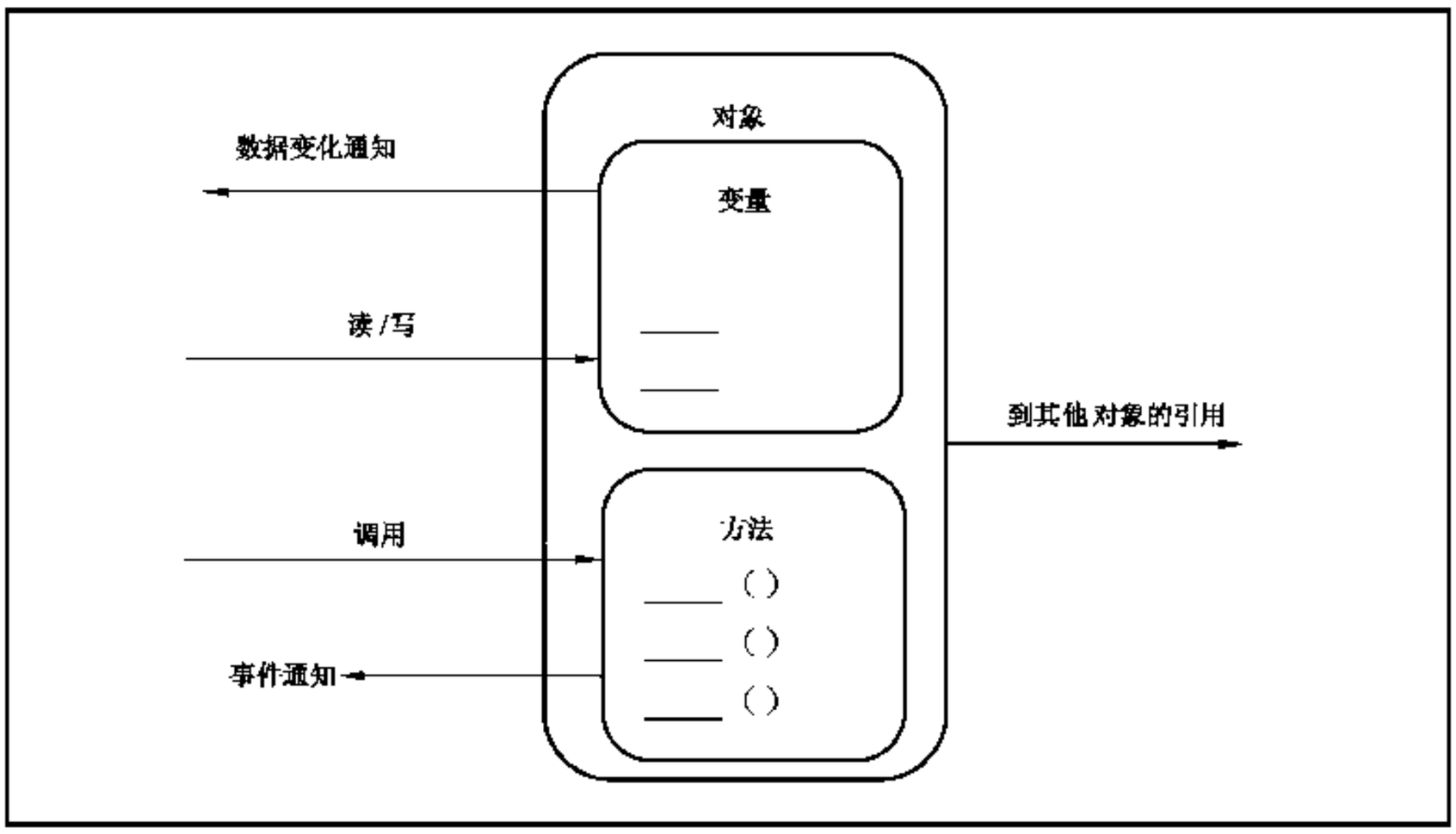


图 2　OPC UA 对象模型

在地址空间中模型的元素被称作节点。为每个节点分配节点类并且每个节点类代表对象模型的不同元素。第 5 章定义了用于代表该模型的节点类。

4.3　节点模型

4.3.1　概述

OPC UA 服务器对客户端可用的对象集合及其相关信息被称作地址空间。用于对象的模型由 OPC UA 对象模型定义(见 4.2)。

对象及其组件在地址空间中表示为节点集合,节点由属性描述并由引用互连。图 3 描述了节点模型和后续章节讨论的节点模型细节。

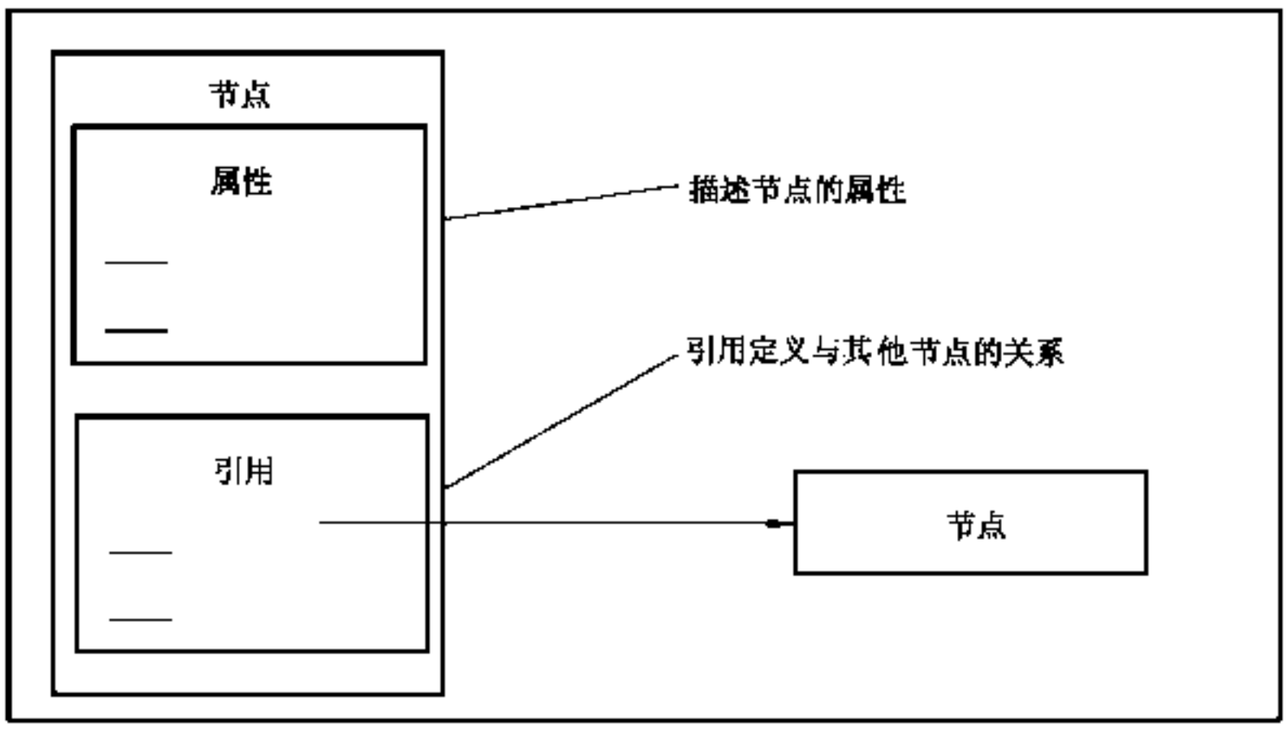


图 3　地址空间节点模型

4.3.2 节点类

节点类依据属性和引用来定义，在地址空间里定义节点时，属性和引用应被实例化(给定值)。4.3.3讨论属性，4.3.4 讨论引用。

第5章定义了用于 OPC UA 地址空间的节点类。这些节点类被全部称作地址空间的元数据。地址空间的每个节点是这些节点类的实例之一。其他节点类不应用于定义节点，因此不允许客户端和服务器定义节点类或者扩展这些节点类的定义。

4.3.3 属性

属性是描述节点的数据元素。客户端可以通过读、写、查询和订阅/监视项服务访问属性值。服务的定义见 IEC 62541 4。

属性是节点类的基本组件。第5章节点类定义包括属性的定义，因此不包括在地址空间中。

每个属性定义由属性 ID(属性 ID 见 IEC 62541-6)、名称、描述、数据类型和必备/可选指示符组成。每个节点类定义的属性集合不应由客户端或服务器扩展。

当节点在地址空间实例化时，要提供节点类属性的值。属性的必备/可选指示符指示属性是否必须被实例化。

4.3.4 引用

引用表示了相关节点间的关系，可以用 IEC 62541-4 定义的浏览和查询服务访问它们。

与属性一样，这些引用被定义为节点的基本组件。与属性不同，引用被定义为引用类型节点的实例。引用类型节点在地址空间内是可见的，并且通过引用类型节点类进行定义(见 5.3)。

包含引用的节点被指定为源节点，被引用的节点被称为目标节点。OPC UA 服务中使用源节点、引用类型和目标节点的结合可以唯一地标识引用。因此，每个节点使用相同的引用类型引用其他节点时只能使用一次。在标识引用时(见 5.3，引用类型的子类型)，对基本的具体引用类型来说，具体引用类型的任何子类型都是一样的。该引用模型见图 4。

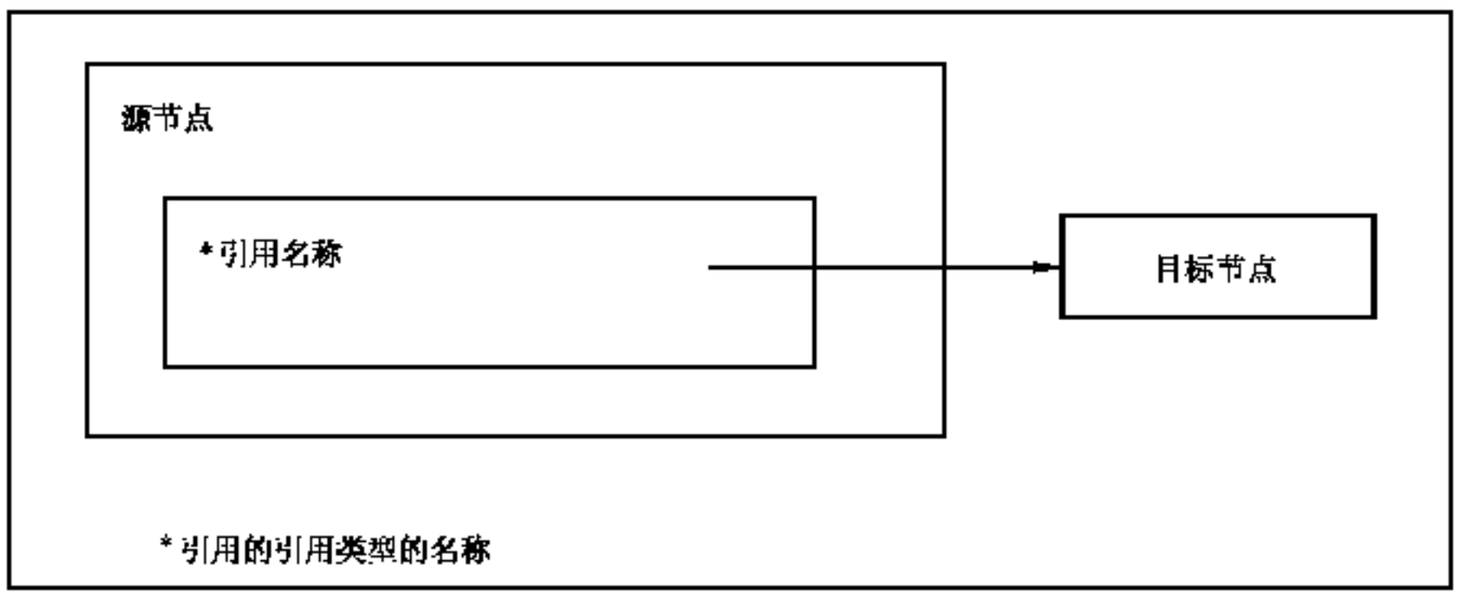


图 4 引用模型

引用的目标节点可以在相同的地址空间里，或者在另一个 OPC UA 服务器的地址空间里。其他服务器的目标节点由 OPC UA 服务标识，使用的是远程服务器名称和由远程服务器分配给该节点的标识符来标识。

OPC UA 并不要求目标节点存在，因而引用可以指向不存在的节点。

4.4 变量

4.4.1 概述

变量用来表示值。定义了两种变量类型:特性和数据变量。它们的区别在于它们代表的数据类型和是否包含其他变量。

4.4.2 特性

特性是服务器定义的对象、数据变量和其他节点的特征。特性与属性的区别在于它们表示的节点代表的内容,如设备或采购订单。属性定义了附加的元数据以便从节点类中实例化所有节点。属性对于节点类的所有节点是通用的,并且仅由本部分定义,而特性能够由服务器定义。

例如,属性定义了变量的数据类型,而特性能够用于规定某些变量的工程单位。

为了防止递归,特性不允许有定义给自身的特性。为了易于标识特性,特性的浏览名称在包含特性的节点中(详见 5.6.3)应是唯一的。

节点及其特性应总是存在于同一服务器。

4.4.3 数据变量

数据变量代表了对象的内容。例如,文件对象可以定义为字节流。字节流可以定义为由字节数据组成的数据变量。特性通常用于公开创建时间和文件对象的拥有者。

例如,如果数据变量由数据结构定义,该结构包括两个字段,"开始时间"和"结束时间",它可以是指向特定数据结构特性,如"最早开始时间"。

另一示例,控制系统的功能块可以表示为对象。功能块的参数,诸如设定点,可以表示为数据变量。功能块对象也可以有特性以描述其执行时间和类型。

数据变量可以有附加的数据变量,但仅限于复杂的情况。在这种情况下,这些数据变量应总是其复杂定义的元素。根据 4.4.2 中对特性的描述实例,服务器能够将"开始时间"和"结束时间"作为数据结构的分离元素提供。

另一示例,复杂的数据变量可以定义由三个分离的温度变送器产生的温度值的集合,这些值在地址空间也是可见的。在这种情况下,复杂数据变量能够定义 HasComponent 引用,该引用从该复杂数据变量到组成它的单个温度值。

4.5 类型定义节点(TypeDefinitionNodes)

4.5.1 概述

OPA UA 服务器应提供用于对象和变量的类型定义。HasTypeDefinition 引用用于连接一个实例,该实例的类型定义由类型定义节点(TypeDefinitionNodes)表示。虽然需要类型定义,但是 IEC 62541-5 定义了基本对象类型、特性类型和基本数据变量类型,因此如果没有可用的特定类型信息,那么服务器可以使用基本类型。对象和变量可以继承由它们的类型定义节点规定的属性(详见 6.4)。

某些情况下,HasTypeDefinition 引用的节点 ID(NodeId)对于客户端和服务器是已知的。组织可以定义工厂内已知的类型定义节点。已知的类型定义节点的节点 ID 提供共性的内容给 OPC UA 服务器,同时允许客户端解释类型定义节点而不必从服务器读取。因此,服务器可以使用已知的节点 ID 而不用在它们的地址空间响应类型定义节点。但是,应提供类型定义节点给一般客户端。这些类型定义节点可以存在于其他服务器中。

图 5 描述了使用 HasTypeDefinition 引用的示例。本示例中,设定值参数"SP"表示地址空间里的数据变量。数据变量是对象的一部分,并没有在图中表示出来。

为了提供一个能被另一对象使用的通用设置值定义,要使用特定的变量类型。每个使用此通用定

义的设定值数据变量将有 HasTypeDefinition 引用以标识通用的“设定值”变量类型。

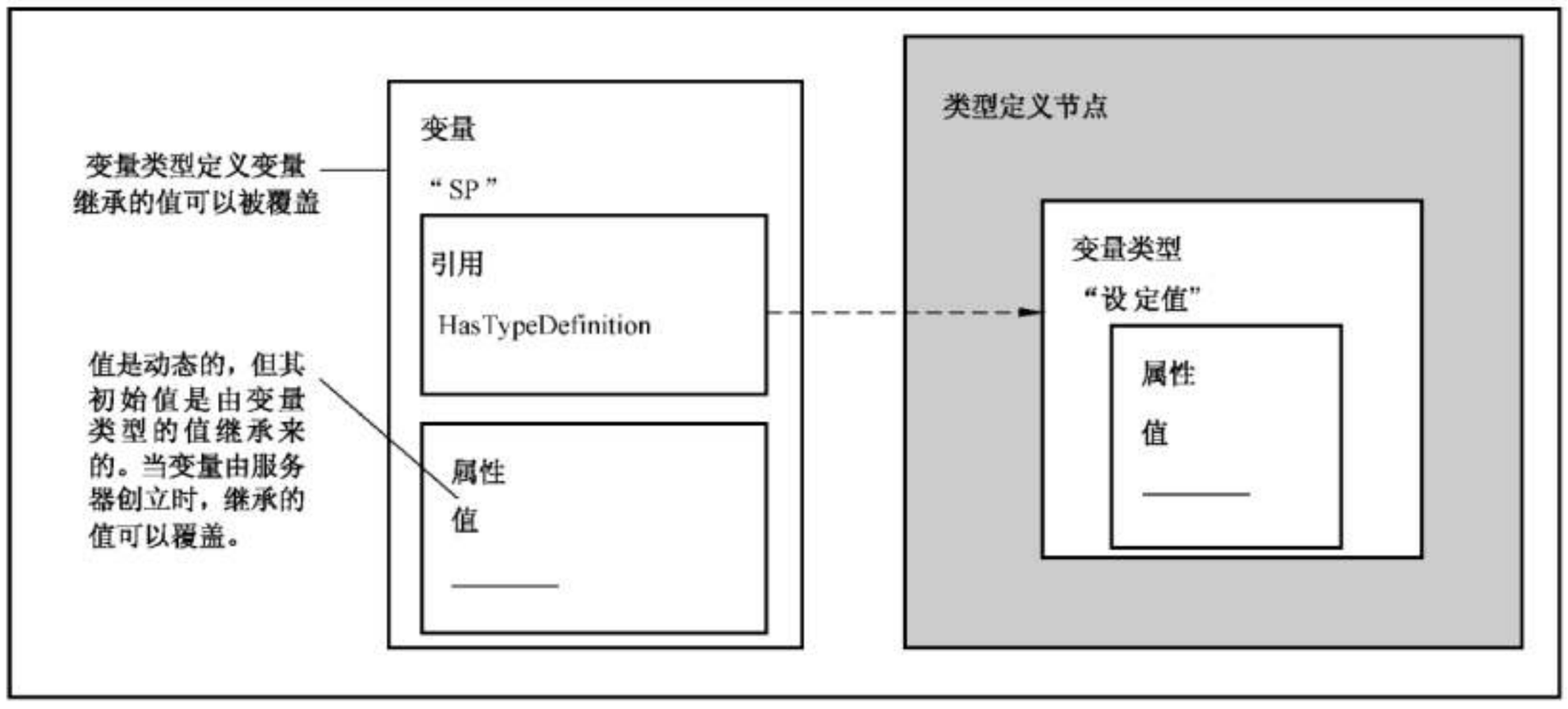


图 5　变量类型定义的变量示例

4.5.2　复杂的类型定义节点及其实例声明

类型定义节点可能是复杂的。复杂类型定义节点也定义到其他节点的引用作为类型定义的一部分。6.4.4 定义的建模规则规定了在建立类型定义的实例时这些节点是如何处理的。

类型定义节点(TypeDefinitionNodes)引用实例而不是其他类型定义节点，这可允许相同类型的几个实例使用唯一的名称、允许定义缺省值以及允许对特定实例(为此复杂类型定义节点特定的而不是为该实例的特定的)增加引用，特定实例指复杂的类型定义节点，而不是本实例中的类型定义节点。例如，图 6 中，对象类型“AL_BLK_TYPE”，表示一个功能块，有 HasComponent 引用到变量类型“SetPoint”的变量“SP”。“AL_BLK_TYPE”能够有附加的设置值变量，这些变量类型相同名称不同。可以对不是通过其类型定义节点“SetPoint”定义的变量增加特性，也可以给“SP”定义缺省值，也就是说，“AL_BLK_TYPE”的每个实例可以有初始设置该值的变量“SP”。

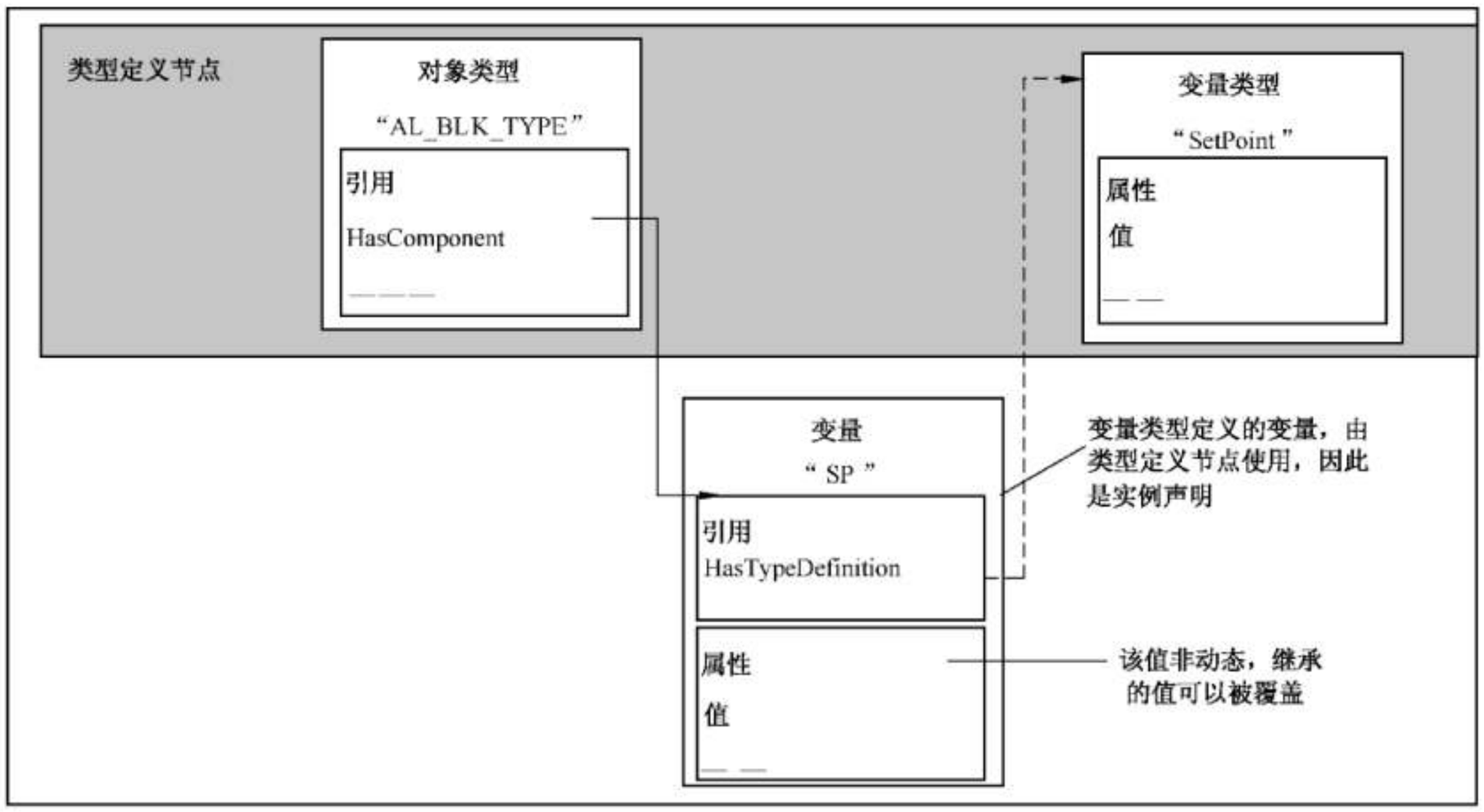


图 6　复杂类型定义示例

该方法通常用于面向对象的编程语言，在这种语言中，类变量被定义为其他类的实例。当实例化该

类时，每个变量也被实例化，但具有为包含类定义的缺省值（构造函数值）。也就是说，通常组件类的构造函数先运行，随后是包含类的构造函数运行。包含类的构造函数可以覆盖由组件类设置的组件值。

为区别代表实际数据的实例和类型定义的实例，类型定义的实例称做实例声明。但是，该术语用于简化该规范，如果实例是实例声明或在随后引用中的地址空间里不是唯一可见的，一些实例可以分享并且可以被类型定义节点、实例声明和实例所引用。这与面向对象编程语言中的类变量类似。

4.5.3 子类型化（Subtyping）

本部分允许类型定义的子类型化。子类型化规则见第6章。对象类型和变量类型的子类型化允许：

仅知道父类型的客户端能够处理子类型的实例，如果它是父类型的实例；

父类型实例能够被子类型实例代替；

继承基本类型的通用特性的专用类型。

换句话说，子类型反映了由其父类型定义的结构，但是可以增加附加的特征。例如，供应商可能希望通过增加提供下次的维护间隔的特性以扩展常用“温度传感器”变量类型。供应商可以通过建立新的变量类型，该变量类型是来自初始变量类型的 HasSubtype 引用的目标节点（TargetNode），然后为该变量类型增加新的特性。

4.5.4 复杂的类型定义节点的实例化

复杂类型定义节点的实例化依赖于6.4.4定义的建模规则。然而，类型定义实例的意图是反映由类型定义节点定义的结构。图7示出了类型定义节点“AL_BLK_TYPE”的一个实例，6.4.4.5.2定义的必备建模规则适用于其包含的变量。因此，“AL_BLK_TYPE”的一个实例，被称做 AL_BLK_1，有一个到“AL_BLK_TYPE”的 HasTypeDefinition 引用。它也包含变量“SP”，与类型定义节点使用的变量“SP”有相同的浏览名称，因此反映了类型定义节点定义的结构。

知道对象类型“AL_BLK_TYPE”的客户端能够使用该知识直接浏览该类型每个实例的包含节点。这允许针对类型定义节点进行编程。例如，客户端能够对图示元素进行编程，用同样的方式通过显示“SP”的值来处理“AL_BLK_TYPE”的所有实例。

有几个约束条件和针对类型定义节点的编程相关。类型定义节点或实例声明应绝不在前向方向使用层次引用，来引用具有相同浏览名称的两个节点。基于实例声明的实例应总是保持与派生它们的实例声明相同的浏览名称。在 IEC 62541-4 中定义的被称做 TranslateBrowsePathsToNodeId 的特定服务可以用于标识基于实例声明的实例。由于浏览名称的唯一性仅要求用于类型定义节点和实例声明，而不用于其他实例，因而使用简单的浏览服务可能是不够的。因此，“AL_BLK_1”可以有另一个具有浏览名称“SP”的变量，虽然这一个变量不是从类型定义节点的实例声明派生的。

从实例声明派生的实例应是同一类型定义节点或此类型定义节点的子类型。

类型定义节点及其实例声明应总是处于同一服务器中。但是，实例可以用它们的 HasTypeDefinition 引用指向不同服务器中的类型定义节点上。

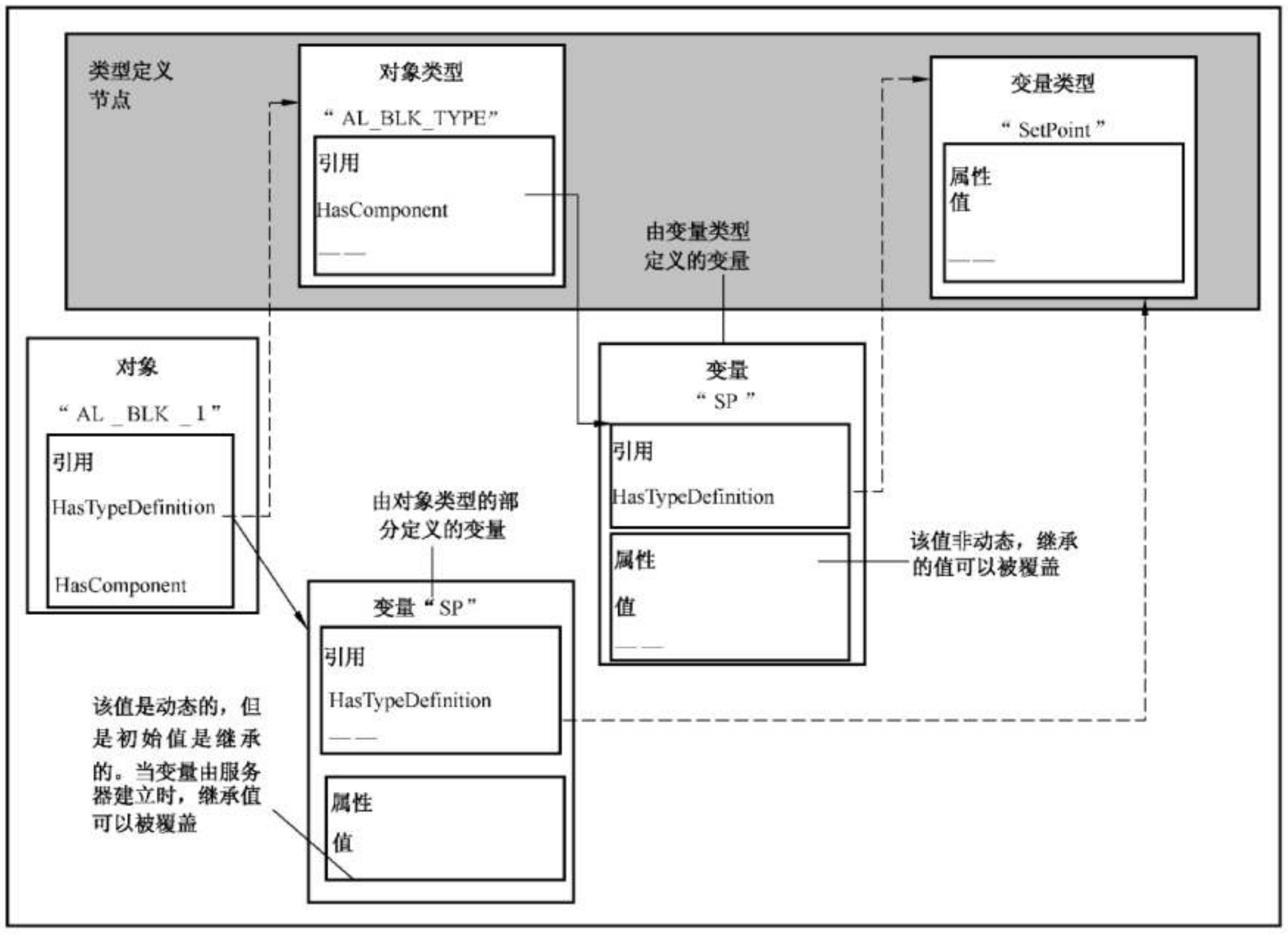


图 7　对象类型定义的对象及其组件

4.6　事件模型

4.6.1　概述

事件模型定义了可以用于不同领域市场的通常目的的事件处理系统。

事件代表了特定瞬间状态。系统组态变更和系统错误都是事件的例子。事件通知报告了事件的发生。本部分定义的事件在 OPC UA 地址空间是不直接可见的。可以用对象和视图来订阅事件。那些节点的 EventNotifier 属性标识节点是否允许订阅事件。客户端向该类节点订阅，以接收事件发生的通知。

事件订阅使用 IEC 62541-4 定义的监视和订阅服务以订阅节点的事件通知。

任何支持事件处理的 OPC UA 服务器应公开至少一个节点做为 EventNotifier。在 IEC 62541-5 定义的服务器对象用于该目的。服务器产生的事件通过该服务器对象获取。如果到事件源的连接因为某些原因关闭，那么不能期望服务器会产生事件(例如，系统离线)。

事件也可以通过地址空间里的任何其他节点来公开。这些节点(通过 EventNotifier 属性标识)提供了由服务器产生的事件的某些子集。在地址空间中的位置确定了子集的内容。例如，代表过程功能区域的过程区域对象仅能够提供起源于过程区域的事件。应当注意的是这仅仅是例子，并且这也全部是由服务器决定哪个节点应提供什么事件。

4.6.2　事件类型

每个事件都具有一个特定的事件类型。服务器可以支持许多类型。本部分定义了基本事件类型，以作为其他事件类型的来源。预计其他兼容规范将通过以本部分定义的基本类型为源，来定义其附加

的事件类型。

服务器支持的事件类型在服务器的地址空间里被公布。事件类型被表示为地址空间里的对象类型，并且没有特定的节点类与其相关。IEC 62541-5 定义了服务器公布事件类型的细节。

本部分定义的事件类型都是抽象的，因而从不在地址空间里实例化。那些事件类型的事件发生也仅通过订阅公布。事件类型存在于地址空间里，以便允许客户端发现事件类型。当与事件订阅建立并工作时，客户端会使用这一信息。由本标准的其他部分或其他兼容规范定义的事件类型，和服务器特定的事件类型一样，可以被定义为非抽象的，那么那些事件类型的实例在地址空间里可见，虽然那些事件类型的事件也可以通过事件通知机制进行访问。

标准的事件类型描述见第 9 章。它们在地址空间里的表示方法见 IEC 62541-5。

4.6.3 事件分类

事件可通过生成新的事件类型进行分类，这些新的事件类型是现存事件类型的子类型而不是现存类型的扩展。它们仅用于标识事件为新的事件类型。例如，事件类型 DeviceFailureEventType 能够被分成子类型：TransmitterFailureEventType 和 ComputerFailureEventType。这些新的子类型单纯地用于事件的分类，它们不会增加新的特性或改变从 DeviceFailureEventType 继承的语义。

也可以通过使用 7.18 和 7.20 描述的事件引用类型将事件源分组。例如，服务器可以在地址空间里定义代表事件的对象，这些事件与物理设备、或者工厂的事件区域或服务器中包含的功能性相关。事件引用应被用于确定哪些事件源表示物理设备，哪些表示基于服务器的功能。此外，可以使用引用将物理设备或基于服务器的功能性分组成有层次的事件区域。在某些情况下，事件源可分为设备和服务器功能。在这种情况下，要建立两个关系。参见事件引用类型中对附加例子的描述。

客户端能够选择事件的一种分类或几种分类，通过定义包括规定事件的事件类型项的过滤器或通过事件源分组来进行选择。这两种机制允许以多种模式分类单一事件。客户端可获取与物理设备相关的所有事件或特定设备的所有失效。

4.7 方法

方法是“轻量级”的功能，它们的范围由所属(见注)对象来界定，与面向对象编程中基于类的方法相类似；或由所属(见注)对象类型来界定，与类的静态方法相类似。方法被客户端调用，在服务器上完成，然后返回结果到客户端。方法调用实例的生命周期开始于客户端调用方法，在结果返回后结束。

注：当调用该方法时，所属对象或对象类型在服务请求中规定。

当方法可以影响到所属对象的状态时，它们没有它们自己的显性状态。在这种情况下，它们处于无状态。方法有可变数量的输入参数和返回结果参数。每个方法由方法节点类的节点描述。该节点包含元数据，以标识方法的参数和描述其行为。

通过 IEC 62541-4 定义的 Call 服务来调用方法。每个方法在已存在的会话内被调用。如果在方法执行期间，会话被终止，那么方法执行的结果不能够返回到客户端，并且被丢弃。在这种情况下，方法的执行是未定义的，即方法可以被执行直到其结束或者中断。

客户端发现服务器支持的方法，通过浏览标识其支持的方法的所属对象引用。

5 标准的节点类(NodeClasses)

5.1 概述

本章定义节点类，该节点类用于定义 OPC UA 地址空间中的节点。节点类来源于通用的基本节点类。首先定义基本节点类，接下来定义用于组织地址空间的节点类，然后定义用于代表对象的节点类。

定义代表对象的节点类分为三类：一类用于定义实例，一类用于定义那些实例的类型，一类用于定

义数据类型。6.3 描述用于子类型化的规则,6.4 描述用于类型定义实例化的规则。

5.2 基本节点类

5.2.1 概述

OPC UA 地址空间模型定义了派生所有其他节点类的基本节点类。该基本节点类代表了 OPC UA 对象模型(见 4.2)的各类组件。表 2 规定了基本节点类的属性。没有为基本节点类规定引用。

表 2 基本节点类

名称	使用	数据类型	描述
属性			
NodeId	M	NodeID	见 5.2.2
NodeClass	M	NodeClass	见 5.2.3
BrowsName	M	QualifiedName	见 5.2.4
DisplayName	M	LocalizedText	见 5.2.5
Description	O	LocalizedText	见 5.2.6
WriteMask	O	UInt32	见 5.2.7
UserWriteMask	O	UInt32	见 5.2.8
引用			没有为此节点类规定引用

5.2.2 节点 ID(NodeId)

使用称为 NodeId 的结构性标识符可明确标识节点。一些服务器可以接收在属性中表示的标准 NodeId 和其他 NodeId。NodeId 的结构定义见 8.2。

5.2.3 节点类(NodeClass)

NodeClass 属性标识节点的节点类(NodeClass)。其数据类型的定义见 8.30。

5.2.4 浏览名称(BrowsName)

节点具有 BrowsName 属性,为了创建 BrowsName 的路径,在浏览地址空间时,该属性被用作为非本地化人员可读的名称。IEC 62541-4 定义的 TranslateBrowsePathsToNodeId 服务,可被用于跟踪 BrowsName 构建的路径。

BrowsName 不宜用于显示节点名称。而 DisplayName 宜用于此目的。

不像 NodeId,不能使用 BrowsName 来唯一确定标识节点。不同的节点可以有相同的 BrowsName。

8.3 定义了 BrowsName 的结构。它包含了命名空间和字符串。虽然 BrowsName 在服务器的上下文中不是唯一的,但在某些情况下在节点(例如,节点特性)的上下文中 BrowsName 是唯一的。如果不同的组织为特性定义 BrowsName,则由组织提供的 BrowsName 的命名空间使得 BrowsName 唯一,虽然不同的组织可以使用相同的但含意稍微不同的字符串。

服务器经常选择使用相同的命名空间用于 NodeId 和 BrowsName。然而,如果它们想提供标准的特性,其 BrowsName 应有标准组织的命名空间,虽然节点 ID 的命名空间反映了另外一些内容,比如本地服务器。

建议定义标准类型定义的标准组织将其命名空间用于类型定义节点的 NodeId 和用于类型定义节点的 BrowsName。

5.2.5 显示名称(DisplayName)

DisplayName 属性包含了节点的本地化名称。如果客户端想显示节点名称给用户,它们宜使用该属性。它们不宜使用 BrowsName 用于此目的。服务器可以维护用于每个 DisplayName 的一个或多个本地化表示。当客户端开放与服务器的会话时,它们要协商被返回的区域。会话建立和区域的描述参见 IEC 62541-4。8.5 定义了 DisplayName 的结构。DisplayName 的字符串部分被限定为 512 字符。

5.2.6 描述(Description)

可选的 Description 属性应在本地化的文本中解释节点的含义,使用与在 5.2.5 中描述的用于 DisplayName 的本地化机制相同。

5.2.7 写入掩码(WriteMask)

可选的 WriteMask 属性公开了客户端写入节点属性的可能性。WriteMask 属性并不考虑任何的用户访问权,也就是说,虽然属性是可写的,但这可被限定为一定的用户/用户组。

如果 OPC UA 服务器没有能力从下面的系统得到 WriteMask 信息用于特定的属性,宜陈述其是可写的。如果在属性上调用写操作,服务器宜传输该请求并且,如果该请求被拒绝,宜返回响应状态代码。状态代码的定义见 IEC 62541-4。

WriteMask 属性是一个 32 位无符号整数,其结构定义见表 3。如果某个比特被置为 0,说明该属性是不可写的,如果置为 1,说明是可写的。如果节点不支持特定属性,那么相应比特应被置为 0。

表 3 用于 WriteMask 和 UserWriteMask 的位掩码

字段	位	描述
AccessLevel	0	指示访问等级属性是否可写
ArrayDimensions	1	指示数组大小属性是否可写
BrowseName	2	指示浏览名称属性是否可写
ContainedNoLoops	3	指示不包含环路属性是否可写
DataType	4	指示数据类型属性是否可写
Description	5	指示描述属性是否可写
DisplayName	6	指示显示名称属性是否可写
EventNotifier	7	指示事件通知器属性是否可写
Executable	8	指示可执行的属性是否可写
Historizing	9	指示历史化属性是否可写
InverseName	10	指示反转名称属性是否可写
IsAbstract	11	指示是否抽象属性是否可写
MinimumSamplingInterval	12	指示最小化样本间隔属性是否可写
NodeClass	13	指示节点类属性是否可写
NodeId	14	指示节点 ID 属性是否可写
Symmetric	15	指示对称的属性是否可写

表 3（续）

字段	位	描述
UserAccessLevel	16	指示用户访问等级属性是否可写
UserExecutable	17	指示用户可执行属性是否可写
UseWriteMask	18	指示用户写入掩码属性是否可写
ValueRank	19	指示值尺寸属性是否可写
WriteMask	20	指示写入掩码属性是否可写
ValueForVariableType	21	指示用于变量类型的值属性是否可写。由于这是由用于变量的访问等级和用户访问等级属性处理，因此它并不适用于变量。在变量中，该位应置 0
保留的	22:32	用于将来使用。应总是置为 0

5.2.8 UserWriteMask

在考虑用户访问权限的同时，可选的 UserWriteMask 属性公开客户端写节点属性的可能性。它使用和表 3 定义的 WriteMask 属性相同的位掩码。

通常情况下对每个用户来说 UserWriteMask 属性设置为不可写时，它仅能进一步限制 WriteMask 属性。

5.3 引用类型节点类

5.3.1 概述

引用被定义为引用类型节点的实例。引用类型节点在地址空间是可见的并且使用表 4 中定义的引用类型节点类来定义。与此相反，引用是节点的固有部分并且没有节点类被用于代表引用。

本部分定义了作为 OPC UA 地址空间模型固有部分的引用类型集合。这些引用类型定义见第 7 章，它们在地址空间的表示法见 IEC 62541-5。服务器也可以定义引用类型。此外，IEC 62541-4 定义了节点管理服务以允许客户端增加引用类型到地址空间。

表 4 引用类型节点类

名称	用法	数据类型	描述
属性			
BaseNodeClass Attributes	M		由基本节点类继承。见 5.2
IsAbstract	M	Boolean	本属性具有以下的值： TRUE 为抽象引用类型，即：不存在该类型的引用，仅是子类型的引用。 FALSE 不是抽象引用类型，即：该类型的引用是存在的
Symmetric	M	Boolean	本属性具有以下的值： TRUE 引用类型的含义与从源节点和目标节点中看到的相同。 FALSE 从目标节点中看到的引用类型的含义与从源节点中看到的相反

表 4（续）

名称	用法	数据类型	描述
InverseName	O	LocalizedText	引用的反向名称，引用类型的含义与目标节点的一样
引用			
HasProperty	0.. *		用于标识特性（见 5.3.3.2）
HasSubtype	0.. *		用于标识子类型（见 5.3.3.3）
标准的特性			
NodeVersion	O	String	NodeVersion 特性用于表示节点的版本。 每次增加引用或删除特性属于的节点时，都要更新 NodeVersion 特性。属性值的改变不会造成 NodeVersion 改变。客户端可以读 NodeVersion 特性或订阅它以确定节点的结构被改变的时间

5.3.2 属性

引用类型节点类继承了来自 5.2 定义的基本节点类的基本属性。被继承的 BrowseName 属性用于规定引用类型的含义，与从源节点看到的一样。例如，具有 BrowseName “包含”的引用类型被用在引用里以规定源节点包含目标节点。被继承的 DisplayName 属性包含了 BrowseName 的翻译。

引用类型的 BrowseName 在服务器中应是唯一的。不允许两个不同的引用类型有相同的 BrowseName。

IsAbstract 属性指示引用类型是否是抽象的。抽象的引用类型不能被实例化并且仅被用于组织的原因，例如规定一些常用的语义或限制继承到子类型。

Symmetric 属性被用于指示引用类型的含义对于源节点和目标节点来说是否是相同的。

如果引用类型是对称的，应省略 InverseName 属性。对称引用类型的例子是“Connects To（连接到）”和“Communicates With(进行通信)”。两者暗示了来自源节点或目标节点的语义相同。因此这两个方向被认为是前向引用。

如果引用类型是非对称的也是非抽象的，那么应设置 InverseName 属性。InverseName 属性规定了从目标节点来的引用类型的含义。非对称引用类型的例子包括了“Contains(包含)”和“Contained In(被包含到)”，和“Receives From(接收来自)”和“Sends To(发送到)”。

引用使用 InverseName，诸如“被包含到”引用，被称作反向引用。

图 8 提供了对称、非对称引用和使用 BrowseName、InverseName 的例子。

对服务器来说，并不总是能同时实例化图 8 中所示的用于非对称引用类型的前向引用和反向引用。如果出现这种情况，则该引用被称做双向的。虽然并不需要这样，但推荐将所有的层次引用实例化为双向的，以确保浏览的关联性。双向的引用被模型化为两个独立的引用。

作为单向引用的例子，经常的情况是订阅者知道其发行商，但是发行商不知道订阅者。订阅者会有“订阅到”引用到发行商，而发行商不用响应“发行到”反向引用到其订阅者。

DisplayName 和 InverseName 是唯一被标准化的指出引用类型语义的属性。与这些属性里表示的语义（例如：HasSubtype 的语义）相比，可能有更加复杂的语义与某个引用类型相关联。该部分不规定语义宜如何被公开。但是，描述属性能够被用于该目的。本部分为第 7 章中规定的引用类型提供语义。

引用类型包含限制其使用的约束条件。例如，可以规定从节点 A 开始并且仅跟随该引用类型的引用，或其子类型中的一个将不能返回 A，也就是说，“无环路”约束。

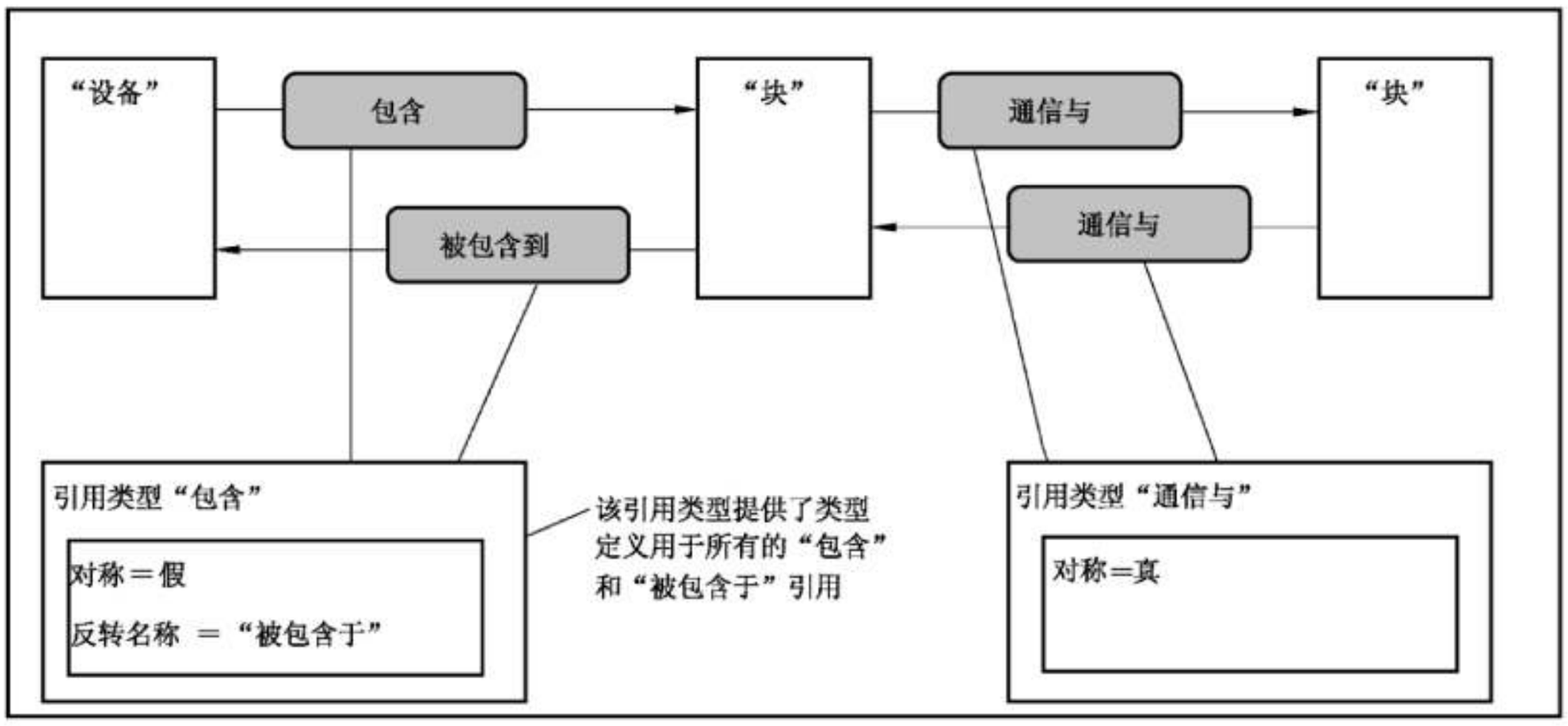


图 8　对称和非对称引用

本部分不规定那些约束能够或宜如何在地址空间里是可用的。然而，对于标准的引用类型，在第7章中规定了一些约束。本部分不限制对引用类型有效的约束种类。例如，它也能够影响对象类型。引用类型仅能被用于一些已被定义的有关联的节点类的相关节点，这是对引用类型的一种特殊约束。

5.3.3　引用

5.3.3.1　概述

HasSubtype 引用和 HasProperty 引用是唯一能以 ReferenceType 节点用作为 SourceNode 的引用类型。ReferenceType 节点不应是其他引用类型的 SourceNode。

5.3.3.2　**HasProperty 引用**

HasProperty 引用被用于标识 ReferenceType(引用类型)的特性，并且应仅与变量 NodeClass 的节点有关。

特性 NodeVersion(节点版本)被用于指示 ReferenceType 的版本。

本部分没有定义用于 ReferenceType 的附加特性。本系列标准的附加部分可以对 ReferenceType 定义附加的属性。

5.3.3.3　**HasSubtype 引用**

HasSubtype 引用被用来定义 ReferenceType 的子类型。给父类型提供 HasSubtype 引用不是必需的，但是子类型提供反向引用给其父类型是必需的。以下规则用于子类型：

——ReferenceType 的语义(例如：“跨层次”)继承给其子类型并且可以被改进(例如：“跨特定层次”)。DisplayName 以及用于非对称引用类型的 InverseName，反映了特殊性。

——如果 ReferenceType 规定了一些约束(例如：“允许无环路”)，则被继承并且仅能够被改进(例如：继承“无环路”能被改进为“应是树型－仅一个根”)，但是不能降级(例如：“允许环路”)。

——关于哪些 NodeClasses 能够被引用的约束也被继承的并且仅能被进一步限制。即，如果 ReferenceType “A”与某个 ObjectType 的对象相关，则此约束对它的子类型也适用。

——ReferenceType 应有一个父类型，在 7.2 定义的 ReferenceType 作为 ReferenceType 层次的根类型。ReferenceType 层次不支持多重继承。

5.4 视图节点类(View NodeClass)

下层的系统通常很庞大,并且客户端经常仅对数据的特殊子集感兴趣。它们不需要,或不想负担视图化那些它们不感兴趣的地址空间里的节点。

针对这个问题,本部分定义了视图的概念。每个视图定义了地址空间里的节点子集。全部地址空间是缺省视图。视图里的每个节点可以仅包含该引用的子集,与视图的创建者定义的一样。视图节点作为视图里的节点的根。使用视图节点类定义视图,规定见表5。

当浏览视图的上下文时,开始于视图节点,视图中的所有节点应是可以访问的。浏览可以多次跳转,例如,没有必要从视图节点直接浏览所有包含的节点。

视图节点不仅用可用做项进入地址空间的附加输入项,而且可作为组织地址空间的一种结构,因而可作为进入地址空间子集的唯一入口点。因此,当公开地址空间时,客户端不应忽视视图节点。用于过滤目的而不处理视图的简单客户端,可以(例如)如同处理 FolderType 类型的对象(见 5.5.3)那样处理一个视图节点。

表5 视图节点类

名 称	用法	数据类型	描 述
属性			
BaseNodeClass Attributes	M	—	由基本节点类继承,见 5.2
ContainsNoLoops	M	Boolean	如果设置为"TRUE",该属性表示视图上下文中的后继引用不包含环路,即,从包含在视图中的节点"A"开始,接下来在视图上下文中转发引用,不再返回到节点"A"。并没有规定仅有一条路径从视图节点开始到达视图中的节点。 如果设置为"FALSE",该属性表示视图上下文中的后继引用可以导致环路
EventNotifier	M	Byte	属性被用于表示节点是否能够被用于订阅事件或者能够读 EventNotifier/写历史事件。EventNotifier 是 8 位无符号整数,其结构定义见下表。 字段 / 位 / 描述 SubscribeToEvents / 0 / 表示是否可用于订阅事件(0 指示不能被用于订阅事件,1 指示能够被用于订阅事件) 保留 / 1 / 将来用。应总是 0 HistoryRead / 2 / 表示事件的历史是否是可读的(0 指示不可读,1 指示可读) HistoryWrite / 3 / 表示事件的历史是否是可写的(0 指示不可写,1 指示可写) 保留 / 4:7 / 将来用。应总是为 0 第 3 位和第 4 位也表示事件的历史在 OPC UA 服务器上是否是可用的

表 5（续）

名　　称	用法	数据类型	描　　述
引用			
HierarchicalReference	0..*		视图中的顶层节点被层次引用所引用(见 7.3)
HasProperty	0..*		HasProperty 引用标识视图的特性
标准的特性			
NodeVersion	O	String	NodeVersion 特性被用于表示节点的版本。每次增加或者删除特性所属的节点的引用时，都要更新 NodeVersion 特性。属性值的改变不会造成 NodeVersion 的改变。客户端可以读 NodeVersion 特性或者订阅它以决定节点的结构被改变的时间
ViewVersion	O	UInt32	用于视图的版本号。当从视图中增加或移除节点时，就要更新 ViewVersion 特性的值。客户端使用该特性可以检测到视图结构的变化。ViewVersion 的值应永远大于 0

视图节点类继承了 5.2 中定义的基本节点类的基本属性。它还定义了两个附加属性。

如果服务器不能够标识视图是否包含环路，那么必备的 ContainsNoLoops 属被性置为 FALSE。

必备的 EventNotifier 属性标识视图能否被用于订阅事件，事件或者发生在视图的内容里，或是视图内容的 ModelChangeEvents，或读/写事件的历史。支持事件的视图应提供在任何对象里发生的所有事件，这些时间被用作为视图内容部分的 EventNotifier。此外，也应提供发生在视图内容里的所有 ModelChangeEvents。

为避免递归(recursion)，即：获取服务器的所有事件，IEC 62541-5 中定义的服务器对象不应是任何视图的部分，因为它提供了服务器的所有事件。

视图由服务器定义。IEC 62541-4 定义的浏览和查询服务期望视图节点的节点 ID 在视图的上下文中提供这些服务。

HasProperty 引用被用于标识视图的特性。NodeVersion 特性被用于表示视图节点的版本。ViewVersion 特性表示视图内容的版本。与 NodeVersion 相反，即使从视图中增加或删除不被视图节点直接引用的节点，也要更新 ViewVersion 特性。因为对服务器来说不太可能检测到视图内容里的变化，因而该特性是可选的。如果从视图增加或删除节点，服务器也可以产生模型变化事件，描述见 9.30。本部分没有定义附加的特性用于视图。本系列标准的附加部分可以定义用于视图的附加特性。

视图可以是任何层次引用的源节点。它们不应是任何非层次引用的源节点。

5.5 对象

5.5.1 对象节点类(NodeClass)

对象被用于代表系统、系统组件、现实世界的对象和软件对象。使用对象节点类定义对象，规定见表 6。

表 6　对象节点类

<table>
<tr><th>名　　称</th><th>方法</th><th>数据类型</th><th>描　　述</th></tr>
<tr><td>属性</td><td></td><td></td><td></td></tr>
<tr><td>BaseNodeClass Attributes</td><td>M</td><td></td><td>由基本节点类继承。见 5.2</td></tr>
<tr><td>EventNotifier</td><td>M</td><td>Byte</td><td>EventNotifier 属性被用于表示节点能否被用于订阅事件或读/写历史事件。EventNotifier 是 8 位无符号整数，其结构定义见下表：
<table>
<tr><th>字段</th><th>位</th><th>描述</th></tr>
<tr><td>SubscribeTo Events</td><td>0</td><td>表示是否能够被用于订阅事件
(0 指示不能够被用于订阅事件，1 指示能够被用于订阅事件)</td></tr>
<tr><td>保留</td><td>1</td><td>保留将来用，应总是为 0</td></tr>
<tr><td>HistoryRead</td><td>2</td><td>表示事件的历史是否可读
(0 指示不可读，1 指示可读)</td></tr>
<tr><td>HistoryWrite</td><td>3</td><td>表示事件的历史是否可写
(0 指示不可写，1 指示可写)</td></tr>
<tr><td>保留</td><td>4:7</td><td>保留将来用，应总是为 0</td></tr>
</table>
第 3 位和第 4 位表示通过 OPC UA 服务器事件的历史是可读的</td></tr>
<tr><td>引用</td><td></td><td></td><td></td></tr>
<tr><td>HasComponent</td><td>0.. *</td><td></td><td>HasComponent 引用标识对象里包含的数据变量、方法和对象</td></tr>
<tr><td>HasProperty</td><td>0.. *</td><td></td><td>HasProperty 引用标识对象的特性</td></tr>
<tr><td>HasModellingRule</td><td>0..1</td><td></td><td>使用 HasModellingRule 引用，对象可以指向最多一个建模规则对象(建模规则详见 6.4.4)</td></tr>
<tr><td>HasTypeDefinition</td><td>1</td><td></td><td>HasTypeDefinition 引用指向对象的类型定义。每个对象应明确有一个类型定义，并且因此应是一个指向对象模型的 HasTypeDefinition 引用的源节点。类型定义详见 4.5</td></tr>
<tr><td>HasModelParent</td><td>0..1</td><td></td><td>HasModelParent 引用指向对象的父模型(父模型详见 6.6)</td></tr>
<tr><td>HasEventSource</td><td>0.. *</td><td></td><td>HasEventSource 引用指向对象的事件来源。该类型的引用仅被用于 EventNotifier 属性中将它们的“SubscribeToEvent”置位的对象。详见 7.18</td></tr>
<tr><td>HasNotifier</td><td>0.. *</td><td></td><td>HasNotifier 引用指向对象的通知器。该类型的引用仅被用于 EventNotifier 属性中将它们的“SubscribeToEvent”置位的对象。详见 7.20</td></tr>
<tr><td>Organize</td><td>0.. *</td><td></td><td>该引用宜仅被用于对象类型为文件夹类型的对象(见 5.5.3)</td></tr>
<tr><td>HasDescription</td><td>0..1</td><td></td><td>该引用应仅被用于对象类型为数据类型编码类型的对象(见 5.8.4)</td></tr>
</table>

表 6（续）

名　　称	方法	数据类型	描　　述
<其他引用>	0..*		对象可以包含其他引用
标准的特性			
NodeVersion	()	String	NodeVersion 特性被用于表示节点的版本。 每次引用被增加或删除特性属于的节点，都要更新 NodeVersion 引用。属性值的改变不会造成 NodeVersion 的改变。客户端可以读 NodeVersion 特性或订阅它以决定节点结构改变的时间
Icon	()	Image	当显示节点时，Icon 特性提供图片以便被客户端使用。期望 Icon 特性能包含相对较小的图片
NamingRule	()	Naming RuleType	NamingRule 特性定义了建模规则的命名规则（详见 6.4.4.2.1）。该特性仅应被用于 6.4.4 定义的类型建模规则的对象

对象节点类继承了 5.2 定义的基本节点类的基本属性。

必备的 EventNotifier 属性标识对象是否可被用于订阅事件或者读和写事件的历史。

对象节点类使用 HasComponent 引用来定义数据变量、对象和对象的方法。

使用 IHasProperty 引用来定义对象的特性。NodeVersion 特性被用于表示对象的版本。Icon 特性提供对象的图标。NamingRule 特性定义了 ModellingRule 的 NamingRule，并且应仅被用于类型 ModellingRuleType 的对象。本部分没有为对象定义附加的特性。本系列标准的附加部分可以为对象定义附加的特性。

为了规定其建模规则，对象能使用最多一个指向建模规则对象的 HasModellingRule 引用。建模规则定义见 6.4.4。

对象应使用最多一个 IHasModelParent 引用来规定其父模型（详见 6.6）。

HasNotifier 和 HasEventSource 引用被用来提供关于事件的信息，并且仅能适用于作为事件通知器的对象。详见 7.18 和 7.20。

IHasTypeDefinition 引用指向被用作对象的类型定义的对象类型。

对象可以使用任何附加的引用来定义与其他节点的关系。对被使用的引用类型和可以被引用的节点的节点类来说不进行约束。但是，除了其用于对象之外，可以由引用类型来定义限制。标准的引用类型描述见第 7 章。

如果对象被用作实例声明（见 4.5），前向层次引用所引用的所有节点在该对象的上下文里应仅有唯一的浏览名称。

如果基于实例声明创建对象，应有相同的浏览名称作为其实例声明。

5.5.2　对象类型节点类

对象类型为对象提供定义。使用对象类型节点类来定义对象类型，规定见表 7。

表 7　对象类型节点类

名　　称	用法	数据类型	描　　述
属性			
BaseNodeClass Attributes	M		继承自基本节点类。见 5.2
IsAbstract	M	布尔	布尔属性，具有以下值： True 抽象的对象类型，即：该类型应没有对象存在，仅有其子类型。 False 不是抽象的对象类型，即：该类对象可能存在
引用			
HasComponent	0.. *		HasComponent 引用标识包含在对象类型里的数据变量、方法和对象。 当该类型的对象被实例化时，是否以及如何实例化引用节点，其规定见 6.4
HasProperty	0.. *		HasProperty 引用标识对象类型的特性。 当该类型的对象被实例化时，是否以及如何实例化特性，其规定见 6.4
HasSubtype	0.. *		HasSubtype 引用标识对象类型，其为该类型的子类型。反向引用的子类型标识该类型的父类型
GeneratesEvent	0.. *		GeneratesEvent 引用标识该类型的事件实例化的类型可以创立
<其他引用>	0.. *		对象类型可以包含其他引用以便被该对象类型定义的对象来实例化
标准的特性			
NodeVersion	()	字符串	NodeVersion 特性被用于表示节点的版本。 每次从特性所属节点增加或删除引用时，都要更新 NodeVersion 特性。属性值的改变不会造成 NodeVersion 的改变。客户端可以读 NodeVersion 特性或者订阅它来决定节点结构已经改变的时间
Icon	()	图片	当显示节点时，Icon 特性提供了图片可以被客户端来使用。期望 Icon 特性包含相对较小的图片

对象类型节点类从 5.2 中定义的基本节点类继承基本属性。附加的 IsAbstract 属性表示该对象类型是否是抽象的。

对象类型节点类使用 HasComponent 引用来定义用于它的数据变量、对象和方法。

HasProperty 引用被用于标识特性。特性 NodeVersion 表示对象类型的版本。特性 Icon 提供了对象类型的图标。本部分没有为对象类型定义附加的特性。本系列标准的其他部分可以为对象类型定义附加特性。

HasSubtype 引用用于子类型对象类型。对象类型子类型从父类型继承了常用语义。子类型的常用规则见第 6 章。不要求提供 HasSubtype 引用给子类型，但是要求子类型提供反向引用给其父类型。

GeneratesEvent 引用标识对象类型的实例可以产生的事件类型。这些对象可以是特定类型的事件的源或者其子类型的一个。服务器宜创立事件引用成为双向引用。但是，当服务器不可能公开反方向

时(从该事件类型指向支持该事件类型的每个对象类型时),可以允许单方向。**注意**:对象的 EventNotifier 属性和其对象类型的 GeneratesEvent 引用是完全不相关的。为了获取相应事件通知,可产生事件的对象可能不被用作客户端订阅的对象。

GeneratesEvent 引用是可选的,即:对象可以产生不被其对象类型公开的事件类型的事件。

对象类型可以使用任何附加引用来定义与其他节点的关系。可以对使用的引用类型或可以被引用节点的节点类不加以限制。但是,除对于对象类型的使用外,可以通过引用类型来定义限制。第 7 章描述了标准的引用类型。

用层次引用引用的所有节点应在对象类型的上下文中有唯一的 BrowseName(见 4.5)。

5.5.3 标准的对象类型文件夹类型(ObjectType FolderType)

对象类型文件夹类型的定义见 IEC 62541-5。其目的是提供与组织地址空间相比没有其他语义的对象。特殊的引用类型被引入到那些文件夹对象(Folder Objects),组织引用类型(Organizes ReferenceType)。这样引用的源节点应总是对象类型文件夹类型的视图或对象;目标节点可以是任何节点类。组织引用能够用于 HasChild(HasComponent、HasProperty 等;见 7.5)和允许环路的任何结合。因此,它们能被用于跨越多个层次。

5.5.4 对象类型的对象的客户端生成

对象总是基于对象类型,即:它们有指向其对象类型的 HasTypeDefinition 引用。

客户端能够使用 IEC 62541-4 中定义的 AddNodes 服务创立对象。该服务要求规定对象的 TypeDefinitionNode(类型定义节点)。通过 AddNodes 服务创立的对象包含所有组件,这些组件由为组件规定的对象类型依赖的建模规则所定义。但是,服务器可以为不是由该对象类型定义的对象和其组件增加附加的组件和引用。这种行为是服务器决定的。对象类型仅规定应存在于该对象类型的每个对象的组件的最小集合。

除了 AddNodes 服务,ObjectTypes(对象类型)可能有具有 BrowseName(浏览名称)"Create"的特定方法。这种方法用于创立这种对象类型的对象。这种方法对于对象的创立是有用的,创立的语义宜不同于与 AddNodes 服务上下文中所期望的默认行为。例如,这些值宜直接区别于缺省值或附加的对象宜被增加等等。这种方法的输入和输出参数依赖于对象类型;唯一公共的是 BrowseName,它标识这种方法将基于该对象类型创立一个对象。除了的是创建 ObjectTypes 的对象之外,服务器不宜对具有浏览名称"Create"的对象类型提供方法用于任何其他目的。

5.6 变量

5.6.1 概述

定义两个类型的变量:特性(Properties)和数据变量(DataVariables)。虽然它们的方法不同(4.4 中描述)以及具有不同的约束(在后续条中描述),它们使用 5.6.2 描述的相同的节点类(NodeClass)。基于此节点类的特性的约束见 5.6.3,数据变量的约束见 5.6.4。

5.6.2 变量节点类

变量用于代表可能简单或复杂的值。变量由变量类型(VariableTypes)定义,规定见 5.6.5。

变量总是被定义为地址空间里其他节点的特性或者数据变量。它们从不定义自身。变量总是至少一个其他节点的一部分,但是可以与任意数量其他节点相关。使用变量节点类(Variable NodeClass)来定义变量,规定见表 8。

表 8　变量节点类

<table>
<tr><th>名　　称</th><th>用法</th><th>数据类型</th><th>描　　述</th></tr>
<tr><td>属性</td><td></td><td></td><td></td></tr>
<tr><td>BaseNodeClass Attributes</td><td>M</td><td></td><td>继承自基本节点类，见 5.2</td></tr>
<tr><td>Value</td><td>M</td><td>由 DateType 属性定义</td><td>服务器有变量的最近的值。它的数据类型由 DateType 属性定义。这是仅有的不与数据类型相联系的属性。这允许所有变量有一个被相同 Value 属性定义的值</td></tr>
<tr><td>DateType</td><td>M</td><td>NodeID</td><td>用于 Value 属性的数据类型定义的节点 ID。标准的数据类型的定义见第 8 章</td></tr>
<tr><td>ValueRank</td><td>M</td><td>Int32</td><td>这个属性表示变量的 Value 属性是否为数组，并且数组的维数是多少。
它可以有以下值：
n>1：值是具有规定维数的数组；
一维(1)：值是一维的数组；
一或多维(0)：值是一或多维的数组；
标量(−1)：值不是数组；
任意(−2)：值可以是标量或任意维数的数组；
标量或一维(−3)：值可以是标量或一维数组</td></tr>
<tr><td>ArrayDimensions</td><td>O</td><td>UInt32[]</td><td>该属性规定了用于数组值的每个维度的长度。属性意图描述变量的能力，而不是当前的大小。
每个元素的个数应与 ValueRank 属性的值相等。如果 ValueRank≤0，那么应为空。
单个维数值为 0 表示该维度长度可变。
例如，如果后面的 C 数组定义了变量：
Int32 myArray[346]；
那么该变量的数据类型能够指向 Int32，变量的 ValueRank 值为 1 并且 ArrayDimensions 表示具有一个元素的一维数组，值为 346</td></tr>
<tr><td>AccessLevel</td><td>M</td><td>字节</td><td>AccessLevel 属性表示变量的值如何被访问(读/写)，同时它是否包含当前的和/或历史的数据。AccessLevel 并不考虑任何用户的访问权力，即：虽然变量是可写的，但这会限制在特定的用户/用户组。
访问等级是 8 位无符号整数，其结构定义见下表：
<table>
<tr><th>字段</th><th>位</th><th>描述</th></tr>
<tr><td>CurrentRead</td><td>0</td><td>表示当前值是否可读
(0 表示不可读，1 表示可读)</td></tr>
<tr><td>CurrentWrite</td><td>1</td><td>表示当前值是否可写
(0 表示不可写，1 表示可写)</td></tr>
<tr><td>HistoryRead</td><td>2</td><td>表示历史值是否可读
(0 表示不可读，1 表示可读)</td></tr>
<tr><td>HistoryWrite</td><td>3</td><td>表示历史值是否可写
(0 表示不可写，1 表示可写)</td></tr>
<tr><td>SemanticChange</td><td>4</td><td>表示用于引用的变量是否创建语义变化事件(见 9.31)</td></tr>
<tr><td>保留</td><td>5:7</td><td>将来用。应总是 0</td></tr>
</table>
前面两个位也表示该变量的当前值是否是可用的，并且从位 2 开始的两个位表示变量的历史通过 OPC UA 服务器是否是可用的</td></tr>
</table>

表 8（续）

<table>
<tr><th>名　　称</th><th>用法</th><th>数据类型</th><th>描　　述</th></tr>
<tr><td>UserAccessLevel</td><td>M</td><td>Byte</td><td>UserAccessLevel 属性表示变量的值如何被访问（读/写），并且考虑到用户访问的权力，是否包含当前的或历史的数据。
UserAccessLevel 是 8 位无符号整数，其结构定义见下表：
<table>
<tr><th>字段</th><th>位</th><th>描述</th></tr>
<tr><td>CurrentRead</td><td>0</td><td>表示当前的值是否可读
（0 表示不可读，1 表示可读）</td></tr>
<tr><td>CurrentWrite</td><td>1</td><td>表示当前的值是否可写
（0 表示不可写，1 表示可写）</td></tr>
<tr><td>HistoryRead</td><td>2</td><td>表示历史的值是否可读
（0 表示不可读，1 表示可读）</td></tr>
<tr><td>HistoryWrite</td><td>3</td><td>表示历史的值是否可写
（0 表示不可写，1 表示可写）</td></tr>
<tr><td>保留</td><td>4:7</td><td>将来用。应总是 0</td></tr>
</table>
前面两个位也表示该变量的当前值是否可用，并且从位 2 开始的两个位表示变量的历史通过 OPC UA 服务器是否可用的</td></tr>
<tr><td>MinimumSamplingInterval</td><td>O</td><td>Duration</td><td>MinimumSamplingInterval 属性表示变量的“当前的”值是如何被保持的。它规定了（以毫秒计）服务器如何快速采样到值的变化（详见 IEC 62541-4 中采样区间的描述）。0 的 MinimumSamplingInterval 表示服务器能够持续性地监视项目。−1 的 MinimumSamplingInterval 指示不确定的</td></tr>
<tr><td>Historizing</td><td>M</td><td>Boolean</td><td>Historizing 属性表示服务器是否主动收集数据用于变量的历史。它与 AccessLevel 属性的区别在于识别变量是否有任何历史数据。值为“TRUE”表示服务器主动收集数据。值为“FALSE”表示服务器不主动收集数据。缺省值为“FALSE”</td></tr>
<tr><td>引用</td><td></td><td></td><td></td></tr>
<tr><td>HasModelingRule</td><td>0..1</td><td></td><td>使用 HasModelingRule 引用（建模规则详见 6.4.4），变量能够指向最多一个建模规则对象</td></tr>
<tr><td>HasProperty</td><td>0..*</td><td></td><td>HasProperty 引用标识数据变量的特性。
不允许特性用于 HasProperty 引用的源节点</td></tr>
<tr><td>HasComponent</td><td>0..*</td><td></td><td>HasComponent 引用用于复杂数据变量以标识它们图示的数据变量。
不允许特性用于该引用</td></tr>
<tr><td>HasTypeDefinition</td><td>1</td><td></td><td>HasTypeDefinition 引用指向变量的类型定义。每个变量应有一个明确地类型定义，因而应是指向变量类型的明确的 HasTypeDefinition 引用的源节点。该类型定义的描述见 4.5</td></tr>
<tr><td>HasModelParent</td><td>0..1</td><td></td><td>HasModelParent 引用指向变量的父模型（父模型见 6.6）</td></tr>
<tr><td><其他引用></td><td>0..*</td><td></td><td>数据变量可以是任何其他引用的源节点。
属性仅用于任何非分层引用的源节点</td></tr>
</table>

表 8（续）

名　　称	用法	数据类型	描　　述
标准的特性			
NodeVersion	O	String	NodeVersion 特性表示数据变量的版本。它不适用于特性。当在特性所属节点中增加或删除引用时，都要更新 NodeVersion 特性。除了 DataTpye 属性，属性值改变不会造成 NodeVersion 变化。客户端可读 NodeVersion 特性或者订阅来决定节点结构何时发生变化。 虽然变量及其数据类型的关系不使用引用建模，变为变量类型的 DataType 属性会导致 NodeVersion 特性的更新
LocalTime	O	TimeZone DataType	LocalTime 特性仅用于数据变量。它不适用于特性。 该特性是包含偏移量和 DaylightSavingInOffset 标记的结构。偏移量规定了与值相联系的源时间戳(UTC)和值被包含的地理位置的时间的区别。源时间戳的定义见 IEC 62541-4。如果 DaylightSavingInOffset 为"TRUE"，那么来自源位置的标准时/夏令时会受到影响，同时偏移量包含夏令时修正。如果为"FALSE"，那么偏移量不包括夏令时修正，并且可能影响到夏令时
DataTypeVersion	O	String	仅用于变量类型的变量 5.8 描述了数据类型字典类型和数据类型描述类型
DictionaryFragment	O	ByteString	仅用于变量类型的变量，5.8 描述了数据类型描述类型
AllowNulls	O	Boolean	AllowNulls 特性仅用于数据变量。并不适用于特性。 该特性规定了是否允许"NULL"用于数据变量的值属性。如果设置为"TRUE"，服务器可以返回"NULL"并且接收"NULL"的写入。如果设置为"FALSE"，服务器不应返回"NULL"，并且应拒绝任何写入"NULL"的要求。 如果不提供该属性，那么由服务器规定是否允许"NULL"

变量节点类从 5.2 定义基本节点类继承基本属性。变量节点类也定义属性集合以便描述变量运行时间值。Value 属性代表变量值。DataType、ValueRank 和 ArrayDimensions 属性提供描述简单和复杂值能力。

不用考虑用户访问权力，AccessLevel 属性表示变量值的可访问性。如果 OPC UA 服务器没有能力从下面系统得到 AccessLevel(访问等级)信息，则宜声明它是可读和可写的。如果在变量上调用读或写操作，服务器宜传输该要求，当该要求被拒绝时，返回相应状态编码(StatusCode)。状态编码定义见 IEC 62541-4。

当特性描述拥有该特性的节点的语义，并且特性值的改变将会产生 SemanticChangeEvents(语义改变事件)时，AccessLevel 属性的 SemanticChange(语义改变)比特应被设置。例如，描述数据变量的工程单位置位，然而包含数据变量的 Icon 特性却没有置位。这种行为确实与 IEC 62541-4 定义的状态编码的 SemanticChange 位的描述相同。然而，如果要订阅变量，则为了接收该信息，在处理变量的值以前宜查看状态编码以标识语义是否已经改变。

考虑到用户访问权力，UserAccessLevel 属性表示变量值的访问能力。如果 OPC UA 服务器没有能力从下面的系统得到任何与信息相关的服务用户访问权力，它宜使用与 AccessLevel 属性里相同的位掩码。UserAccessLevel 属性能限制 AccessLevel 属性里表示的访问能力，但是不能超过它。

MinimumSamplingInterval 属性规定了服务器能够多快地采集到变化的值。系统装载和其他因素能够大大地影响该值的精确度(服务器的质量可达到"最好情况"的性能)。

Historizing 属性指示服务器是否主动收集数据用于变量的历史。历史化变量的内容详见 IEC 62541-11。

客户端可以读或写变量值，或者监视它们值的变化，规定见 IEC 62541-4。当为自动化数据使用服务时，IEC 62541-8 定义了附加规则。

为了规定变量的 ModellingRule(建模规则)，它能使用最多一个指向建模规则对象的 HasModellingRule 引用。6.4.4 定义建模规则。

变量应使用最多一个 HasModeleParent 引用以规定其父模型(详见 6.6)。

如果基于 InstanceDeclaration(实例声明)(见 4.5)创立变量，它应有与其实例声明相同的浏览名称。

其他引用在后续的特性和数据变量中分别描述。

5.6.3 特性

特性用于定义节点的特征。使用表 8 中规定的变量节点类来定义特性。但是，限制它们的使用。

特性是任何层次的结点，因此它们不应是任何分层引用的源节点。这包括 HasComponent 或 HasProperty 引用，即：特性不包含特性也不能公开它们的复杂结构。但是，它们可以是任何非层次引用的源节点。

HasTypeDefinition 引用指向特性的变量类型。既然特性被它们的 BrowseName 唯一标识，那么所有的特性应指向特性类型，定义见 IEC 62541-5。

特性应总是在另一个节点的上下文中定义，并且应至少是一个 HasProperty 引用的目标节点。为了将它们与变量区别开，它们不应是任何 HasComponent 引用的目标节点。因此，指向变量节点的 HasProperty 引用定义该节点为特性。

特性的 BrowseName 在节点上下文中总是唯一的。对于节点不允许使用 BrowseName 相同的 HasProperty 引用指向两个变量。

5.6.4 数据变量

数据变量表示对象的内容。使用表 8 变量节点类定义数据变量。

数据变量用 HasProperty 引用来标识它们的特性。复杂的数据变量使用 HasComponent 引用来公开它们的组件数据变量。

NodeVersion 特性表示数据变量的版本。LocalTime 特性表示了，在获得值的地点上，值的源时间戳和标准时间的区别。DataTypeVersion 特性仅用于 5.8 定义的数据类型字典(DataTypeDictionaries)和数据类型描述(DataTypeDescriptions)。标准的 DictionarySegment 特性仅用于 5.8 定义的数据类型描述。AllowNull 引用表示"Null"是否允许用于 Value 属性。本部分不为数据变量定义附加的特性。本系列标准的其他部分可以为数据变量定义附加的特性。IEC 62541-8 定义了用于数据变量的特性集合。

数据变量可以用附加的引用来定义与其他节点的关系。对于可能被引用的引用类型或者节点的节点类不进行限制。然而，除了其用于数据变量，引用类型可以定义限制。标准的引用类型详见第 7 章。

数据变量拟在对象的上下文中定义。然而，复杂的数据变量可以公开其他数据变量，并且对象类型和复杂变量类型也可以包含数据变量。因此每个数据变量应是来自对象、对象类型、数据变量或者变量类型的至少一个 HasComponent 引用的目标节点。数据变量不应是任何 HasProperty 引用的目标节点。因此，指向变量节点的 HasComponent 引用标识它为数据变量。

HasTypeDefinition 引用指向用于数据变量定义类型的变量类型。如果数据变量被用作实例声明(见 4.5),那么用层次引用来引用的所有节点在向前方向上,应在该数据变量上下文中有唯一的 BrowseName。

5.6.5 变量类型节点类

变量类型用于提供变量的类型定义。使用表 9 中规定的变量类型节点类来定义变量类型。

表 9 变量类型节点类

名　　称	用法	数据类型	描　　述
属性			
BaseNodeClass Attributes	M		继承自基本节点类,见 5.2
Value	O	由 DataType 属性定义	缺省值用于该类型的实例
DataType	M	NodeID	数据类型定义的节点 ID 用于该类型的实例
ValueRank	M	Int32	这个属性表示变量的 Value 属性是否为数组,并且数组的维数是多少。 它可以有以下值: $n>1$:值是具有规定维数的数组; 一维(1):值是一维的数组; 一或多维(0):值是一或多维的数组; 标量(−1):值不是数组; 任意(−2):值可以是标量或任意维数的数组; 标量或一维(−3):值可以是标量或一维数组
ArrayDimensions	O	UInt32	该属性规定了用于数组值的每个维度的长度。属性意图描述变量的能力,而不是当前的大小。 每个元素个数应与 ValueRank 属性值相等。如果 ValueRank ≤0,那么应为空。 单个维数值为 0 表示该维度长度可变。 例如,如果后面的 C 数组定义了变量: Int32 myArray[346]; 那么该变量的数据类型能够指向 Int32,变量的 ValueRank 值为 1 并且 ArrayDimensions 表示具有一个元素的一维数组,值为 346
IsAbstract	M	Boolean	布尔属性具有以下值: TRUE　仅为其子类型的抽象变量类型,即:该类型的变量不应存在 FALSE　不是抽象变量类型,即:该类型的变量能够存在
引用			
HasProperty	0.. *		HasProperty 引用被用于表示变量类型的特性。根据 6.4.4 定义的建模规则,该类型的实例可以实例化被引用的节点
HasComponent	0.. *		HasComponent 引用用于复杂的变量类型以标识它们包含的数据变量。复杂的变量类型仅能被用于数据变量。根据 6.4.4定义的建模规则,该类型的实例可以实例化被引用的节点

表 9（续）

名　　称	用法	数据类型	描　　述
HasSubType	0..*		HasSubType 引用标识该类型的子类型的变量类型。反向的引用子类型标识该类型的父类型
GeneratesEvent	0..*		GeneratesEvent 引用标识该类型的事件实例的类型可以创建
<其他引用>	0..*		变量类型可以包含其他引用，由该变量类型定义的变量来实例化其他引用。建模规则见 6.4.4
标准的特性			
NodeVersion	O	String	NodeVersion 特性表示节点的版本。 每次在特性所属的节点中增加或删除引用时，都要更新 NodeVersion 特性。除了 DataType 属性，属性值的改变不会造成 NodeVersion 变化。客户端可以读 NodeVersion 特性或者订阅它来决定节点的结构何时发生变化。 虽然变量及其数据类型的关系不使用引用建模，变为变量类型的 DataType 属性会导致 NodeVersion 特性的更新

变量类型节点类从 5.2 定义的基本节点类继承了基本属性。变量类型节点类也定义了属性集合以描述其实例变量的缺省或者初始值。Value 属性代表了缺省值。DataType、ValueRank 和 ArrayDimension 属性提供描述简单值和复杂值的能力。IsAbstract 属性定义了该类型能否被直接实例化。

变量类型节点类使用 HasProperty 引用来定义特性，同时使用 HasComponent 引用来定义数据变量。它们是否被实例化依赖于 6.4.4 定义的建模规则。

特性 NodeVersion 表示变量类型的版本。本部分不为变量类型定义附加的特性。本系列标准的其他部分可以为变量类型定义附加特性。IEC 62541-8 定义了用于变量类型的特性的集合。

HasSubType 引用被用于子类型变量类型。变量类型子类型从父类型继承了常用语义。子类型的通用规则的定义见第 6 章。没有要求提供 HasSubType 引用给父类型，但是要求子类型提供反向引用给其父类型。

GeneratesEvent 引用标识变量类型的变量可以是特定事件类型的事件的源或者是其子类型中的一个。服务器应为 GeneratesEvent 引用提供双向的引用。然而，当服务器不能公开从事件类型指向支持事件类型的每个变量类型的反向指向时，(引用)可以允许是单方向的。

GeneratesEvent 引用是可选的，即：变量可以创建不被其变量类型公开的事件类型的事件。

变量类型可以使用任何附加的引用来定义其与其他节点的关系。在可以被引用的引用类型上或者在可以被引用的节点的节点类上不设限制。然而，除了其用于变量类型，引用类型可以定义限制。标准的引用类型详见第 7 章。

具有分层引用的被引用的所有节点应在变量类型上下文里有唯一的 BrowseNames。

5.6.6　变量类型的变量的客户端生成

变量总是基于变量类型，即：它们有指向其变量类型的 HasTypeDefinition 引用。

客户端能使用 IEC 62541-4 中定义的 AddNode 服务创建变量。该服务要求规定该变量的 TypeDefinitionNode(类型定义节点)。通过 AddNode 服务创建的变量包含依赖建模规则规定的组件的变量类型定义的所有组件。然而，服务器可以增加附加的组件和引用到不是由该变量类型定义的变量及其组件。该行为是取决于服务器。变量类型仅规定应存在于变量类型的每个变量中的最小组件集合。

5.7 方法节点类

方法定义可调用的功能。使用 IEC 62541-4 定义的 Call 服务调用方法。方法调用不在地址空间里表述。方法调用总是运行到结束并且总是在结束时返回响应。使用方法节点类定义方法，规定见表 10。

表 10 方法节点类

名　　称	用法	数据类型	描　　述
属性			
BaseNodeClass Attributes	M		继承自基本节点类。见 5.2
Executable	M	Boolean	Executable 属性表示方法当前是否可执行。("False"指示不可执行，"True"指示可执行) Executable 属性不考虑任何用户的访问权力，即：虽然方法是可执行的，这可能被限制在一定的用户/用户群内
UserExecutable	M	Boolean	UserExecutable 属性表示，考虑到用户的访问权力，方法是否是当前可执行的("False"表示不可执行的，"True"表示可执行的)
引用			
HasProperty	0..*		HasProperty 引用标识特性用于方法
HasModellingRule	0..1		方法可以使用 HasModellingRule 引用(建模规则详见 6.4.4)指向最多一个建模规则对象
HasModelParent	0..1		HasModelParent 引用指向方法的父模型(父模型详见 6.6)
GeneratesEvent	0..*		GeneratesEvent 引用标识事件的类型，方法无论何时被调用，事件都将被建立
AlwaysGeneratesEvent	0..*		AlwaysGeneratesEvent 引用标识事件的类型，方法无论何时被调用，事件都将被建立
<其他引用>	0..*		
标准的特性			
NodeVersion	O	String	NodeVersion 特性表示节点的版本。 每次在特性所属的节点中增加或删除引用时，都要更新 NodeVersion 特性。除了 DataType 属性，属性值的改变不会造成 NodeVersion 变化。客户端可以读 NodeVersion 特性或者订阅它来决定节点的结构何时发生变化
InputArgument	O	Argument[]	当调用方法时，InputArgument 特性被用于规定客户端将要使用的参数
OutArgument	O	Argument[]	OutArgument 特性规定从方法调用返回的结果

方法节点类从 5.2 定义的基本节点类继承基本属性。方法节点类不定义附加的属性。

Executable 属性表示方法是否可执行，而在不考虑用户访问权。如果 OPC UA 服务器不能从下面的系统等到 Executable 的信息，那么它宜声明它是可执行的。如果方法被调用，则服务器应传输该请求，并且如果请求被拒绝时返回相应的状态编码。状态编码定义见 IEC 62541-4。

UserExecutable 属性表示考虑到用户访问权时方法是否可执行。如果 OPC UA 服务器不能够从下面的系统得到任何与用户权限相关的信息，那么它应使用与 Executable 属性里相同的值。UserExecutable 属性能被置位为“False”，即使 Executable 属性被置位为“True”，但是如果 Executable 属性被置位为“False”，那么它将被置位为“False”。

使用 HasProperty 引用可以为方法定义特性。InputArgument 特性和 OutputArgument 规定了方法的输入参数和输出参数。两者包含在 8.6 中规定的数据类型参数的数组。一个空的数组(没有提供任何特性)表示该方法没有输入参数或者输出参数。NodeVersion 特性表示方法的版本。本部分没有为方法定义附加的特性。本系列标准的其他部分可以为方法定义附加的特性。

为了规定其建模规则，方法能够使用最多一个 HasModellingRule 引用以指向建模对象。建模规则定义见 6.4.4。

方法应使用最多一个 HasModelParent 引用以规定它的父模型(详见 6.6)。

GeneratesEvent 引用标识方法对于其每次调用可以创建一个特定事件类型的事件或者它的子模型中的一个。当方法被成功调用时，服务器可以为每个引用类型创建一个事件。

AlwaysGeneratesEvent(总是创建事件)引用标识方法对方法的每次调用应创建被规定事件类型的事件或者它的子类型中的一个。当方法被成功调用时，服务器总将为每个被引用事件类型创建一个事件。

服务器宜使 GeneratesEvent 引用双方向引用。然而，当服务器不能够公开从事件类型指向创建事件类型的每个方法的反向方向时，可以允许是单方向的。

GeneratesEvent 引用是可选的，即方法的调用可以生成事件类型的事件，该事件不被方法通过 GeneratesEvent 引用来引用。

方法可以使用附加的引用来定义与其他节点的关系。在被使用的引用类型上或者在可以被引用的节点的节点类上不进行限制。然而，除了其用于方法外，引用类型可以定义限制。标准的引用类型描述见第 7 章。

方法应总是至少一个 HasComponent 引用的目标节点，这些 HasComponent 源节点应是一个对象或对象类型。如果调用方法，其中一个节点的节点 ID 应作为参数放到 IEC 62541-4 定义的 Call 服务中，以检测方法调用的上下文。

如果方法被用作实例声明(见 4.5)，层次引用的被引用的节点在其向前的方向上应在其方法的上下文中有唯一的 BrowseName。

5.8 数据类型

5.8.1 数据类型模型

数据类型模型用于定义简单的和复杂的数据类型。数据类型用于描述变量的 value 属性的结构及其变量类型。因而，每个变量以及变量类型以其 DataType 属性指向 DataType NodeClass(数据类型节点类)的节点，见图 9。

在许多情况下，数据类型节点的 NodeId(节点 ID)-DataTypeId(数据类型 ID)-将会在客户端和服务器中已知。第 8 章定义了数据类型，IEC 62541-6 定义了它们的数据类型 ID。此外，其他组织可以定义行业内已知的数据类型。数据类型 ID 为 OPC UA 服务器提供共性，同时允许客户端不用从服务器读类型描述就可以解释值。因此，服务器可以使用已知的数据类型 ID 而无需在地址空间里表现相应的数据类型节点。

在其他情况下，数据类型和它们相应的数据类型 ID 可以由供货商定义。服务器宜试图公开该数据类型节点和有关那些数据类型的结构相关的信息用于服务器读取，虽然该信息可能不总是对服务器有用。

图 10 描述了地址空间里使用的节点以描述数据类型的结构。数据类型指向类型 DataTypeEncodingType(数据类型编码类型)的对象。每个数据类型可以有几个数据类型编码，例如“缺省的”“UA 二进制”和“XML”编码。IEC 62541-4 中的服务允许客户端请求编码或者选择“缺省的”编码。每个数据

类型编码非常明确地被一个数据类型使用。也就是说，不允许两个数据类型指向相同的数据类型编码。数据类型编码对象明确地指向一个类型数据类型描述类型的变量。数据类型描述变量属于数据类型字典变量。

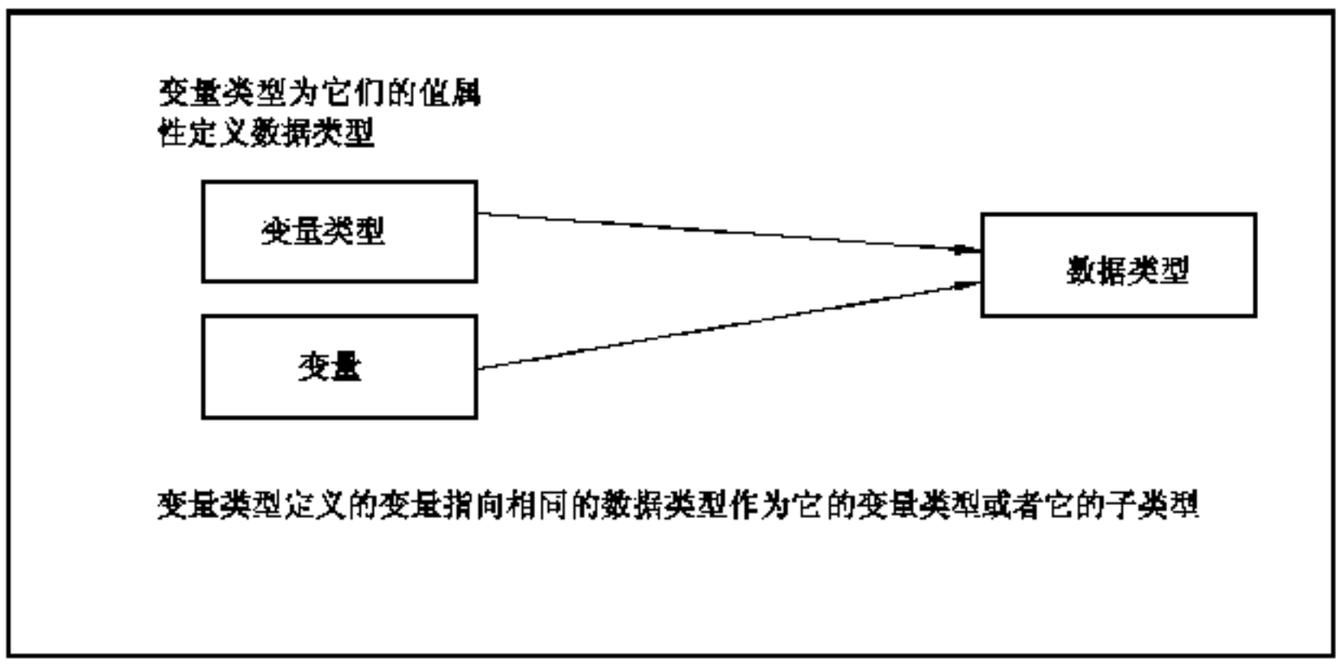


图 9　变量、变量类型及其数据类型

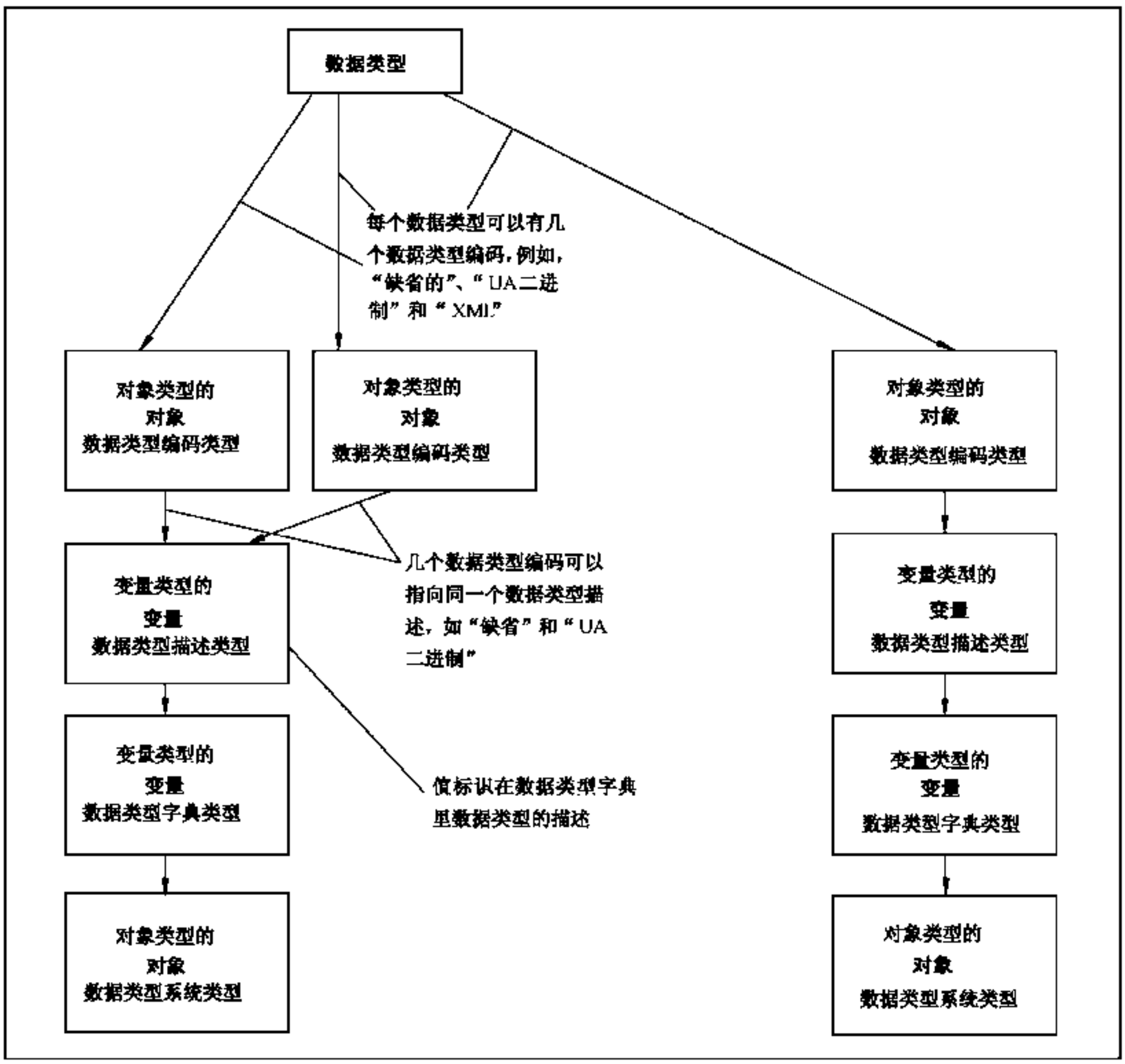


图 10　数据类型模型

由于数据类型编码的节点 ID 将在一些映射中用来标识 IEC 62541 6 中定义的数据类型和其编码，

那么那些节点 ID 对于那些已知的数据类型 ID 也可以是已知的。

数据类型字典充分详细地描述了数据类型集,以允许客户端解析/解释它们收到的变量值和构造它们发送的值。数据类型字典在地址空间里被表示为 DataTypeDictionaryType(类型数据类型字典类型)的变量,关于数据类型的描述在其 Value 属性里。地址空间里公开的所有包含的数据类型被表示为类型 DataTypeDescriptionType(数据类型描述类型)的变量。这些变量之一的值在数据类型字典的 Value 属性里标识为一个数据类型的描述。

DataTypeDictionary 变量的数据类型总是一个 ByteString(字节字符串)。在字节字符串里用于定义数据类型的格式和约定由 DataTypeSystems(数据类型系统)定义。数据类型系统由节点 ID 标识。它们在地址空间里被表示为对象类型 DataTypeSystemType(数据类型系统类型)的对象。代表数据类型字典的每个变量引用数据类型系统对象以标识它们的数据类型系统。

客户端应识别数据类型系统以解析任何类型描述信息。不识别数据类型系统的 OPC UA 客户端将不能够理解它的类型描述,必然地,也不理解被它们描述的值。在这些情况下,客户端将这些值理解为不可读的字节字符串。

OPC 二进制和 W3C XML 结构(Schema)是数据类型系统的例子。OPC 二进制数据类型系统的定义见附录 C。OPC 二进制使用 XML 来描述二进制数据值。对 W3C XML 结构的规定见 XML 结构的第 1 部分和 XML 结构的第 2 部分。

5.8.2 不同数据类型的编码规则

根据不同种类的数据类型编码及其在地址空间里是否被表述,不同种类的数据类型是有区别的并且处理时也不相同。

内置的数据类型是数据类型的固定集合(内置的数据类型的完整列表见 IEC 62541 6)。虽然编码宜为所有的 OPC UA 产品所熟知,它们在地址空间里的编码是不可见的。内置式数据类型的例子是 Int32(见 8.26)和 Double(见 8.12)。

简单数据类型是内置式数据类型的子类型。它们像内置式数据类型一样被处理,即:不能从它们内置的父类型识别它们。关于编码虽然它们像内置数据类型一样被处理,但是它们没有在地址空间里定义的编码。客户端能够读取变量的 DataType 属性或者变量类型以标识 Value 属性的简单数据类型。简单数据类型的例子是持续时间。将它作为 Double 进行处理,但是客户端可以读取 DataType 属性因而理解持续时间(见 8.13)定义的值。

结构化数据类型是表现结构化数据的数据类型并且没有被定义为内置的数据类型。结构化数据类型直接或间接继承自 8.33 定义的数据类型结构。结构化数据类型可以有几种编码,同时编码被公开在地址空间里。如何定义结构化数据类型的编码见 IEC 62541-6。结构化数据类型的编码随着每个值被发送,因而,客户端不用读取 DataType 属性就可以知道数据类型。编码应被发送以便客户端能够解释该数据。结构化数据类型的例子是参数(见 8.6)。

枚举型数据类型表示被命名值的离散集合的数据类型。枚举型总是按 Int32 进行编码,定义见 IEC 62541-6。枚举型数据类型直接或间接继承自 8.14 定义的数据类型枚举。枚举型在地址空间里的编码不公开。为了公开人们可读取的枚举值的表示,数据类型节点可以有包含了 LocalizedText 数组的 EnumString 特性。该枚举值的整数表示法指向了该数组的位置。枚举数据类型的例子是 8.30 定义的 NodeClass 节点类。

除了以上描述的数据类型,也支持抽象的数据类型,该类型没有任何编码并且不能够在线路上进行交换。变量和变量类型使用抽象数据类型以表示它们的值可以是抽象数据类型的子类型中的任何一个。抽象数据类型的例子见 8.24 定义的 Integer(整型)。

5.8.3 数据类型节点类

数据类型节点类描述了变量值的语法。数据类型可以是简单的或者复杂的,这取决于数据类型系统。使用数据类型节点类定义数据类型,规定见表 11。

表 11　数据类型节点类

名　　称	使用	数据类型	描　　述
属性			
BaseNodeClass Attributes	M		继承自基本节点类，见 5.2
IsAbstract	M	Boolean	布尔属性，具有以下值： True　是抽象数据类型 False　不是抽象数据类型
引用			
HasProperty	0.. *		HasPropert 引用表示特性用于数据类型
HasSubType	0.. *		HasSubType 引用可以用于跨越数据类型分层
HasEncoding	0.. *		HasEncoding 引用表示数据类型的编码被看作类型对象类型编码类型的对象。 仅具体的结构化数据类型可以使用 HasEncoding 引用。抽象的，内置的，枚举的和简单的数据类型不允许用于 HasEncoding 引用的源节点。 每个具体的结构化数据类型将指向最少一个数据类型编码对象，其浏览名称“缺省二进制”或“缺省 XML”，命名空间索引 0。数据类型编码对象的 BrowseName 在数据类型的上下文中应是唯一的，即：数据类型不应指向具有相同 BrowseName 的两个数据类型编码
标准的特性			
NodeVersion	()	String	NodeVersion 特性表示节点的版本。 每次在特性所属的节点中增加或删除引用时，都要更新 NodeVersion 特性。除了 DataType 属性，属性值的改变不会造成 NodeVersion 变化。客户端可以读 NodeVersion 特性或者订阅它来决定节点的结构何时发生变化
EnumString	()	Localized Text[]	EnumString 特性仅用于枚举数据类型。它不应适用于其他数据类型。在该特性中，本地化文本的数组的每个条目表示了被枚举值是人们可读取的。枚举值的整形表示指向数组的位置

数据类型节点类从 5.2 定义的基本节点类继承了基本属性。IsAbstract 属性规定了数据类型是否是抽象数据类型。抽象数据类型可以用于地址空间，即：变量和变量类型可以用它们的 DataType 属性指向一个抽象数据类型。然而，具体值从不是抽象的数据类型，并且总是抽象数据类型的具体子类型。

HasProperty 引用被用于标识数据类型的特性。NodeVersion 特性表示数据类型的版本。该版本不受数据类型字典的 DataTypeVersion 特性和数据类型描述的影响。EnumString 特性包含人可读的枚举值的表示并且仅用于枚举数据类型。本部分没有为数据类型定义附加的特性。本系列标准的其他部分可以为数据类型定义附加的特性。

HasSubType 引用可以用于公开地址空间里的数据类型层次结构。该层次应反映数据类型字典里规定的层次。子类型化的语义依赖于数据类型系统。服务器不需要提供 HasSubType 引用，即便它们的数据类型跨越类型层次。客户端不宜对具有那些信息的任何其他语义做假设，而是通过数据类型字典提供。例如，不太可能把一个数据类型的值投向它的基本数据类型。

HasEncoding 引用从数据类型指向其数据类型编码。根据这样的引用，为了被使用的编码，客户端

可以浏览到描述数据类型结构的编码类型字典。每个具体的结构化数据类型可以指向许多数据类型编码，但是每个数据类型编码应发送一个数据类型，也就是说，不允许使用 HasEncoding 引用，将两个数据类型节点指向相同的数据类型编码对象。

抽象数据类型不是 HasEncoding 引用的源节点。抽象数据类型的数据类型编码由其具体子类型提供。

数据类型节点不应是引用的其他类型的源节点。然而，它们可以是其他引用的目标节点。

5.8.4 数据类型字典、数据类型描述、数据类型编码和数据类型系统

数据类型字典(DataTypeDictionaries)是包含了类型描述集合的实体，诸如，XML 结构。数据类型字典被定义为变量类型 DataTypeDictionaryType(数据类型字典类型)的变量。

数据类型系统(DataTypeSystems)规定了在数据类型字典里定义数据类型的格式和约定。数据类型系统被定义为对象类型 DataTypeDictionaryType(数据类型系统类型)的对象。

为使对象类型 DataTypeSystemType 的对象和变量类型 DataTypeDictionaryType 的变量发生关系的引用类型是 HasComponent 引用类型。因而，变量总是 HasComponent 引用的目标节点，这是对变量的要求。然而，对于数据类型字典因为当处理数据类型字典时，非常有必要知道数据类型系统，服务器将总是提供反向引用。

数据类型字典的例子是包含 XML 结构的 XML 文档。既然这样，数据类型系统是 W3C XML 结构，在这个结构文档中的上层元素声明是数据类型描述。这些描述的每一个都定义在 XML 结构的不同版本中，使用的是相同的 XML 目标命名空间。在服务器的地址空间里，这个目标命名空间被用作数据类型 ID 的命名空间组件。因为能在其他 XML 结构中使用相同的目标命名空间，客户端应知道具有相同命名空间的两个数据类型 ID 没必要定义在一个相同的数据类型字典里。

改变可以是对类型描述的变化的结果，但是这更像是字典的改变是类型描述的增加或删除的结果。这包括了当服务器离线时所做的改变，那么当服务器再启动时，新版本是可用的。客户端可以订阅 DataTypeVersion 特性以便决定数据类型字典从它上次被读取后是否已经改变了。

服务器可以，但并不是必要的，通过 Value 属性使得数据类型字典的内容对于客户端是可用的。客户端宜假设数据类型字典的内容相对来说是很大的，并且如果它们自动地读取数据类型字典内容，每次它们遇到特定数据类型的实例时，它们将遇到性能问题。客户端宜使用 DataTypeVersion 特性来决定本地的缓存副本是否有效。如果客户端探测到 DataTypeVersion 的改变，那么应重读数据类型字典。这意味着，即使是重新启动，DataTypeVersion 应由服务器更新，即便客户端可以持续地存储本地的缓存副本。

包含类型描述的数据类型字典的值属性是字节字符串，其格式由数据类型系统定义。对于“XML 结构”数据类型系统，字节字符串包含一个有效的 XML 架构文档。服务器应确保字节字符串内容的任何改变都要与 DataTypeVersion 特性的相应的改变相匹配。换言之，客户端可以安全地使用数据类型字典的缓存副本，只要数据类型版本保持不变。

数据类型字典是复杂的变量，它用 HasComponent 引用公开它们的数据类型描述为变量。数据类型描述提供了必要信息以发现数据类型字典范围内的数据类型的正式描述。数据类型描述的值依赖于数据类型字典的数据类型系统。当使用“OPC 二进制”字典时，值应被命名为类型描述的名称。当使用“XML 结构”字典时，值应为 XPATH，该值指向结构文件中的 XML 元素。

像数据类型字典一样，每个数据类型描述提供了 DataTypeVersion 特性，表示数据类型的类型描述是否已经改变。DataTypeVersion 的改变可以影响订阅的操作。如果 DataTypeVersion 改变变量，该变量正在为了订阅而受监视，并且它使用这个数据类型描述，那么发送给变量的下一个数据改变通知将会包含在数据类型描述中表示改变的状态。

数据类型的数据类型编码对象使用 HasDescription 引用来引用它们的数据类型字典的数据类型描述。然而，并不需要服务器提供数据类型描述到数据类型编码对象的反向引用。如果数据类型节点公开在地址空间中，它应提供其数据类型编码；如果数据类型字典被公开，它宜公开其所有的数据类型

描述。在这些引用中两两间应是双向的。

数据类型字典类型和数据类型描述类型的变量类型、数据类型系统类型和数据类型编码类型的对象类型的定义见 IEC 62541-5。

图 11 提供了在地址空间里如何对数据类型建模的例子。

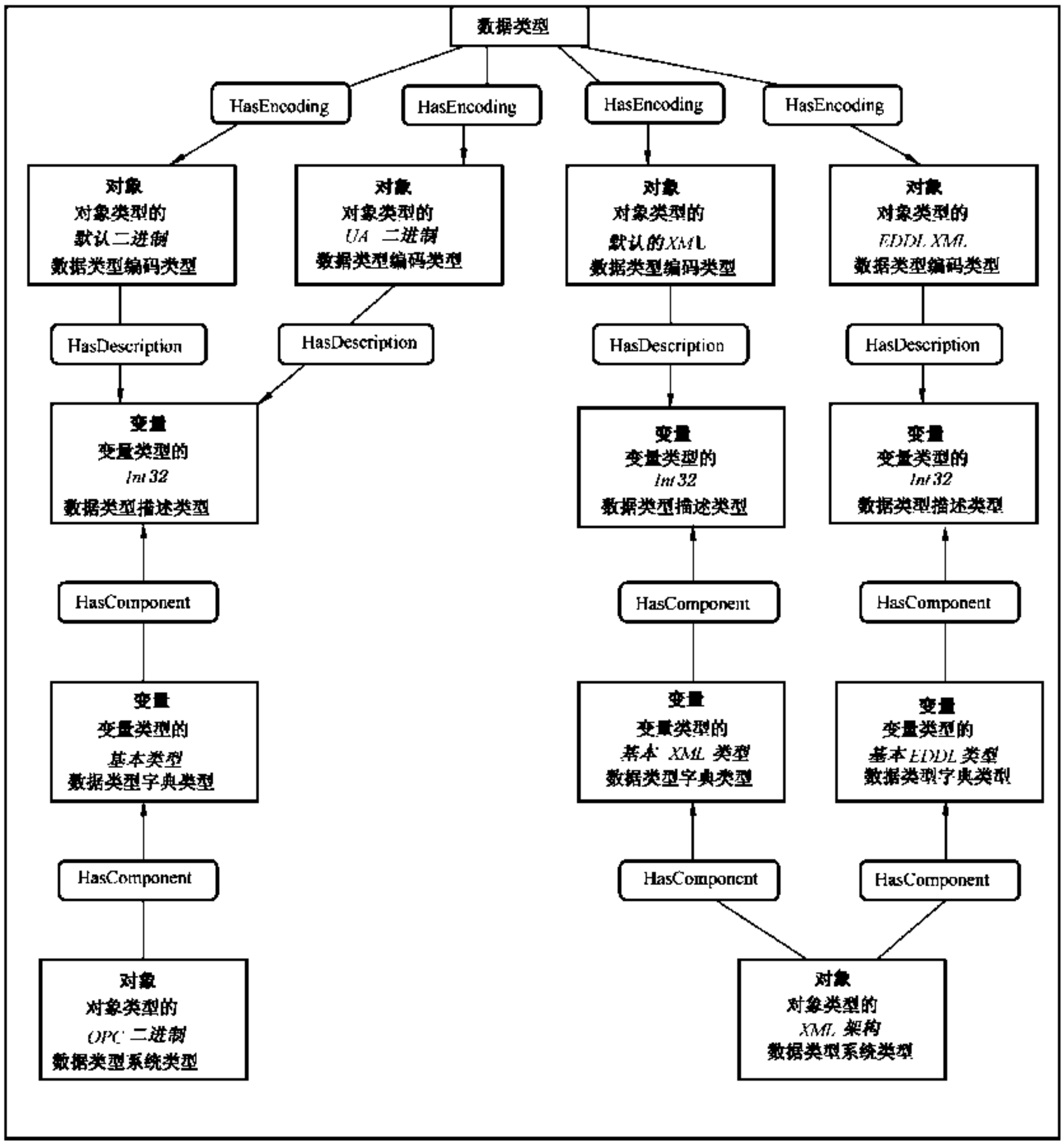


图 11 数据类型建模例子

在一些场景下，OPC UA 服务器因为资源的限制使得它不能公开大型的数据类型字典。在这些场景下，服务器可以提供访问描述给单个的数据类型，即使整个字典不能被读取。因为这个原因，本部分定义了特性用于被称作 DictionaryFragment(见 5.6.2)数据类型描述。该特性是字节字符串包含了数据类型字典的子集，这描述了与数据类型描述相关的数据类型的格式。因此，服务器将大型数据类型字典分成几个小部分，并且客户端可以访问而不会影响到系统的整体性能。

然而，如果可能，服务器宜立即提供所有的数据类型字典。通常更有效的是立刻读取全部的数据类型字典，而不是读取单个的部分。

5.9 节点属性的总结

表 12 概述了本部分中定义的所有属性，并且指出哪些节点类使用属性时是可选的(O)还是必备的(M)。

表 12 属性概述

属性	变量	变量类型	对象	对象类型	引用类型	数据类型	方法	视图
AccessLevel	M							
ArrayDimensions	O	O						
BrowseName	M	M	M	M	M	M	M	M
ContainsNoLoops								M
DataType	M	M						
Description	O	O	O	O	O	O	O	O
DisplayName	M	M	M	M	M	M	M	M
EventNotifier			M					M
Executable							M	
Historizing	M							
InverseName					O			
IsAbstract		M		M	M	M		
MinimumSamplingInterval	O							
NodeClass	M	M	M	M	M	M	M	M
NodeId	M	M	M	M	M	M	M	M
Symmetric					M			
UserAccessLevel	M							
UserExecutable							M	
UserWriteMask	O	O	O	O	O	O	O	O
Value	M	O						
ValueRank	M	M						
WriteMask	O	O	O	O	O	O	O	O

6 对象类型和变量类型的类型模型

6.1 概述

在后续章节中，从子类型化和实例化两方面来对对象类型和变量类型的类型模型进行定义。

6.2 定义

6.2.1 实例声明

实例声明是一个对象、变量或方法。它用 HasModellingRule 来引用建模规则(ModellingRule)，并

且是来自类型定义节点(TypeDefinitionNode)或另一个实例声明(InstanceDeclaration)的分层引用的目标节点。

6.2.2 无建模规则的实例

如果没有建模规则,那么该节点既不认为是一种类型的实例化,也不认为是用于子类型化。

如果类型定义节点引用的节点没有引用建模规则,则它指示此节点仅属于类型定义节点,而不属于实例。例如,一个对象类型(ObjectType)节点可包含描述了该类型适用场景的属性。当生成该类型的实例时不应考虑此属性。对于类型化也是如此,即该类型定义的子类型将不继承该引用节点。

6.2.3 实例声明层次结构(InstanceDeclarationHierarchy)

TypeDefinitionNode(类型定义节点)的 InstanceDeclarationHierarchy(实例声明层次)包含该 TypeDefinitionNode 和所有实例声明,这些实例声明是该 TypeDefinitionNode 在正向使用 Hierarchy 引用直接或间接引用的。

6.2.4 实例声明的相似节点

实例声明的相似节点是与该实例声明具有相同的 BrowseName 和 NodeClass(节点类)的节点,如果是变量和对象的情况,则只有相同的类型定义节点或子类型。

6.2.5 浏览路径

TypeDefinitionNode(类型定义节点)的正向层次引用的所有目标在该 TypeDefinitionNode 内唯一的 BrowseName。相同的限制适用于任何实例声明正向层次引用的目标。这意味着实例声明层次中任何实例声明可通过 BrowseName 序列唯一地标识。此 BrowseName 序列称为浏览路径。

6.2.6 实例声明的属性处理

当实例声明被覆盖或实例化时,关于实例声明的属性存在一些限制。BrowseName 浏览路径和 NodeClass 应保持不变,并一直和原始的实例声明相同。

此外,6.2.7 定义的规则适用于 NodeClass 变量的实例声明。

6.2.7 变量和变量类型的属性处理

对于变量类型或变量的属性有一些限制,其中该变量用作关于 Value 属性数据类型的实例声明。

当变量用作实例声明或变量类型被覆盖或实例化时,采用如下规则:

a) 如果新的数据类型是原来用的数据类型的子类型时,数据类型属性仅能变为新的数据类型;

b) ValueRank 属性可能进一步限定为:

   1) "Any"可设为任何其他值;

   2) "ScalarOrOneDimension"可设为"Scalar"或"OneDimension";

   3) "OneOrMoreDimensions"可设为维度的具体值(值>0);

   4) 此属性的所有其他值应保持不变。

c) 如果没有提供 ArrayDimension 属性,则可增加 ArrayDimension 属性,或将数组登录项的值从 0 改为不同值时。数组中所有其他值应保持不变。

6.3 对象类型和变量类型的子类型

6.3.1 概述

HasSubtype 引用类型定义类型的子类型。子类型化仅能发生在相同节点类的节点间。子类型化引用类型的规则见 5.3.3.3。对于子类型化 DataType 没有通用的定义,见 5.8.3。以下条款规定了单重

继承 ObjectTypes(对象类型)和 VariableTypes(变量类型)的子类型化规则。

6.3.2 属性

子类型继承父类型除 NodeId 外的属性值。子类型可覆盖继承的属性值,且应覆盖 BrowseName 和 DisplayName 的值。6.2.7 定义了变量类型某些属性适用的特定规则。父类型没有提供的可选属性可增加到子类型。

6.3.3 实例声明

6.3.3.1 概述

子类型完全继承了父类型的实例声明。

只要那些实例声明没被覆盖,它们就没有被子类型引用。实例声明可通过增加引用、改变对引用不同节点的引用、改变作为原始引用类型的子类型的引用、改变属性值、或增加可选属性所覆盖。为了获得子类型的完整信息,继承的实例声明应从所有类型中收集,这些类型可通过从子类型进行后续递归反向的 HasSubtype 引用发现。此实例声明的集合称为子类型的完全继承的 InstanceDeclarationHierarchy(实例声明层次)。

后续条款定义如何构建完全继承的实例声明层次,以及实例声明层次是如何被覆盖的。

6.3.3.2 完整继承的 InstanceDeclarationHierarchy(实例声明层次结构)

类型定义节点的实例由类型定义节点的完全继承实例声明层次所描述。生成完全继承实例声明层次可通过用类型定义节点的实例声明层次开始,并且合并其父类型的完全继承实例声明层次。

合并实例声明层次的过程一直是前向的,并且可用图 12 的示例来说明。图 12 规定了类型定义节点的"BetaType",其中该 BetaType 是"AlphaType"的子类型。每个文本框中的名称为 BrowseName,数字为 NodeId。

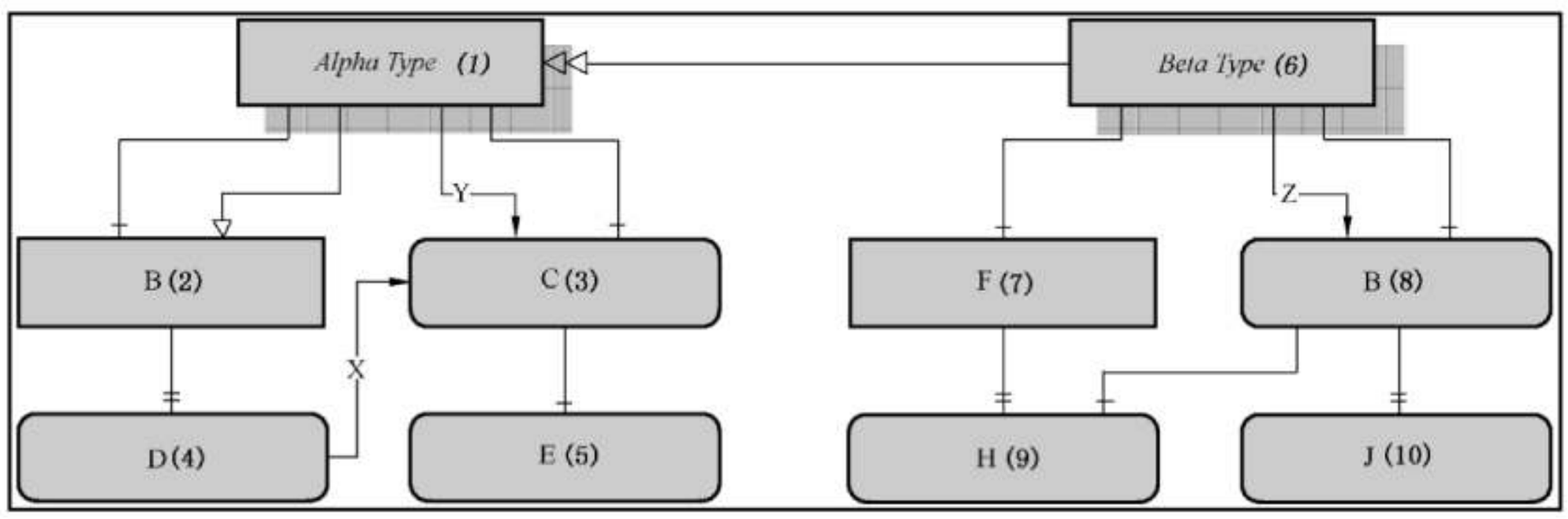


图 12 子类型化 TypeDefinitionNodes(类型定义节点)

实例声明层次可被完全描述为一张节点表,由它们的 BrowsePath 用一张对应的引用表标识。"BetaType"的实例声明层次见表 13,其中表的上半部分为节点表,下半部分为引用表(为了清楚起见,HasModellingRule 引用已从表中删除,除 1、6、5 以外的所有节点都有建模规则)。表中增加了 InstanceDeclarationHierarchy 实例声明层次的所有实例声明,以及所有从这样的实例声明用非层次引用所引用的所有节点。不考虑无建模规则的对于节点的层次引用。

表 13　BetaType 的实例声明层次

浏览路径	节点 Id
/	6
/F	7
/B	8
/F/H	9
/B/J	10
/B/H	9

源路径	引用类型	目标路径	目标节点 Id
/	HasComponent	/F	—
/	HasComponent	/B	
/	Z	/B	
/	HasTypeDefinition		BetaType
/F /F	HasComponent HasTypeDefinition	/F/H	 Base 对象类型
/B	HasProperty	/B/J	
/B	HasTypeDefinition		Base 对象类型
/F/H	HasTypeDefinition		特性类型
/B/J	HasTypeDefinition		特性类型
/B	HasComponent	/B/H	—
/B/H	HasTypeDefinition	—	BaseData 变量类型

对相同节点的多重浏览路径应视作独立节点。实例可对每个浏览路径提供不同的节点。

BetaType 的完全继承的实例声明层次可通过合并"AlphaType"的实例声明层次来构建。结构见表 14，其中从"AlphaType"增加的登录项标为灰色底色。

表 14　BetaType 完全继承的实例声明层次

浏览路径	节点 Id
/	6
/F	7
/B	8
/F/H	9
/B/J	10
/B/H	9
/B/D	4
/C	3

源路径	引用类型	目标路径	目标节点 Id
/	HasComponent	/F	

表 14（续）

源路径	引用类型	目标路径	目标节点 Id
/	HasComponent	/B	—
/	Z	/B	—
/	HasTypeDefinition	—	BetaType
/F	HasComponent	/F/H	—
/F	HasTypeDefinition	—	Base 对象类型
/B	HasProperty	/B/J	—
/B	HasTypeDefinition	—	Base 对象类型
/F/H	HasTypeDefinition	—	特性类型
/B/J	HasTypeDefinition	—	特性类型
/B	HasComponent	/B/H	—
/B/H	HasTypeDefinition	—	BaseData 变量类型
/	HasNotifier	/B	—
/B	HasProperty	/B/D	—
/	HasComponent	/C	—
/	Y	/C	—
/C	HasTypeDefinition	—	BaseData 变量类型
/B/D	HasTypeDefinition	—	特性类型
/B/D	X	/C	—

浏览路径“/B”已存在表中，因此不需要增加它。然而没有从“/”到“/B”的 HasNotifier 引用，所以增加此内容。

表 14 定义的节点和引用可用来生成图 13 中的完全继承实例声明层次，如图 13 所示。完全继承实例声明层次包含所有关于类型定义节点复杂结构的必要信息，而不需要来自父类型的如何其他信息。

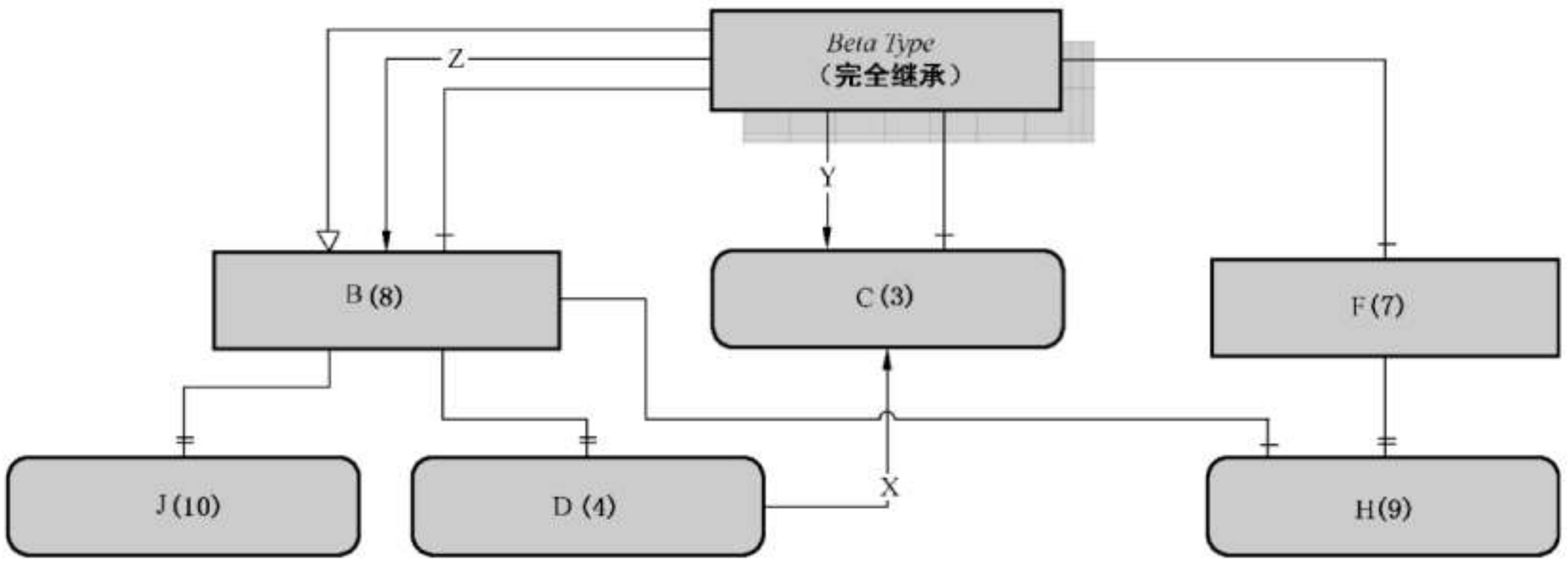


图 13　BetaType 的完全继承实例声明层次

6.3.3.3 覆盖实例声明

子类型通过指定的具有相同浏览路径规定实例声明来覆盖实例声明。覆盖的实例声明应有相同的NodeClass 和 BrowseName。覆盖的实例声明的类型定义节点应与父类型规定的类型定义节点相同，或是其子类型。

当覆盖实例声明时，有必要提供层次引用，它将新节点连回子类型(引用用来确定节点的浏览路径)。

仅可能覆盖类型定义节点直接引用的实例声明。如果必须要覆盖非直接引用的实例声明，如图 13中的"J"，那么在情况"B"中应先覆盖含有"J"的直接引用的实例声明，然后下一步"J"可被覆盖。

如果引用是在两个覆盖节点之间，并且与父类型定义的引用具有相同的引用类型，那么替换该引用。子类型规定的引用可以是父类型所用的引用类型的子类型。

对覆盖的实例声明所规定的任何非层次引用被视作新引用，除非该引用类型仅允许每个SourceNode(源节点)一个单引用。如果出现这种情况，子类型可改变引用的目标，但是新目标应具有与在父类型中规定的该类型的相同 NodeClass(节点类)，且对于对象和变量也应该具有与在父类型中规定的该类型的相同类型或子类型。

覆盖的节点可规定新的节点属性值，而非节点类或 BrowseName，然而 6.2.6 规定的属性限制适用。覆盖的实例声明所提供的任何属性应通过覆盖实例声明来提供，可增加其他可选属性。

可改变覆盖的实例声明的建模规则，见 6.4.4.3。

每个覆盖的实例声明需要其自身的 HasModellingRule 和 HasTypeDefinition 引用，即使它们没有变化。

子类型应不覆盖节点，除非它需要改变它。

某类型定义节点和引用类型的语义可增加其他关于覆盖节点的限制。

6.4 对象类型和变量类型的实例

6.4.1 概述

类型定义节点(TypeDefinitionNode)的任何一个实例将是反映该类型定义节点的实例声明层次(InstanceDeclarationHierarchy)的那个层次结构的根。该实例层次结构中的每个节点都有一个浏览路径，它可能与该类型定义节点的层次结构中的实例声明之一的浏览路径相同。具有相同浏览路径的实例声明称为该节点的实例声明。如果某个节点有一个实例声明，那么该节点应该有与该实例声明相同的 BrowseName 和 NodeClass，对于变量和对象，它应该有相同的类型定义节点或其子类型。

实例可用相同的浏览路径引用多个节点。对于需要区分基于实例声明层次的节点和非基于实例声明层次的节点的客户，他们可使用 IEC 62541-4 定义的 TranslateBrowsePathsToNodeId 服务来实现。

6.4.2 创建实例

自某个类型定义节点实例化来的实例继承与该类型定义节点有相同的属性的初始值，但 NodeClass 和 NodeId 除外。

当服务器创建类型定义节点的实例时，它应依据各实例声明的建模规则，在新目标或变量下创建相同的节点层次结构。标准建模规则的定义见 6.4.4.5。刚创建的层次结构中的节点可以是实例声明的副本、实例声明自身、或地址空间中有相同类型定义节点和 BrowseName 的另一个节点。如果创建了新的副本，实例声明的属性值用作初始值。

图 14 给出了类型定义节点和实例的一个简单示例。无建模规则的类型定义节点引用的节点没有出现在该实例中。实例可以有用复制的 BrowseName 的孩子，但这些孩子中仅有一个与实例声明相对应。

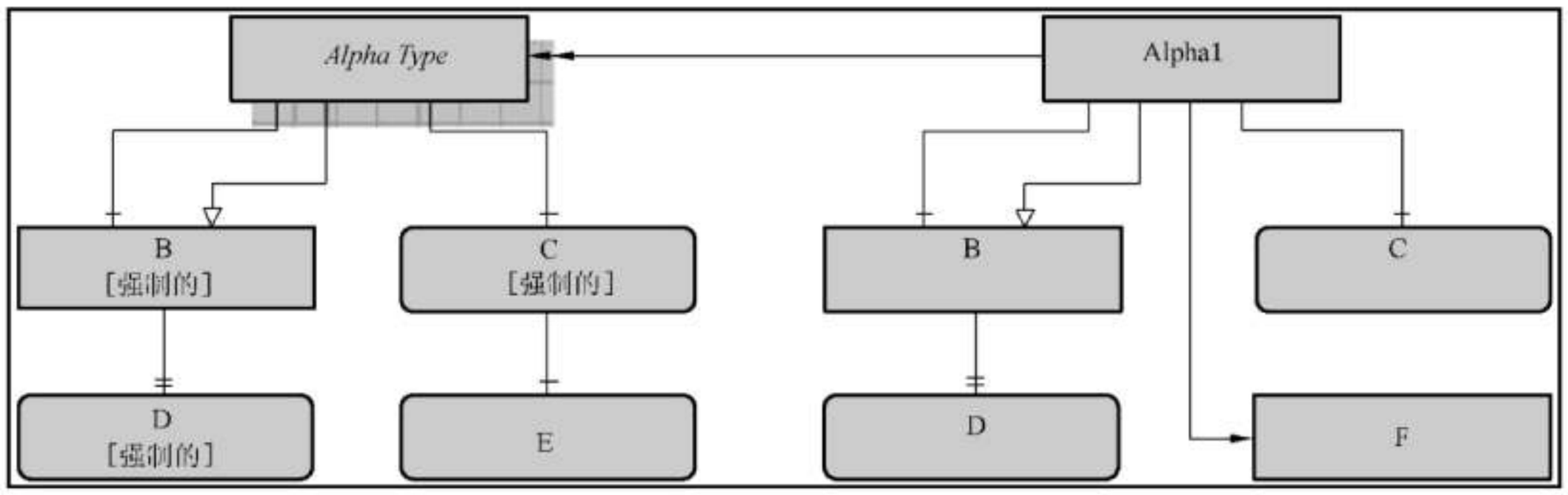


图 14　实例及其类型定义节点

由服务器决定哪个实例声明出现在任何单个实例中。有些情况下，服务器不定义整个实例，并提供对另一个服务器中节点的远程引用。6.4.4.5 所述的建模规则允许服务器指出一些节点总会出现，但客户端应准备当节点存在于不同服务器的情况。

客户端可使用类型定义节点的信息来访问实例层次结构中的节点。它应将实例的 NodeId 和基于类型定义节点的子节点 BrowsePath 传递给 TranslateBrowerPathsToNodeId 服务（见 IEC 62541-4）。此服务返回各子节点的 NodeId。如果存在子节点，那么 BrowseName 和 NodeClass 应与实例声明匹配。在对象或变量的情况下，类型定义节点也应匹配或作为原始类型定义节点的子类型。

6.4.3　对实例的限制

对象和变量可在创建后改变其属性值。对某些属性适用特殊规则，见 6.2.6。

对节点可以增加其他引用，并且只要对该类型定义节点的实例声明定义的建模规则仍被执行，则可删除这些引用。

对于变量和对象，HasTypeDefinition 引用应总指向与实例声明或其子类型相同的类型定义节点。

如果完整继承实例声明层次的两个实例声明已用若干引用直接相连，那么所有那些引用应连接相同节点。图 15 给出一个示例。实例 A1 和 A2 是允许的，由于 B1 引用了与两个引用相同的节点，然而 A3 是不允许的，由于引用了两个不同节点。请注意此限制仅适用于直接连接的节点。例如，A2 直接引用 C1，而另一个 C1 引用通过 B1。

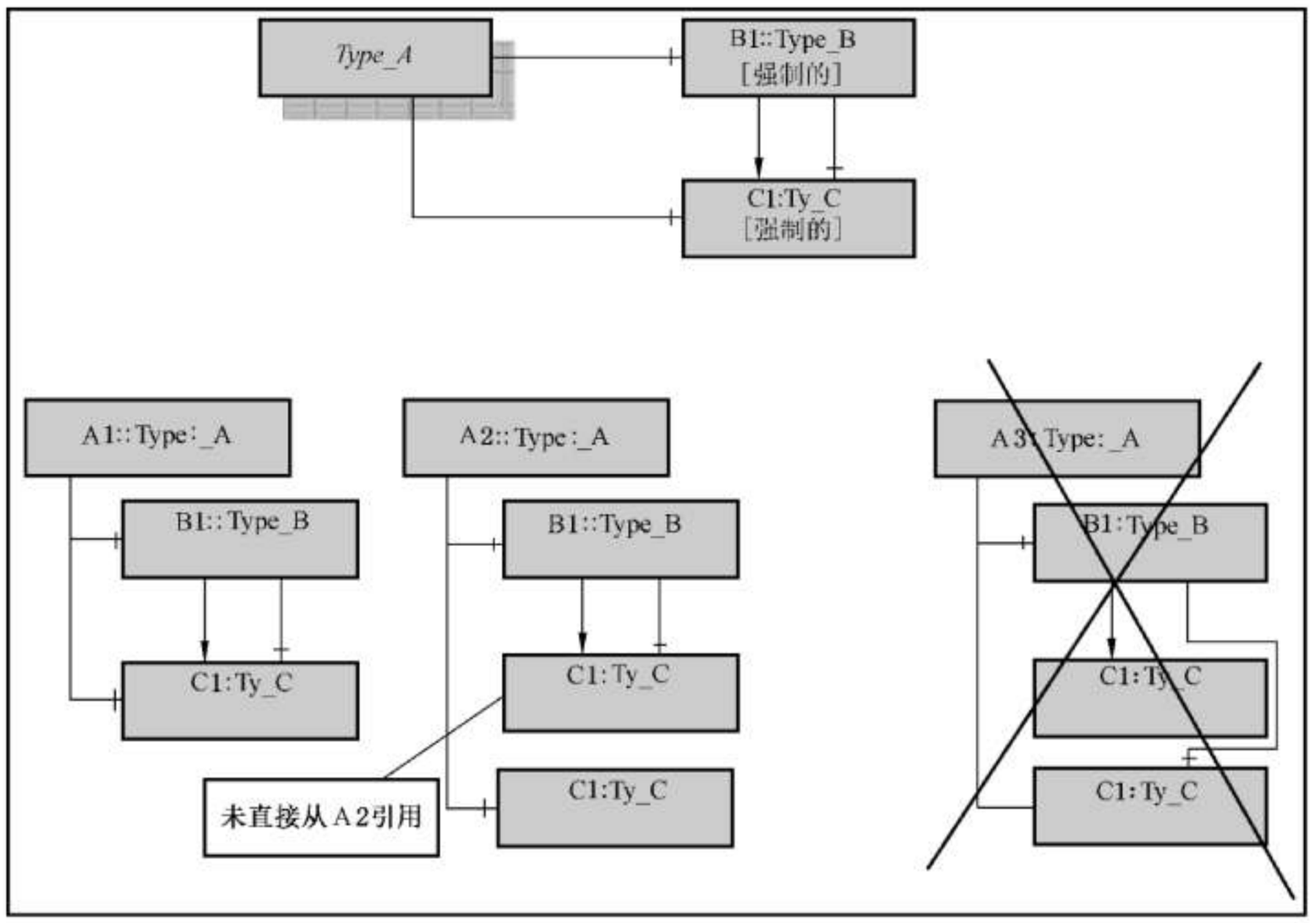


图 15　实例声明间若干引用的示例

6.4.4 建模规则

6.4.4.1 概述

建模规则定义见 6.4.5。本部分的其他部分可能定义了另外的建模规则。建模规则在 OPC UA 中是一个可扩展的概念,因此供应商可定义自己的建模规则。

请注意本部分定义的建模规则并没有定义如何处理实例声明间的非层次结构引用,也就是说,那些引用是否存在于某个实例层次结构中,是服务器特定的。其他建模规则也可定义实例声明间非层次结构引用的行为。

在地址空间中建模规则用 ModellingRuleType 的对象表示。有一些特性定义了建模规则的通用语义。本部分仅定义了建模规则的一个特性。未来版本可能会定义建模规则的其他特性。IEC 62541-5 规定了建模规则对象的表示、它们的特性和它们在地址空间的类型。建模规则的特性语义见 6.4.4.2。

6.4.4.4 定义了当实例化关于特性的实例声明时,如何改变建模规则。6.4.4.3 定义了当覆盖关于特性的实例声明子类型时,如何改变建模规则。

6.4.4.2 描述建模规则的特性

6.4.4.2.1 命名规则

NamingRule 是建模规则的必备属性。它规定了实例声明的目的。各实例声明引用一个建模规则,因而每个实例声明定义一个 NamingRule。

建模规则的 NamingRule 允许使用 3 个值:可选的、必备的和限制的。

如下语义在实例的整个生命周期中是有效的,该实例基于拥有实例声明的类型定义节点。

对于类型定义节点 AlphaType 的实例 A1,带有实例声明的 B1,其建模规则使用可选的 NamingRule,以下规则适用:对从 AlphaType 到 B1 的各浏览路径,A1 可以有一个具有相同浏览路径且与 B1 相似的节点(见 6.2.4)。如果存在这样的节点,TranslateBrowerPathsToNodeId 服务(见 IEC 62541-4)返回此节点在列表中的第一个输入项。

对于类型定义节点 AlphaType 的实例 A1,带有实例声明的 B1,其建模规则使用必备的 NamingRule,以下规则适用:对从 AlphaType 到 B1 的各浏览路径,A1 应有一个与浏览路径相同的 B1 相似的节点(见 6.2.4),如果浏览路径的所有节点都存在。例如,如果浏览路径的节点有可选的 NamingRule,并且实例忽略该 NamingRule,那么此节点的全部子节点被忽略该 NamingRule,独立于他们的建模规则。

如果一个实例声明的建模规则使用限制的 NamingRule,则它指示实例声明的 BrowseName 没有意义,但其他语义使用建模规则进行定义。典型地,不可用 TranslateBrowerPathsToNodeId 服务(见 IEC 62541-4)基于那些实例声明来访问实例。

6.4.4.3 建模规则特性的子类型化规则

允许子类型覆盖其实例声明的建模规则。作为子类型化的通用规则,限制应只能加严,而不能放松。因此,允许对子类型作如下规定,实例应有名称(NamingRule 为必备的),子类型为可选的(NamingRule 为可选的)。表 15 规定了当交换了类型建模规则时,对特性允许的修改。

表 15　子类型化时用于建模规则特性的规则

	父类型值	子类型值
NamingRule	必备的	必备的
NamingRule	可选的	必备的或可选的
NamingRule	限制的	限制的

6.4.4.4　建模规则属性的实例化规则

当基于类型定义节点“A_Type”创建实例“A”时，有两种不同的使用情况。或者“A”用作正常实例，或者它用作另一个类型定义节点的实例声明。

对于第一种情况，不要求基于实例声明新创建或引用的实例有建模规则。然而，也允许它们有任何建模规则，独立于其实例声明的建模规则。

图 16 给出一个例子。尽管 A1 的 B 没有建模规则，而且 A3 的 B 的建模规则不同于 Type_A 的 B，但实例 A1、A2 和 A3 都是 Type_A 的有效实例。

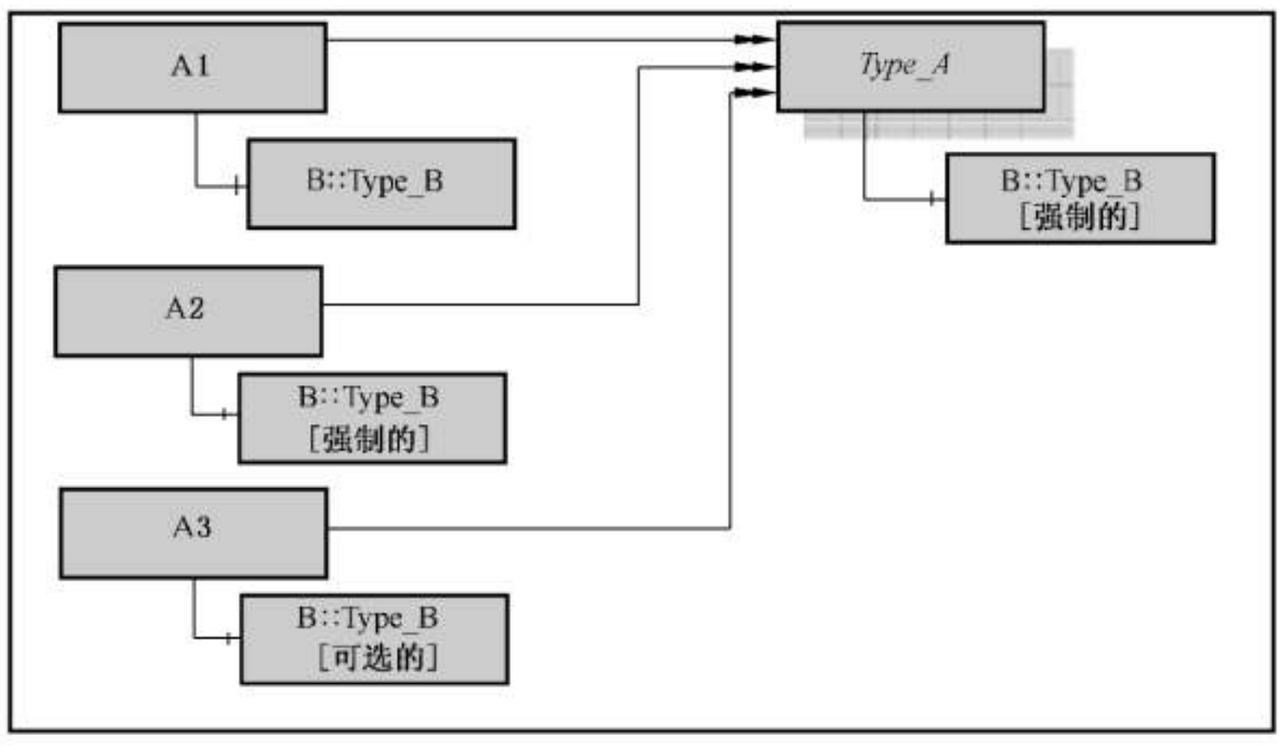


图 16　基于实例声明改变实例的例子

对于第二种情况，基于"A_Type"实例声明从"A"直接或间接引用的所有实例首先维护相同的建模规则作为其实例声明。可更新该建模规则；对这些节点的建模规则允许的修改与6.4.4.3子类型化中定义的相同。

图17给出了此种情况下的例子。Type_B使用基于Type_A的实例声明(图的上半部分)。后面在实例声明的建模规则上，A1发生了改变(图的下半部分)。A1成为必备的NamingRule(原来为可选的)。

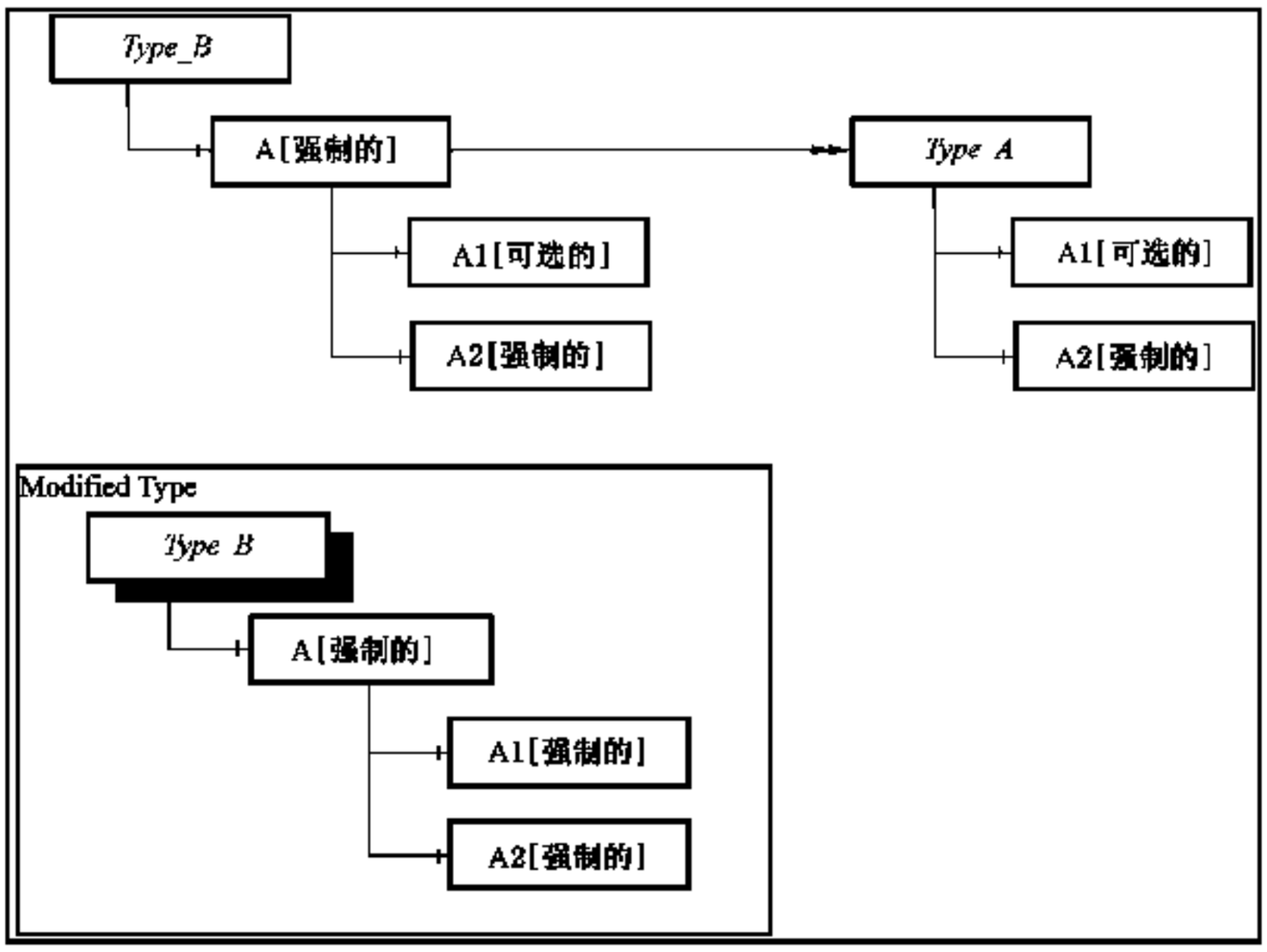


图17 基于实例声明改变实例声明的例子

6.4.4.5 标准建模规则

6.4.4.5.1 标准建模规则标题

如下条款定义了建模规则。表16总结了那些建模规则的特性。

表16 建模规则特性

标题	NamingRule
Mandatory	必备的
Optional	可选的
ExposesItsArray	限制的

6.4.4.5.2 必备的

标记为必备ModellingRule必备的实例声明完全符合为必备NamingRule定义的语义。它意味着对于实例上各现有浏览路径都存在一个相似节点，但没有定义是否创建一个新节点或引用现有节点。

例如，功能块"AI_BLK_TYPE"的类型定义节点将有一个设定值"SP1"。此"AI_BLK_1"类型的实例将有一个新创建的设定值"SP1"，作为对实例声明SP1的相似节点。图18给出了该例子。

6.4.4.5.3给出一个有必备建模规则和可选建模规则的复杂例子。

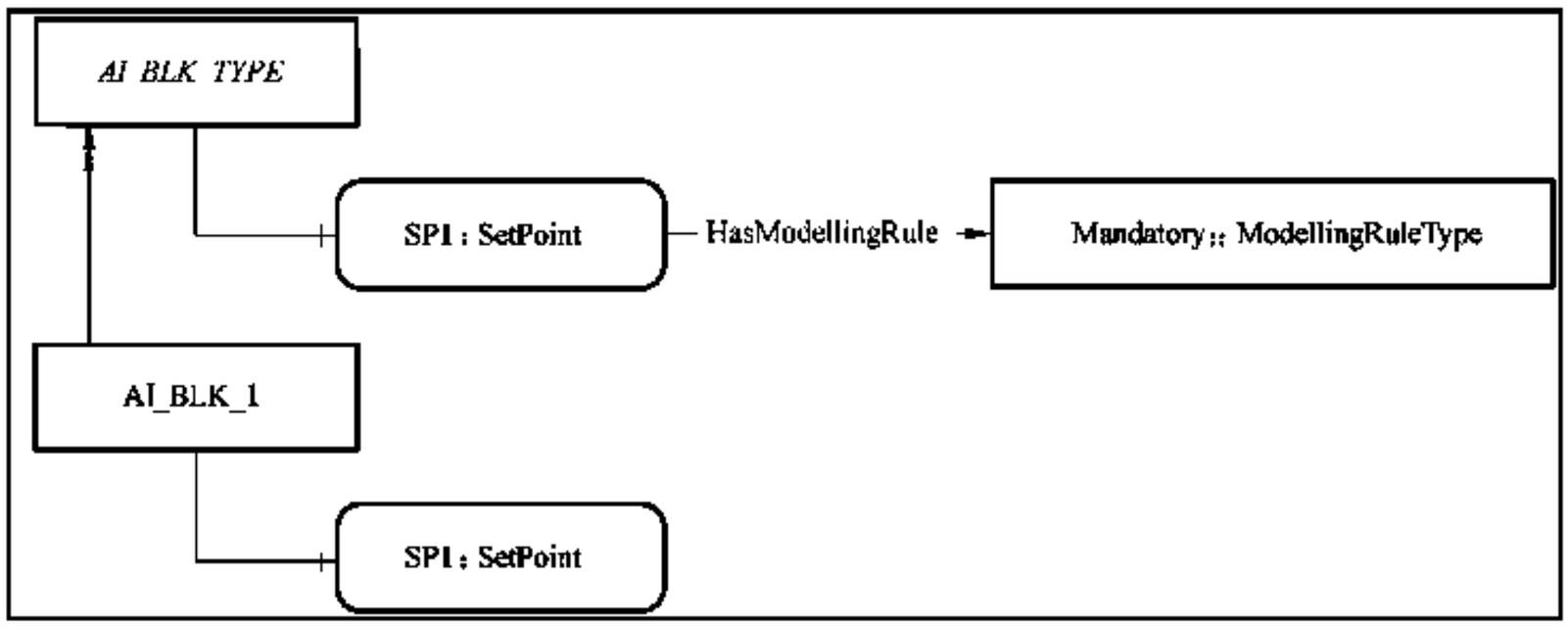


图 18　新标准建模规则的使用

6.4.4.5.3　可选的

标记为可选建模规则的实例声明完成的正是可选 NamingRule 定义的语义。它意味着对于实例上各现有浏览路径都存在一个相似节点，但没有定义是否创建一个新节点或引用现有节点。

图 19 给出的例子使用可选的和必备的建模规则。该例子包含对象类型 A_Type 和所有 A1～A13 有效的实例组合。请注意如果提供了可选的 B，则必须也提供必备的 E，反之不然。F 由 C 和 D 引用。在此实例上，它可以是相同节点，或有相同 BrowseName 名称的两个不同节点（与实例声明 F 相似的节点）。在该例子中没有考虑该实例是否有建模规则。假设每个 F 都与实例声明 F 相似。

如果在实例声明层次中的 E 和 F 间有非层次结构引用，没有规定它是否发生在实例层次结构。在 A13 情况下，可以有来自 F 的引用，而非来自于其他，或者来自于两者或两者都不是。

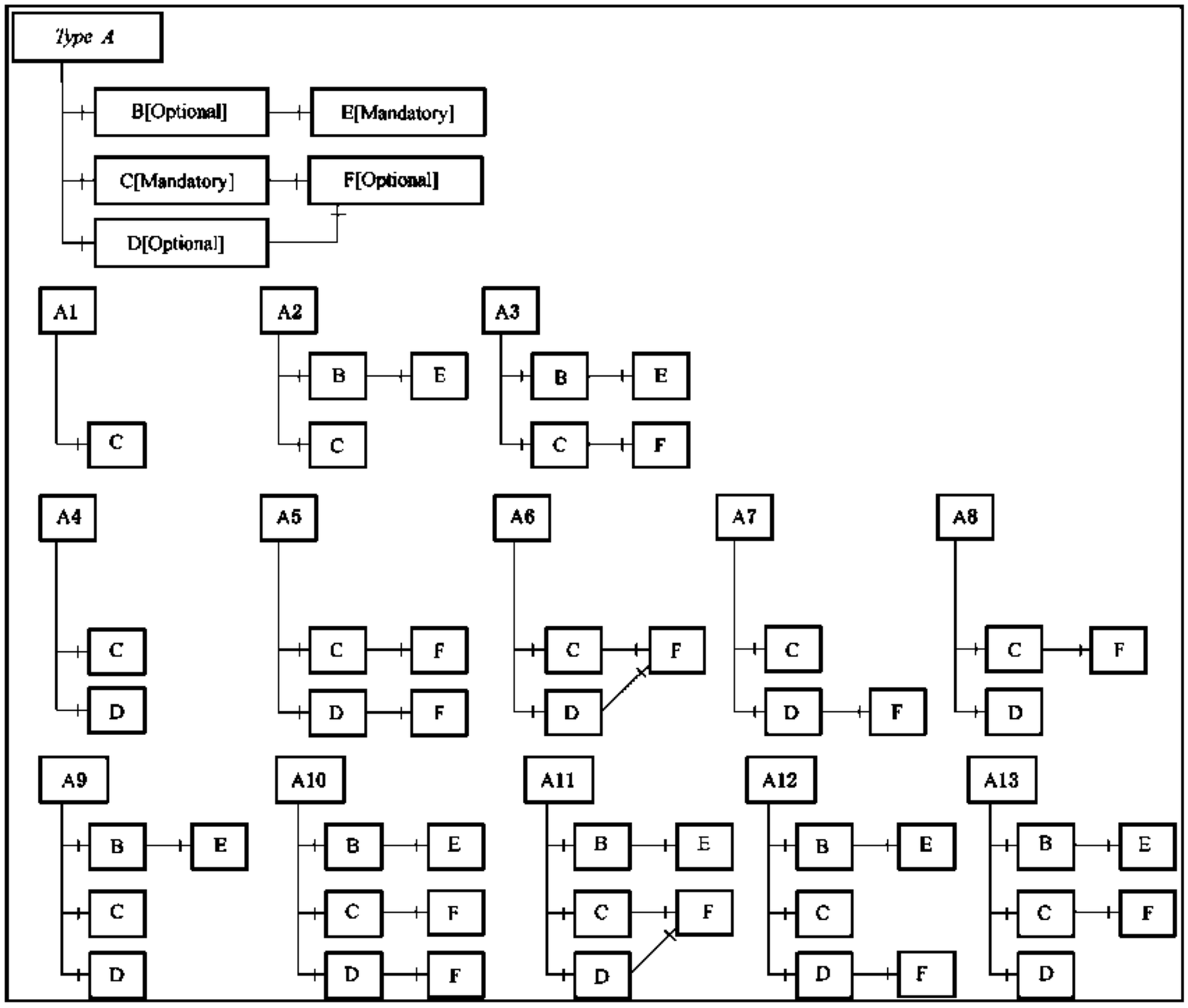


图 19 使用可选和必备的标准建模规则的示例

6.4.4.5.4 ExposesItsArray

ExposesItsArray 建模规则揭示了变量类型的一种特定语义，该变量类型的数据类型是一个单维或多维的数组。它指出该数组的每个值也将公开为地址空间中的一个变量。

ExposesItsArray 建模规则仅能用于 NodeClass 变量的实例声明，该变量是具有单维或多维数组作为数据类型的变量类型的一部分。

具有此建模规则的变量 A 应从变量类型 B 以向前方向通过 hierarchical 引用来引用。B 应具有一个等于或大于零的 ValueRank 值。A 应具有一种数据类型，它至少反映出在数组 B 中管理的数据的各部分。B 的每个实例应为其数组元素的每一个引用 A 的一个实例。所使用的引用应与连接 B 和 A 或其子类型的层次引用的类型相同。如果在 A 和 B 之间有多于一个向前方向的 hierarchical 引用，那么所有基于 B 的实例应以所有那些引用来引用。

图 20 给出一个示例。A 是在其值数组中拥有两个登入项的 Type_A 的实例。因此，它引用与实例声明 ArrayExpose 有相同类型的两个实例。那些实例的 BrowseName 未通过该 ModellingRule 来定义。一般来讲，不可能获得代表数组中特定登入项的变量(如第二个)。典型地，客户端或者获得数组，或者直接访问变量，因此没必要提供此信息。

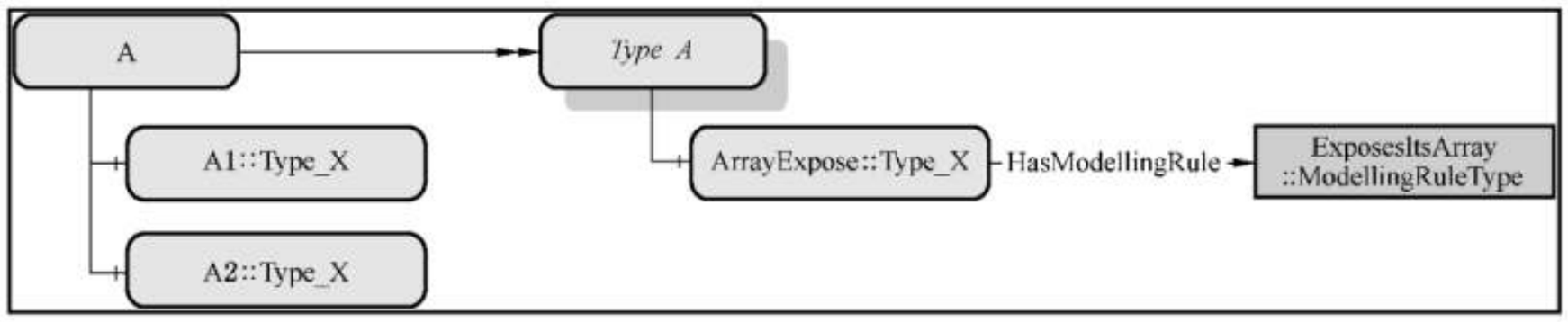


图 20　使用 ExposesItsArray 的示例

也允许由其他实例声明引用 A。那些引用应反映在基于 A 的各实例上。

图 21 给出一个示例。属性 EUUnit 被 ArrayExpose 引用，因此基于 ArrayExpose 的各实例引用基于实例声明 EUUnit 的实例。

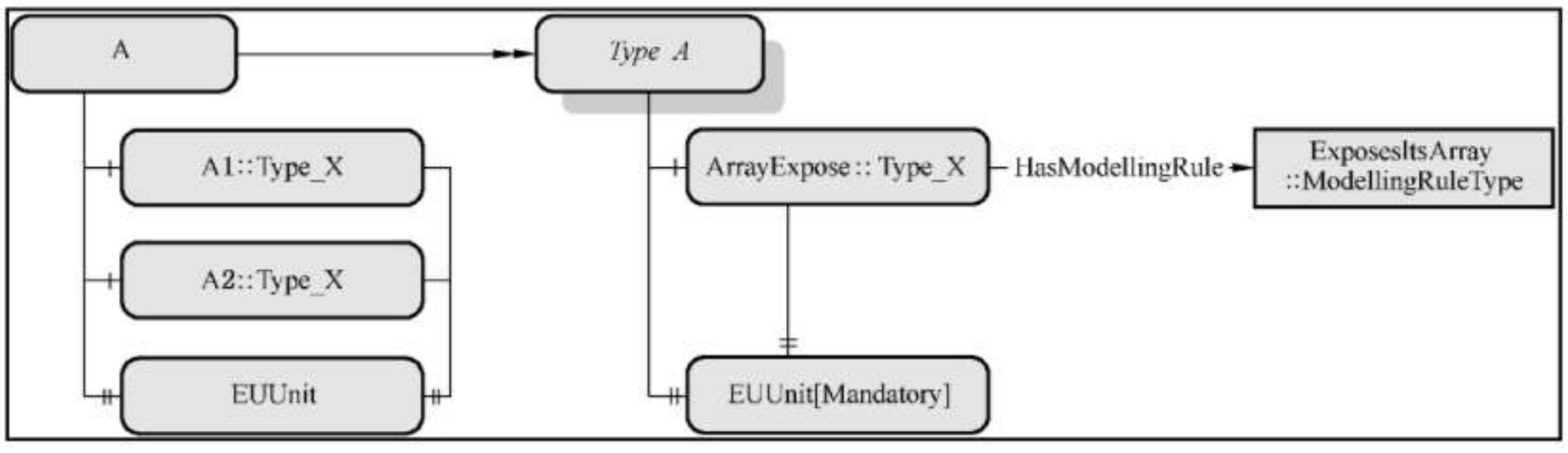


图 21　使用 ExposesItsArray 的复杂示例

6.5　改变已使用的类型定义

在改变对象类型和变量类型时，关于子类型和实例不存在特定的行为。如果那些改变反映在该类型的子类型和实例上，则其行为取决于服务器。然而，为子类型和实例定义的所有限制应被满足。例如，如果此类型的实例没有特性，不允许使用必备建模规则对此类型增加特性。在这种情况下，服务器或者必须对此类型的所有实例增加特性，或者必须拒绝对该类型增加特性。

6.6　父模型(ModelParent)

每个对象、变量和方法可被若干 hierarchical(层次引用)来引用。为了表示此节点“A”的范围，它可以公开其父模型。“A”的父模型是“A”定义的范围。当客户端想要改变“A”，它可使用父模型来决定它是否有正确的范围。

例如，TypeDefinitionNode “Type_A”可以有所有实例共享的实例声明，称为“Icon”。因而“Icon”的父模型是“Type_A”。客户端可浏览实例“A”，它是“Type_A”的实例来改变“A”的“Icon”值。“A”引用了共享的实例声明“Icon”，并且通过辨识其父模型，客户端可计算出该节点的范围是“Type_A”而不是“A”。因此，该客户端可以不改变该节点值，而创建一个副本，引用该副本而不是实例声明，再改变副本值。

为了辨识父模型，对象、变量或方法使用引用类型 HasModelParent 引用代表父模型节点。父模型应在向前方向以 hierarchical 引用来引用对象、变量或方法。

本部分定义的所有建模规则应提供 HasModelParent 引用。不要求向其他建模规则显示 HasModelParent 引用，但建议这么做。允许向没有 HasModellingRule 引用的对象、变量或方法提供 HasMod-

clParent 引用。然而，由于可能没有清晰定义的父模型，因此它并不是必备要求的，并且并不总是存在的。

图 22 示出类型定义节点的 HasModelParent 关系和两个实例。在此例子中，节点"C"的父模型是类型定义节点，而且所有实例用层次引用来引用它。

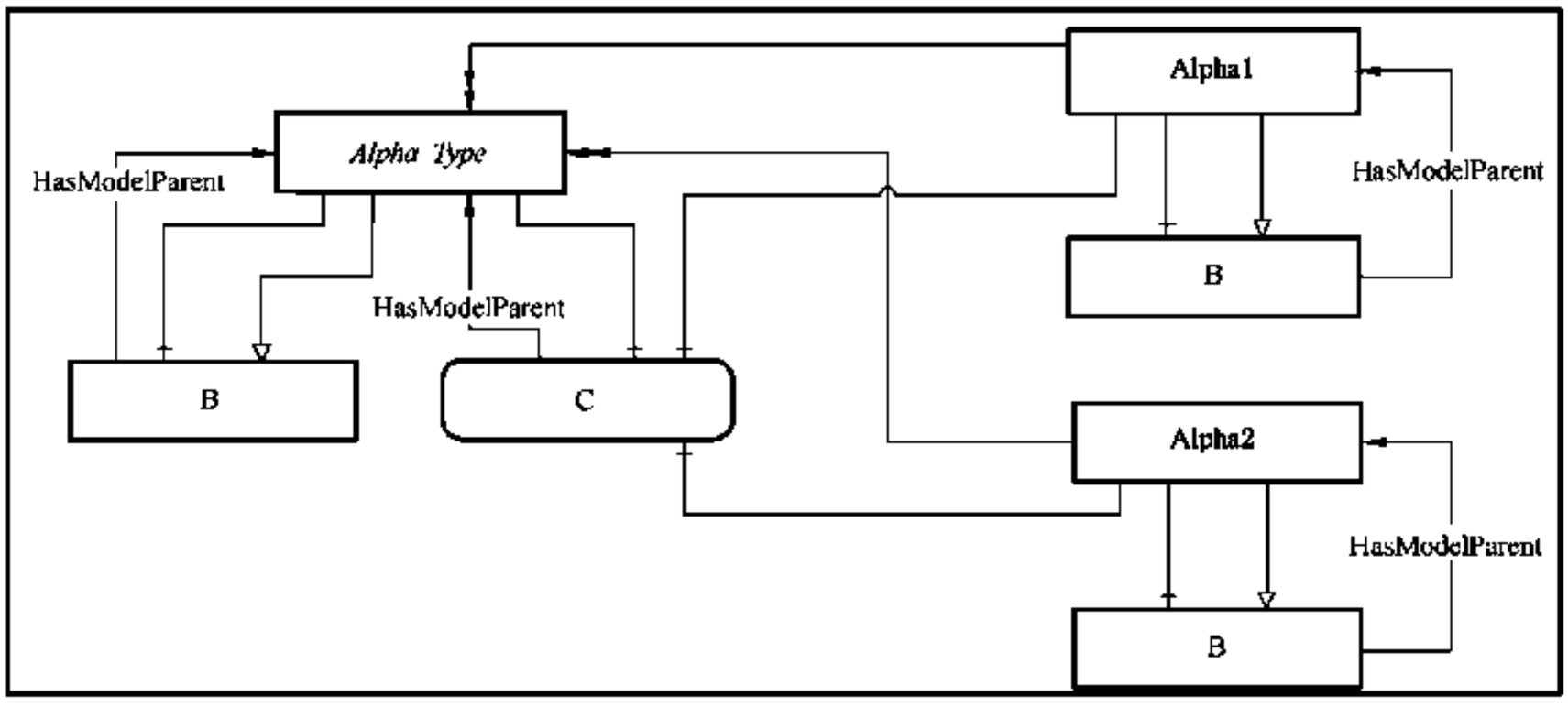


图 22　父模型示例

7　标准引用类型

7.1　概述

本部分定义了引用类型作为 OPC UA 地址空间模型的继承部分。图 23 简略地描述了这些引用类型的层次结构。本系列标准的其他部分可能规定了另外的引用类型。如下条款定义了引用类型。IEC 62541-5 定义了它们在地址空间的表示。

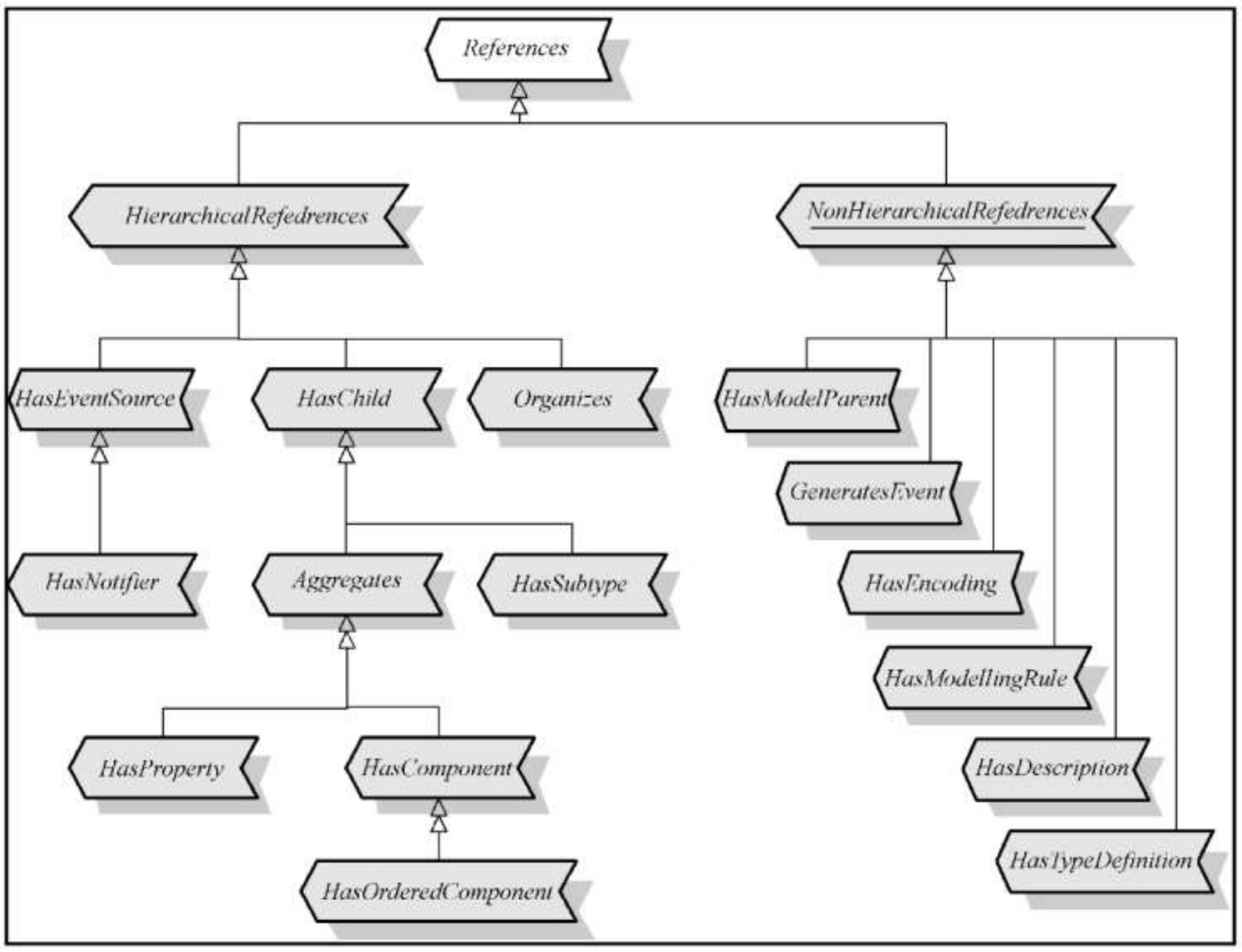


图 23　标准引用类型层次结构

7.2　References 引用类型

References 引用类型是一个抽象的引用类型，仅可使用其子类型。

没有与此引用类型相关的语义。这是所有引用类型的基础类型。所有引用类型应是此基础引用类型直接的或间接的子类型。此引用类型的主要目的是允许在 IEC 62541-5 的相应服务中进行简单的过滤和查询。

没有定义对此抽象引用类型的限制。

7.3　HierarchicalReference 引用类型

HierarchicalReference 引用类型是一种抽象引用类型，仅它的子类型可以使用。

HierarchicalReference 的语义指示了 HierarchicalReference 的引用跨越了一个层次结构。这意味着它对于以类似层次结构的方式来表示与此类型相关的节点是有用的。HierarchicalReference 不禁止环路。例如，从节点“A”开始，接着 HierarchicalReference 可再次浏览节点“A”。

不允许属性作为此抽象引用类型任何子类型的引用源节点。

不允许 HierarchicalReference 引用类型引用的源节点和目标节点相同，也就是说，不允许用 HierarchicalReference 进行自身引用。

7.4　NonHierarchicalReference 引用类型

NonHierarchicalReference 引用类型是一种抽象引用类型，仅它的子类型可以使用。

NonHierarchicalReference 的语义指示了其子类型没有跨越一个层次结构，并当试图提出一个层次结构时，不应对其进行跟踪。区别层次结构的和非层次结构的引用，所有具体引用类型应从 Hierarchi-

calReference 或 NonHierarchicalReference 直接或间接继承。

没有定义对此抽象引用类型的限制。

7.5 HasChild 引用类型

HasChild 引用类型是一种抽象引用类型，仅它的子类型可以使用。它是 HierarchicalReference 的子类型。

其语义指示了此类型的引用跨越了非环路的层次结构。

从节点"A"开始，且后续仅为 HasChild 引用类型的子类型的引用应永远不能返回"A"。但是允许后续引用有多条路径通向另一个节点"B"。

7.6 Aggregates 引用类型

Aggregates 引用类型是一种抽象引用类型；仅它的子类型可以使用。它是 HasChild 的子类型。

其语义指示一部分(目标节点)属于源节点。它没有规定目标节点的所有权。

没有定义对此抽象引用类型的限制。

7.7 HasComponent 引用类型

HasComponent 引用类型是一种可直接使用的具体引用类型。它是 Aggregates 引用类型的子类型。

其语义表示包含的关系。HasComponent 引用类型引用的目标节点是源节点的一部分。此引用类型用来关联对象或对象类型与其包含的对象、数据变量和方法，以及复杂变量或带有数据变量的变量类型。

与其他所有引用类型一样，此引用类型没有规定任何关于各部分的所有权，尽管它代表包含关系的语义。没有规定当删除源节点时，HasComponent 引用类型引用的目标节点是否被删除。

该引用类型的目标节点应是变量、对象或方法。

如果目标节点是一个变量，那么其源节点应为对象、对象类型、数据变量或变量类型。通过使用 HasComponent 引用，变量定义为 D 数据变量。

如果目标节点是对象或方法，其源节点应为对象或对象类型。

7.8 HasProperty 引用类型

HasProperty 引用类型是一种可直接使用的具体引用类型。它是 Aggregates 引用类型的子类型。

其语义是标识节点的特性。特性描述见 4.4.2。

此引用类型的源节点可为任何节点类。目标节点应为变量。通过使用 HasProperty 引用，变量定义为特性。由于特性不应具有特性，特性应绝非 HasProperty 引用的源节点。

7.9 HasOrderedComponent 引用类型

HasOrderedComponent 引用类型是一种可直接使用的具体引用类型。它是 HasComponent 引用类型的子类型。

HasOrderedComponent 引用类型的语义——除 HasComponent 引用类型的语义外——是当从节点浏览并接着此类型或子类型的应用时，在 IEC 62541-5 定义的浏览服务中以明确定义的顺序返回所有引用。该顺序是服务器特定的，但客户端可假设服务器总以相同顺序返回它们。

没有定义对此抽象引用类型的附加限制。

7.10 HasSubtype 引用类型

HasSubtype 引用类型是一种可直接使用的具体引用类型。它是 HasChild 引用类型的子类型。

此引用类型的语义是表示类型的子类型关系。它用来跨越引用类型层次结构，其语义规定见5.3.3.3。5.8.3规定了数据类型的层次结构，其他子类型层次结构规定见第6章。

此类型引用的源节点应是对象类型、变量类型、数据类型或引用类型，并且其目标节点不应与源节点的节点类相同。各引用类型应是类型HasSubtype的至多一个引用的目标节点。

7.11 Organizes引用类型

Organizes引用类型是一种可直接使用的具体引用类型。它是HasChild引用类型的子类型。

此引用类型的语义是组织地址空间中的节点。它可用来跨越多个层次结构，独立于非环路Aggregates引用创建的任何层次结构。

此类型引用的源节点应是对象或视图。如果它是对象，则它应是对象类型FolderType或其子类型之一的对象(见5.5.3)。

此引用类型的目标节点可以是任何节点类。

7.12 HasModellingRule引用类型

HasModellingRule引用类型是一种可直接使用的具体引用类型。它是NonHierarchicalReference的子类型。

此引用类型的语义是绑定建模规则到一个对象、变量或方法。建模规则机制的描述见6.4.4。

此引用类型的源节点应是对象、变量或方法。目标节点应是对象类型"建模规则"或其子类型之一的对象。

各节点应是至多一个HasModellingRule引用的源节点。

7.13 HasModelParent引用类型

HasModelParent引用类型是一种可直接使用的具体引用类型。它是NonHierarchicalReference的子类型。

此引用类型的语义是公开对象、变量或方法中的父模型。父模型机制的描述见6.6。

此引用类型的源节点应是对象、变量或方法。

各节点应是至多一个HasModelParent引用的源节点。

7.14 HasTypeDefinition引用类型

HasTypeDefinition引用类型是一种可直接使用的具体引用类型。它是NonHierarchicalReference的子类型。

此引用类型的语义是分别绑定对象或变量到其对象类型或变量类型。描述类型和实例间的关系见4.5。

此引用类型的源节点应是对象或变量。如果源节点是对象，那么目标节点应是对象类型；如果源节点是变量，那么目标节点应是变量类型。

各变量和对象应只是一个HasTypeDefinition引用的源节点。

7.15 HasEncoding引用类型

HasEncoding引用类型是一种可直接使用的具体引用类型。它是NonHierarchicalReference的子类型。

此引用类型的语义是引用数据类型的DataTypeEncoding。

此引用类型的目标节点应是对象类型DataTypeEncodingType或其子类型之一的对象(见5.8.4)。

7.16 HasDescription 引用类型

HasDescription 引用类型是一种可直接使用的具体引用类型。它是 NonHierarchicalReference 的子类型。

此引用类型的语义是引用 DataTypeEncoding 的 DataTypeDescription。

此引用的源节点应是对象类型 DataTypeEncodingType 或其子类型之一的对象。

此引用类型的目标节点应是变量类型 DataTypeDescriptionType 或其子类型之一的变量。

7.17 GeneratesEvent

GeneratesEvent 引用类型是一种可直接使用的具体引用类型。它是 NonHierarchicalReference 的子类型。

此引用类型的语义是标识事件类型，该事件由对象类型或变量类型的实例产生，或在每个方法调用时由方法产生。

此引用类型的源节点应是对象类型、变量类型或方法。

此引用类型的目标节点应是代表事件类型的对象类型，即 BaseEventType 或其子类型之一。

7.18 AlwaysGeneratesEvent

AlwaysGeneratesEvent 是一种可直接使用的具体引用类型。它是 GeneratesEvent 的子类型。

此引用类型的语义是标识在每个方法调用时必须生成的事件方法类型。

此引用类型的源节点应为方法。

此引用类型的目标节点应是代表事件类型的对象类型，即 BaseEventType 或其子类型之一。

7.19 HasEventSource

HasEventSource 是一种可直接使用的具体引用类型。它是 HierarchicalReference 的子类型。

此引用类型的语义是关联非环路组织的层次结构的事件源。此引用类型和任何子类型旨在用于发现服务器中的事件产生。不要求在场的服务器产生其源节点中和其通知节点中的事件。特别地，服务器的根通知器(在 IEC 62541-5 中定义的服务器对象)总是能提供来自服务器的所有事件，并因此对服务器中的每个事件源具有隐含的 HasEventSource 引用。

此引用类型的源节点应为对象，该对象是事件订阅的源。事件订阅的源是自身有在 EventNotifier 属性中对“SubscribeToEvents”比特置位的对象。

此引用类型的目标节点可以是任何节点类的节点，它可通过对引用源的订阅生成事件通知。

从节点“A”开始，且后续仅 HasEventSource 引用类型或其子类型的引用，应绝不能返回到“A”。但允许后续该引用有多条路径通向另一个节点“B”。

7.20 HasNotifier

HasNotifier 引用类型是一种具体的引用类型并可直接使用。它是 HasEventSource 的子类型。

此引用类型的语义是关联有通知器的对象节点和其他通知器对象节点。该引用类型用来建立事件通知对象的层次结构组织。它是 7.19 中定义的 HasEventSource 引用类型的子类型。

此引用类型的源节点应为事件订阅源的对象或视图。此引用的目标节点应为事件订阅源的对象。事件订阅的源是自身有在 EventNotifier 属性中对“SubscribeToEvents”比特置位的对象。

如果此类型引用的目标节点生成一个事件，那么该事件也应在引用的源节点中提供。

可能的事件引用的组织示例见图 24。在此示例中，指向“液位传感器”对象的未过滤事件订阅将提供事件源“低限”和“高限”给订阅者。指向“区域 1”对象的未过滤事件订阅将提供来自“储罐 A”的事件

源和所有低于“储罐 A”的通知器源。

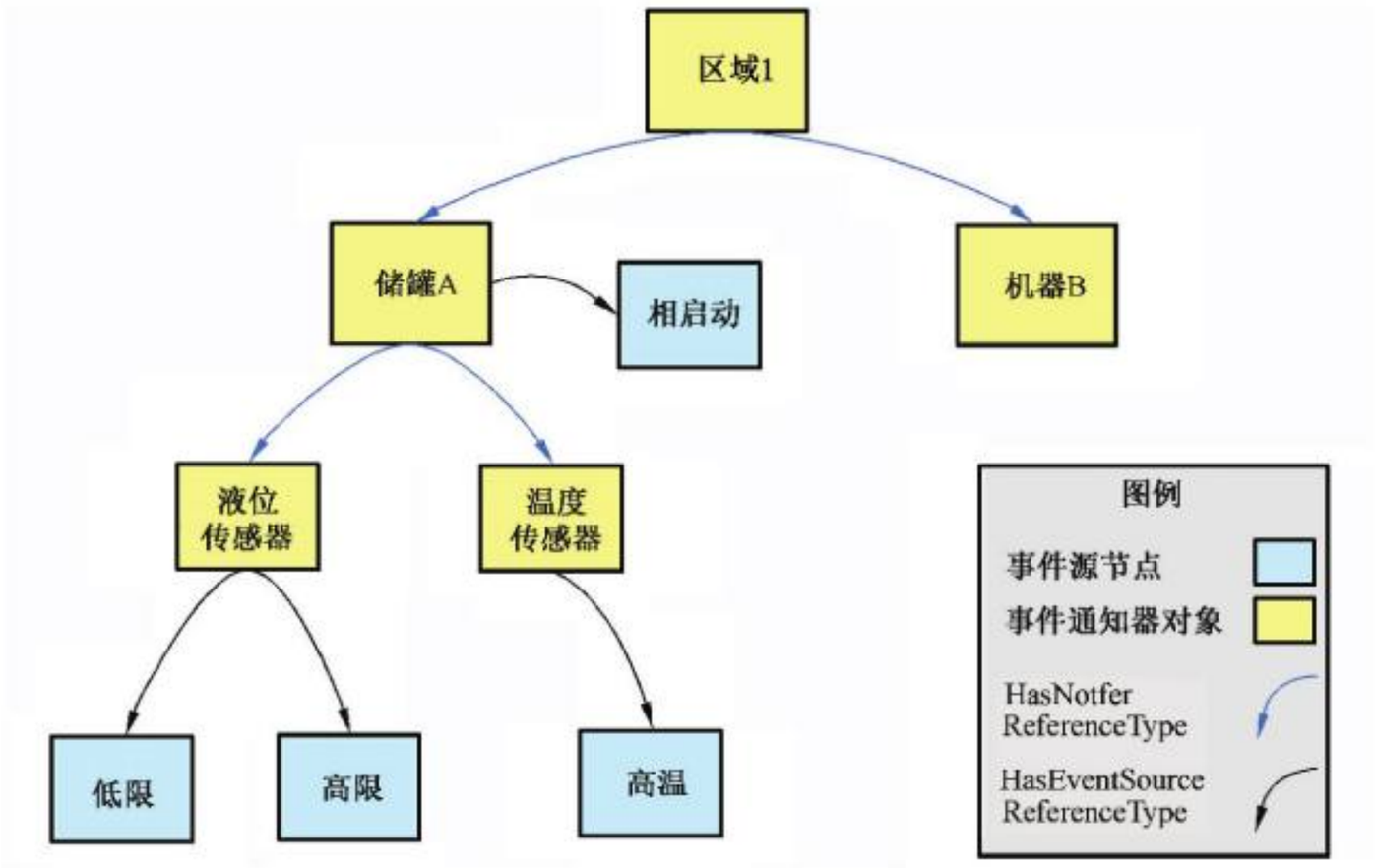


图 24　事件引用示例

图 25 示出第二个更复杂的事件引用组织的示例。在此示例中，包含了来自服务器服务对象的显式引用，该服务对象是所有服务事件的源。引入第二个事件组织来收集“Tank Farm 1”相关的事件。指向“Tank Farm 1”对象的未过滤事件订阅将提供来自“Tank B”、“Tank A”以及所有低于“Tank B”和“Tank A”的通知器源的事件源。

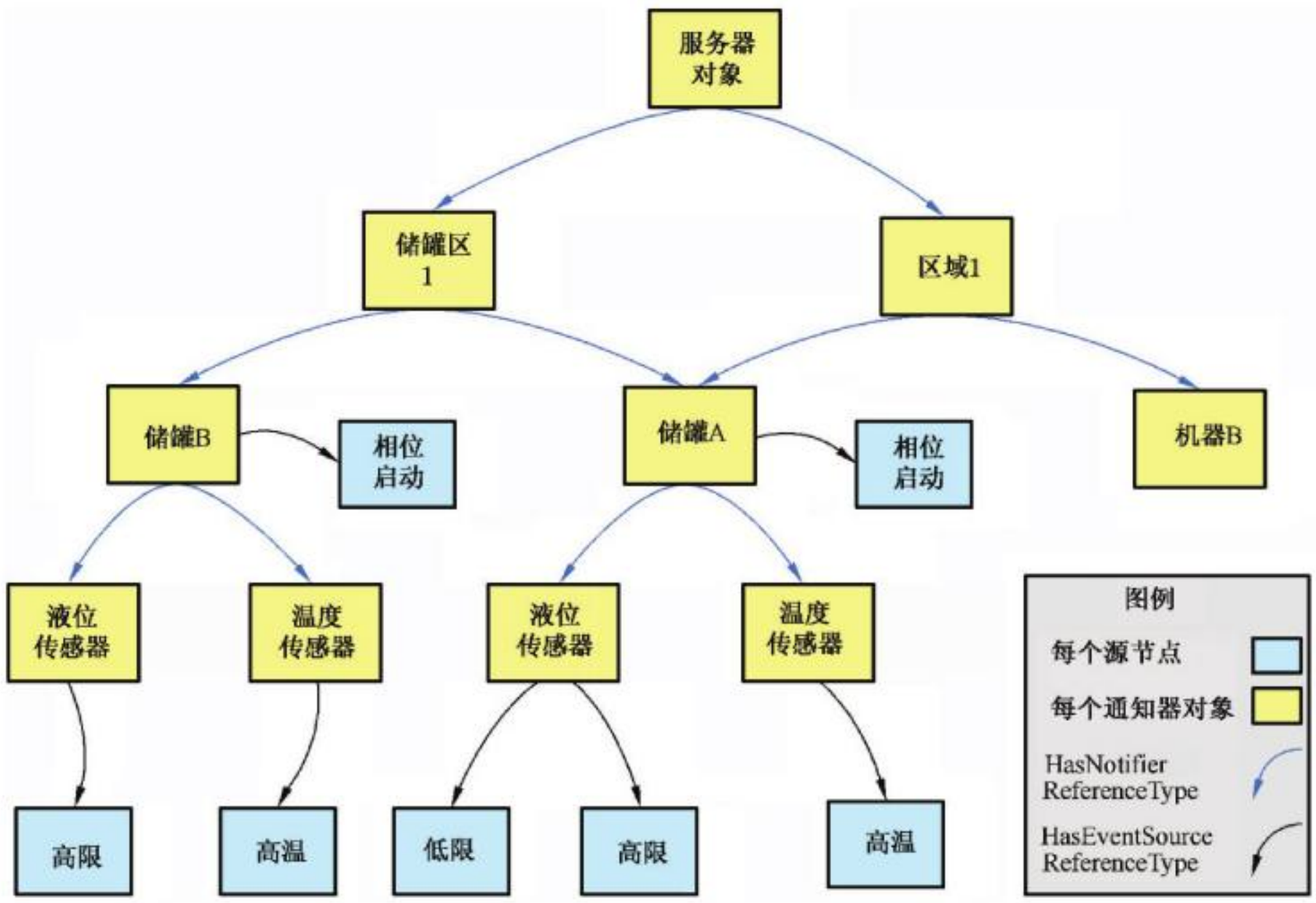


图 25　复杂事件引用示例

8 标准数据类型

8.1 概述

如下章节定义了数据类型。IEC 62541 5 规定了它们在地址空间和数据类型层次结构中的表示。此标准的其他部分可能规定了其他的数据类型。

8.2 NodeId

8.2.1 概述

该内置数据类型包括了服务器中标识节点的 3 个元素。它们的定义见表 17。

表 17 NodeId 定义

名称	类型	描述
NodeId	structure	
NamespaceIndex	UInt16	命名空间 URI 的索引，见 8.2.2
IdentifierType	Enum	Identifier 的格式和数据类型，见 8.2.3
Identifier	*	OPC UA 服务器地址空间中的节点标识符，见 8.2.4
“*”表示类型可选。		

将标识符编码成 OPC UA 消息的描述见 IEC 62541-6。

8.2.2 NamespaceIndex

命名空间是一个 URI，它标识了负责分配 NodeId 标识符元素的命名权威结构。命名权威结构包括本地服务器、底层系统、标准组织和团体。建议大部分节点将使用服务器的或底层系统的 URI。

使用命名空间 URI 允许多个 OPC UA 服务器附属到相同的底层系统，以使用相同标识符来标识相同对象。它使连接到那些服务器的客户端能够识别它们共有的对象。

命名空间 URI，例如服务器名称，由 OPC UA 服务中的数值标识，以允许更有效的传输和处理（如查表）。用来标识命名空间的数值与 NamespaceArray 中的索引相对应。NamespaceArray 是作为地址空间中服务对象的一部分的变量。（其定义见 IEC 62541-5。）

用于 OPC UA 命名空间的 URI 是：“http://opcfoundation.org/UA/”。

它在命名空间表中对应的索引是 0。

8.2.3 IdentifierType

IdentifierType 元素标识了 NodeId 的类型、格式和范围。它的值见表 18。

表 18 IdentifierType 值

值	描 述
NUMERIC_0	数值
STRING_1	字符串值
GUID_2	全球唯一标识符
OPAQUE_3	命名空间特定格式

通常 NodeId 的范围是定义它们所位于的服务器。对于 NodeId 的某些类型，NodeId 可唯一标识系统中的一个节点，或跨系统的一个节点(如 GUID)。系统范围的和全球唯一的标识符允许客户端跟踪节点，例如工作指令，它们在 OPC UA 服务器间移动就像它们工作在同一系统里。

OPAQUE 标识符是指标识符格式为自由格式的字节字符串，并且可能是人无法解释的。

8.2.4 Identifier(标识符)值

Identifier 值元素用在前三个元素中来标识节点。其数据类型和格式由 IdType 定义。

IdType STRING_1 的标识符值限定为 4 096 个字符。IdType OPAQUE_3 的标识符值限定为 4 096个字节。

NULL NodeId 有特殊的意义。例如，如果以参数形式传递 NULL NodeId 时，IEC 62541-4 定义的许多服务定义了特殊的行为。各 NodeId 有一组代表 NULL NodeId 的标识符值。这些值在表 19 中给出。

表 19 NodeId NULL 值

IdType	标 识 符
NUMERIC_0	0
STRING_1	NULL 或空字符串("")
GUID_2	初始化为 0 的 Guid(如 00000000-0000-0000-0000-000000)
OPAQUE_3	长度为 0 的字节字符串

通常 NULL NodeId 的 NamespaceIndex 等于 0。

地址空间中的节点应不把 NULL 作为其 NodeId。

8.3 QualifiedName

此内置数据类型包含了一个限定名称。例如，它用作 BrowseName。其元素在表 20 中定义。QualifiedName 的名称部分限于 512 个字符。

表 20 QualifiedName 定义

名称	类型	描 述
QualifiedName	Structure	
NamespaceIndex	UInt16	定义了名称的命名空间的索引。 此索引是本地服务器 NamespaceArray 中该命名空间的索引。 客户端可以读取 NamespaceArray 变量以访问命名空间中的字符串值
Name	String	QualifiedName 的文本部分

8.4 LocaleId

简单数据类型规定为由语言部分和国家/地区部分组成的字符串，规定见 IETF RFC 3066。<国家/地区>部分通常以连字符为前缀。LocaleId 字符串的格式如下：

<语言>[-<国家/地区>]，其中<语言>是代表语言的两字母 ISO 639 代码，<国家/地区>是代表国家/地区的两字母 ISO 3166 代码。

IETF RFC 3066 规定的构造 LocaleId 的规则限定如下：

a) 本规范仅允许零或一个<国家/地区>在<语言>之后；

b) 本规范也允许"-CHS"和"-CHT"3 个字母作为<国家/地区>代码，表示中文的"简体"和"繁体"；

c) 本规范也允许使用其他客户端或服务器认为必要的<国家/地区>代码。

表 21 给出了 OPC UA LocaleId 的例子。客户端和服务器通常提供明确标识语言和国家/地区的 LocaleId。

表 21 LocaleId 示例

地点	OPC UA LocaleId
英语	en
英语(美国)	en-US
德语	de
德语(德国)	de-DE
德语(奥地利)	de-AT

空的或 NULL 字符串表示该 LocaleId 未知。

8.5 LocalizedText

此内置数据类型定义了一个含有本地特定翻译的字符串的结构，它在用于地点的标识符中规定。其元素定义见表 22。

表 22 LocalizedText 定义

名称	类型	描　　述
LocalizedText	Structure	
text	String	本地化的文本
locale	LocaleId	用于地点的标识符(如"en-US")

8.6 参数

该结构化数据类型定义了方法输入或输出参数规范。例如用在用于方法的输入和输出参数属性。其元素描述见表 23。

表 23 参数定义

名称	类型	描述
Arguemnt	Structure	
name	String	参数的名称
dataType	NodeId	此参数数据类型的 NodeId
valueRank	Int32	指示该数据类型是否是一个数组，并且该数组的维数是多少。 它可有如下值： n>1：数据类型是规定维数的数组； OneDimension(1)：数据类型是一维数组； OneOrMoreDimensions(0)：数据类型是一维或多维数组； Scalar(−1)：数据类型不是数组； Any(−2)：数据类型可以是标量或任意维数的数组； ScalarOrOneDimension(−3)：数据类型可以是标量或一维数组
arrayDimension	UInt32[]	规定了数组数据类型各维的长度。它用于描述数据类型的容量，而非当前大小。 元素的个数应等于 valueRank 的值。如果 valueRank< 0，则它必须为 NULL。 单个维数值为 0 指示维数长度可变
Description	LocalizedText	参数的本地化描述

8.7 BaseDataType

此抽象数据类型定义了一个可有任意有效数据类型的值。
它定义了一个特殊值 NULL，它指示值不存在。

8.8 Boolean

此内置数据类型定义了一个为 TRUE 或 FALSE 的值。

8.9 Byte

此内置数据类型定义了一个范围在 0 到 255 的值。

8.10 ByteString

此内置数据类型定义了一个为字节序列的值。

8.11 DateTime

此内置数据类型定义了公历日期。此数据类型详见 IEC 62541-6。

8.12 Double

此内置数据类型定义了符合 IEEE 754 1985 双精度数据类型定义的值。

8.13 Duration

此简单数据类型是定义了毫秒级时间间隔(可使用分数来定义亚毫秒值)的 Double。负值通常是

无效的，但在使用 Duration 时可有特殊意义。

8.14 Enumeration

此抽象数据类型是所有枚举型数据类型的基础数据类型，如 8.30 定义的节点类。所有继承此数据类型的数据类型对编码有特殊处理，见 IEC 62541-6。所有枚举型数据类型应继承此 DataType。

8.15 Float

此内置数据类型定义了符合 IEEE 754-1985 单精度数据类型定义的值。

8.16 Guid

此内置数据类型定义的值是 128 位全球唯一标识符。此数据类型详见 IEC 62541-6。

8.17 SByte

此内置数据类型定义了一个在－128 到 127 之间的有符号整数值。

8.18 IdType

此数据类型为标识 NodeId 的 IdType 的枚举。其值在表 18 中定义。此数据类型在 NodeId 中的使用描述见 8.2.3。

8.19 Image

此抽象数据类型定义了表示图形的 ByteString。

8.20 ImageBMP

此简单数据类型定义了表示 BMP 格式图形的 ByteString。

8.21 ImageGIF

此简单数据类型定义了表示 GIF 格式图形的 ByteString。

8.22 ImageJPG

此简单数据类型定义了表示 JPG 格式图形的 ByteString。JPG 定义见 ISO/IEC 10918-1。

8.23 ImagePNG

此简单数据类型定义了表示 PNG 格式图形的 ByteString。PNG 定义见 ISO/IEC 15948。

8.24 Integer

此抽象数据类型定义了一个整数，其长度由子类型定义。

8.25 Int16

此内置数据类型定义了在－32 768 到 32 767 之间的有符号整型值。

8.26 Int32

此内置数据类型定义了在－2 147 483 648 到 2 147 483 647 之间的有符号整型值。

8.27 Int64

此内置数据类型定义了在−9 223 372 036 854 775 808 到 9 223 372 036 854 775 807 之间的有符号整型值。

8.28 TimeZoneDataType

此结构化数据类型定义了本地时间,可能有或没有考虑夏令时。其元素描述见表 24。

表 24 TimeZoneDataType 定义

名称	类型	描　　述
TimeZoneDataType	structure	
Offset	UInt16	以分钟表示的与 UtcTime 的偏移量
DaylightSavingInOffset	Boolean	如果为 TRUE,那么夏令时(DST)有效,并且 Offset 含 DST 修正。如果为 FALSE,那么 Offset 不含 DST 修正,并且 DST 可能有效或无效

8.29 NamingRuleType

数据类型是标识 NamingRule(见 6.4.4.2.1)的枚举。其值的定义见表 25。

表 25 NamingRuleType 值

名　　称
MANDATORY_1
OPTIONAL_2
CONSTRAINT_3

8.30 NodeClass(节点类)

数据类型是标识节点类的枚举。其值的定义见表 26。

表 26 NodeClass 值

名　　称
OBJECT_1
VARIABLE_2
METHOD_4
OBJECT_TYPE_8
VARIABLE_TYPE_16
REFERENCE_TYPE_32
DATA_TYPE_64
VIEW_128

8.31 Number

此抽象数据类型定义了一个数字。详细内容由其子类型定义。

8.32 String

此内置数据类型定义了 Unicode(统 -码)字符串,它应不包含非空格的控制字(0x00-0x08.0x0E-0x1F 或 0x7F)。

8.33 Structure

此抽象数据类型是所有结构化数据类型的基础数据类型,如 8.6 定义的参数。所有继承此数据类型的数据类型对编码有特殊处理,见 IEC 62541-6。如果它们没有解释为本标准的源语,所有结构化数据类型应继承此数据类型(如 8.2 定义的 NodeId.NodeId 是结构化的,但以一种特殊方式处理,见 IEC 62541-6)。

8.34 UInteger

此抽象数据类型定义了无符号整型,其长度由子类型定义。

8.35 UInt16

此内置数据类型定义了在 0 到 65535 之间的无符号整型值。

8.36 UInt32

此内置数据类型定义了在 0 到 4294967295 之间的无符号整型值。

8.37 UInt64

此内置数据类型定义了在 0 到 18446744073709551615 之间的无符号整型值。

8.38 UtcTime

此简单数据类型是用来定义协调世界时(UTC)值的 DateTime。所有在 OPC UA 服务器和客户端之间传输的时间值为 UTC 值。客户端应提供 UTC 和本地时间的任一转换。此数据类型详见 IEC 62541-6。

8.39 XmlElement

此内置数据类型用来定义 XML 元素。此数据类型详见 IEC 62541-6。

XML 数据可总是建模成带有单个 DataTypeEncoding 的结构数据类型的子类型,表示定义 XML 元素的 XML 复杂类型(没有必要访问 XML 结构来定义 DataTypeEncoding)。由于这个原因,服务器应绝不定义使用 XmlElement 数据类型的变量,除非该服务器没有关于可能在该变量值中的 XML 元素的信息。

9 标准事件类型(EventType)

9.1 概述

如下章节定义了事件类型。它们在地址空间的表示见 IEC 62541-5。此标准的其他部分可能规定了其他的事件类型。图 26 简略地描述了这些事件类型的层次结构。

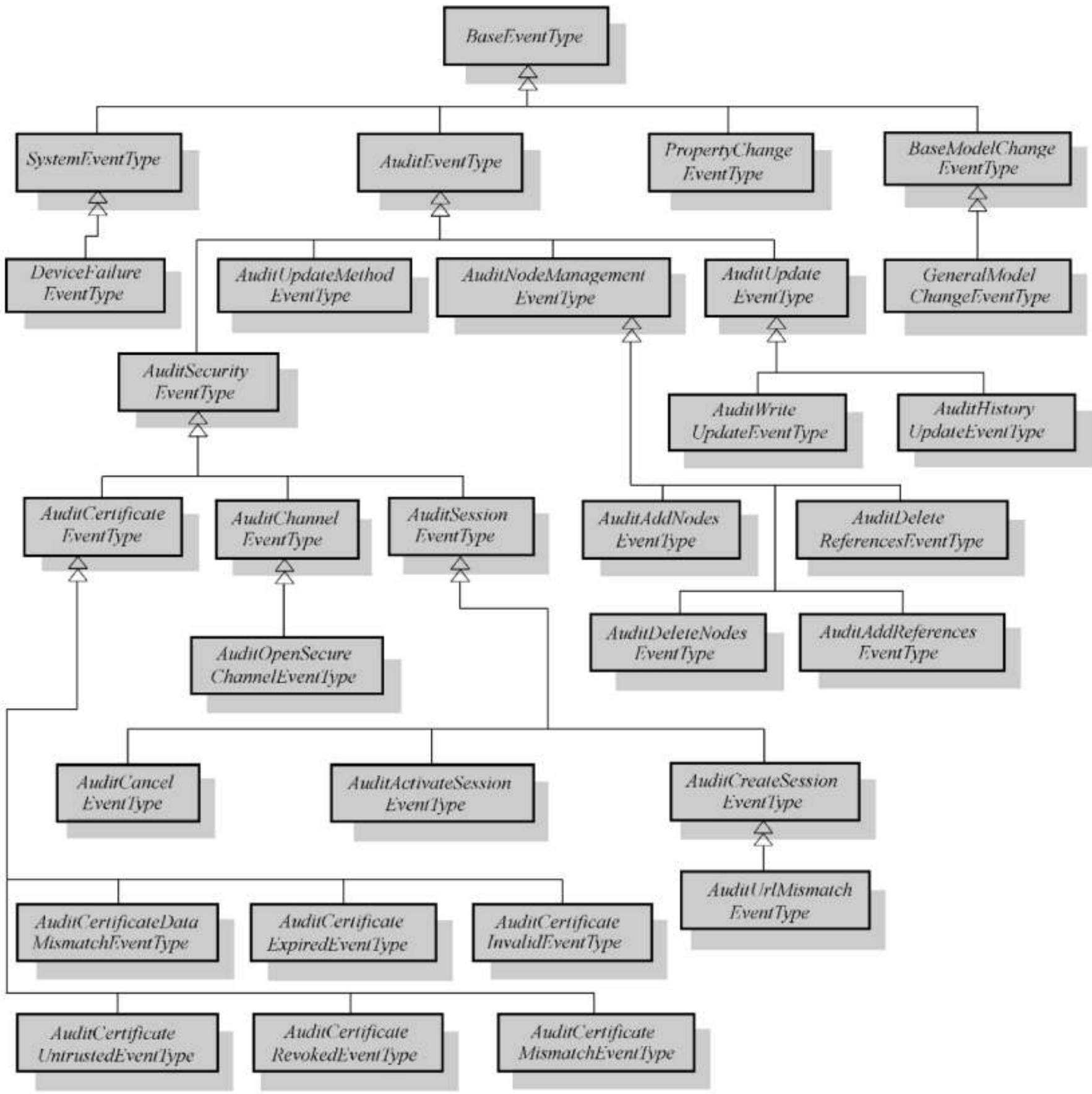


图 26　标准事件类型层次结构

9.2　BaseEventType

BaseEventType 定义了事件的所有通用特性。所有其他事件类型由它衍生。没有关于此类型的其他语义。

9.3　SystemEventType

SystemEvent 是由于一些在服务器中或服务器所表示的系统发生的事件而产生。

9.4　AuditEventType

AuditEvent 是由于服务器的客户端在服务器上采取的动作所产生。例如，响应客户端对变量发出写操作，服务器将生成 AuditEvent，它将变量描述为源，将用户和客户端会话描述为事件发起者。

图 27 说明了定义的 OPC UA 服务器在响应可审核的动作请求时的行为。如果动作被接受，则生成动作 AuditEvent，并由服务器进行处理。如果由于安全原因动作不被接受，则生成安全 AuditEvent，并由服务器进行处理。服务器可能涉及过程中的底层设备或系统，但服务器有责任向任何感兴趣的客

户端提供该事件。客户端可自由地从服务器订阅事件,且将接收到响应常规发布请求的 AuditEvent。

所有动作请求包含可读的 AuditEntryId。该 AuditEntryId 包含在 AuditEvent 中,以允许用户将事件关联到发起的动作。AuditEntryId 典型地包含了谁发起该动作和它在哪里被发起的。

服务器可选择在必备事件订阅发给客户端之外,可选地坚持 AuditEvent。

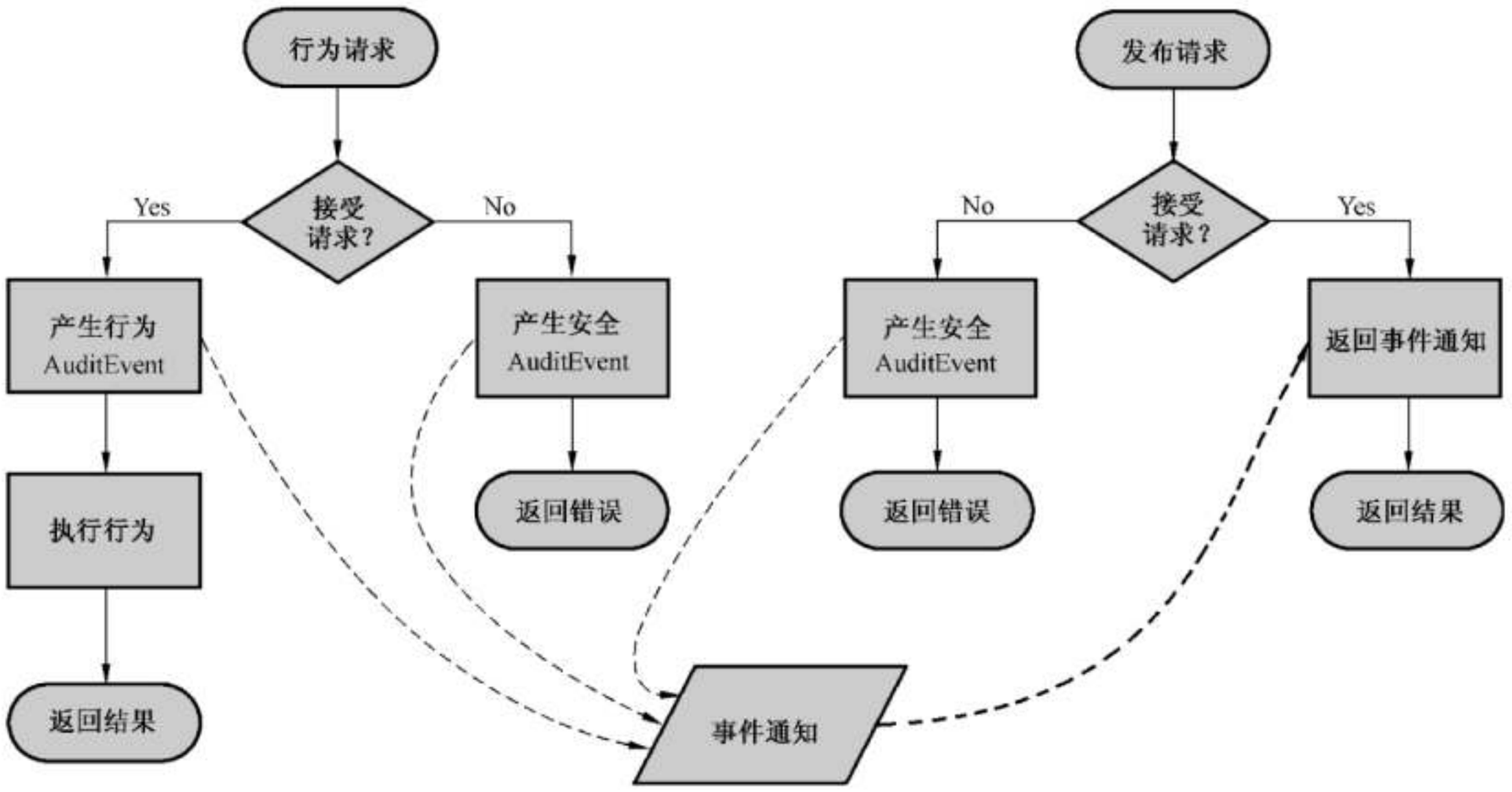


图 27 服务器审核行为

图 28 说明了聚合服务器在响应可审计动作请求时的预期行为。此用例包含了该聚合服务器将动作传递给其聚合的服务器之一。也就是说,该请求可能失败,并在聚合服务器中生成一个安全 AuditEvent。常规过程是将动作传递给一个聚集的服务器来处理。该聚合服务器将依次跟着此行为或通

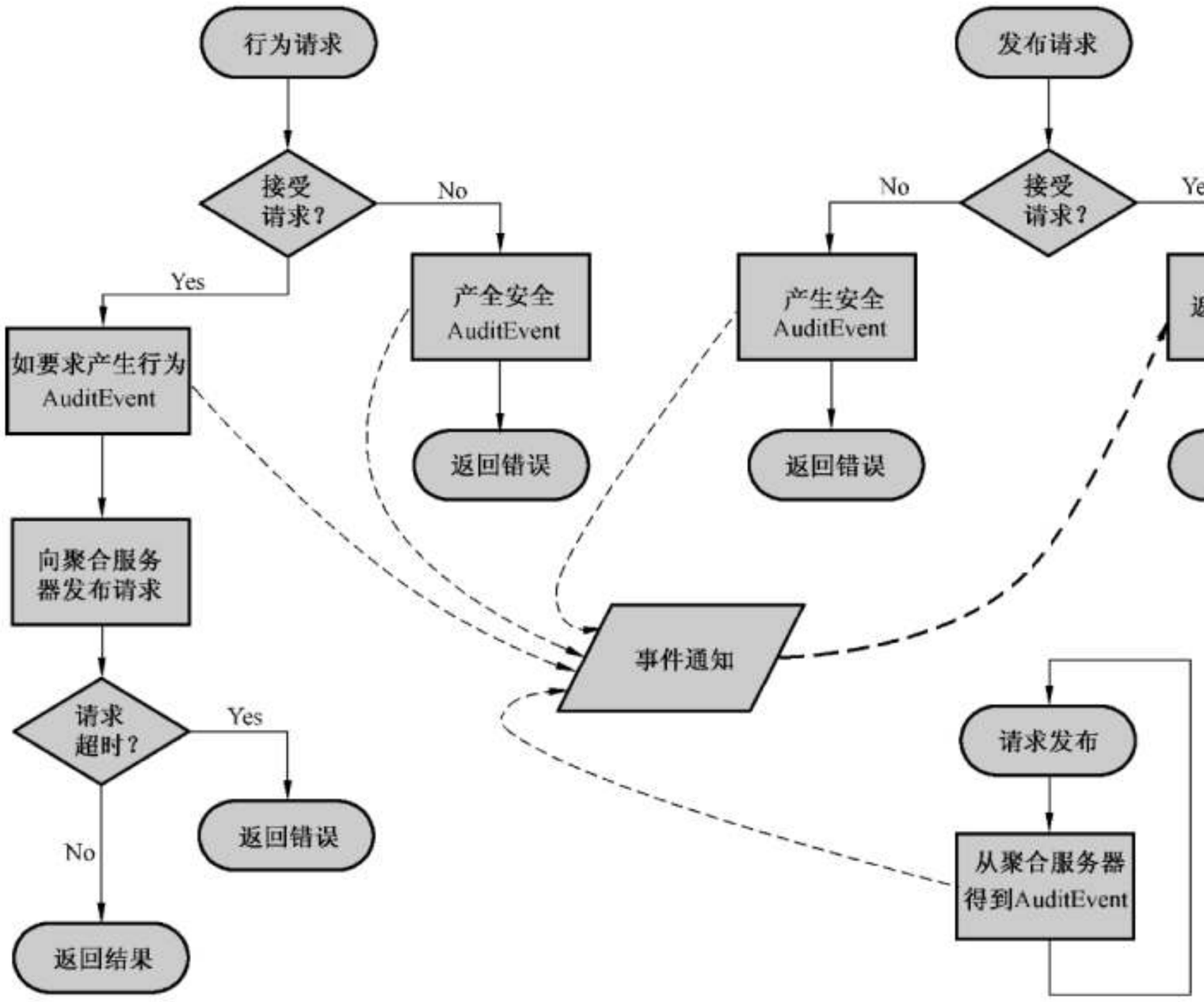


图 28 聚合服务器的审核行为

用行为，并生成适当的 AuditEvent。聚合服务器周期性地向聚合服务器发布出版请求。这些收集的事件与自生成事件合并，并且可用于订阅客户端。如果聚合服务器支持可选的坚持 AuditEvent，那么坚持收集的事件和本地生成的事件。

当将请求传递给聚合服务器时，聚合服务器可映射已认证的用户账户，以向一个或其自己的账户发出请求。然而，它应在传递的时候保存 AuditEntryId 像收到时一样。聚合服务器也可在传递给聚合服务器之前生成其自己的 AuditEvent 用于请求，特别地，如果聚合服务器需要将该请求分成多个请求，而各请求都指向分别的聚合服务器，或如果部分请求由于聚合服务器的安全而被否认时。

9.5 AuditSecurityEventType

它是 AuditEventType 的子类型，仅用于对安全相关事件进行分类。此类型具备其父类型的所有行为。

9.6 AuditChannelEventType

它是 AuditSecurityEventType 的子类型，用于对 IEC 62541-4 定义的 SecureChannel 服务集的安全相关事件进行分类。

9.7 AuditOpenSecureChannelEventType

它是 AuditChannelEventType 的子类型，用于在调用 IEC 62541-4 定义的 OpenSecureChannel 服务时生成的事件。

9.8 AuditSessionEventType

它是 AuditSecurityEventType 的子类型，用于对 IEC 62541-4 定义的 Session 服务集的安全相关事件进行分类。

9.9 AuditCreateSessionEventType

它是 AuditSessionEventType 的子类型，用于在调用 IEC 62541-4 定义的 CreateSession 服务时生成的事件。

9.10 AuditUrlMismatchEventType

它是 AuditCreateSessionEventType 的子类型，用于在调用 IEC 62541-4 定义的 CreateSession 服务时，如果服务调用中使用的 EndpointUrl 不匹配服务器的主机名称(详见 IEC 62541-4)生成的事件。

9.11 AuditActivateSessionEventType

它是 AuditSessionEventType 的子类型，用于在调用 IEC 62541-4 定义的 ActivateSession 服务时生成的事件。

9.12 AuditCancelEventType

它是 AuditSessionEventType 的子类型，用于在调用 IEC 62541-4 定义的 Cancel 服务时生成的事件。

9.13 AuditCertificateEventType

它是 AuditSecurityEventType 的子类型，仅用于对认证相关事件进行分类。此类型的行为与其父类型的所有行为一致。在其他服务调用相关的 AuditEvent 以外，生成这些 AuditEvent 用于证书错误。

9.14 AuditCertificateDataMismatchEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果 URL 中用于连接服务器的主机名称与认证中规定的主机名称不相同，或者如果应用和软件证书包含应用或与证书提供的 ApplicationDescription 规定的 URI 与产品 URI 不匹配，则生成此 AuditEvent。关于证书的更多详细内容见 IEC 62541-4。

9.15 AuditCertificateExpiredEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果当前时间没在证书的有效期内，则生成此 AuditEvent。

9.16 AuditCertificateInvalidEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果该证书结构是无效的或者如果证书有无效签名，则生成此 AuditEvent。

9.17 AuditCertificateUntrustedEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果证书不被信任，也就是说，如果证书发行人是未知的，则生成此 AuditEvent。

9.18 AuditCertificateRevokedEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果证书被撤销，或者如果撤销列表不可用（即网络干扰阻止应用访问该列表），则生成此 AuditEvent。

9.19 AuditCertificateMismatchEventType

它是 AuditCertificateEventType 的子类型，仅用于对证书相关事件进行分类。此类型的行为与其父类型的所有行为一致。如果有用的证书集不匹配证书的使用请求（即应用、软件或认证机构），则生成此 AuditEvent。

9.20 AuditNodeManagementEventType

它是 AuditEventType 的子类型，用于对节点管理相关事件进行分类。此类型的行为与其父类型的所有行为一致。

9.21 AuditAddNodesEventType

它是 AuditNodeManagementEventType 的子类型，用于在调用 IEC 62541-4 定义的 AddNode 服务时生成的事件。

9.22 AuditDeleteNodesEventType

它是 AuditNodeManagementEventType 的子类型，用于在调用 IEC 62541-4 定义的 DeleteNode 服务时生成的事件。

9.23 AuditAddReferencesEventType

它是 AuditNodeManagementEventType 的子类型，用于在调用 IEC 62541-4 定义的 AddReference

服务时生成的事件。

9.24 AuditDeleteReferencesEventType

它是 AuditNodeManagementEventType 的子类型，用于在调用 IEC 62541-4 定义的 DeleteReference 服务时生成的事件。

9.25 AuditUpdateEventType

它是 AuditEventType 的子类型，用于对更新相关事件进行分类。此类型的行为与其父类型的所有行为一致。

9.26 AuditWriteUpdateEventType

它是 AuditUpdateEventType 的子类型，用于对写更新相关事件进行分类。此类型的行为与其父类型的所有行为一致。

9.27 AuditHistoryUpdateEventType

它是 AuditUpdateEventType 的子类型，用于对历史更新相关事件进行分类。此类型的行为与其父类型的所有行为一致。

9.28 AuditUpdateMethodEventType

它是 AuditUpdateEventType 的子类型，用于对方法相关事件进行分类。此类型的行为与其父类型的所有行为一致。

9.29 DeviceFailureEventType

DeviceFailureEvent 表示底层系统设备中的失效。

9.30 ModelChangeEvent

9.30.1 概述

生成 ModelChangeEvent 来表示地址空间结构的变化。该变化可包含增加或删除节点或引用。尽管变量或变量类型与其数据类型的关系没有使用引用来建模，对变量或变量类型的数据类型属性的改变也考虑成模型变化，因此如果数据类型属性改变，则生成一个 ModelChangeEvent。

9.30.2 NodeVersion 属性

ModelChangeEvent 与节点的 NodeVersion 属性之间相互关联。每当给节点发出 ModelChangeEvent，其 NodeVersion 应改变，并且每当 NodeVersion 改变时，则应生成一个 ModelChangeEvent。服务器应支持 ModelChangeEvent 和 NodeVersion 属性，或两者都不支持，但绝不能仅支持其中之一。

9.30.3 视图

ModelChangeEvent 总是在包含缺省视图的视图环境中生成，其中考虑到整体地址空间。因此，仅报告 ModelChangeEvent 的通知是视图节点和表示缺省视图的服务器对象。由于 ModelChangeEvent 可能影响了不同的视图，每个生成 ModelChangeEvent 的动作可能导致几个事件。例如，如果从地址空间删除一个节点，并且此节点也含在视图“A”中，那么就会有一个事件以地址空间作为上下文，另一个事件以视图“A”作为上下文。如果节点仅从视图“A”中移除，但仍存在于地址空间，那么它将仅生成

ModelChangeEvent 用于视图“A”。

如果客户端不想接收变化的副本，那么它应使用事件订阅的过滤机制仅过滤缺省视图，并禁止 ModelChangeEvent 用其他视图作为上下文。

当发布视图的 ModelChangeEvent，并且该视图支持 ViewVersion 属性，那么应更新该 ViewVersion。

9.30.4 事件压缩

每当有更新时，不要求用实现来发布更新事件。OPC UA 服务器能够按更大单元对一系列交互或简单更新分组。这一系列可组成一组逻辑变化或一组暂时变化。单个 ModelChangeEvent 可在该系列的最后一次变化后发布，以覆盖所有的变化，这被称为事件压缩。NodeVersion 和 ViewVersion 的变化可因而反映出一组变化，而不是单个变化。

9.30.5 BaseModelChangeEventType

BaseModelChangeEventType 是 ModelChangeEvent 的基础类型，且不包含关于变化的信息，但仅指示发生了变化。因此，客户端应假设任何或所有节点可能发生了变化。

9.30.6 GeneralModelChangeEventType

GeneralModelChangeEventType 是 BaseModelChangeEventType 的子类型。它包含关于变化节点的信息，以及发生 ModelChangeEvent 的动作(如增加一个节点或删除一个节点等)。如果受影响的节点是个变量或对象，那么也会出现类型定义节点。

为了允许事件压缩，GeneralModelChangeEventType 包含了一组这样结构。

9.30.7 ModelChangeEvent 指南

定义了两类 ModelChangeEvent：没有包含任何关于变化信息的 BaseModelChangeEventType，和通过数组标识改变的节点的 GeneralModelChangeEventType。所用的精度取决于 OPC UA 服务器的能力和更新的性质。OPC UA 服务器可根据环境使用任一 ModelChangeEvent 类型。它也可定义增加了附加信息的事件类型的子类型。

为了保证互操作性，应遵守如下对事件的指南。

——如果出现 GeneralModelChangeEventType 数组，那么它应标识从上述的 ModelChangeEvent 之后的每个发生变化的节点。

OPC UA 服务器应为一次更新或一系列更新发出一个 ModelChangeEvent。它不应为相同更新发布多个类型的 ModelChangeEvent。

——响应 ModelChangeEvent 的任何客户端应响应 BaseModelChangeEventType 的任何事件，包含例如 GeneralModelChangeEventType 的子类型。

如果客户端不能解释 BaseModelChangeEventType 子类型的附加信息，那么它应以处理 BaseModelChangeEventType 的事件相同的方式处理这些类型的事件。

9.31 SemanticChangeEventType

9.31.1 概述

生成 SemanticChangeEventType 来指示地址空间语义的变化。该变化包含特性的 Value 属性变化。

SemanticChangeEventType 包含了关于拥有该特性发生变化的节点信息。如果它是个变量或对

象，则也有类型定义节点。

特性的 AccessLevel 属性的 SemanticChange 比特指示了 SemanticChangeEvent 是否考虑了该特性值的变化。

9.31.2 ViewVersion 和 NodeVersion 特性

ViewVersion 和 NodeVersion 特性不会因为 SemanticChangeEvent 的发布而变化。

9.31.3 视图

SemanticChangeEvent 在视图上下文中用 ModelChangeEvent 相同的方式进行处理，见 9.30.3。

9.31.4 事件压缩

SemanticChangeEvent 可用 ModelChangeEvent 相同的方式进行压缩，见 9.30.4。

附 录 A
（资料性附录）
如何使用地址空间模型

A.1 概述

本附录列出了如何使用地址空间模型的一些通常考虑。本附录只是资料性信息，也就是说，每个服务器供应商能以满足其需求的适当方式对其数据进行建模。然而，它给出了这些服务器供应商可以考虑的某些提示。

典型地，OPC UA 服务器提供来自底层系统（如设备、组态、OPC COM 服务器等）的数据。因此，数据建模取决于底层系统的模型，以及访问 OPC UA 服务器的客户端的要求。期待在 OPC UA 之上制定相应的规范，以规定如何对数据建模的附加规则。然而，下面的章条也给出了关于对数据进行建模的 OPC UA 的不同概念，以及何时使用它们的通常考虑。

IEC 62541-5:2011 的附录 A 给出了对 IEC 62541-5:2011 中定义的服务器信息进行建模时，所做的设计决策的概述。

A.2 类型定义

在同一系统中类型信息可能多次使用，或者为了支持同一类型定义的不同系统间的互操作的任何时候，都应使用类型定义。

A.3 对象类型

5.5.1 规定“对象用于表示系统、系统组件、现实对象和软件对象。”因此，如果这些对象的类型定义有用时（见 A.2），就应使用对象类型。

更抽象地讲，对象用于对地址空间中的变量和其他对象进行分组。因此，在描述某些共同结构、对象组和/或变量组时，应该使用对象类型。客户端能够使用该知识来对对象类型结构进行编程，并且使用 IEC 62541-4 中定义的针对实例的 TranslateBrowsePathsToNodeIds 服务。

只具有一个值的简单对象（如简单热传感器）也可以建模为变量类型。然而，应考虑扩展机制（如复杂热传感器可以具有多个值）和对象应被作为客户端的 GUI 的一个对象或者只是一个值。无论何时，建模者都应质疑使用只有一个变量的对象类型的解决方案是否是最佳的。

A.4 变量类型

A.4.1 概述

变量类型只用于数据变量[1]，并且在多个变量具有同一语义（如设定值）时使用。没有必要只是为了反映变量的数据类型（如“Int32 变量类型”）而定义一个变量类型。

1） 变量类型而不是特性类型，属性类型用于所有特性。

A.4.2 特性或数据变量

除了第4章描述的特性和数据变量的语义上的不同之外，还存在语法上的不同。特性由其BrowseName标识，也就是说，如果多次使用具有同一语义的特性，那么他们应该总是有同样的BrowseName。数据变量的同一语义表现为变量类型。

如果不清楚基于第4章中描述的语义应使用何种概念，不同的语法会有帮助。下面几方面表示了它应该是数据变量。

——如果它是复杂变量或者它包含特性形式的附加信息；

类型定义可以被改进(子类型)；

如果需要使用类型定义，那么客户端应使用IEC 62541-4中定义的AddNodes服务来创建新的类型定义实例；

如果它是复杂变量的一个部件，表现为复杂变量值的一部分。

A.4.3 许多变量和/或复杂数据类型

当客户端需要使用复杂数据结构时，一般有三种不同的方法：

a) 使用总是反映简单结构部分的简单数据类型创建几个简单变量。根据数据结构，使用对象对变量进行分组；

b) 创建一个复杂数据类型，并使用这个数据类型创建一个简单变量；

c) 创建一个复杂数据类型，并使用这个数据类型创建一个复杂变量，使用简单数据类型将这个复杂数据结构作为复杂变量的变量。

第一种方法的优点是数据的复杂结构在地址空间中可见；一般的客户端无需了解用户定义的数据类型，就可以容易地访问那些数据；客户端能够访问复杂数据的单个部分。这种方法的缺点是对单个数据的访问不提供任何的事务上下文；对特定客户端，服务器首先要对数据进行转换，然后客户端需要再次对数据进行转换，以得到底层系统提供的数据结构。

第二种方法的优点是数据在事务上下文中访问，这样复杂数据类型能以如下方式建立，服务器不必转换数据就能直接将它们传递给能够直接使用他们的特定客户。缺点是通常客户端不能访问和解释数据，或者至少在读取数据类型描述并解释数据上有困难。数据结构在地址空间中不可见；因为地址空间中不存在足够的空间，所以不能加入描述数据结构的额外特性。如果不访问全部数据结构，就无法读取数据的单个部分。

第三种方法组合了上面两种方法。因此，特定客户端能够访问事务上下文中原始格式的数据，同时一般客户端能够访问复杂变量的部件的简单数据类型。缺点是服务器应能够提供原始格式，同时也能够解释该数据并在简单数据类型中提供信息。

推荐使用第一种方法。当需要事务上下文，或客户端应能得到大量数据而不只是几个单个值时，第三种方法是适当的。然而，服务器并不总能解释底层系统的复杂数据，因此不得不使用第二种方法只是将数据传递给能够解释该数据的特定客户端。

A.5 视图(View)

服务器定义的视图能用于展现适用于特定类型客户端(如维护客户端、工程客户端等)的部分地址空间。视图只提供该客户端所需的信息，并隐藏不必要的信息。

A.6 方法(Method)

当服务器需要某些输入和发送一个结果时,应使用方法。此时,应避免使用变量来写输入值,也避免使用变量来得到输出值,这应在 OPC COM 中完成,因为没有可用的方法。然而,简单 OPC COM 包可用于完成这些。

也能够使用方法来触发服务器中不需要输入和/或输出参数的某些操作。

全局方法,也就是不能直接分配给特定对象的方法,应被分配给 IEC 62541-5 中定义的服务器对象。

A.7 定义引用类型

只有在预先定义的引用类型不适用时,才应定义新的引用类型。无论何时定义一个新的引用类型,都应使用最适当的引用类型作为其子类型。

期望服务器只有新定义的分层的引用类型来表示不同的层次关系,并只有非分层的引用类型来表示地址空间中节点间的相互关系。

A.8 定义建模规则

如果对于服务器表示的模型,预先的建模规则不适用,那么应定义新的建模规则。

根据底层系统使用的模型,服务器可能需要定义新的建模规则。因为 OPC UA 服务器可能只需要将数据传递给底层系统,然后该系统可以使用其自身的用于实例、子类型等的内部规则。

除此之外,预先定义的建模规则可能不足以规定实例和子类型必须的行为。

附 录 B
（资料性附录）
UML 表示的 OPC UA 元模型

B.1 背景

使用 UML 类和固定格式为<<TypeExtension>>的 UML 对象来表示 OPC UA 元模型(OPC UA 地址空间模型)。固定格式的 UML 对象表示数据类型或引用类型。域模型包含用户定义的引用类型和数据类型,也表示为<<TypeExtension>>。另外,域模型包含表示为 UML 对象的对象类型、变量类型等(见图 B.1)。

OPC 基金会不只规定了 OPC UA 元模型,而且定义了按照 IEC 62541-5 组织地址空间并为服务器提供信息的某些节点。

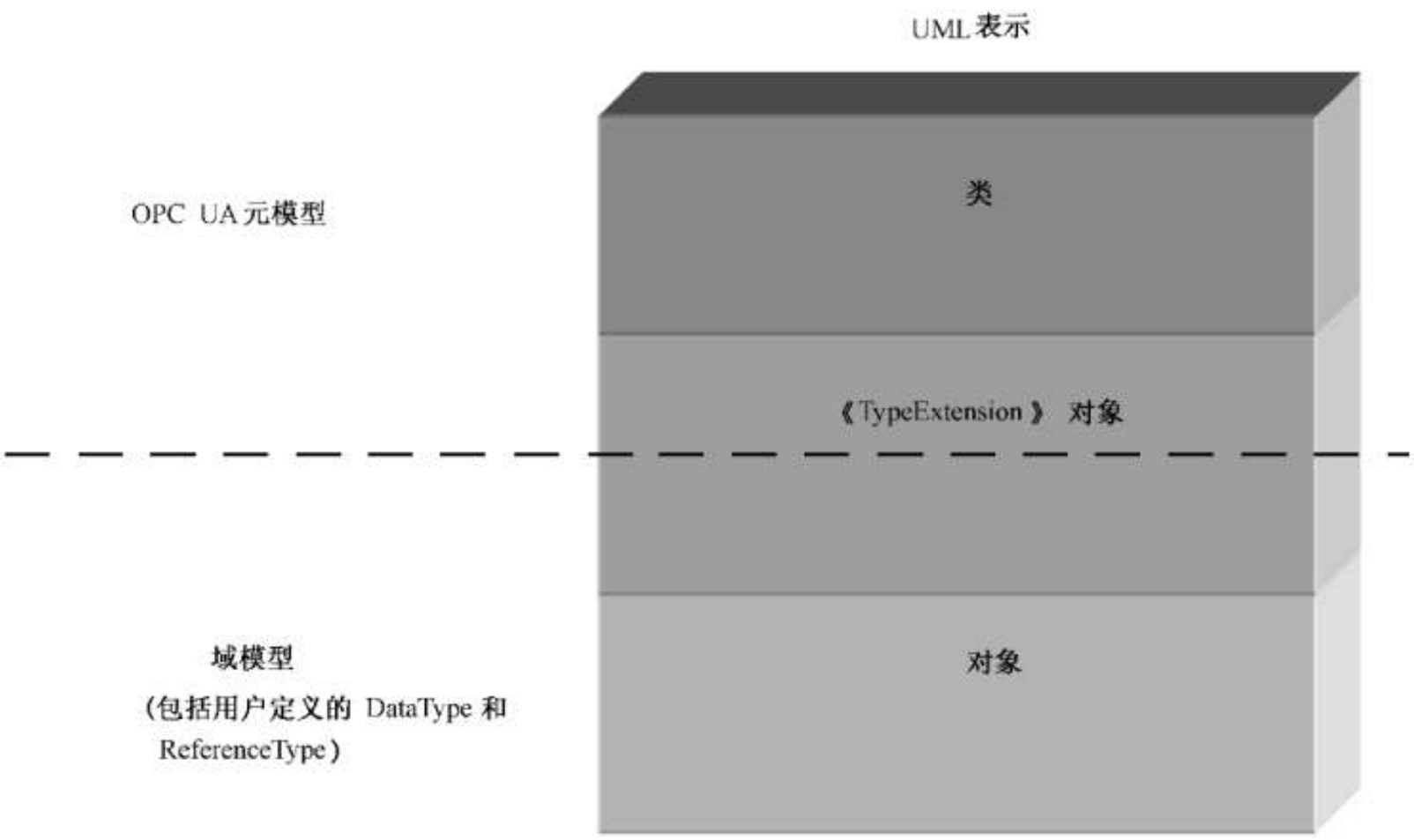


图 B.1 OPC UA 元模型的背景

B.2 标记

在图 B.2 的 UML 类图中给出表示 OPC UA 概念“基(Base)”的 UML 类的例子。OPC 的属性继承自抽象类属性,且具有标识其数据类型的值。他们组合用于一个节点,可选(0..1)或者必须(1),类似图 B.2 中 BrowseName 用于基一样。

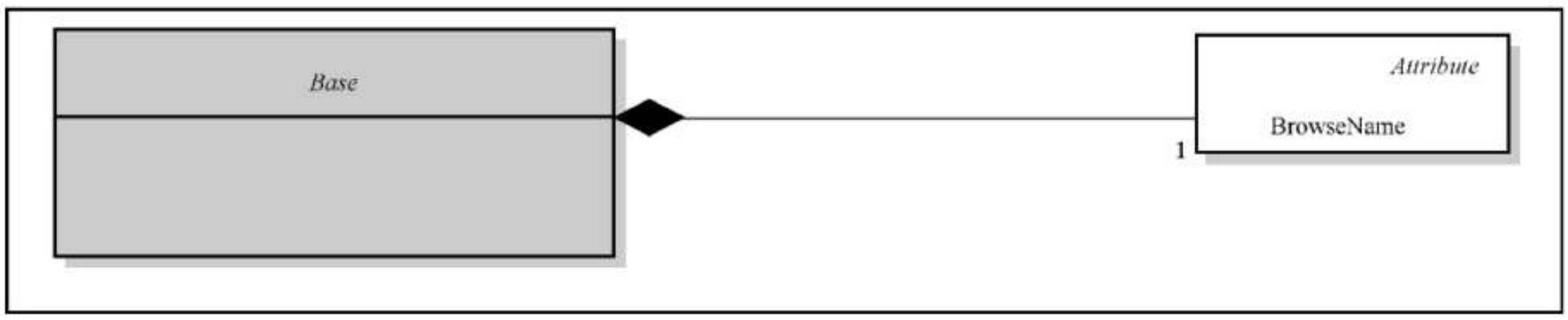


图 B.2 标记(Ⅰ)

UML 对象图用于显示<<TypeExtension>>对象(如图 B.3 中的 HasComponent)。在对象图中,OPC 属性表示为不带数据类型的 UML 属性,标记为固定格式<<Attribute>>,类似 UML 对象 HasComponent 中的 InverseName。他们有值,类似对于 HasComponnet 的 InverseName=ComponentOf。为了保持对象图的简洁,不需要表示出所有的属性(如 HasComponent 的 NodeId)。

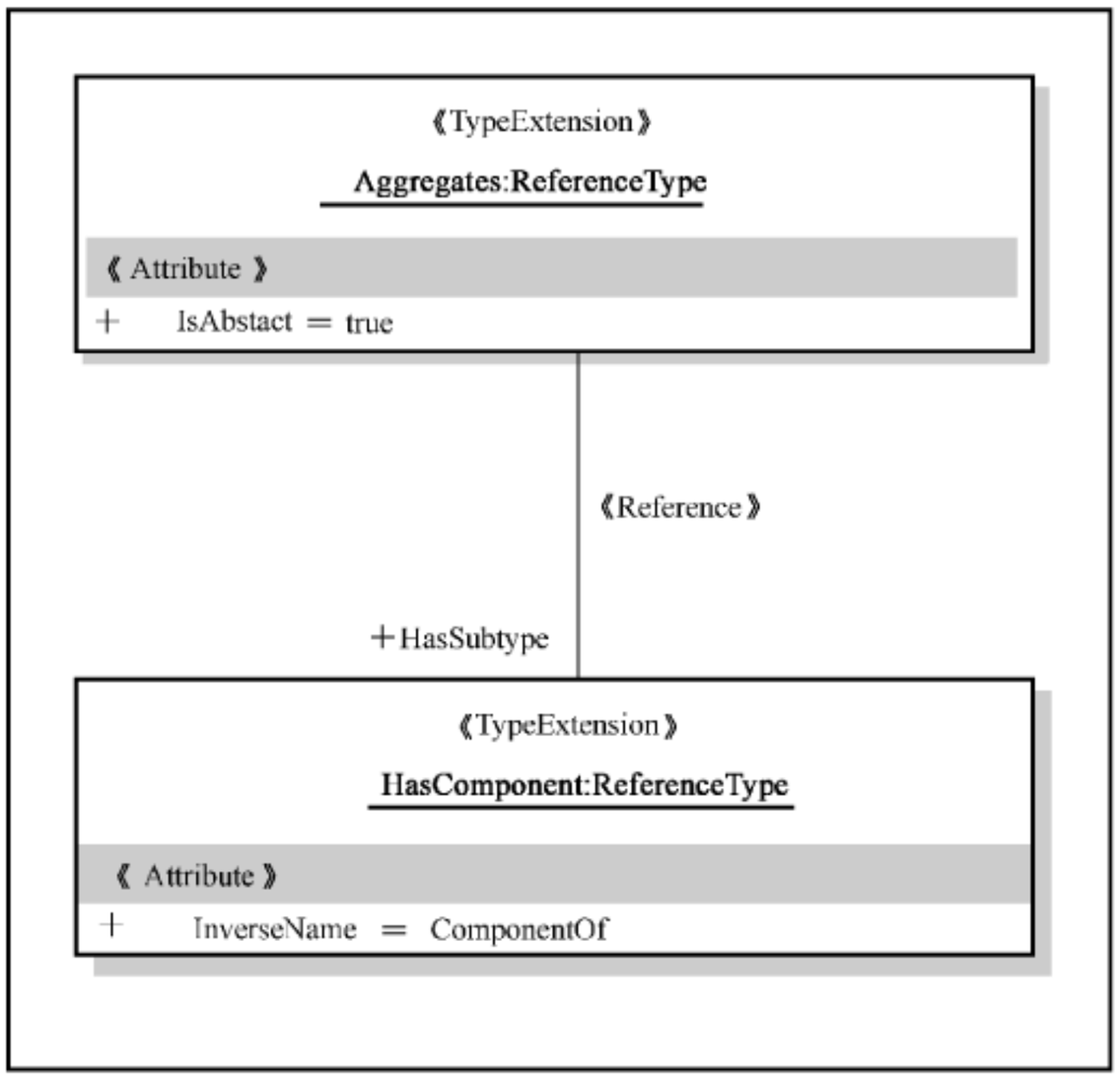


图 B.3 标记(Ⅱ)

OPC 引用表示为 UML 联系,标记为固定格式<<Reference>>。如果使用特殊的引用类型,其名称将被用作角色名称,以标识引用的方向(如 Aggregate 有子类型 HasComponent)。为了简化,反向的角色名称未被列出(如本例中的 SubclassOf)。当没有提供角色名称时,意味着可以使用任何的引用类型(只对类图有效)。

在 OPC UA 中有一些包含 NodeId 的特殊属性,因此可以引用另一个节点。这些属性被表示为关联,固定格式为<<Attribute>>。属性的名称显示为目标节点的角色名称。

使用 UML 对象名称来表示 OPC 属性的 BrowseName 的值,如图 B.3 中 UML 对象 HasComponent 的 BrowseName 为“HasComponent”。

为了强调类图中解释的类,用灰色标记他们(如图 B.2 中的基)。只有这些类具有和图中其他类及属性的所有关系。对于其他类,只需提供用于理解图中主要类的那些属性和关系。

B.3 元模型

本标准的其他部分可以通过增加属性和定义新的引用类型来扩展 OPC UA 元模型。

B.3.1 基

基节点见图 B.4。

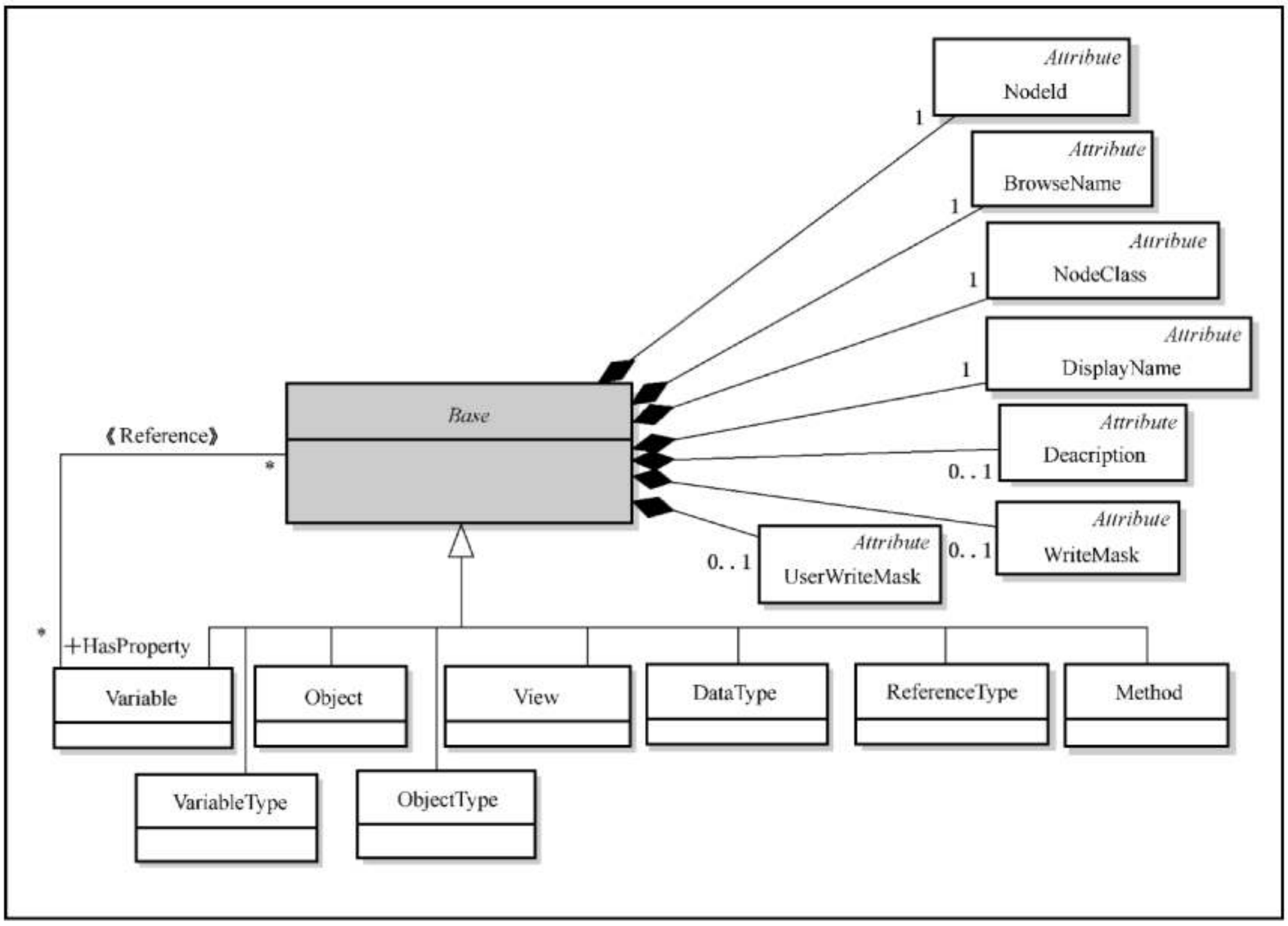


图 B.4　基节点

B.3.2　引用类型

引用和引用类型见图 B.5。

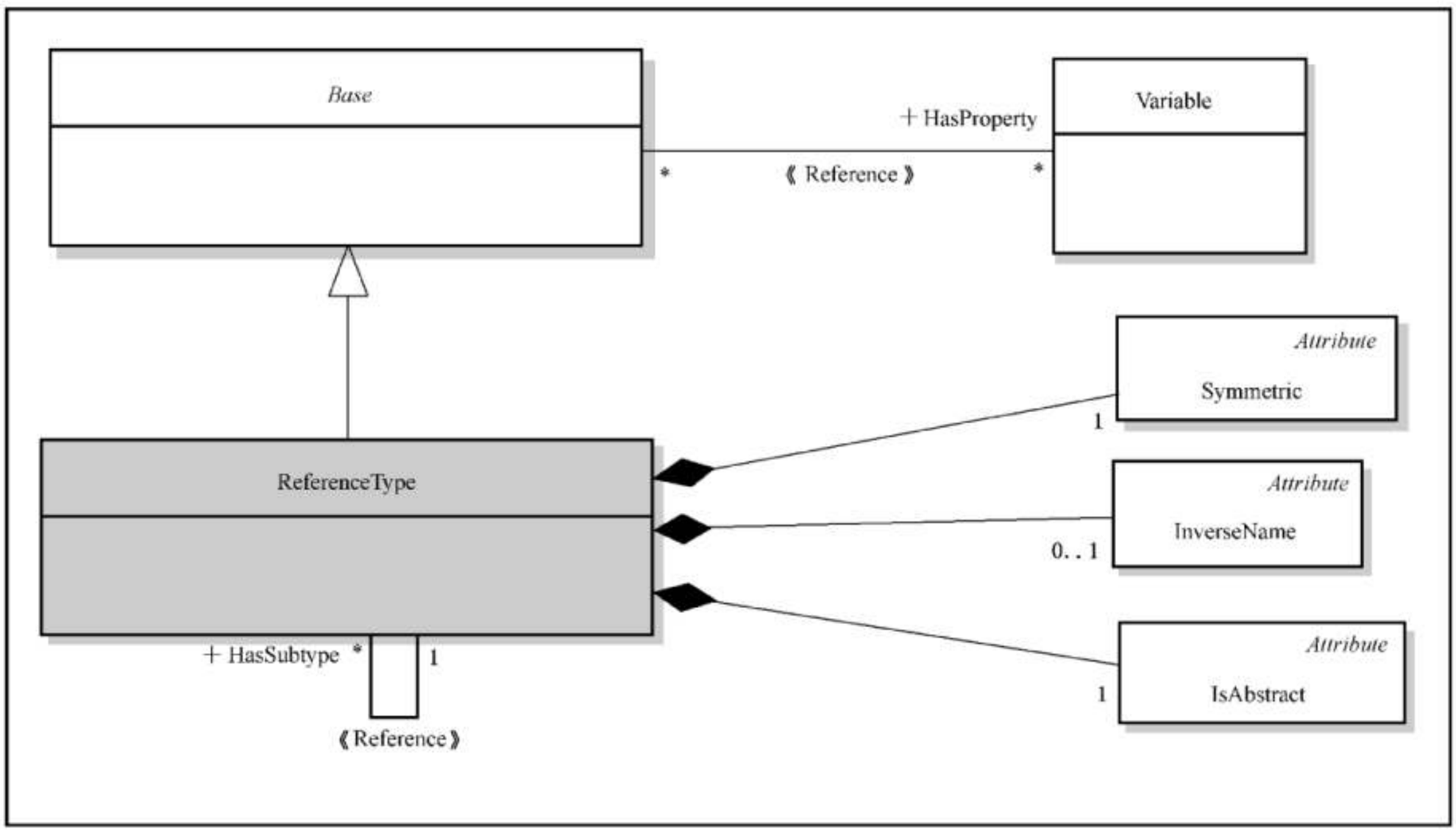


图 B.5　引用和引用类型

如果 Symmetric 为“false”且 IsAbstract 为“false”，那么就应提供 InverseName。

B.3.3 预先定义的引用类型

预先定义的引用类型见 B.6。

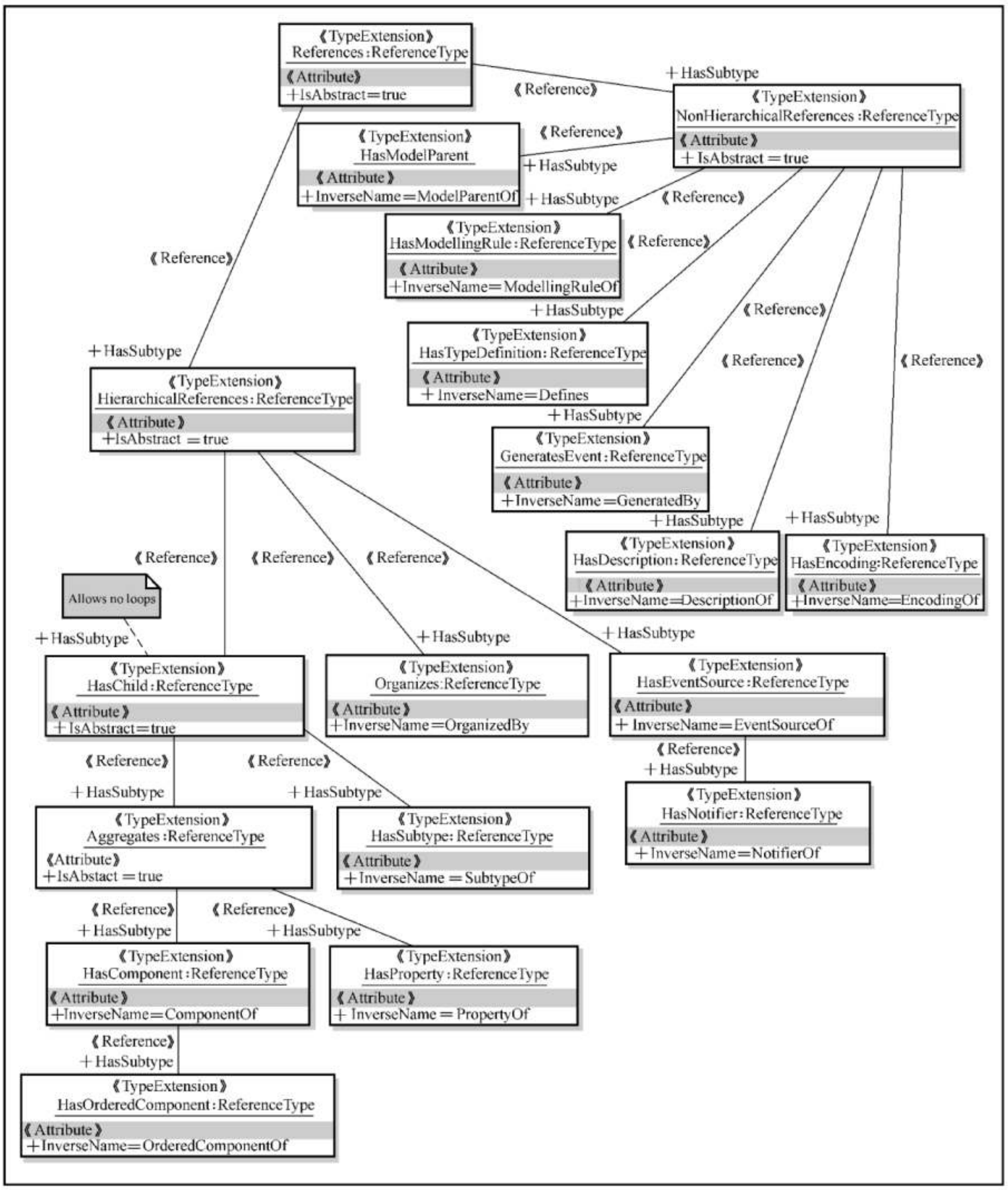


图 B.6 预先定义的引用类型

B.3.4 属性

属性见图 B.7。

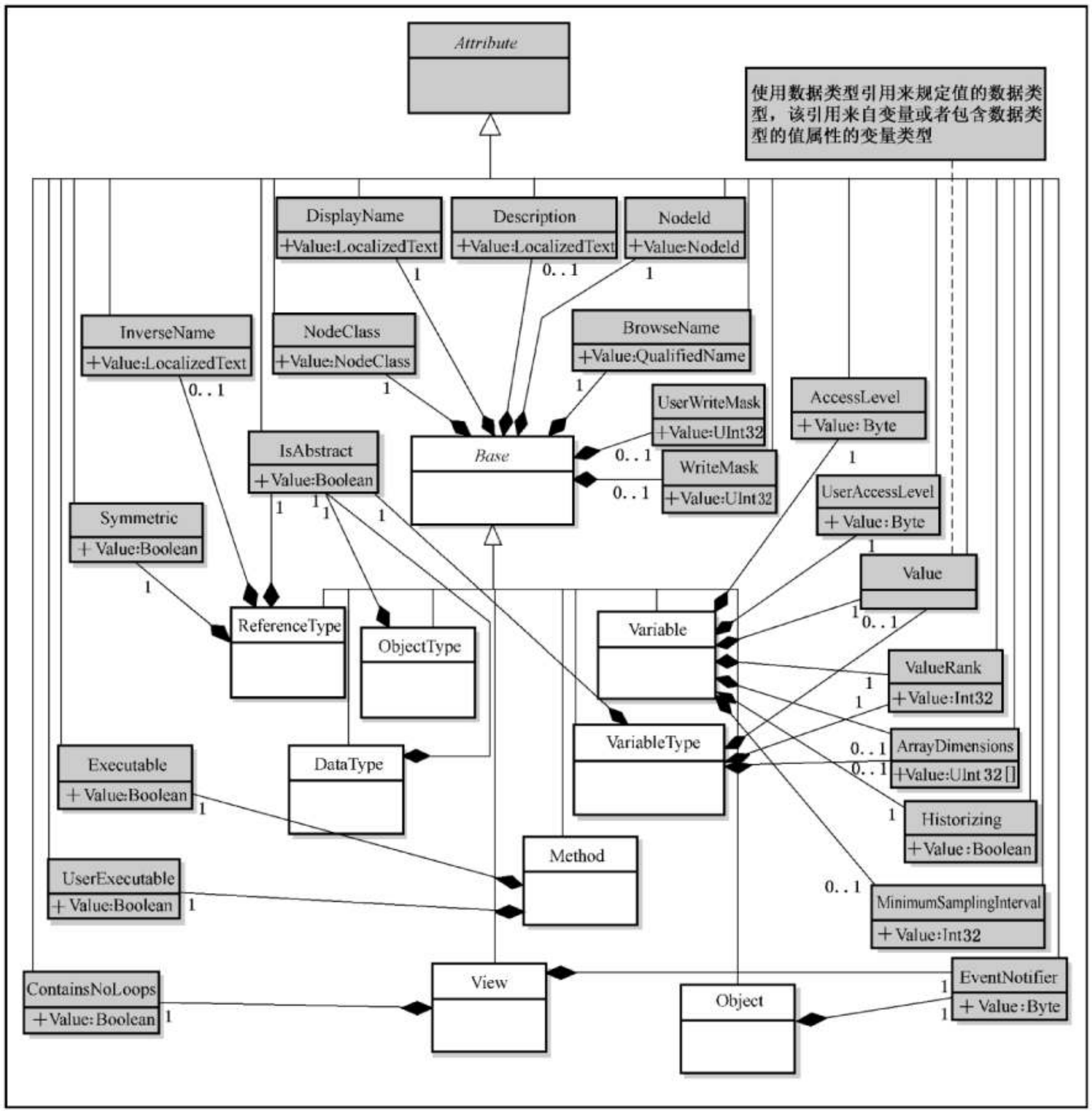


图 B.7 属性

本标准的其他部分中可能定义更多的属性。

用于引用的属性具有 NodeId 作为其数据类型，在本图中没有显示，而在其他图中表示为固定格式的关联。

B.3.5 对象和对象类型

对象和对象类型见图 B.8。

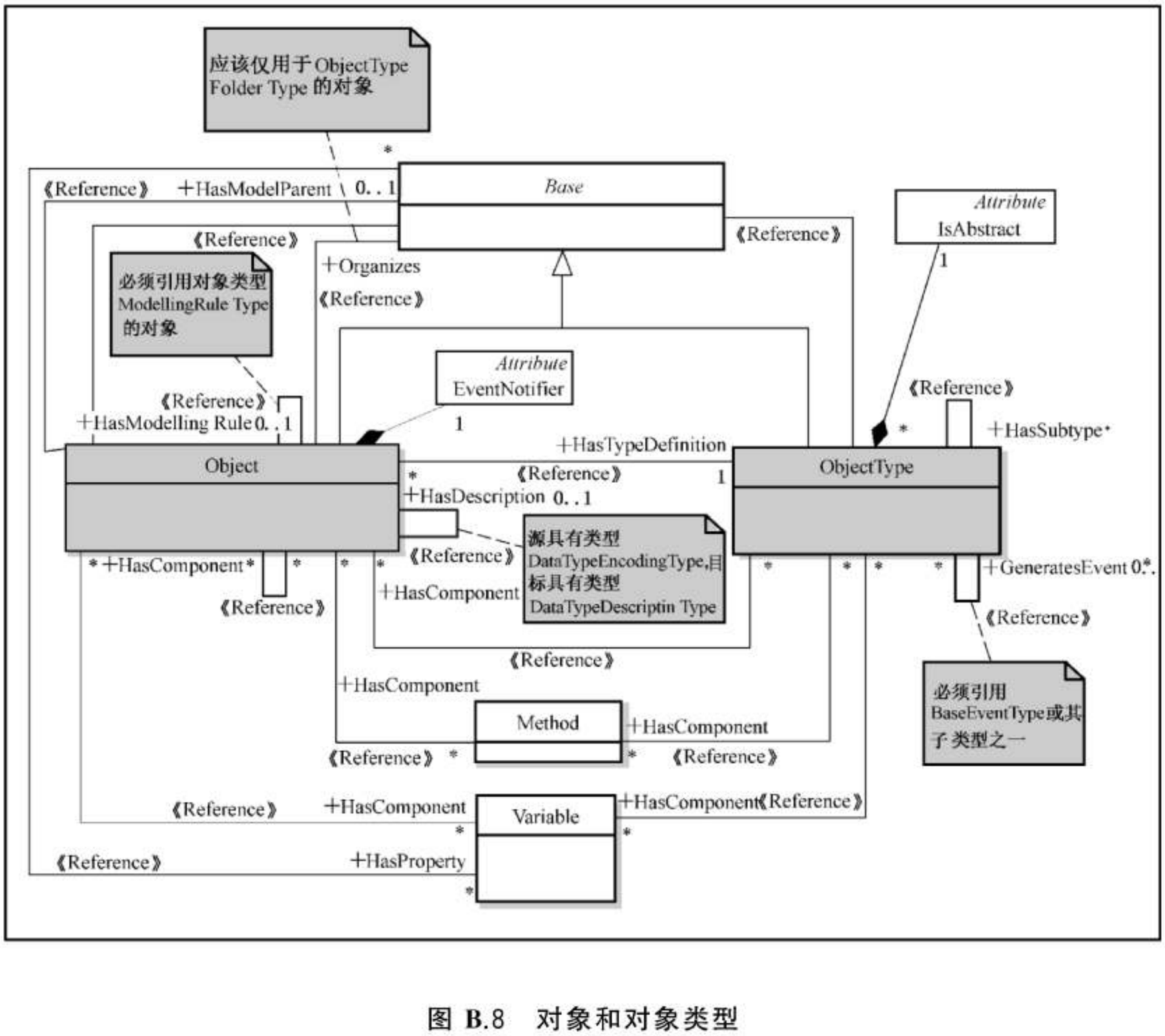


图 B.8　对象和对象类型

B.3.6　EventNotifier

EventNotifier 见图 B.9。

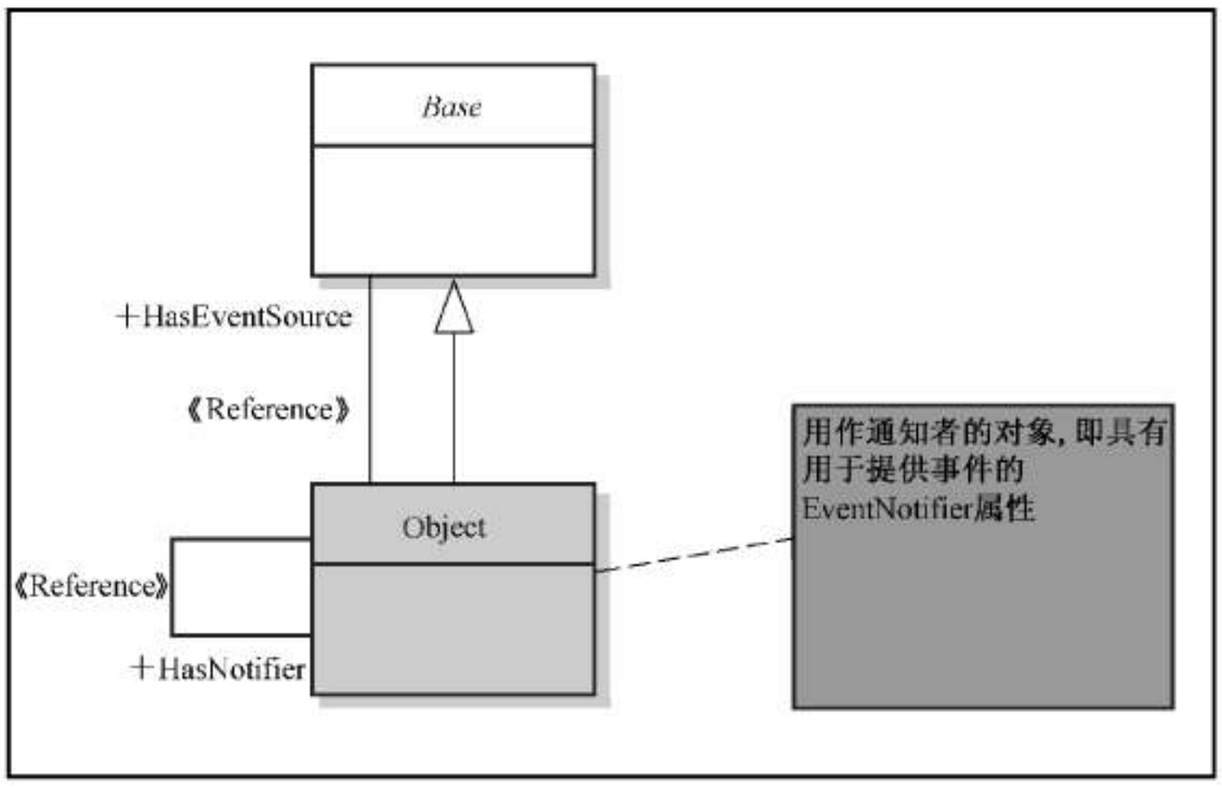


图 B.9　EventNotifier

B.3.7 变量和变量类型

变量和变量类型见图 B.10。

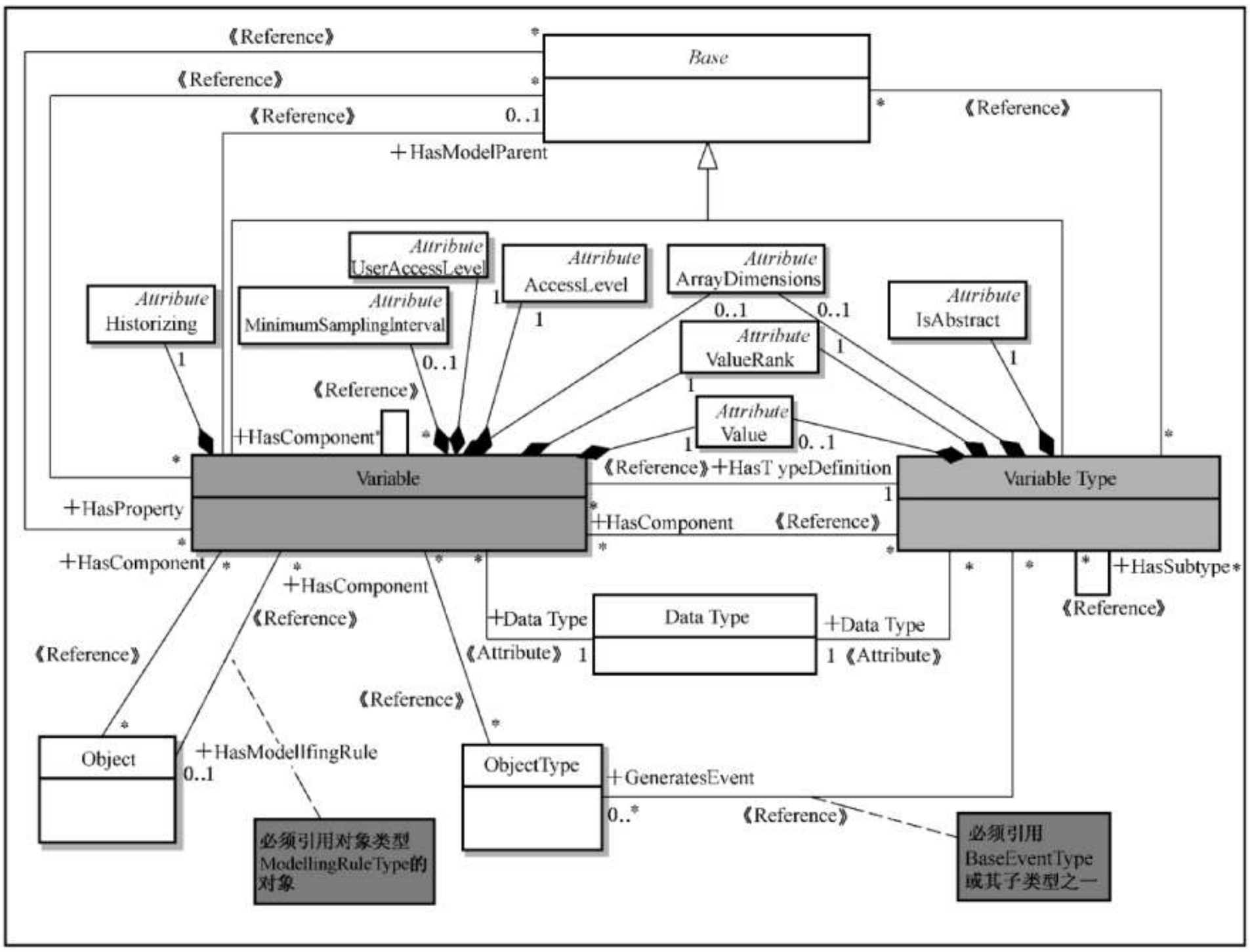


图 B.10 变量和变量类型

变量的数据类型应是一致的，或者应是其变量类型（使用 HasTypeDefinition 引用）的数据类型的一个子类型。

如果 HasProperty 指向来自基“A”的变量，那么下面的限制适用：

变量不应是 HasProperty 的源节点，或者任何其他 HierarchicalReference 的引用。

具有“A”作为 HasProperty 引用的源节点的所有变量都应在“A”的上下文中有唯一的 BrowseName。

B.3.8 方法

方法见图 B.11。

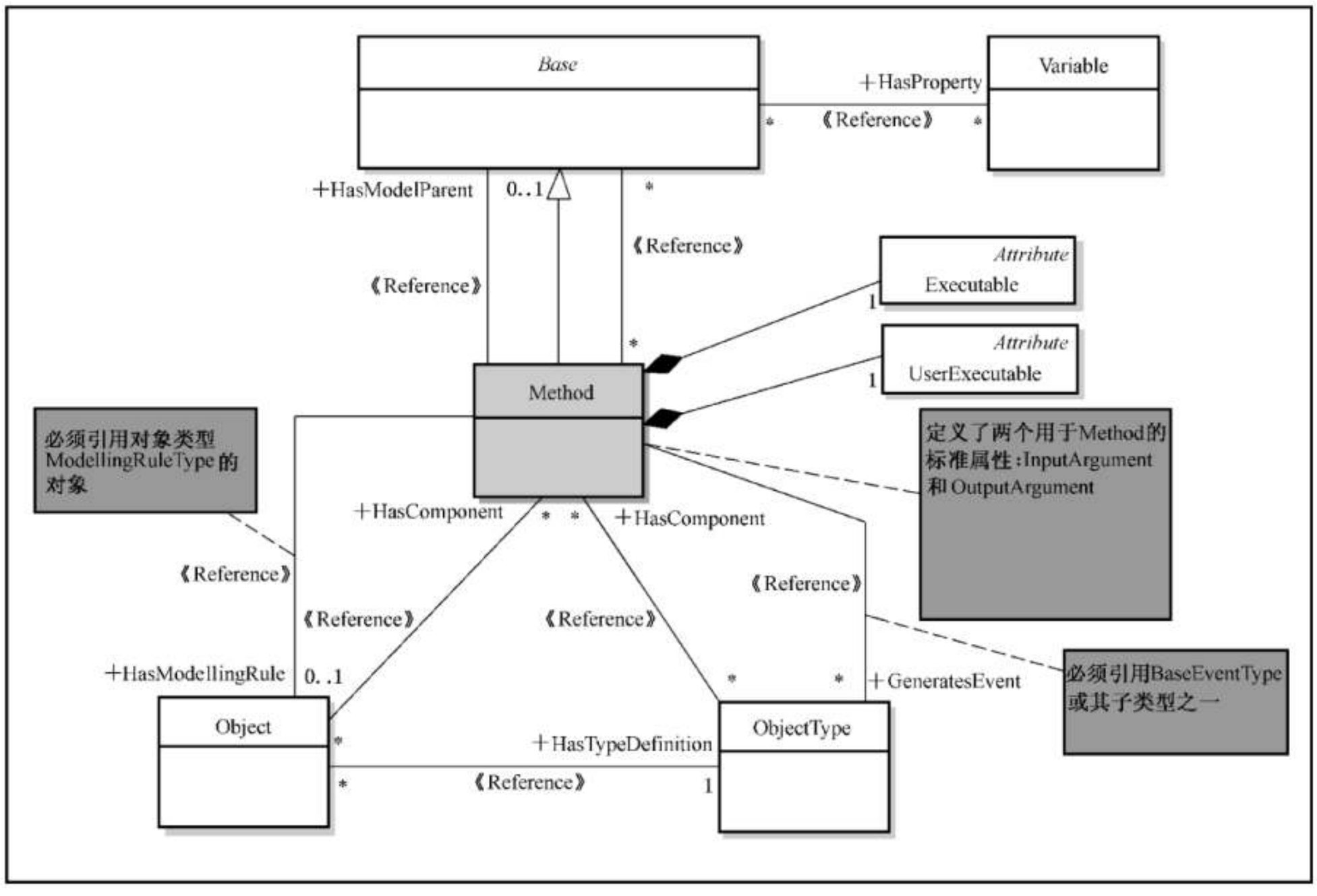


图 B.11 Method

B.3.9 数据类型

数据类型见图 B.12。

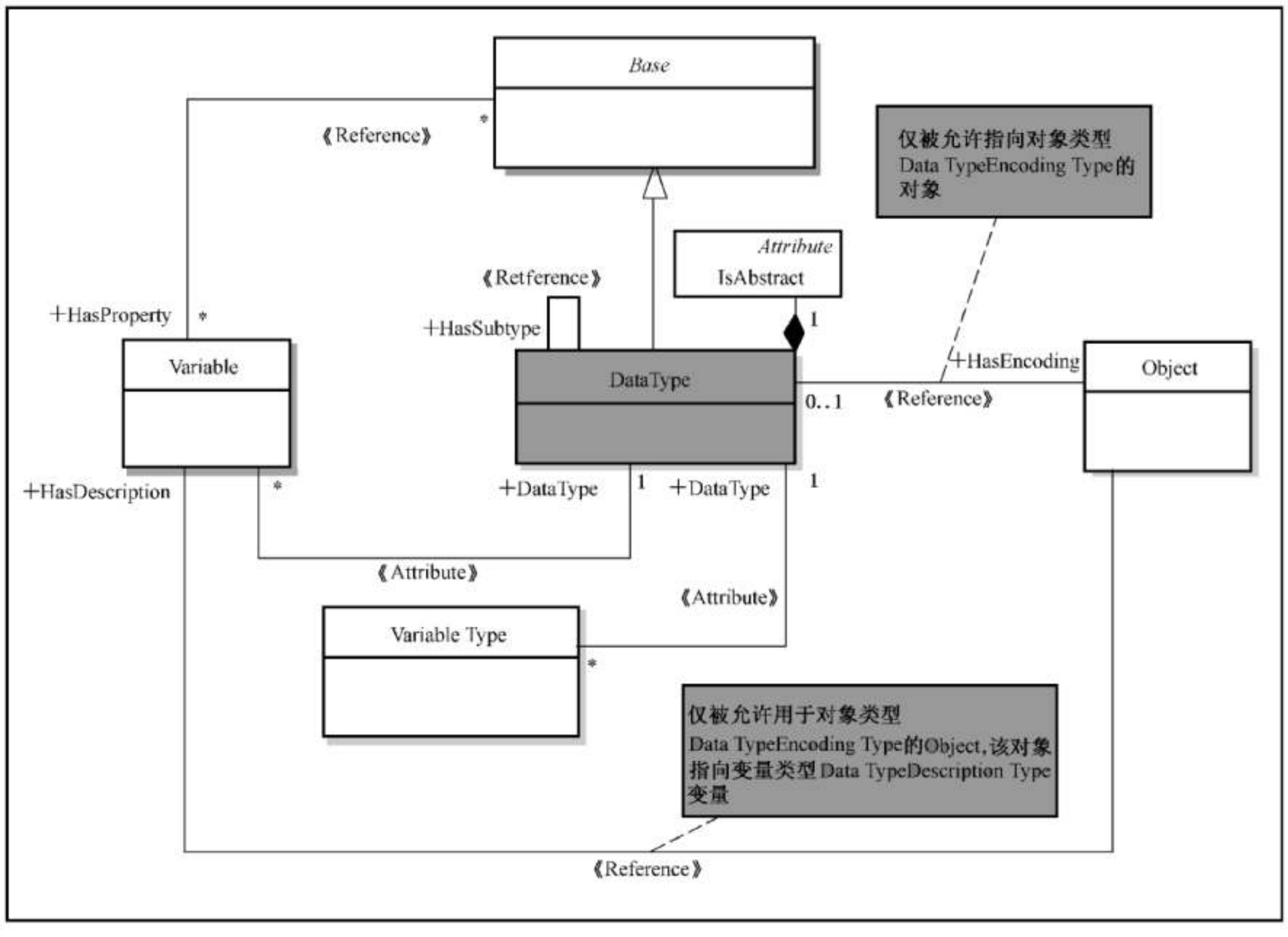


图 B.12　数据类型

B.3.10　视图

视图见图 B.13。

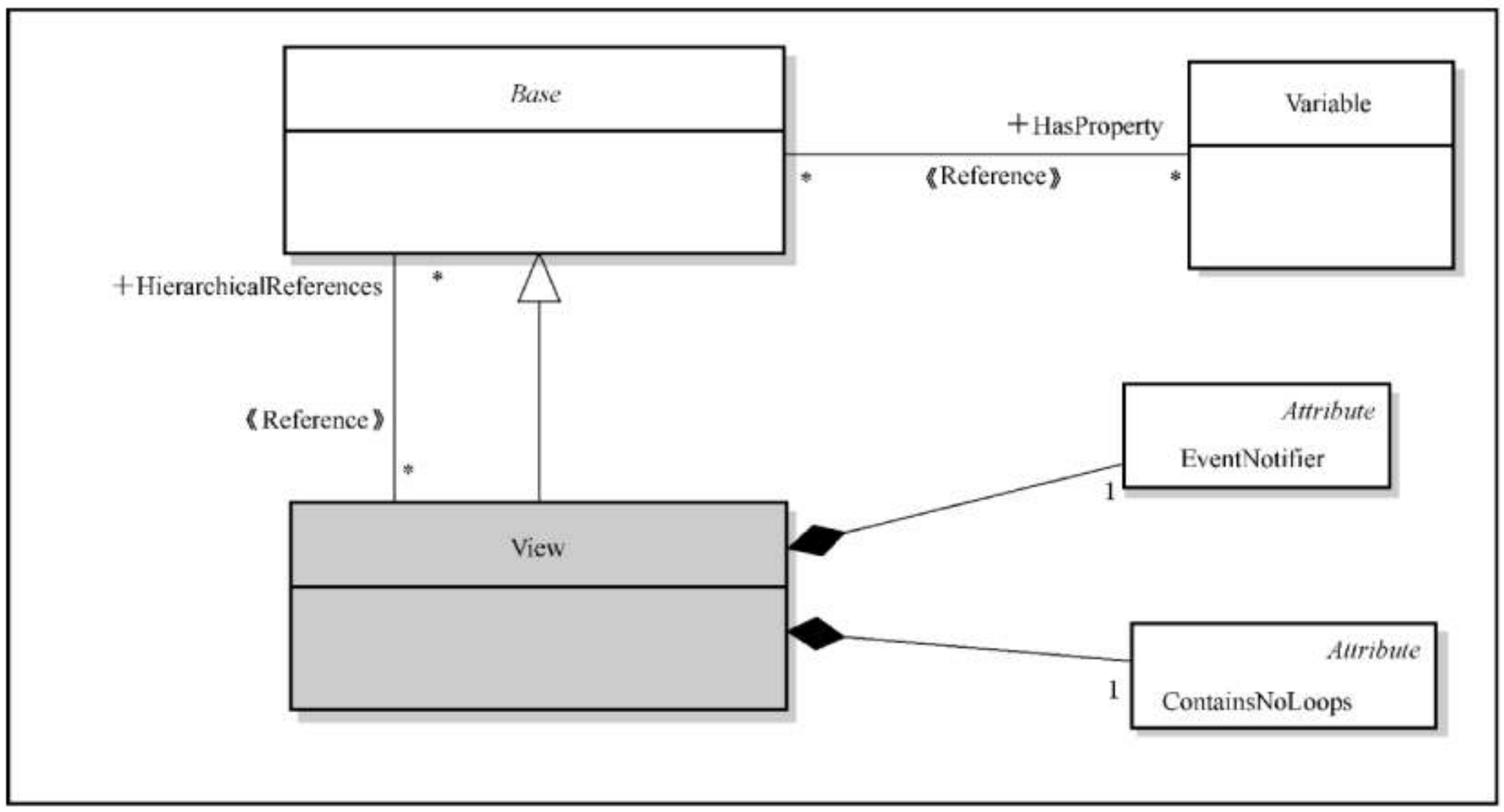


图 B.13　视图

附 录 C
（规范性附录）
OPC 二进制类型描述系统

C.1 概念

OPC 二进制 XML 结构定义了 OPC 二进制类型字典的格式。每个 OPC 二进制类型字典是一个 XML 文档，该文档包含一个或多个用于描述二进制编码值的格式的类型描述。事先不了解特定二进制编码的应用程序能够使用 OPC 二进制类型描述来解释或建立一个值。

OPC 二进制类型描述系统并不定义以二进制编码数据的标准机制。它只提供描述现有二进制编码的标准方法。许多二进制编码都有描述可被编码类型的机制；然而，这些描述只对那些了解每个二进制编码已使用的类型描述系统的应用程序有用。OPC 二进制类型描述系统是一个通用的语法，可被任何应用程序用于解释任何二进制编码。

OPC 二进制类型描述系统最初在 OPC 复杂数据规范中定义。本附录中描述的 OPC 二进制类型描述系统是不同的，它是 OPC 二进制描述系统的第 2.0 版。

每个类型描述由一个类型名称标识，该类型名称在定义它的类型字典中应是惟一的。每个类型字典也有一个目标命名空间，该目标命名空间在所有 OPC 二进制类型字典中也应是惟一的。以字典的目标命名空间限定的类型名称应是用于类型描述的全局惟一标识符。

图 C.1 解释了 OPC 二进制类型字典的结构。

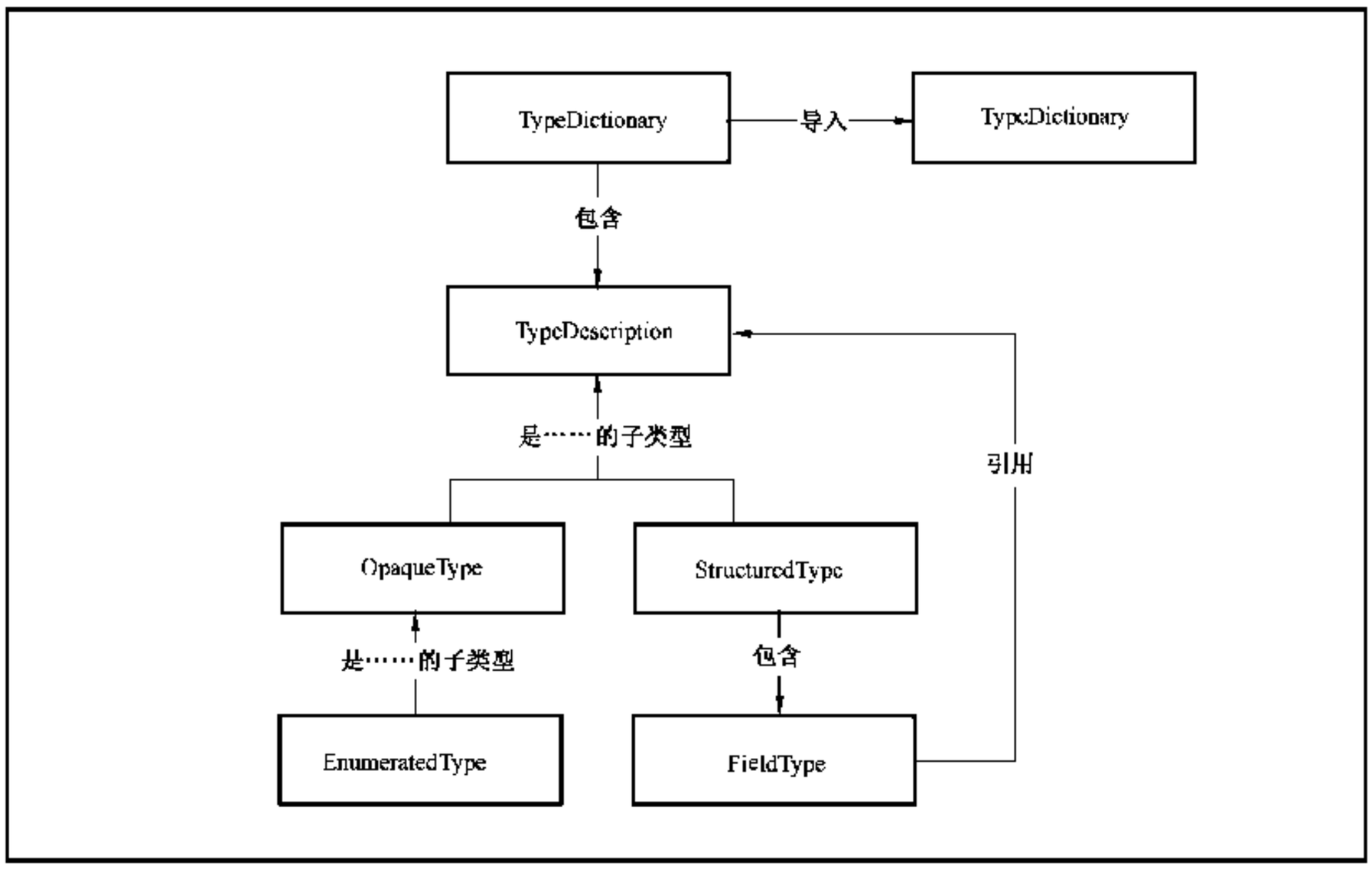


图 C.1 OPC 二进制字典结构

每个二进制编码都由一系列难懂的构建块组成，他们或者是具有固定长度的原始类型，或者是具有太复杂以致于无法以 XML 文档正确描述的结构的可变长度类型。使用 OpaqueType 来描述这些构建

块。这些构建块之一的一个实例是二进制编码值。

OPC 二进制类型描述系统定义了一系列标准的 OpaqueType，所有 OPC 二进制类型字典都应使用他们来构建自己的类型描述。这些标准类型描述在 C.3 中定义。

在某些情况下，由 OpaqueType 描述的二进制编码可能有固定的长度，这允许一个应用程序可以忽略它无法理解的编码值。这种情况下，OpaqueType 的 LenthInBit 属性应被规定。如果类型字典的作者需要定义没有固定长度的新 OpaqueType，他们应该将该元素文档化，以描述如何编码该类型的二进制值。该描述应提供足够的细节，以允许他人编写能解释该类型实例的程序。

StructuredType 将复杂值分解为一系列由字段类型描述的值。每个字段类型有一个名称、类型和大量限定符，这些限定符规定该字段何时使用，以及存在该类型的实例的数量。字段类型在 C.2.6 中详细描述。

EnumeratedType 描述了一个数值，它包括有限的一系列可能值，其中每个都有一个描述性的名称。枚举类型提供一种获得与那些难懂的数值相关的语义信息的便利方法。

C.2 结构描述

C.2.1 类型字典

类型字典元素是 OPC 二进制字典的根元素。该元素的组成见表 C.1。

表 C.1 类型字典组成

名称	类型	描述
Documentation	Documentation	包含可读文本和 XML 的元素，它提供字典中所包含内容的概述
Import	ImportDirective[]	0 或更多元素，它规定字典中定义的结构类型所引用的其他类型字典。每个输入元素规定正被输入的类型字典的 NamespeaceURI。类型字典元素应声明每个导入命名空间的 XML 命名空间前缀
TargetNamespace	xs:string	规定用于限定字典中定义的所有类型描述的 URI
DefaultByteOrder	ByteOrder	为所有 ByteOrderSignificant 属性设为“true”的类型描述规定缺省的字节顺序。 在任何导入类型字典中，该值将覆盖设置值。 该值被类型描述中规定的 DefaultByteOrder 所覆盖
TypeDescription	TypeDescription []	一个或更多元素，它描述二进制编码值的结构。 类型描述是一个抽象类型。一个字典可以只包含 OpaqueType、EnumeratedType 和 StructuredType 元素

C.2.2 类型描述

类型描述是对二进制编码值结构的描述。类型描述是一个抽象的基础类型，只有子类型的实例可以出现在类型字典中。类型描述的组成在表 C.2 中描述。

表 C.2 类型描述组成

名称	类型	描述
Documentation	Documentation	包含描述类型的可读文本和 XML 的元素。该元素收集能帮助理解该值包含何种信息的任何语义信息
Name	xs: NCName	规定类型描述的名称的属性,它在字典中是惟一的。结构类型的字段通过字典命名空间 URI 限定的该名称引用类型描述
DefaultByteOrder	ByteOrder	规定类型描述缺省字节顺序的属性。该值覆盖引用该类型描述的任何类型字典或者任何 StructruedType 中的设置

C.2.3 OpaqueType

OpaqueType 描述一个二进制编码值,他们或者是具有固定长度的原始类型,或者是具有太复杂以致于无法在 OPC 二进制类型字典中获得的结构。类型字典的作者应避免定义没有固定长度的 OpaqueType,因为这将使得使用这些类型但又没有内置 OpaqueType 知识的应用程序无法解释该值。OPC 二进制类型描述系统定义了许多标准的 OpaqueType,这也允许作者将大多数二进制编码值描述为 StructuredType。

OpaqueTypeDescription 的组成见表 C.3。

表 C.3 OpaqueType 组成

名称	类型	描述
TypeDescription	TypeDescription	OpaqueType 继承表 C.2 中为类型描述定义的所有元素和属性
LengthInBits	xs:string	以位为单位规定 OpaqueType 长度的属性。应该规定该值。如果未规定该值,那么文档元素应以可理解的方式描述编码
ByteOrderSignificant	xs:boolean	指示对于该类型字节顺序是否重要的属性。如果字节顺序是重要的,那么应用程序在解释编码值前应确定当前上下文所使用的字节顺序。 应用程序通过寻找为内置的 StructuredType 或者类型字典规定的 DefaultByteOrder 属性来确定字节顺序。如果内部规定了 StructuredType,那么内部的 StructuredType 覆盖外部描述的字节顺序。 如果为 OpaqueType 规定了 DefaultByteOrder 属性,那么字节顺序是固定的,而且不会随上下文改变。 如果该属性为"true",那么应规定 LengthInBit 属性,而且它应是 8 位的整数倍

C.2.4 枚举类型

枚举类型描述了二进制数值,它具有固定的一系列有效值。EnumberatedType 描述的二进制编码值总是一个无符号整数,其长度由 LengthInBit 属性规定。

每个枚举值的名称不要求解释二进制编码,但是它们组成类型文档的一部分。

枚举类型的组成见表 C.4。

表 C.4 枚举类型组成

名称	类型	描述
OpaqueType	OpaqueTypeDescription	枚举类型继承表 C.2 中为类型描述和表 C.3 中为 OpaqueType 定义的所有元素和属性。 应总是规定 LengthInBits 属性
EnumeratedValue	EnumeratedValue	描述类型实例的可能值的一个或多个元素

C.2.5 StructuredType

StructuredType 将类型描述为一序列二进制编码值。序列中的每个值称为一个字段。每个字段引用一个描述该字段中出现的二进制编码值的类型描述。一个字段可以规定 0 个、1 个或更多个由 StructuredType 描述的序列中出现的类型的实例。

类型字典的作者应该使用 StruturedType 来描述各种通用数据结构,包括 array(数组)、union(共同体)和 structure(结构)。

某些字段的长度可能不是 8 位的整数倍。上述的一些字段可能出现在一个结构中的一个序列中,但是整个序列所使用的所有位数应是固定的且为 8 位的整数倍。没有固定长度的任何字段都应按字节边界对齐。

还未按字节边界对齐的一序列字段按规定从最低有效位到最高有效位排列。超过一个字节的序列从第一个字节的最高有效位溢出,并进入下一个字节的最低有效位。

StructruedType 的组成见表 C.5。

表 C.5 StructruedType 组成

名称	类型	描述
TypeDescription	TypeDescription	StructuredType 继承表 C.2 中为类型描述定义的所有元素和属性
Field	FieldType	描述该结构字段的一个或多个元素。每个字段应有一个在 StruturedType 中唯一的名称。某些字段可以通过使用该名称来引用 StructruedType 中的其他字段
anyAttribute	任何类型	类型字典的作者可以给任何 StructruedType 增加他们自身的属性,该属性应在该作者定义的命名空间中限定。不应要求应用程序为了解释该类型的二进制编码实例而理解这些属性

C.2.6 FieldType

FieldType 描述 StructuredType 的序列中出现的二进制编码值。每个 FieldType 应引用一个描述该字段编码值的类型描述。

一个 FieldType 可以规定一组编码值。

字段可以是可选的，它们引用其他 FieldType，这指出了它们是否在该类型的任何特定实例中出现。

FieldType 的组成见表 C.6。

表 C.6 FieldType 组成

名称	类型	描述
Documentation	Documentation	包含描述该字段的可读文本和 XML 的元素。该元素应收集能帮助理解该字段所包含内容的任何语义信息
Name	xs:string	规定该字段的名称的属性，其在 StructuredType 中是唯一的。 结构化类型的其他字段通过使用该名称来引用一个字段
TypeName	xs:QName	规定描述该字段内容的类型描述的属性。一个字段可能包含 0 个或多个该类型的实例，这取决于其他属性的设置和其他字段中的值
Length	xs:unsignedInt	指示字段长度的属性。该值可以是编码字节的全部个数，或者也可以是该字段所引用类型实例的个数。IsLengthInByte 属性规定适用哪个定义
LengthField	xs:string	指出 StructuredType 中的哪个其他字段规定了字段的长度。字段的长度可以是字节，也可以是字段所引用的类型实例的个数。IsLengthInByte 属性规定适用哪个定义。 如果该属性指向一个在编码值中不存在的字段，那么长度的缺省值为 1。这种情况发生在所引用的字段是一个可选字段的时候(见 SwitchField 属性)。 长度字段应是一个固定长度的二进制整数。如果长度字段是标准有符号整数类型之一且值为负整数，那么该字段在编码串中不存在。 该属性引用的字段类型应在 StructruedType 字段之前
IsLengthInBytes	xs:boolean	该属性指出 Length 或 LengthField 属性是以字节规定字段长度，还是以该字段所引用类型实例的个数规定字段长度
SwitchField	xs:string	如果规定了该属性，那么该字段是可选的，不会在每个编码值实例中都出现。 该属性规定了控制本字段是否出现在编码值中的另一个字段的名称。该属性引用的字段应是一个整数值(见 LengthField 属性)。 使用 SwitchOperand 对 SwitchField 的当前值和 SwitchValue 属性进行比较。如果结果为"true"，那么字段在字符串中存在。 如果没有规定 SwitchValue 属性且 SwitchField 的值非 0，那么存在该字段。如果字段存在，则忽略 SwitchOperand 字段。 如果 SwitchOperand 属性缺失，且 SwitchField 的值等于 SwitchValue 属性的值，那么字段存在。 该属性引用的字段应在 StructuredType 字段之前

表 C.6（续）

名称	类型	描述
SwitchValue	xs:unsignedInt	该属性规定字段何时在编码值中存在。使用 SwitchOperand 属性将 Switchname 属性引用的字段的值与该值进行比较。如果表达式等于“true”，那么字段存在。否则字段不存在
SwitchOperand	xs:string	该属性规定 SwitchField 的值如何与 SwitchValue 属性进行比较。该字段是一个枚举类型，包括下列值： Equal SwitchField 等于 SwitchValue； GreaterThan SwitchField 大于 SwitchValue； LessThan SwitchField 小于 SwitchValue； GreaterThanOrEqual SwitchField 大于或等于 SwitchValue； LessThanOrEqual SwitchField 小于或等于 SwitchValue； NotEqual SwitchField 不等于 SwitchValue。 不论何种情况，只要表达式为“true”，那么字段存在
Terminator	xs:hexBinary	该属性指出字段包含一个或多个被该字段引用的类型描述实例，而且最后一个值具有该属性的值规定的二进制编码。如果规定了该属性，那么字段引用的类型描述应该或者是有固定的字节顺序（即字节顺序不重要或被明确规定），或者内嵌的 StructruedType 应明确规定字节顺序。 字段数据类型 Terminator 字节顺序 十六进制字符串 Char tab character 不适用 09 WideChar: tab character 高位优先 0009 WideChar: tab character 低位优先 0900 Int16 1 高位优先 0001 Int16 1 低位优先 0100
anyAttribute	任何类型	类型字典的作者可以给任何字段类型增加他们自身的属性，该属性应在该作者定义的命名空间中限定。不应要求应用程序为了解释二进制编码字段值而理解这些属性

C.2.7 EnumeratedValue

EnumeratedValue 描述枚举类型的可能值。

EnumeratedValue 的组成见表 C.7。

表 C.7 EnumeratedValue 组成

名称	类型	描述
Name	xs:string	该属性规定 EnumeratedValue 的描述性名称
Value	xs:unsignedInt	该属性规定二进制编码中可能出现的数值

C.2.8 ByteOrder

ByteOrder 是一个枚举类型，它描述用于类型描述的可能值的字节顺序，这允许使用不同的字节顺序。有两种可能值：高位优先和低位优先。高位优先指最高有效位在二进制编码中先出现。低位优先指最低有效位在二进制编码中先出现。

C.2.9 ImportDirective

ImportDirective 规定了当前字典中定义的字段描述所引用的类型字典。

ImportDirective 的组成见表 C.8。

表 C.8 ImportDirective 组成

名称	类型	描　　述
Namespace	xs:string	该属性规定导入的类型字典的目标命名空间。这可以是一个已知的 URI，它意味着应用程序不必为了识别被引用类型而访问物理文件
Location	xs:string	该属性规定包含导入的类型字典的 XML 文件的物理位置。该值可以是网络资源的 URL、OPC UA 服务器地址空间中的 NodeId 或者本地文件路径

C.3 标准类型描述

OPC 二进制类型描述系统定义了大量标准的类型描述，它们使用 StruturedType 来描述许多通用的二进制编码。标准的类型描述见表 C.9。

表 C.9 标准的类型描述

类型名称	描　　述
Bit	单个位值
Boolean	表示为 8 位值的两状态逻辑值
SByte	8 位有符号整数
Byte	8 位无符号整数
Int16	16 位有符号整数
UInt16	16 位无符号整数
Int32	32 位有符号整数
UInt32	32 位无符号整数
Int64	64 位有符号整数
UInt64	64 位无符号整数
Foat	IEEE 754 1985 单精度浮点值
Double	IEEE 754 1985 双精度浮点值
Char	8 位 UTF-8 字符值
WideChar	16 位 UTF-16 字符值
String	以空字符结尾的一系列 UTF-8 字符

表 C.9（续）

类型名称	描　　述
CharArray	以字符个数开始的一系列 UTF-8 字符
WideString	以空字符结尾的一系列 UTF-16 字符
WideCharArray	以字符个数开始的一系列 UTF-16 字符
DateTime	64 位有符号整数，用于表示从 1601-01-01 00:00:00 开始、以 100 ns 为间隔的个数。这与 WIN32 FILETIME 类型一致
ByteString	以字节表示的长度开始的一系列字节
Guid	128 位结构化类型，用于表示 WIN32 GUID 值

C.4 类型描述示例

C.4.1 128 位有符号整数

```
〈opc:OpaqueType Name="Int128" LengthInBits="128"〉
〈opc:Documentation〉A 128-bit signed integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
```

C.4.2 分为几个字段的 16 位值

```
〈opc:StructuredType Name="Quality"〉
〈opc:Documentation〉An OPC COM-DA quality value.〈/opc:Documentation〉
〈opc:Field Name="LimitBits" 类型名称="opc:Bit" Length="2" /〉
〈opc:Field Name="QualityBits" 类型名称="opc:Bit" Length="6"/〉
〈opc:Field Name="VendorBits" 类型名称="opc:Byte" /〉
〈/opc:StructuredType〉
```

当使用位字段时，字节中的最低有效位应首先出现。

C.4.3 带有可选字段的结构化类型

```
〈opc:StructuredType Name="DataValue"〉
〈opc:Documentation〉A value with an associated timestamp, and
quality.〈/opc:Documentation〉
〈opc:Field Name="ValueSpecified" 类型名称="Bit" /〉
〈opc:Field Name="StatusCodeSpecified" 类型名称="Bit" /〉
〈opc:Field Name="TimestampSpecified" 类型名称="Bit" /〉
〈opc:Field Name="Reserved1" 类型名称="Bit" Length="5" /〉
〈opc:Field Name="Value" 类型名称="Variant" SwitchField="ValueSpecified" /〉
〈opc:Field Name="Quality" 类型名称="Quality" SwitchField="StatusCodeSpecified" /〉
〈opc:Field Name="Timestamp" 类型名称="opc:DateTime"
SwitchField="SourceTimestampSpecified" /〉
〈/opc:StructuredType〉
```

应明确规定要求的任何填充位，以确保后续字段按字节边界排列。

C.4.4 整数数组

〈opc:StructuredType Name="IntegerArray"〉
〈opc:Documentation〉An array of integers prefixed by its length.〈/opc:Documentation〉
〈opc:Field Name="Size" 类型名称="opc:Int32" /〉
〈opc:Field Name="Array" 类型名称="opc:Int32" LengthField="Size" /〉
〈/opc:StructuredType〉

如果 Size 字段的值小于或等于 0,那么该数组未编码。

C.4.5 带有终止符而不是长度前缀的整数数组

〈opc:StructuredType Name="IntegerArray" DefaultByteOrder="LittleEndian"〉
〈opc:Documentation〉An array of integers terminated with a known
value.〈/opc:Documentation〉
〈opc:Field Name="Value" 类型名称="opc:Int16" Terminator="FF7F" /〉
〈/opc:StructuredType〉

终止符是 32767,按低位优先字节顺序转化为十六进制。

C.4.6 简单共同体

〈opc:StructuredType Name="Variant"〉
〈opc:Documentation〉A union of several types.〈/opc:Documentation〉
〈opc:Field Name="ArrayLengthSpecified" 类型名称="opc:Bit" Length="1"/〉
〈opc:Field Name="VariantType" 类型名称="opc:Bit" Length="7" /〉
〈opc:Field Name－"ArrayLength" 类型名称－"opc:Int32"
SwitchField="ArrayLengthSpecified" /〉
〈opc:Field Name="Int32" 类型名称="opc:Int32" LengthField="ArrayLength"
SwitchField="VariantType" SwitchValue="1" /〉
〈opc:Field Name="String" 类型名称="opc:String" LengthField="ArrayLength"
SwitchField="VariantType" SwitchValue="2" /〉
〈opc:Field Name="DateTime" 类型名称="opc:DateTime" LengthField="ArrayLength"
SwitchField="VariantType" SwitchValue="3" /〉
〈/opc:StructuredType〉

ArrayLength 字段是可选的。如果编码值中没有该字段,那么使用 LengthField 设置"ArrayLength"的所有字段的长度为 1。

VariantType 字段具有一个与已定义字段不匹配的值是有效的。这意味着所有可选字段在编码值中不存在。

C.4.7 枚举类型

〈opc:枚举类型 Name="TrafficLight" LengthInBits="32"〉
〈opc:Documentation〉The possible colours for a traffic signal.〈/opc:Documentation〉
〈opc:EnumeratedValue Name="Red" Value="4"〉
〈opc:Documentation〉Red says stop immediately.〈/opc:Documentation〉
〈/opc:EnumeratedValue〉
〈opc:EnumeratedValue Name="Yellow" Value="3"〉

〈opc:Documentation〉Yellow says prepare to stop.〈/opc:Documentation〉
〈/opc:EnumeratedValue〉
〈opc:EnumeratedValue Name="Green" Value="2"〉
〈opc:Documentation〉Green says you may proceed.〈/opc:Documentation〉
〈/opc:EnumeratedValue〉
〈/opc:枚举类型〉

文档元素被用于提供类型和值的可读描述。

C.4.8 可为零(nillable)数组

〈opc:StructuredTypen Name—"NillableArray"〉
〈opc:Documentation〉An array where a length of -1 means null.〈/opc:Documentation〉
〈opc:Field Name="Length" 类型名称="opc:Int32" /〉
〈opc:Field
Name="Int32"
类型名称—"opc:Int32"
LengthField="Length"
SwitchField—"Length"
SwitchValue="0"
SwitchOperand="GreaterThanOrEqual" /〉
〈/opc:StructuredType〉

如果数组长度为—1,那么数组在字符串中不出现。

C.5 OPC 二进制 XML 结构

〈? xml version="1.0" encoding="utf-8" ?〉
〈xs:schema
目标命名空间="http://opcfoundation.org/BinarySchema/"
elementFormDefault="qualified"
xmlns="http://opcfoundation.org/BinarySchema/"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
〉
〈xs:element name="Documentation"〉
〈xs:complexType mixed="true"〉
〈xs:choice minOccurs—"0" maxOccurs—"unbounded"〉
〈xs:any minOccurs="0" maxOccurs="unbounded"/〉
〈/xs:choice〉
〈xs:anyAttribute/〉
〈/xs:complexType〉
〈/xs:element〉
〈xs:complexType name="ImportDirective"〉
〈xs:attribute name="Namespace" type="xs:string" use="optional" /〉
〈xs:attribute name—"Location" type—"xs:string" use—"optional" /〉
〈/xs:complexType〉
〈xs:simpleType name="ByteOrder"〉

```
〈xs:restriction base="xs:string"〉
〈xs:enumeration value="BigEndian" /〉
〈xs:enumeration value="LittleEndian" /〉
〈/xs:restriction〉
〈/xs:simpleType〉
〈xs:complexType name="类型描述"〉
〈xs:sequence〉
〈xs:element ref="Documentation" minOccurs="0" maxOccurs="1" /〉
〈/xs:sequence〉
〈xs:attribute name="Name" type="xs:NCName" use="required" /〉
〈xs:attribute name="DefaultByteOrder" type="ByteOrder" use="optional" /〉
〈/xs:complexType〉
〈xs:complexType name="OpaqueType"〉
〈xs:complexContent〉
〈xs:extension base="类型描述"〉
〈xs:attribute name="LengthInBits" type="xs:int" use="optional" /〉
〈xs:attribute name="ByteOrderSignificant" type="xs:boolean" default="false" /〉
〈/xs:extension〉
〈/xs:complexContent〉
〈/xs:complexType〉
〈xs:complexType name="EnumeratedValue"〉
〈xs:sequence〉
〈xs:element ref="Documentation" minOccurs="0" maxOccurs="1" /〉
〈/xs:sequence〉
〈xs:attribute name="Name" type="xs:string" use="optional" /〉
〈xs:attribute name="Value" type="xs:unsignedInt" use="optional" /〉
〈/xs:complexType〉
〈xs:complexType name="枚举类型"〉
〈xs:complexContent〉
〈xs:extension base="Opaque 类型描述"〉
〈xs:sequence〉
〈xs:element name="EnumeratedValue" type="EnumeratedValueDescription"
maxOccurs="unbounded" /〉
〈/xs:sequence〉
〈/xs:extension〉
〈/xs:complexContent〉
〈/xs:complexType〉
〈xs:simpleType name="SwitchOperand"〉
〈xs:restriction base="xs:string"〉
〈xs:enumeration value="Equals" /〉
〈xs:enumeration value="GreaterThan" /〉
〈xs:enumeration value="LessThan" /〉
〈xs:enumeration value="GreaterThanOrEqual" /〉
```

```
〈xs:enumeration value="LessThanOrEqual" /〉
〈xs:enumeration value="NotEqual" /〉
〈/xs:restriction〉
〈/xs:simpleType〉
〈xs:complexType name="字段类型"〉
〈xs:sequence〉
〈xs:element ref="Documentation" minOccurs="0" maxOccurs="1" /〉
〈/xs:sequence〉
〈xs:attribute name="Name" type="xs:string" use="required" /〉
〈xs:attribute name="类型名称" type="xs:QName" use="optional" /〉
〈xs:attribute name="Length" type="xs:unsignedInt" use="optional" /〉
〈xs:attribute name="LengthField" type="xs:string" use="optional" /〉
〈xs:attribute name="IsLengthInBytes" type="xs:boolean" default="false" /〉
〈xs:attribute name="SwitchField" type="xs:string" use="optional" /〉
〈xs:attribute name="SwitchValue" type="xs:unsignedInt" use="optional" /〉
〈xs:attribute name="SwitchOperand" type="SwitchOperand" use="optional" /〉
〈xs:attribute name="Terminator" type="xs:hexBinary" use="optional" /〉
〈xs:anyAttribute processContents="lax" /〉
〈/xs:complexType〉
〈xs:complexType name="StructuredType"〉
〈xs:complexContent〉
〈xs:extension base="类型描述"〉
〈xs:sequence〉
〈xs:element name="Field" type="字段类型" minOccurs="0" maxOccurs="unbounded"
/〉
〈/xs:sequence〉
〈xs:anyAttribute processContents="lax" /〉
〈/xs:extension〉
〈/xs:complexContent〉
〈/xs:complexType〉
〈xs:element name="类型字典"〉
〈xs:complexType〉
〈xs:sequence〉
〈xs:element ref="Documentation" minOccurs="0" maxOccurs="1" /〉
〈xs:element name="Import" type="ImportDirective" minOccurs="0"
maxOccurs="unbounded" /〉
〈xs:choice minOccurs="0" maxOccurs="unbounded"〉
〈xs:element name="OpaqueType" type="OpaqueType" /〉
〈xs:element name="枚举类型" type="枚举类型" /〉
〈xs:element name="StructuredType" type="StructuredType" /〉
〈/xs:choice〉
〈/xs:sequence〉
〈xs:attribute name="目标命名空间" type="xs:string" use="required" /〉
```

```
〈xs:attribute name="DefaultByteOrder" type="ByteOrder" use="optional" /〉
〈/xs:complexType〉
〈/xs:element〉
〈/xs:schema〉
```

C.6 OPC 二进制标准类型字典

```
〈? xml version="1.0" encoding="utf-8"?〉
〈opc:类型字典
xmlns="http://opcfoundation.org/BinarySchema/"
xmlns:opc="http://opcfoundation.org/BinarySchema/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
目标命名空间="http://opcfoundation.org/BinarySchema/"
〉
〈opc:Documentation〉This dictionary defines the standard types used by the OPC Binary
type description system.〈/opc:Documentation〉
〈opc:OpaqueType Name="Bit" LengthInBits="1"〉
〈opc:Documentation〉A single bit.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Boolean" LengthInBits="8"〉
〈opc:Documentation〉A two state logical value represented as a 8-bit
value.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="SByte" LengthInBits="8"〉
〈opc:Documentation〉An 8-bit signed integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Byte" LengthInBits="8"〉
〈opc:Documentation〉A 8-bit unsigned integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Int16" LengthInBits="16" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 16-bit signed integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="UInt16" LengthInBits="16" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 16-bit unsigned integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Int32" LengthInBits="32" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 32-bit signed integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="UInt32" LengthInBits="32" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 32-bit unsigned integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Int64" LengthInBits="32" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 64-bit signed integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
```

```
〈opc:OpaqueType Name="UInt64" LengthInBits="64" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 64-bit unsigned integer.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Float" LengthInBits="32" ByteOrderSignificant="true"〉
〈opc:Documentation〉An IEEE-754 single precision floating point
value.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Double" LengthInBits="64" ByteOrderSignificant="true"〉
〈opc:Documentation〉An IEEE-754 double precision floating point
value.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:OpaqueType Name="Char" LengthInBits="8"〉
〈opc:Documentation〉A 8 bit character value.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:StructuredType Name="String"〉
〈opc:Documentation〉A UTF-8 null terminated string value.〈/opc:Documentation〉
〈opc:Field Name="Value" 类型名称="Char" Terminator="00" /〉
〈/opc:StructuredType〉
〈opc:StructuredType Name="CharArray"〉
〈opc:Documentation〉A UTF-8 string prefixed by its length in
characters.〈/opc:Documentation〉
〈opc:Field Name="Length" 类型名称="Int32" /〉
〈opc:Field Name="Value" 类型名称="Char" LengthField="Length" /〉
〈/opc:StructuredType〉
〈opc:OpaqueType Name="WideChar" LengthInBits="16" ByteOrderSignificant="true"〉
〈opc:Documentation〉A 16-bit character value.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:StructuredType Name="WideString"〉
〈opc:Documentation〉A UTF-16 null terminated string value.〈/opc:Documentation〉
〈opc:Field Name="Value" 类型名称="WideChar" Terminator="0000" /〉
〈/opc:StructuredType〉
〈opc:StructuredType Name="WideCharArray"〉
〈opc:Documentation〉A UTF-16 string prefixed by its length in
characters.〈/opc:Documentation〉
〈opc:Field Name="Length" 类型名称="Int32" /〉
〈opc:Field Name="Value" 类型名称="WideChar" LengthField="Length" /〉
〈/opc:StructuredType〉
〈opc:StructuredType Name="ByteString"〉
〈opc:Documentation〉An array of bytes prefixed by its length.〈/opc:Documentation〉
〈opc:Field Name="Length" 类型名称="Int32" /〉
〈opc:Field Name="Value" 类型名称="Byte" LengthField="Length" /〉
〈/opc:StructuredType〉
〈opc:OpaqueType Name="DateTime" LengthInBits="64" ByteOrderSignificant="true"〉
```

```
〈opc:Documentation〉The number of 100 nanosecond intervals since January 01,
1601.〈/opc:Documentation〉
〈/opc:OpaqueType〉
〈opc:StructuredType Name="Guid"〉
〈opc:Documentation〉A 128-bit globally unique identifier.〈/opc:Documentation〉
〈opc:Field Name="Data1" 类型名称="UInt32" /〉
〈opc:Field Name="Data2" 类型名称="UInt16" /〉
〈opc:Field Name="Data3" 类型名称="UInt16" /〉
〈opc:Field Name="Data4" 类型名称="Byte" Length="8" /〉
〈/opc:StructuredType〉
〈/opc:类型字典〉
```

附 录 D
（规范性附录）
图形表示法

D.1 概述

本附录定义了 OPC UA 数据的图形表示法。本附录是规范性的，也就是说，在本部分中使用该表示法来表示 OPC UA 数据示例。然而，并不要求使用该表示法来表示所有 OPC UA 数据。

图形表示法能够表示 OPC UA 的所有结构性数据。节点、包括当前值的属性、包括引用类型的节点间的引用都可以被表示。图形表示法没有提供表示事件或历史数据的机制。

D.2 表示法

D.2.1 概述

表示法分为两个部分。简单表示法只提供了关于数据的简化视图，隐藏了类似属性的细节。扩展表示法允许表示 OPC UA 的所有结构化信息，包括属性值。在一个图中，可以组合使用简单表示法和扩展表示法来表示 OPC UA 数据。下面各节描述了简单和扩展表示法。

两种表示法共同的是既没有颜色，也没有线条的粗细和线型与表示法相关。可以使用那些效果来强调图形的某些方面。

D.2.2 简单表示法

根据其节点类，节点可以使用不同的图形形式表示，见表 D.1。

表 D.1 根据节点类的节点表示法

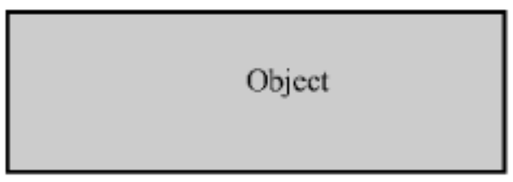
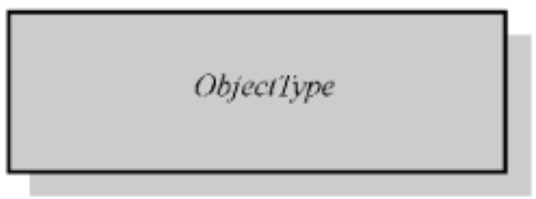
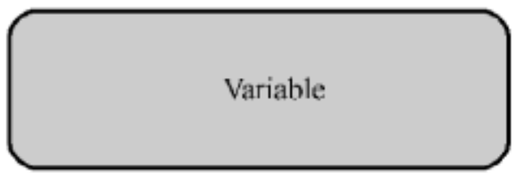
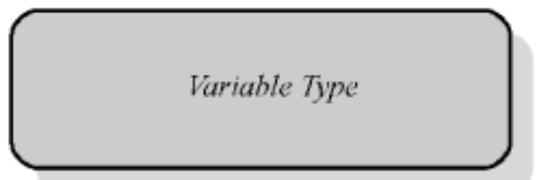
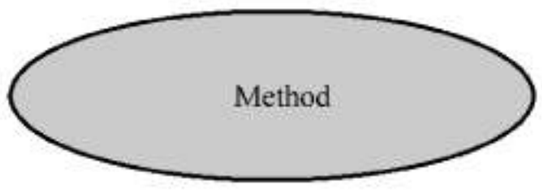
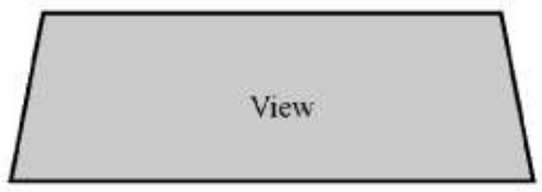
节点类	图形表示法	注 释
Object		包含表示对象的显示名称的字符串部分文本的矩形。字体不应设为斜体
ObjectType		包含表示对象类型的显示名称的字符串部分文本的阴影矩形。字体应为斜体
Variable		包含表示变量的显示名称的字符串部分文本的圆角矩形。字体不应设为斜体
VariableType		包含表示变量类型的显示名称的字符串部分文本的阴影圆角矩形。字体应为斜体

表 D.1（续）

节点类	图形表示法	注　　释
DataType		包含表示数据类型的显示名称的字符串部分文本的阴影六角形
ReferenceType		包含表示引用类型的显示名称的字符串部分文本的阴影六面多边形
Method		包含表示方法的显示名称的字符串部分文本的椭圆
View		包含表示视图的显示名称的字符串部分文本的梯形

被表示节点间连线的引用示例见图 D.1。连线可以根据各自的情况变化。不一定以直线连接节点；它们可以有直角、圆弧等。

图 D.1　连接两个节点的引用示例

表 D.2 定义了通常如何表示对称和非对称引用，也定义了某些引用类型的快捷方式。虽然推荐使用快捷方式，但并不是必须的。如果不使用快捷方式，也可以使用通常解决方法。

表 D.2　根据节点类的节点简单表示法

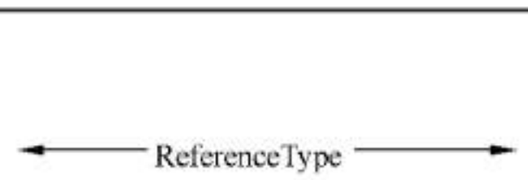
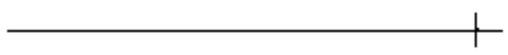
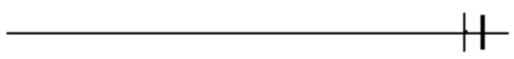
引用类型	图形表示法	注释
任何对称引用类型		对称引用类型表示为节点间的连线，在连线两边使用指向被连接节点的空心和实心箭头。连线附近应该有包含引用类型的 BrowseName 的字符串部分的文本
任何非对称引用类型		非对称引用类型表示为节点间的连线，在连线一边使用指向被连接目标节点的封闭、实心箭头。连线附近应该有包含引用类型的 BrowseName 的字符串部分的文本

表 D.2（续）

引用类型	图形表示法	注释
任何分层引用类型	——— ReferenceType ———→	作为层次引用的子类型的非对称引用类型应表示为与非对称引用类型同样的方式，除了应使用开放箭头外
HasComponent		该表示法提供了左边所示 HasComponent 引用的快捷方式。单独的竖线应靠近目标节点
HasProperty		该表示法提供了左边所示 HasProperty Reference 的快捷方式。双竖线应靠近目标节点
HasTypeDefinition		该表示法提供了左边所示 HasTypeDefinition Reference 的快捷方式。双封闭、实心箭头应靠近目标节点
HasSubtype		该表示法提供了左边所示 HasSubtype Reference 的快捷方式。双封闭箭头应指向 SourceNode
HasEventSource		该表示法提供了左边所示 HasEventSource Reference 的快捷方式。封闭箭头应指向目标节点

D.2.3 扩展表示法

在扩展表示法中，引入了一些其他概念。对于一个图形中的元素，只允许使用这些概念中的某些部分。

下列规则定义了该结构的某些特殊处理：

——通常，所有数据类型的值应由适当的字符串表示。当这些结构中使用了 NamespaceIndex 或 LocaleId 时，他们应被忽略。

——DisplayName 包含一个 LocaleId 和一个字符串。这样的结构可以表达为[<LocaleId>:]<String>，其中 LocalId 是可选项。例如，DisplayName 可以是"en:MyName"。或者，也可以使用"MyName"。无论何时显示 DisplayName，都适用该规则，包括节点的图形表示法中使用的文本。

BrowseName 包含 NamespaceIndex 和一个字符串。这样的结构可以表示为[<NamespaceIndex>:]<String>，其中 NamespaceIndex 是可选项。例如，BrowseName 可以是"1:MyName"。或者，也可以使用"MyName"。无论何时显示 BrowseName，都适用该规则，包括节点的图形表示法中使用的文本。

代替使用 HasTypeDefinition 引用从一个对象或变量指向其对象类型或变量类型，可以将类型定义的名称加入节点所使用的文本中。类型定义带有前缀"::"。图 D.2 给出一个示例，其中"节点 1"使用引用，而"节点 2"使用快捷方式。该图包含了用于某些节点的 HasTypeDefinition 引用和用于其他节点的快捷方式。不允许一个节点既使用快捷方式，同时又是 HasTypeDefinition 的源节点。

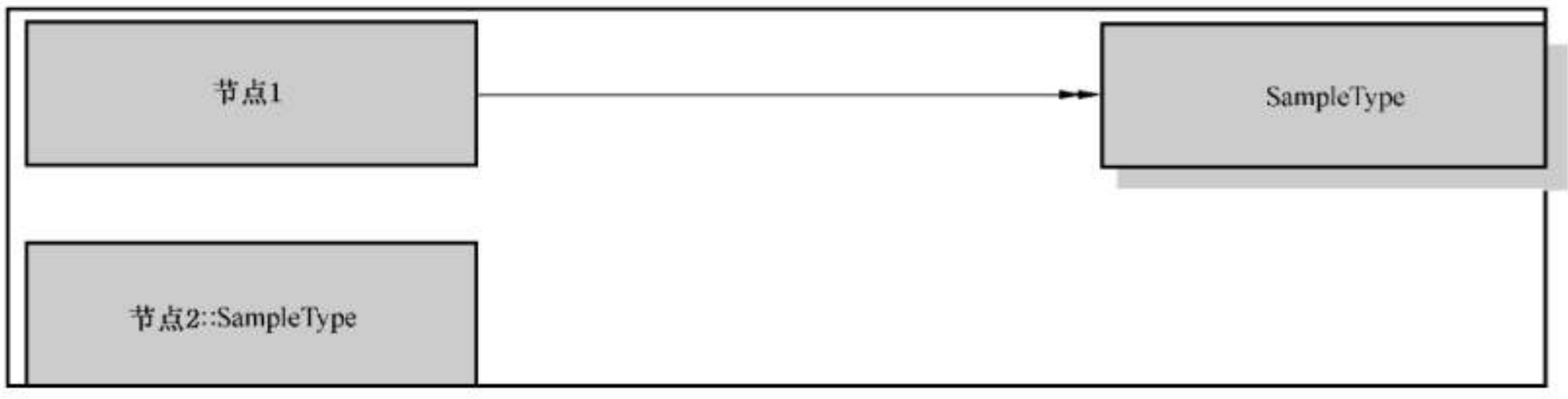


图 D.2　节点内部使用类型定义的示例

为了显示节点的属性，额外的文本可以放入表示节点的表格中，并置于表示 DisplayName 的文本下方。DisplayName 和描述属性的文本应使用水平线分隔。每个属性应设置一个新的文本行。每个文本行应包含属性名称，后面跟随一个“=”和该属性的值。包含一个属性的第一个文本行上方应是一个包含带下划线文本“属性”的文本行。不需要表示一个节点的所有属性。只允许展示属性的一个子集。如果没有提供任何可选的属性，那么属性能用一个贯穿线标记。例如“描述”。表示属性的示例见图 D.3。

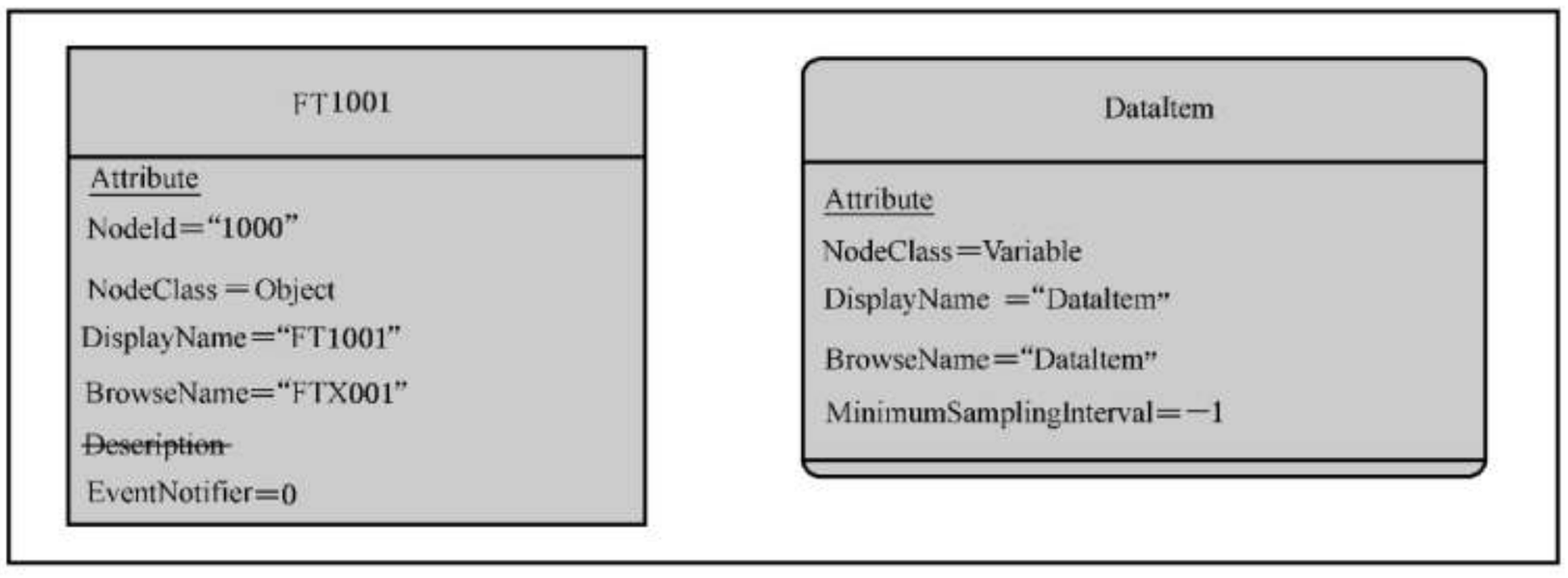


图 D.3　表示属性的示例

为了避免在一个图形中有太多节点，允许在节点内部表示特性，类似于属性。因此，用于表示属性的文本字段得到了扩展。在包含一个属性的最后的文本行下面，应增加一个包含带下划线文本“特性”的新文本行。如果没有提供属性，那么文本将以该文本行起始。该文本行后，每个新的文本行应包含一个特性，以特性的 BrowseName 开始，跟随“=”和该特性的 Value 属性值。图 D.4 展示了内部特性表示的示例。它允许在节点内部表示某些特性，其他特性作为节点。不允许一个特性既在内部表示，又作为一个额外的节点。

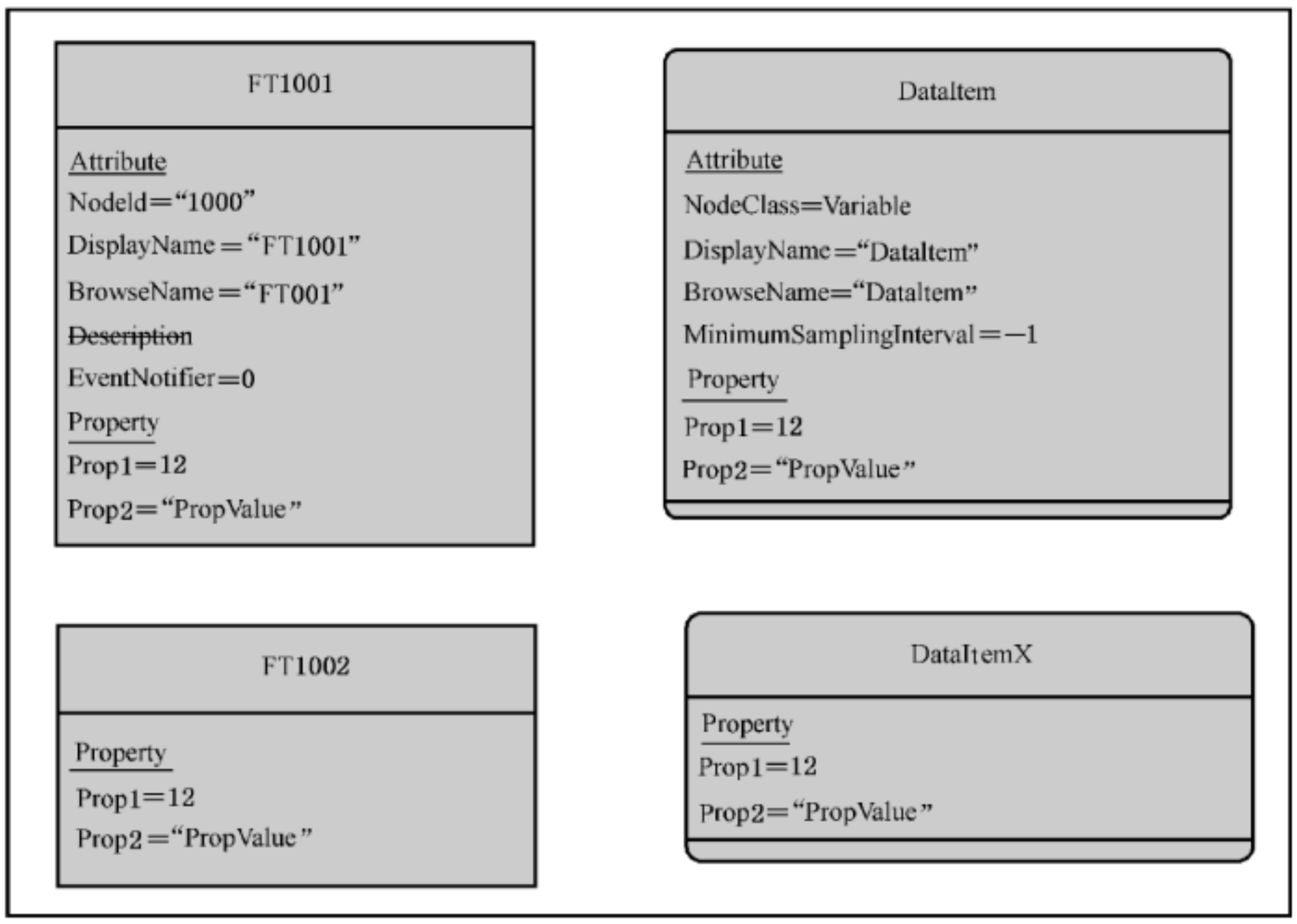


图 D.4　内部表示特性的示例

允许使用图形表示法为图形增加额外的信息，例如图注。

参 考 文 献

[1] IEC/TR 62541-2 OPC Unified Architecture—Part 2:Security Model

[2] IEC 62541-9 OPC Unified Architecture—Part 9:Alarms and Conditions

[3] IEC 62541-11 OPC Unified Architecture—Part 11:Historical Access

中华人民共和国
国家标准
OPC 统一架构 第 3 部分:地址空间模型
GB/T 33863.3 2017/IEC 62541-3:2010
*
中国标准出版社出版发行
北京市朝阳区和平里西街甲 2 号(100029)
北京市西城区三里河北街 16 号(100045)
网址:www.spc.org.cn
服务热线:400 168 0010
2017 年 7 月第一版
*
书号:155066·1-56669

GB/T 33863.3-2017