ICS 35.240.01
A11
备案号：

JR

中 华 人 民 共 和 国 金 融 行 业 标 准

JR/T 0107.6—2014/ISO 17369-6:2013

统计数据和元数据交换（SDMX）第6部分：SDMX技术说明事项

Statistical data and metadata exchange (SDMX)—
Part 6: SDMX technical notes

（ISO 17369-6:2013, SDMX technical notes, IDT）

2014－08－28 发布　　2014－08－28 实施

中国人民银行　发布

目　　次

前　言

JR/T 0107《统计数据和元数据交换（SDMX）》分为七个部分：

——第1部分：框架；

——第2部分：信息模型　UML概念设计；

——第3部分：SDMX-ML　模式和文档；

——第4部分：SDMX-EDI　语法和文档；

——第5部分：注册表规范　逻辑功能和逻辑接口；

——第6部分：SDMX技术说明事项；

——第7部分：Web服务用法指南。

本部分为JR/T 0107的第6部分。

本部分依据GB/T 1.1-2009规则起草。

本部分等同采用ISO 17369-6：2013《统计数据和元数据交换（SDMX） 第6部分：SDMX技术说明事项》。

本部分由中国人民银行提出。

本部分由全国金融标准化技术委员会（SAC/TC180）归口。

本部分主要起草单位：中国人民银行调查统计司、中国金融电子化公司。

本部分主要起草人：盛松成、徐诺金、姚力、巴运红、任全忠、潘润红、李曙光、韩建国、贾树辉、李兴锋、曹小艳、赵志兰、邓琳莹、许宁。

引　言

统计数据和元数据交换（SDMX）标准由SDMX国际组织发起并提出。SDMX国际组织是由国际清算银行（BIS）、经济合作与发展组织（OECD）、欧盟统计局（Eurostat）、欧洲中央银行（ECB）、国际货币基金组织（IMF）、联合国（UN）和世界银行（WB）七个国际组织联合建立，其制定发布的《统计数据和元数据交换》标准规定了统计人员在采集、处理和交换统计数据时所使用的统计概念和方法，规范了对外披露统计信息时统计数据的机构范围、地理区域、存流量性质、时间属性、频度以及文件格式等内容。

SDMX 标准提供了统计数据及元数据交换和共享的标准化格式，可以达到更好地扩展和高效率使用的目的。目前SDMX标准主要应用领域为部分国家中央银行和统计部门。本标准作用在于规范我国金融统计标准体系的内部处理和对外发布，促进金融统计的互联互通、信息共享和业务协同，提高信息共享的效率，满足金融综合统计的需要。

本部分的目的是记录一些对SDMX理解和实施决策非常重要的内容。此处的说明是对SDMX XML语法及信息模型的补充。

本部分可以分为以下几部分：

——数据结构定义和数据集相关的SDMX信息模型的指南；被不同的格式和句法所支持的数据结构定义和数据集在功能上的区别；SDMX格式使用的最优实践，包括时间期间的表示。

——元数据结构定义和元数据集相关的SDMX信息模型的指南。

——其它相关内容：机构、概念角色、范围、约束、部分代码表。

本版本的标准中，XML语法和信息模型中的“关键字族”的提法被数据结构定义（即DSD）所替代。许多人不熟悉该术语，它的名字取自SDMX-EDI模型（之前为GESMES/TS）。一个较熟悉的名字是“数据结构定义”，它在许多文档中被使用，同时也是SDMX-ML和信息模型技术说明书中的技术工具。“关键字族”的提法在SDMX-EDI模型中也仍然被使用。

目前在SDMX社区已经有很多旨在帮助理解和执行此标准的用户指南、教程和其他文档，本规范的目的不在重复那些文档的作用，而是希望能够提供一套简短的、其他文档没有广泛涉及的技术注解。

统计数据和元数据交换（SDMX）
第6部分：SDMX技术说明事项

1 范围

本部分规定了统计数据和元数据交换的技术说明事项，记录了一些对SDMX理解和实施决策非常重要的内容，是对SDMX XML语法及信息模型的补充。

本部分适用于金融统计中数据和元数据的交换和共享。

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。

JR/T 0107.3 统计数据和元数据交换（SDMX） 第3部分：SDMX-ML模式和文档

JR/T 0107.4 统计数据和元数据交换（SDMX） 第4部分：SDMX-EDI 语法和文档

ISO 8859-1 (UNOC)8位字符集

3 SDMX标准指南

3.1 引言

本部分旨在为SDMX格式标准——SDMX信息模型和SDMX-EDI的实施者提供有关数据信息，例如，数据结构定义和数据集。本部分提供信息的目的是，帮助SDMX使用者理解和实施这些标准。本部分内容不是规范性的，不提供任何SDMX标准的使用规则，相关使用规则请见《SDMX-ML：模式和文档》和《SDMX-EDI：语法和文档》。

3.2 SDMX信息模型

本部分的目的是，向关注XML或EDI格式使用情况的读者介绍SDMX信息模型的数据结构定义和数据集。对于希望深入了解信息模型的读者，完整的SDMX信息模型文件及SDMX标准的其他章节将进行详细介绍，并配有通用建模语言图表及辅助性说明。对于不熟悉“DSDs”的读者，SDMX信息模型的附录是较好的入门教程。

SDMX信息模型通常用于描述SDMX数据格式中使用的基础数据和元数据结构。信息模型涉及统计数据和结构化元数据，这些是本部分描述的。结构化元数据和数据都有一些附加的公共元数据，这些数据元与它们的经营和管理相关，不在本部分的描述范围内，也没有涉及SDMX信息模型文档的其它部分。

信息模型的数据结构定义和数据集部分与GESMES/TS版本3.0（SDMX标准中称为SDMX-EDI）一致，但有如下例外：

——不仅仅频率和GESMES/TS中一样，“同级组（Sibling Group）”结构已被泛化到允许任何单个或多个维度作为通配符。它被重命名为“组（Group）”，以便同仅有频率是通配符的“同级组（Sibling Group）”区别开。允许的局部“组（Group）”字集必须在数据结构定义里说明，且

属性应该可以附加到任何一个组关键字。——另外，为保持与版本2.0和SDMX-EDI 的兼容性，“组（Group）”被保留下来，在版本2.1中被“属 性关系（Attribute Relationship）”定义（稍后说明）所取代。

——数据表达式章节目前约定支持与 EDIFACT 语法的互操作性（见 3.3.2 节）。

DSD 特定的数据格式派生自信息模型，同时一些用来声明多重维度的支持性特征被加入到结构化元数据描述中。

根据其它部分。显然，这不是一个巧合。GESMES/TS 数据模型为SDMX-EDI中的EDIFACT信息模型提供依据，而且也是SDMX-ML开发的起点。

注意下面的描述，以斜体字出现的表示符是SDMX信息模型中使用的名称（如DataSet）。

3.3 SDMX-ML 和 SDMX-EDI：表达能力和功能的比较

SDMX提供数个等效的格式用于描述数据和结构元数据，以优化在不同应用中的使用。尽管所有这些格式直接派生自SDMX-IM（因此是等效的），但在应用中，在表述这些模型的语法中放置了一些不同的约束。不同的优化提供不同的能力。本条描述这些不同，并且提供一些应用的规则，这些规则需要支持多于一个SDMX格式或语法。本条限于数据结构定义和数据集。

3.3.1 格式优化和差异

下面这一节提供了对各种 SDMX 格式间差别的简洁性综述。

2.0 版本以 4 种数据信息为特点，每一种都有一种独特的格式：Generic、Compact、Cross-Sectional 以及 Utility。由于设计的关系，某些格式中的数据不能总是被转为另一种格式。在版本 2.1 中，这个问题通过分解为更多的格式或舍弃某些格式来解决。最终，在 SDMX2.1 中只有两种数据格式，GenericData 和 StructureSpecificData（例如：一个特定的数据结构定义）。

目前这两种格式都足够灵活，并且允许数据在多个维度定位，以便于消除歧义观察（与 V2.0 中仅能进行一项时序或横截面测量是不同的）。上述格式也被扩展到允许未分组观测中。

为了兼容只针对时间序列数据的应用，这些格式的衍生已经被两种数据信息的格式包含；GenericTimeSeriesData 和 StructureSpecificTimeSeriesData。需要指出的是这些衍生的格式是基于同样的根结构并且可以作为基础格式用同样的方式来处理，因此他们并不需要额外的处理需求。

a) 结构定义：

SDMX-ML 结构报文支持结构的注解的使用，SDMX-EDI 语法不支持这个。

SDMX-ML 结构报文允许有数据结构定义依赖的结构——就是代码表和概念——或者包含在信息模型中或者由包含数据结构定义的信息模型引用。设计的 XML 语法用于促使 URLs 和其它基于互联网的引用机制的改变，这些用于 SDMX-ML 信息模型中。这些选择对用于 SDMX-EDI 结构信息模型是不可用的。

b) 验证：

SDMX-EDI——因为是典型的 EDIFACT 语法信息模型——把验证留给专门的应用（“验证”是语法检查，数据类型，及结构定义里描述的数据信息模型到结构的粘连度）；

SDMX-ML 通用数据报文把 XML 语法等级上的验证留给应用；

DSD 特定的 SDMX-ML 信息将允许 XML 语法的验证和用于执行一般 XML 语法分析程序的数据类型及在结构定义和适当程度上有同样工具的数据间的执行协议；

c) 更新和删除信息模型及记录信息模型：

所有 SDMX 数据信息模型允许删除信息模型及信息模型仅包含数据或仅包含文档。

d) 字符编码：

所有 SDMX-ML 信息模型用 UTF-8 译码，然而 SDMX-EDI 使用 ISO 8879-1 字符译码。UTF-8 有更

大的容量来表达一些字符集（参见“ISO 8859-1 (UNOC)字符集的映射”）拉丁文 1 或者“WESTERN（西方）”在文档“语法和文档版本 2.0”中。许多转换工具是可用的，其允许用 UTF-8 译码的 XML 示例来表述成 ISO 8879-1-译码的字符，并转换 UTF-8 到 ISO 8879-1。当转换 SDMX-ML 信息模型到 SDMX-EDI 信息模型时应该使用这些工具，反之亦然。

e) 数据类型：

XML 语法和 EDIFACT 语法有不同的数据类型机制。本条下面提供当要求在两种语法中支持信息模型时的一个观察到的惯例集。SDMX-ML 数据表示形式上的更多信息参见 3.3.2 数据类型。

3.3.2 数据类型

XML 语法关于数据类型比有一个与 EDIFACT 语法有非常大差别的机制，这个差别可能给既支持基于 EDIFACT 又支持基于 XML 的 SDMX 数据格式的应用造成困难。本条提供了一个所有格式的数据中的表达式的惯例集，允许它们间完全的互用性。

应该注意，本条不介绍字符译码——假设变换软件包含转换的使用，该转换将是 SDMX-EDI 格式的 ISO 8879-1 译码和 SDMX-ML 格式的 UTF-8 间译码的映射。

下面的惯例可能是从事解除 EDIFACT 和 XML 的数据和元数据表达形式间的互操作。对于 EDIFACT 和 XML 语法间没有转换在其中的执行是可被预见的，下面的约束不需要应用。

a) 身份标识符是：

最多 18 个字符

任意的 A..Z(大写字母)，0..9(数字)，_(下划线)

第一个字符是字母。

b) 名字是：

最多 70 个字符

来自 ISO 8859-1 字符集（包括重读的字符）

c) 说明是：

最多 350 个字符；来自 ISO 8859-1 字符集

d) 编码值是：

最多 18 个字符；任意的 A..Z（大写字母），0..9（数字），_(下划线)，/（斜线），＝（等于符）；-（连字符），不过，给维度提供值的编码值必须仅使用下面的字符：

A..Z（大写字母），0..9（数字），_(下划线)

e) 观察值是：

十进制数字（仅是负数时有负号）；有效数字的最大数量是：整数 15 位；正小数或负整数 14 位；负小数 13 位；可以使用科学计数发。

f) 未编码的统计概念文本值是：

最多 1050 个字符；来自 ISO 8859-1 字符集

g) 时间序列关键字：

原则上，时间序列关键字用于数据交换的最大许可长度不需要限定。然而，对一个有效用图，限制在最多35个字符；在这个长度上，同样（对于SDMX- EDI）一个（分隔符）位置包含在所有的连续维度值之间；这意味着最大长度允许的一个纯序列关键字（维度值的连接）可以少于35个字符。通常分隔符字符是一个冒号（“：”）。

3.4 SDMX-ML 和 SDMX-EDI 最佳实践

3.4.1 报告及分发指南

3.4.1.1 中心机构和它们在统计数据交换中的角色

中心机构是其它合作机构向其“报告”统计数据的组织。这些统计数据或被中心机构用于收集汇总或把它们放在一起使之以统一的方式使用（如在线或在 CD-ROM 上或通过文件传输）。因此，中心机构从其它机构接收数据，且，通常，它们也“分发”数据给个人和/或机构最终使用。在一个国家里，一个 NSI 或一个国家中央银行（NCB）自然就扮演一个中心机构的角色，因为其从其它实体收集数据且它分发统计信息给最终用户。在 SDMX 中，中心机构的角色是非常重要的，所有统计信息模型基于基础的结构定义（统计概念，代码表，数据结构定义），基础的结构定义由一个特定的机构设计，通常是中心机构。这样一个机构为对应的被交换的信息模型扮演引用“结构定义维护机构的角色”。自然，两个可以使用/提交给结构信息交换数据的机构由第三个机构设计。

中心机构可以扮演两个角色：

收集并帮助分发统计数据；

设计用于数据交换的结构定义。

3.4.1.2 定义数据结构定义（DSDs）

为构建一个数据结构定义建议下面的准则。不过，当设计新的数据结构定义时，中心机构将遵守这些准则。

a) 维度、属性和代码表

避免维度对所有数据结构定义中的时间序列是不恰当的。如果一些维度对一些序列是不恰当的（如代码表中有一个标识为“not applicable”、“not relevant”或“total”的代码），则考虑移动这些序列到一个新的数据结构定义中，在这个数据结构定义中这些维度下移自数据结构定义。当有时很难在不大量增加 DSD 的情况下实现这一操作时，这是一个判断标准。

数据公开展示方面，设计 DSD 时使用尽可能少的维度。对于非专业用户来讲，一个超过 6 或 7 个维度的 DSD 通常是很难理解的。这些情况下，使用大量带有更小数据立方体结构的 DSDs 或者在更高层面消除和聚合数据应该会更好些。对 SDMX 数据标准来说，网络上的数据传播是一个不断增长的用例：通过维度来区分观测值，这一点对于统计学家和经济学家是必要的，但对于公共消费者而言经常会感觉晦涩，他们可能并不总是理解语义的分化。

避免混合维度。每个维度应相当于数据的一个简单特性，不是特性的一个组合。

考虑下列属性的包含关系。一旦数据结构定义的关键结构被确定，数据结构定义的属性集也需要被定义。总之，从所有数据结构定义到包含的有效信息，一些统计概念被认为非常必要。例如：

一个描述性的题目（这对于数据展示特别有用，例如，在网上）

收集（如：时期的终点，超期的平均值或总和）

单位（如：货币的面额）

复合单位（如：使用“百万”来计算）

用户（一个序列的可用对象）

小数位（如：在数值型数据中小数位的数量）

观测数据的状态（如：估计、暂时的、标准等）

此外，一些特殊的数据结构定义时，一些额外的属性可能是必需的。

避免在已存在代码表的地方建立新的代码表。强烈建议结构定义和代码表国际上已同意的标准方法一致，不管它们存在于哪里，如国民会计系统 1993（System of National Accounts 1993）；贸易支付差额手册 15 版（Balance of Payments Manual，Fifth Edition）；货币和财政统计手册（Monetary and Financial Statistics Manual）；政府财政统计手册等等。当按照一个新的数据交换，当使用代码表时，建议用下面的优先顺序。

——国际标准代码表；

——其它国际和/或区域机构增补的国际代码表；

——国际机构已经使用的标准化列表；

——两个国际的或区域的机构同意使用的新代码表；

——新明确的代码表。

同样的代码表在一个数据结构定义中或交叉的数据结构定义中可以用于几个统计概念。SDMX 已经认识到这些分类非常宽泛，而且在任意一个 DSD 中用到的代码只是全代码列表的一个小小的缩影。在本版标准中，全代码列表中部分代码列表的交换是可能的，对于一个特殊的 DSD 来说，这将使维度的值有效。

b) 数据结构定义结构

当定义一个新的数据结构定义时，结构定义维护机构应说明下面的项目：

数据结构定义身份验证：

数据结构定义身份标识符

数据结构定义名

元数据概念的列表分配与数据结构定义的维度一样。对每个：

（统计）概念身份标识符

关键字概念中维度的序数（只在 SDMX-EDI 中）

如果表达形式是编码的，代码表身份标识（ID，版本，维护机构）

（统计）概念列表分配与数据结构定义的属性一样。对每个：

（统计）概念身份标识符

如果概念是编码的，代码表身份标识符

任务状态：必选的或条件的

附件等级

非编码概念的最大文本长度

编码概念的最大编码长度

用于数据结构定义的代码表的列表。对每一个：

代码表身份标识符

代码表名

代码表值和说明

数据流定义的定义。两个（或更多）合作者在某些环境下执行数据交换需要商定：

数据集身份标识符列表应被使用；

对每个数据流：

它的内容和说明

相关的 DSD，这些 DSD 根据数据流定义（dataflow definition）定义了所报告或发布数据的结构。

3.4.1.3 交换属性----序列、组和数据集等级上的属性

静态特性

当建造序列时，发送者必须提供接受者所有必选属性的值。在它们时可用的情况下，可选属性的值应也被提供。但是，开始该信息可能有除了 SDMX-ML 或者 SDMX-EDI 信息模型外的方式提供（如纸质文件或电话），可以预料的是，合作机构将在一个位置上随着时间的过去用 SDMX-ML 或 SDMX-EDI 格式提供该信息；

一个中心可能同意它的数据交换伙伴的特别程序授权设定属性的初始值；

数据集等级上的属性值由中心管理的交换的数据集设置和维护；

给中心的变化通信；

下面一个序列的创建，发送者不必再报告属性值，如果它们没有改变；

每当序列（或同级组）的属性值变化发生，报告机构应或者再次报告所有的属性值（这是推荐的选择），或者仅报告改变的属性值。这个应用与必选的和条件的属性（例如，如果先前报告的条件的属性的值不再有效）必须报告到中心；

一个中心可能同意它的数据交换伙伴的特别程序授权属性值的修改；

观察值等级属性“观察值状态”，“观察值机密性”，“观察值破坏之前”的通讯；

在 SDMX-EDI 中，观察值等级属性“观察值状态”是用于观察值报告的 ARR 部分的固定语法的一部分。每当交换一个观察值，对应的观察值状态也必须附加在观察值上交换，不管它从上次数据交换后已经改变与否。这个准则也应用于 SDMX-ML 格式，尽管语法不必须要求这个；

如果“观察值状态”改变且观察值保持不变，应必须报告两个组件；

数据结构定义也有观察值等级定义的属性“观察值机密性”和“观察值破坏之前”，这个准则也应用于这些属性：如果一个机构接收到另一个机构的仅附加观察值属性的观察值，则意味着关联的观察值机密性和破坏前观察值属性或者不存在或者从现在起它们没有该观察值的值。

3.4.2 批量数据交换的最佳实践

3.4.2.1 引言

批量数据交换是在对应的实体间交换和维护整个数据库，这是一个通常使用 SDMX-EDI 格式的活动，也可能使用 SDMX-ML DSD 特定数据集下面指出了两种格式等价的应用。

3.4.2.2 维度“频率（Frequency）”的定位

“频率”维度的定位定义在数据结构定义中。此外，大多数中心机构决定把这个维度安排在关键字结构中的第一个位置。这有利于简化该维度的识别，有些东西在数据库系统中和在同级上的附加属性中对频率的决定性角色是必要的。

3.4.2.3 数据结构定义（DSDs）的标识

为了便于直接简便地认出结构定义维护机构二定义一个数据结构定义，大多数中心机构设计结构定义使用数据结构定义标识符的第一个字符来标识它们的机构：如 BIS_MACRO、EUROSTAT_BOP_01、ECB_BOP1 等。

3.4.2.4 数据流的标识

为了便于直接简便地认出管理一个数据流定义的机构，许多中心机构宁愿使用数据流定义标识符的第一个字符来标识它们的机构：如 BIS_MACRO、ECB_BOP1、ECB_BOP1T 等。

在 GESMES/TS 中数据集扮演数据流定义的角色（见 SDMX 信息模型中的 DataSet）。

SDMX 信息模型中的统计信息分解成两个基本规范——结构元数据（包括，数据结构定义及相关联的概念和代码表）和观察数据（数据集）——见标准的框架。这是一个最终的区别，由于具体术语关联到每一部分。数据——在时间的具体点上的典型的数字观察值的集——被组织到数据集(DataSet)中。这些数据集按照一个具体的数据结构定义（DataStructureDefinition）结构化，且被描述在数据流定义（DataflowDefinition）中。数据结构定义描述允许理解什么表述在数据集中的元数据，同时，数据流定义提供身份标识符和其它重要信息（如报告的周期），这对所有它的组件数据集是公用的。

数据流，模型中叫做数据流定义和数据集的角色在模型中是非常明确的，在所有的组织中使用的术

语可能不同，尤其术语数据集在 SDMX 中和在 GESMES/TS 中不一样。本质上 GESMES/TS 术语“数据集”是 SDMX 中的“数据流定义”，同时，术语“数据集”在 SDMX 中用于描述数据示例的“容器”。

3.4.2.5 需要特别讨论的问题----“频率”相关问题

专用的频率。在一个低于每天的频率（如每年 24 个或 36 个或 48 个观察值，在一年中不定期的日子）上的具体（定期或不定期的）时间间隔上收集的数据专题不在这里广泛讨论，对于数据交换目标：

a) 这样的数据可以用日报的频率映射到一个序列;这个每人的序列将仅包含测量事件的位置这些日子的观察值。

b) 如果收集间隔是定期的，将来可能增加附加值到存在频率的代码表。

分笔数据。在一个高于每天的频率（如滴答滴答数据）定期间隔上收集的数据专题也不在这里讨论。不管怎样，对于数据交换目标，这样的序列已经可能在 SDMX-EDI 格式中通过使用选择发送观察值及关联的时间戳交换了。

4 实施者说明

本条讨论除 SDMX-ML 和 SDMX-EDI 数据集交换以外的大量内容。这些内容包括 SDMX 中涉及的元数据机制，结构集和报告分类系统的使用，处理的使用，讨论时间、数据类型及 SDMX-ML 结构信息模型中关于版本和外部引用的一些常规机制。

本条没有深入讨论这些内容,但提供了有关这些内容的概述性描述来帮助实施者深入了解和使用它们相关部分。

4.1 表达形式

SDMX-ML中有几种不同的表达形式，取自XML模式（XML Schemas）和其它公用程序语言。下面描述SDMX-ML中不同的表达形式，它们是等效的（内容见表1）。

表 1

SDMX-ML Data Type	XML Schema Data Type	.NET Framework Type	Java Data Type
String	xsd:string	System.String	java.lang.String
Big Integer	xsd:integer	System.Decimal	java.math.BigInteger
Integer	xsd:int	System.Int32	int
Long	xsd: long	System.Int64	long
Short	xsd: Short	System.Int16	short
Decimal	xsd: decimal	System.Decimal	java.math.BigDecimal
Float	xsd:float	System.Single	float
Double	xsd:double	System.Double	Double
Boolean	Xsd:boolean	System.Boolean	Boolean
URI	Xsd:anyURI	Syetem.Uri	Java.net.URI or java.lang.String
DataTime	Xsd:dataTime	Syetem.DataTime	Javax.xml.datatype.XMLGregorianCalendar
Time	Xsd:time	System.DataTime	Javax.xml.datatype.XMLGregorianCalendar
GregorianYear	Xsd:gYear	System.DataTime	Javax.xml.datatype.XMLGregor

			ianCalendar
GregorianMonth	Xsd:gYearMonth	System.DataTime	Javax.xml.datatype.XMLGregorianCalendar
GregorianDay	Xsd:data	System.DataTime	Javax.xml.datatype.XMLGregorianCalendar
Day,MonthDay,Month	Xsd:g*	System.DataTime	Javax.xml.datatype.XMLGregorianCalendar
Duration	Xsd:duration	System.TimeSpan	Javax.xml.datatype.Duraion

也有大量的没有这些直接对应关系的 SDMX-ML 数据类型，通常因为它们是复合表达形式或者更宽泛的数据类型限制。对于绝大多数来讲，有一些可以从 SDMX 语法中参照的简单类型，对于其它的来说，需要一些派生的简单类型：

——字母数字（通常：字符数字类型，字母 A-Z 以及数字 0-9）；

——字母（通常：字母类型，A-Z）；

——数字（通常：数字类型，0-9 之间的字符串，但不是数值所以可以以 0 开头）；

——计数（xs:整数，以“1”为间隔的序列）；

——包含值范围（xs: 包含最小值和最大值界限的十进制数值）；

——不包含值范围（xs: 包含最小值和最大值界限的十进制数值）；

——增长值（xs: 指定间隔的十进制数值；间隔）；

——时间范围（通常：时间范围类型，开始日期时间+时长）；

——观测时期（通常：观测时期类型，一个标准时间和时间范围的组合）；

——标准时期（通常：标准时期类型，基本时期和时间范围的联合）；

——基本时期（通常：基本时期类型，公历时期和日期时间的联合）；

——公历时期（通常：公历时期类型，公历年、月、日的组合）；

——报告时期（通常：报告时期类型，报告年、半年、季度、月、周、日）；

——报告年（通常：报告年类型）；

——报告半年（通常：报告半年类型）；

——报告三个月（通常：报告三个月类型）；

——报告季度（通常：报告季度类型）；

——报告月（通常：报告月类型）；

——报告周（通常：报告周类型）；

——报告日（通常：报告日类型）；

——XHTML（通常：结构化的正文，允许多语言的正文内容）；

——关键值（通常：数据关键类型）；

——可标识参考（每一个可标识对象的类型）；

——数据集参考（通常：数据参考类型）；

——附件约束参考（通常：附件约束参考类型）。

数据类型同样包含一个方面的集合：

- isSequence = true | false (indicates a sequentially increasing value)
- minLength = positive integer (# of characters/digits)
- maxLength = positive integer (# of characters/digits)
- startValue = decimal (for numeric sequence)
- endValue = decimal (for numeric sequence)
- interval = decimal (for numeric sequence)
- timeInterval = duration
- startTime = BasicTimePeriod (for time range)
- endTime = BasicTimePeriod (for time range)
- minValue = decimal (for numeric range)
- maxValue = decimal (for numeric range)
- decimal = Integer (# of digits to right of decimal point)
- pattern = (a regular expression, as per W3C XML Schema)
- isMultiLingual = boolean (for specifying text can occur in more than one language)

应该注意，代码表除了它们本身的代码列举，可能还会有指定给它的原文表示。

4.2 时间和时间格式

4.2.1 引言

首先，认识到大多数观测时间是一个时期。SDMX 详细地指出时间是如何处理的。

时间的表示被分割为几个具有等级的表示方式组合。一个数据结构定义可以使用任何等级的表示方式。这使得一个特殊的数据结构定义的时间维度只允许默认的表示方式的一个子集。

时间格式的等级结构（加粗表示一个由多种格式组成的目录，斜体表示一个特殊的格式）如下：

- **Observational Time Period**
  - **Standard Time Period**
    - **Basic Time Period**
      - **Gregorian Time Period**
      - *Date Time*
    - **Reporting Time Period**
  - *Time Range*

这些时期分类目录和特殊格式的细节将在以下章节详述。

4.2.2 观测时期

这是 SDMX 中所有时间表示的超集。这允许时间用任何允许的格式来表示。

4.2.3 标准时期

这是任何预定义的时期或者一个时间点的超集。一个时期由一个不同的起点和终点。如果一个时期的起点和终点是用日期而不是绝对的日期时间，那么默认起始时间就是起始日的开始点（00：00：00）同时终止时间就是终止日的终点（23：59：59）。

4.2.4 公历时期

一个公历时期经常用年、年-月或者日的方式表示。这些都是基于 ISO8601 标准。在 SDMX-ML 信息中，每一个用公历时期表示的周期如下：

公历年：

表示：xs:gYear(YYYY)

周期：1 月 1 日至 12 月 31 日

公历月：
表示：xs：gYearMonth(YYYY-MM)
周期：每月第一天到最后一天
公历日：
表示：xs：data(YYYY-MM-DD)
周期：每天（00：00：00）到（23：59：59）

4.2.5 日期时间

这个被用来明白地声明，一个日期周期代表了某一个时间点的一个观测。因此，如果想对一个不同时间点的而不是超过报告超过一个周期的数据使用 SDMX，日期时间表示方式就可以使用。

表示方式：xs：日期时间（YYYY-MM-DDThh：mm：ss）

4.2.6 标准报告时间

标准报告周期是一个报告年的对应时间。每一个这些标准的报告周期有一个与之对应持续时间（基于 ISO8601 定义）。通用的报告周期的格式如下：

[REPORTING_YEAR]-[PERIOD_INDICATOR][PERIOD_VALUE]

这里:

REPROTING_YEAR 表示报告年份，用四位数字表示

PERIOD_INDICATOR 标识决定周期持续时间的周期类型

PERIOD_VALUE 标识年内的实际周期

下面的部分具体描述 SDMX 中的标准报告周期。

报告年:
时期指标: A
时期长度：P1Y（1 年）
每年限值：1
表示:通常:报告年类型(YYYY-A1, e.g. 2000-A1)
报告半年:
时期指标: S
时期长度: P6M (6 个月)
每年限值 r: 2
表示:通常:报告半年类型(YYYY-Ss, e.g. 2000-S2)
报告三个月:
时期指标: T
时期长度: P3M (3 个月)
每年限值: 3
表示: 通常：报告三个月类型(YYYY-Tt, e.g. 2000-T3)
报告季度:
时期指标: Q
时期长度: P4M (4 个月)
每年限值: 4
表示: 通常：报告季度类型(YYYY-Qq, e.g. 2000-Q4)
报告月:
时期指标: M

时期长度：P1M（1 个月）

每年限值：1

表示：通常：报告月类型(YYYY-Mmm, e.g. 2000-M12)

注：报告月通常表示为两位数，因此 1-9 由 0 填充（例如，01）。这允许数值可适用文本分类方法按时间顺序排序使用。

报告周:

时期指标: W

时期长度: P7D (7 天)

每年限值: 53

表示：通常：报告周类型(YYYY-Www, e.g. 2000-W53)

注：一个报告年有 52 个周或 53 个周，这是基于 ISO 8601 有关一周（星期一至星期六）的定义，其报告年的第一周被定义为含有第一个星期四或报告年起始日后的一周。报告周通常表示为两位数，因此 1-9 由 0 填充（例如，01）。这允许数值可适用文本分类方法按时间顺序排序使用。

报告日：

时期指标: D

时期长度: P1D (1 天)

每年限值: 366

表示：通常:报告日类型 (YYYY-Dddd, e.g. 2000-D366)

注：一个报告年有 365 天或 366 天，主要取决于报告年是否包含闰日（2 月 29 日）。报告日通常表示为三位数，因此 1-99 由 0 填充（例如，001）。这允许数值可适用文本分类方法按时间顺序排序使用。

一个报告年的方式总是基于那年的开始日并且需要报告年作为周期开始的那一年。对于一个报告年，这个开始日总是相同的，经常被表达为一天或一个月（如 7 月 1 日）。因此，由 7 月 1 日开始的报告年 2000 开始于 2000 年 7 月 1 日。

为了联系报告年的开始日，生成了一个指定的特殊属性（报告年起始日）。这个属性有一个固定的标识符（REPORTING_YEAR_START_DAY）以及一种固定的表示方式（xs:gMonthDay）这样就可以在数据信息中容易地将其标识并处理。虽然这个属性只存在于特定的子类中，其发挥的作用与其他任意标识符和表示方式之外的属性是一样的。标识符必须取自概念并且声明与数据结构定义的其它成分的关系。声明这种关系的能力允许这个报告年起始日在一个数据信息的合适的层面存在。没有这个属性，报告年的起始日将被假定为 1 月 1 日；因此，如果报告年考虑历法的话，这个属性就不是必需的。

既然持续期和和报告年起始日对于任意报告周期都是已知的，那么将任意的报告周期关联到一个特定的历法周期就是可能的。实际上报告期覆盖的公历周期可以通过以下计算得到（基于标准格式[REPROTING_YEAR]- [PERIOD_INDICATOR][PERIOD_VALUE]和报告期起始日[REPORTING_YEAR_START_DAY]）：

1.确定[REPORTING_YEAR_BASE]：

将报告周期值[REPORTING_YEAR] （YYYY）与[REPORTING_YEAR_START_DAY] (MM-DD)组合得到一个日期（YYYY-MM-DD）。

这就是[REPORTING_YEAR_START_DATE]

a) If the [PERIOD_INDICATOR] is W:

1) If [REPORTING_YEAR_START_DATE] is a Friday, Saturday, or Sunday:
Add3 (P3D, P2D, or P1D respectively) to the [REPORTING_YEAR_START_DATE],The result is the
[REPORTING_YEAR_BASE].

2) If [REPORTING_YEAR_START_DATE] is a Monday, Tuesday, Wednesday, or Thursday:

Add3 (P0D, -P1D, -P2D, or -P3D respectively) to the [REPORTING_YEAR_START_DATE]. The result is the [REPORTING_YEAR_BASE].

b) Else:

The [REPORTING_YEAR_START_DATE] is the [REPORTING_YEAR_BASE].

2. 确定 [PERIOD_DURATION]:

a) If the [PERIOD_INDICATOR] is A, the [PERIOD_DURATION] is P1Y.

b) If the [PERIOD_INDICATOR] is S, the [PERIOD_DURATION] is P6M.

c) If the [PERIOD_INDICATOR] is T, the [PERIOD_DURATION] is P4M.

d) If the [PERIOD_INDICATOR] is Q, the [PERIOD_DURATION] is P3M.

e) If the [PERIOD_INDICATOR] is M, the [PERIOD_DURATION] is P1M.

f) If the [PERIOD_INDICATOR] is W, the [PERIOD_DURATION] is P7D.

g) If the [PERIOD_INDICATOR] is D, the [PERIOD_DURATION] is P1D.

3. 确定 [PERIOD_START]:

Subtract one from the [PERIOD_VALUE] and multiply this by the [PERIOD_DURATION]. Add3 this to the [REPORTING_YEAR_BASE]. The result is the [PERIOD_START].

4. 确定[PERIOD_END]:

Multiply the [PERIOD_VALUE] by the [PERIOD_DURATION]. Add3 this to the [REPORTING_YEAR_BASE] add3 -P1D. The result is the [PERIOD_END].

对于所有的这些范围，边界包括了[PERIOD_START] (i.e. 00:00:00)的开始和[PERIOD_END] (i.e. 23:59:59)的结尾。

示例：

2010-Q2, REPORTING_YEAR_START_DAY = --07-01 (July 1)

1. [REPORTING_YEAR_START_DATE] = 2010-07-01

c) [REPORTING_YEAR_BASE] = 2010-07-01

2. [PERIOD_DURATION] = P3M

3. (2-1) * P3M = P3M

2010-07-01 + P3M = 2010-10-01

[PERIOD_START] = 2010-10-01

4. 2 * P3M = P6M

2010-07-01 + P6M = 2010-13-01 = 2011-01-01

2011-01-01 + -P1D = 2010-12-31

[PERIOD_END] = 2011-12-31

实际覆盖了2010-Q2的历法范围（假设报告年分开始于7月1日）是2010-10-01T00:00:00/2010-12-31T23:59:59。

2011-W36, REPORTING_YEAR_START_DAY = --07-01 (July 1)

1. [REPORTING_YEAR_START_DATE] = 2010-07-01

a) 2011-07-01 = Friday

2011-07-01 + P3D = 2011-07-04

[REPORTING_YEAR_BASE] = 2011-07-04

2. [PERIOD_DURATION] = P7D

3. (36-1) * P7D = P245D

2011-07-04 + P245D = 2012-03-05

[PERIOD_START] = 2012-03-05

4. 36 * P7D = P252D

2011-07-04 + P252D =2012-03-12

2012-03-12 + -P1D = 2012-03-11

[PERIOD_END] = 2012-03-11

实际覆盖了2011-W36的时间范围（假设报告年开始于7月1日）是

2012-03-05T00:00:00/2012-03-11T23:59:59。

4.2.7 独立范围

如果一个报告期不能与以上描述的周期相匹配，将使用一个独立的时间范围。这个范围的值基于ISO8601 时间间隔格式的开始和持续。开始可以用一个 ISO8601 日期或一个日期时间来表示，持续可以用一个 ISO8601 持续来表示。但是，持续时间只能是正值。

4.2.8 时间格式

在 SDMX2.0 版本中，推荐使用时间格式属性来提供信息中代表时间的方式额外信息。在有用性评价之后这一点已经不再需要。但是，如果需要，仍然有可能在 SDMX-ML 中包含时间格式属性。（见表 2）

表 2

Code	Format
OTP	Observational Time Period: Superset of all SDMX time formats (Gregorian Time Period, Reporting Time Period, and Time Range)
STP	Standard Time Period: Superset of Gregorian and Reporting Time Periods
GTP	Superset of all Gregorian Time Periods and date-time
RTP	Superset of all Reporting Time Periods
TR	Time Range: Start time and duration (YYYY-MM-DD(Thh:mm:ss)?/<duration>)
GY	Gregorian Year (YYYY)
GTM	Gregorian Year Month (YYYY-MM)
GD	Gregorian Day (YYYY-MM-DD)
DT	Distinct Point: date-time (YYYY-MM-DDThh:mm:ss)
RY	Reporting Year (YYYY-A1)
RS	Reporting Semester (YYYY-Ss)
RT	Reporting Trimester (YYYY-Tt)
RQ	Reporting Quarter (YYYY-Qq)
RM	Reporting Month (YYYY-Mmm)
RW	Reporting Week (YYYY-Www)
RD	Reporting Day (YYYY-Dddd)

4.2.9 SDMX-ML 与 SDMX-EDI 之间的转换

在将 SDMX-ML 数据结构定义转换到 SDMX-EDI 数据结构定义时，只有时间格式属性的标识符将被保留。属性的表示方式将被由 SDMX-ML 格式转换为 SDMX-EDI 代码表。如果 SDMX-ML 数据结构定义没有定义时间格式属性，那么需要用标识符“TIME-FORMAT”自动创建一个。

在转换 SDMX-ML 数据至 SDMX-EDI 时，源时间格式属性将是不恰当的。由于 SDMX-ML 的时间表示类

型容易引起歧义，目标时间格式可以直接从源时间值得到。例如，如果 SDMX-ML 时间是 2000Q2，SDMX-EDI 格式将总是 608/708（取决于目标序列是否包含一个观测值或一组观测值）。

当转换一个 SDMX-EDI 的数据结构定义时，时间格式属性应该被忽略，因为它在 SDMX-ML 中没什么作用。

当转换 SDMX-EDI 的数据至 SDMX-ML 时，源时间格式只在决定目标时间值的格式时是必要的。例如，一个格式为 604 的时间格式将形成一个以 YYYY-Ss 格式的目标时间，同时，一个格式为 608 的形成一个 YYYY-Qq 的目标时间值。

4.2.10 时区

在与 ISO8601 标准的组合过程中，SDMX 允许在所有时间周期内的时区的特殊化，并且在报告年的开始日。如果一个时区在一个报告年的开始日提供，那么相同的时区应该为每一个报告时间周期报告。如果报告年开始日与报告周期时区不同，那么报告周期的时区将有优先权。每种时区格式的实例如下：

Time Range (start date): 2006-06-05-05:00/P5D

Time Range (start date-time): 2006-06-05T00:00:00-05:00/P5D

Gregorian Year: 2006-05:00

Gregorian Month: 2006-06-05:00

Gregorian Day: 2006-06-05-05:00

Distinct Point: 2006-06-05T00:00:00-05:00

Reporting Year: 2006-A1-05:00

Reporting Semester: 2006-S2-05:00

Reporting Trimester: 2006-T2-05:00

Reporting Quarter: 2006-Q3-05:00

Reporting Month: 2006-M06-05:00

Reporting Week: 2006-W23-05:00

Reporting Day: 2006-D156-05:00

Reporting Year Start Day: --07-01-05:00

在 ISO8601 中，一个没有时区的日期被认为是“当地时间”。SDMX 假设当地时间是由信息发送者确定的。在 SDMX 版本中，为了指定一个时区，一个可选域被添加到发送者定义的头部。这个域的默认值是“Z”，当地时间的这种确定方式应用到一个消息的所有日期中。

4.2.11 其它时间跨度的表示

从 SDMX2.0 开始，一个元素可以指定一个时间跨度来表示。基于数据信息的格式，这将导致或者拥有两个 XML 属性的元素来指定开始和持续时间，或者两个分别的基于潜在的成分标识符的 XML 属性。例如，如果 REF_PERIOD 被给予一个时间跨度的表示方式，那么在一个简洁的数据格式中，其将会通过两个 XML 属性来表示：REF_PERIODStartTime（开始）和 REF_PERIOD（持续）。如果一个新的简单类型在保留 ISO 8601 的 SDMX 语法中被引入，那么这将不再是必须的。如下所示：

<Series REF_PERIODStartTime="2000-01-01T00:00:00" REF_PERIOD="P2M"/>

可以被这样表示：

<Series REF_PERIOD="2000-01-01T00:00:00/P2M"/>

4.2.12 关于格式的注解

这些格式中都没有歧义，所以任意给定的时间值，周期（包括预期的时期范围）的分类总是很清晰的。同样应该注意到，通过使用 ISO8601 格式，以及松散地基于它的报告周期格式，时间值可以容易地

按照先后顺序分类，而不用任何解析。

4.2.13 时间区间的作用

所有 SDMX-ML 的数据信息都可以以类似于 SDMX-EDI 的方式来使用，如果观测层级的维度是时间：第一个观测值的时间周期可以开始而观测值的其余部分可以忽略其时间值,因为可以开始时间和频率中得到。由于频率可以在基于所有的时间值的实际格式来确定，除了时间和时间区间的特殊点，这样处理起来就更加简单，因为时间区间之间的间隔被直接从时间值中得到。

4.2.14 信息查询中的时间

当查询时间值的时候，时间参数的值可以作为观测时间周期格式的任意一种来提供，同时需要一个相应的算子。此外，一个报告年起始日期的具体值可以被提供，或者被设为任意值。本节将具体详述处理查询信息的系统如何理解这些参数。

在一个查询信息中处理一个时间值参数的基础是理解所有的时期都应该被作为一个特殊的时间范围来处理。由于查询中的时间参数都带有一个运算符，这也同样有效地表示了特定的时间范围。因此，一个处理查询的系统必须先简单地匹配数据，这里请求参数的时期被查询参数生成的时期所包含。以下表格（表 3）详细说明了对一个参数提供的任意时期运算符是如何被解释的。

表 3

Operator	Rule
Greater Than	Any data after the last moment of the period
Less Than	Any data before the first moment of the period
Greater Than or Equal To	Any data on or after the first moment of the period
Less Than or Equal To	Any data on or before the last moment of the period
Equal To	Any data which falls on or after the first moment of the period and before or on the last moment of the period

报告时期作为查询参数,是基于 XML 属性“reportingYearStartDay” 的值是否是具体的某月某天或者任意。

如果时间参数为 XML 属性“reportingYearStartDay”提供了具体的某月某天的值，那么参数值就会被转化成一个特定的范围，并且像其他时间周期那样处理。

如果时间参数为 XML 属性“reportingYearStartDay”是一个任意值，那么报告周期边界内的所有值都会匹配，不论报告年度的实际起始日期。此外，非平常历法周期的数据报告也可以匹配，如果其可以落在基于报告年起始日 1 月 1 日的时间参数范围内。当确定一个报告周期是否落在一个报告周期查询参数的范围内时，需要将实际时间周期考虑进去，来比较周、日与更高顺序的报告周期。这将在下面示例中阐述。

示例：

公历时间

查询参数：Greater than 2010

解释说明：起始日期在2010-12-31T23:59:59之后的任意数据

匹配的实例：

2011 or later

2011-01 or later

2011-01-01 or later

2011-01-01/P[Any Duration] or any later start date

2011-[Any reporting period] (any reporting year start day)
2010-S2 (reporting year start day --07-01 or later)
2010-T3 (reporting year start day --07-01 or later)
2010-Q3 or later (reporting year start day --07-01 or later)
2010-M07 or later (reporting year start day --07-01 or later)
2010-W28 or later (reporting year start day --07-01 or later)
2010-D185 or later (reporting year start day --07-01 or later)

具有具体起始日的报告期

查询参数：Greater than or equal to 2009-Q3, reporting year start day = "-- 07-01"

解释说明：起始日期发生在2010-01- 01T00:00:00之后的任意数据（注意在本例中， 由于报告年起始日的值，2009-Q3 被转换为具体的日期范围2010-01-01/2010-03-31）。

匹配的实例：同上。

具有任意起始日的报告期

查询参数：Greater than or equal to 2010-Q3, reporting year start day = "Any"

解释说明：对于相同的报告年起始日，报告期的起始日期在2010-Q3的起始日期当天及之后的数据或者起始日期在2010-07-01当天及以后的数据。

匹配的实例：

2011 or later
2010-07 or later
2010-07-01 or later
2010-07-01/P[Any Duration] or any later start date
2011-[Any reporting period] (any reporting year start day)
2010-S2 (any reporting year start day)
2010-T3 (any reporting year start day)
2010-Q3 or later (any reporting year start day)
2010-M07 or later (any reporting year start day)
2010-W27 or later (reporting year start day --01-01)
2010-D182 or later (reporting year start day --01-01)
2010-W28 or later (reporting year start day --07-01)
2010-D185 or later (reporting year start day --07-01)

4.3 结构化元数据查询最佳实践

在查询结构化元数据的时候，声明参照应如何解决的能力是非常强大的。但是这个机制通常不是必要的，而且如果使用不当，可能为系统处理查询增加一个过度的负担。

任何包含一个对象的参照结构化元数据对象都可以基于那个参照来查询。例如，一个目录参照了一个目录和它所归类的对象。在这种情况下，可以查询目录化的特定对象的目录，或者是区别于特定目录的目录。这种机制应该在参照对象已知的情况下被使用。

当参照的对象不是已知的，那么可以使用参照解决机制。例如：假设希望能够发现一个给定的维护机构所有目录语法以及相关的目录方式。这种情况下，可以通过维护机构查询目录语法，并且指出父和兄弟级参照需要解决。这可能会产生在匹配的语法下返回参照目录的目录，以及它们目录化的对象。

4.4 版本和外部引用

在 SDMX-ML 结构报文中，应指出的版本和外部引用模式。标识符取决于版本号——即一个标识符含

有 Agency(机构)“A”、ID“X”及版本“1.0”的对象是与 Agency(机构)“A”,ID“X”及版本“1.1”的对象不同的。

假定可标识的对象/资源的产品版本是静态的——就是它们的 isFinal 属性设置成“true”。一旦形成产品，对象就不能以任何方式改变，或必须把它版本化。对于对象不是静态的这种情况，isFinal 属性值必须是“false”，但没有完工的对象应该不能用于规定的设计来适应它们的系统之外。在大多数场合，所有的对象在当成产品使用之前应该宣布最终完成。

这个机制是一个“早期封装”——版本化标识的所有东西的数量是已知的，且这些东西是不能改变的。有必要指出在有些情况下，关系是稳定的单向引用：一个例证是类别。尽管类可能被许多数据流和元数据流引用，许多引用自流对象的附加不给类建立版本。这是因为流不是类的特性——它们仅仅是产生到它的引用。如果类的名字改变，或者它的子类改变，则建立版本是必须的。

在 SDMX 信息模型中，建立版本操作在可建立版本和可维护对象的等级中。如果任何对象的子女在这些等级上改变，则对象自己被版本化。

受版本计划影响大的一个方面是引用外部对象的能力。尽管大量依赖于 SDMX 中不同结构的对象，它对拥有外部引用计划是有用的。这些操作于可维护对象的等级（数据结构定义，代码表，概念方案等等。）在 SDMX-ML 结构报文中，每当“isExternalReference”属性设置成真（true），则应用必须解决关联的“uri”属性提供的地址及使用 SDMX-ML 结构报文存储的位置（用于对象的全定义）等问题。如果已经提供注册表“urn”属性，则注册表可能被轮流地用于提供对象的全部的详细资料。

因为版本号是一个对象的标识符的一部分，版本是决定取得一个给定资源的必须部分。应该注意，每当不提供版本号，则假定它是“1.0”（“x.x”版本记号SDMX常规中的惯例，但不是要求的）。

5 元数据结构定义

5.1 范围

本版本中，元数据结构定义（MSD）的范围大大扩大，可以更好地支持附加元数据的结构类型。尤其是允许为数据集中任何关键字或部分关键字规定一个附件。这对于网络传播非常有用，在此可用元数据表示数据，但元数据只是与数据相关，不会与数据一起存储。为了满足这一点，在 MSD 中元数据作为附件被指定到一个关键字或部分关键字非常必要,同时实际的关键字和部分关键字将会在元数据集中被标识。

除了增加 MSD 中所包含对象的范围，本版本中标识符的运行机制、使用的专门术语都非常简单。

5.2 元数据所附目标类型特征

下面的例子显示了 MSD 元素在定义全关键字和部分关键字的使用实例的结构和命名。

一个 MSD 相应的语法结构如下（图 1）所示：

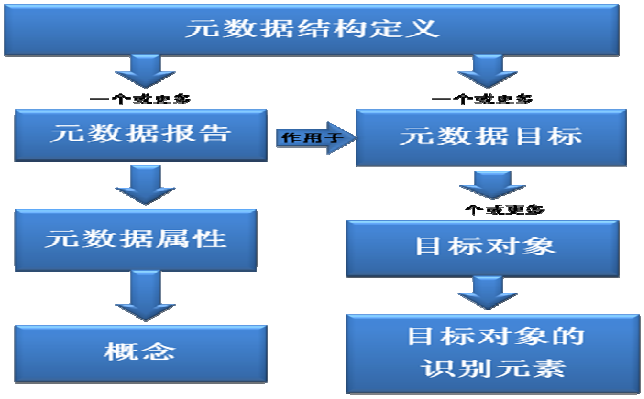

图 1 元数据结构定义示意图 MSD

包含了一个元数据集中元数据可以公布的对象类型的规定，和用元数据属性组成的用来标识概念的报告结构，此概念是在一个元数据集中可能被公布的。重要的是，一个报告结构为相关的元数据目标提供了参照。一个报告结构可以映射多个元数据目标，例如报告结构可以被不同的目标对象使用（见图 2）。

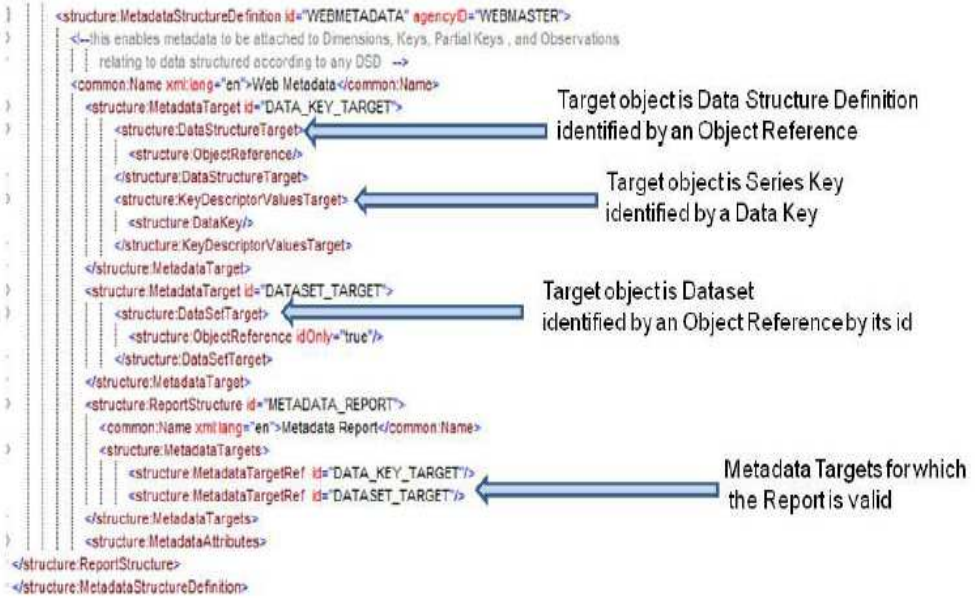

图 2 显示元数据目标的 MSD 用例

注：对于每个可以标识的对象类型来说，SDMX-ML 语法有一些明显的 XML 元素， 例如，一个可维护的对象与一个可标识对象有不同的属性，这个可标识对象必须同时包含机构 ID、版本以及所在的维护机构的 ID。

5.3 报告结构

下面是一个具体实例：

```
<structure:MetadataStructureDefinition id="WEBMETADATA" agencyID="WEBMASTER">
  <!--this enables metadata to be attached to Dimensions, Keys, Partial Keys , and Observations
      relating to data structured according to any DSD  -->
  <common:Name xml:lang="en">Web Metadata</common:Name>
    <structure:MetadataTarget id="DATA_KEY_TARGET">
    <structure:MetadataTarget id="DATASET_TARGET">
    <structure:ReportStructure id="METADATA_REPORT">
      <common:Name xml:lang="en">Metadata Report</common:Name>
      <structure:MetadataTargets>
        <structure:MetadataTargetRef  id="DATA_KEY_TARGET"/>
        <structure:MetadataTargetRef  id="DATASET_TARGET"/>
      </structure:MetadataTargets>
      <structure:MetadataAttributes>
        <structure:MetadataAttribute isPresentational="true">
          <structure:ConceptReference>
            <common:ConceptSchemeRef>
              <common:Ref id="IMETADATA_CONCEPTS" agencyID="WEBMASTER" version="1.0"/>
            </common:ConceptSchemeRef>
            <common:ConceptRef id="SOURCE"/>
          </structure:ConceptReference>
          <structure:MetadataAttribute>
            <structure:ConceptReference>
              <common:ConceptSchemeRef>
                <common:Ref id="IMETADATA_CONCEPTS" agencyID="WEBMASTER" version="1.0"/>
              </common:ConceptSchemeRef>
              <common:ConceptRef id="SOURCE_TYPE"/>
            </structure:ConceptReference>
          </structure:MetadataAttribute>
          <structure:MetadataAttribute>
            <structure:ConceptReference>
              <common:ConceptSchemeRef>
                <common:Ref id="METADATA_CONCEPTS" agencyID="WEBMASTER" version="1.0"/>
              </common:ConceptSchemeRef>
              <common:ConceptRef id="COLLECTION_SOURCE_NAME"/>
            </structure:ConceptReference>
          </structure:MetadataAttribute>
          <structure:MetadataAttribute>
          <!-- and so on for the reamaining metadata attribute -->
        </structure:MetadataAttribute>
      </structure:MetadataAttribute>
    </structure:MetadataAttributes>
  </structure:ReportStructure>
</structure:MetadataStructureDefinition>
```

这个实例显示了以下的元数据属性分级情况。

Source-只是表示存在此级并不包含任何元数据。

——Source Type

——Collection Source Name

5.4 元数据集

根据上面描述的 MSD，一个报告元数据实例列示如下：

```
<g:MetadataSet>
  <c:MetadataStructureDefinitionReference>
    <c:Ref id="WEBMETADATA" agencyID="WEBMASTER" version="1.0"/>
  </c:MetadataStructureDefinitionReference>
  <g:AttributeValueSet>
    <g:ReportRef>METADATA_REPORT</g:ReportRef>
    <!-- This is a partial key report (combination of codes from different dimensions) -->
    <g:TargetValues>
      <g:MetadataTargetValue id="DATA_KEY_TARGET">
        <g:ReferenceValue>
          <c:DataStructureReference>
            <c:Ref id="FINANCE_DSD" agencyID="WEBMASTER" version="1.0"/>
          </c:DataStructureReference>
        </g:ReferenceValue>
        <g:ReferenceValue>
          <c:DataKey>
          <c:DataKeyValue dimensionID="ECONOMICCONCEPT">
            <c:DimensionValue>ENDA</c:DimensionValue>
          </c:DataKeyValue>
          <c:DataKeyValue dimensionID="DATASOURCE">
            <c:DimensionValue>IFS</c:DimensionValue>
          </c:DataKeyValue>
        </c:DataKey>
        </g:ReferenceValue>
      </g:MetadataTargetValue>
    </g:TargetValues>
    <g:ReportedAttribute id="SOURCE">
      <g:ReportedAttribute id="SOURCE_TYPE">
        <g:Value>Market Values</g:Value>
      </g:ReportedAttribute>
      <g:ReportedAttribute id="COLLECTION_SOURCE_NAME">
        <g:Value>These series are typically the monthly average of market rates or official rates of the reporting countr
are not available, they are the monthly average rates in New York. Or if the latter are not available, they are estimates basec
averages of the end-of-month market rates quoted in the reporting country.
        </g:Value>
      </g:ReportedAttribute>
    </g:ReportedAttribute>
  </g:AttributeValueSet>
</g:MetadataSet>
```

本例显示：

a) MSD、元数据报告以及元数据目标（MetadataTargetValue）之间的参照；

b）报告的元数据属性（AttributeValueSet）。

6 维护机构

SDMX 中所有的机构化元数据被维护机构拥有并维护。在 agencyID 上没有分歧对于保持结构化元数据的完整性是非常重要。为了达到这一点，SDMX 采用以下规则：

a) 机构用一种机构语法（Agency Scheme：一种组织语法的子类）来维护；

b) Agency Scheme 的维护机构也必须用一种 Agency Scheme 来声明；

c) 最高等级的机构是 SDMX，并且它的机构语法由 SDMX 维护；

d) 在高级语法中注册的机构可以自己维护一种单独的语法。在 SDMX 机构语法中，SDMX 是一个机构。这个语法中的机构可以自行维护一种机构语法；

e) AgencyScheme 没有版本更新，所以默认为 1.0，还没有最终版；

f) 任何一个机构只能有一种 AgencyScheme，有一个 AgencyScheme 的固定 ID；

g) 机构标识的格式是 agencyId 或 agencyID 等。这种标识机制中最高等级的机构是注册在 SDMX 机构语法中的。也就是说，SDMX 不是机构中分等级的 ID 结构中的一部分，SDMX 本身就是一个维护机构。这样就支持了一种分等级结构的 agencyID。

下面是一个实例（图 3）。

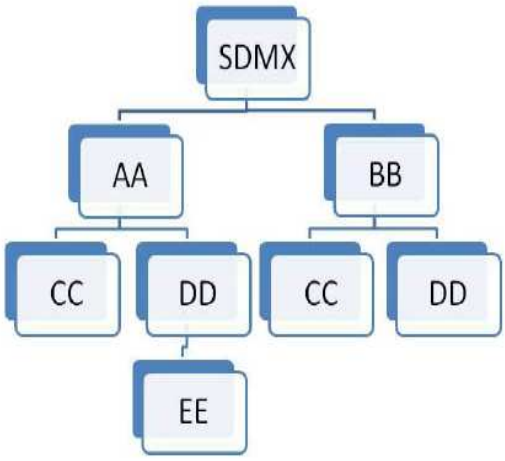

图 3 机构的分级结构实例

除了 SDMX，每个机构都可以由其完整的层次结构来标识。

表示这种结构的 XML 如下：

```
<structure:Organisations>
  <structure:AgencyScheme agencyID="SDMX" id="AGENCY_SCHEME">
    <common:Name>name</common:Name>
    <structure:Agency id="AA">
      <common:Name>AA Name</common:Name>
    </structure:Agency>
    <structure:Agency id="BB">
      <common:Name>BB Name</common:Name>
    </structure:Agency>
  </structure:AgencyScheme>
<structure:AgencyScheme agencyID="AA" id="AGENCY_SCHEME">
    <common:Name>name</common:Name>
    <structure:Agency id="CC">
      <common:Name>CC Name</common:Name>
    </structure:Agency>
    <structure:Agency id="DD">
      <common:Name>DD Name</common:Name>
    </structure:Agency>
  </structure:AgencyScheme>
<structure:AgencyScheme agencyID="BB" id="AGENCY_SCHEME">
    <common:Name>name</common:Name>
    <structure:Agency id="CC">
      <common:Name>CC Name</common:Name>
    </structure:Agency>
    <structure:Agency id="DD">
      <common:Name>DD Name</common:Name>
    </structure:Agency >
  </structure:AgencyScheme>
<structure:AgencyScheme agencyID="AA.CC" id="AGENCY_SCHEME">
    <common:Name>name</common:Name>
    <structure:Agency id="EE">
      <common:Name>EE Name</common:Name>
    </structure:Agency >
  </structure:AgencyScheme>
</structure:Organisations>
```

结构定义的实例：

```
<structure:Codelists>
<structure:Codelist id"=CL_BOP" agencyID="SDMX" version="1.0"
 urn="urn:sdmx:org.sdmx.infomodel. codelist.Codelist =SDMX:CL_BOP[1.0]">
    <common:Name>name</common:Name>
 </structure:Codelist>
<structure:Codelist id"=CL_BOP" agencyID="AA" version="1.0"
 urn="urn:sdmx:org.sdmx.infomodel. codelist.Codelist =AA:CL_BOP[1.0]" >
    <common:Name>name</common:Name>
 </structure:Codelist>
<structure:Codelist id"=CL_BOP" agencyID="AA.CC" version="1.0"
 urn="urn:sdmx:org.sdmx.infomodel.codelist.Codelist=AA.CC:CL_BOP[1.0]" >
    <common:Name>name</common:Name>
 </structure:Codelist>
<structure:Codelist id"=CL_BOP" agencyID="BB.CC" version="1.0"
 urn="urn:sdmx:org.sdmx.infomodel. codelist.Codelist =BB.CC:CL_BOP[1.0]">
    <common:Name>name</common:Name>
 </structure:Codelist>
 </structure:Codelists>
```

每一个这样的维护机构都有一个标识的代码表，id 为 CL_BOP。但是，每一个都是通过层级机构结构来唯一标识。

7 概念角色

7.1 概览

维度和属性元素在 DSD 中扮演着非常特殊的角色，并且对于某些应用来说非常重要。例如，以下一些实例中的角色：

——频率：在数据集中，这个元素的内容包含了观察值的频率信息；

——地理：在数据集中，这个元素的内容包含了观察值的地理位置信息；

——计量单位：在数据集中，这个元素的内容包含了观察值的计量单位信息。

为了使这些角色可扩展，也能确保用户团体的大众角色，该角色被保留在一个可控的词汇表中，在 SDMX 中，这个词汇表被作为一个概念语法中的概念使用。如果需要明确角色，那么这个元素可以随意地参照此概念。

注：一个元素可以承担不止一个角色，因此可同时参照多个角色概念。

7.2 信息模型

信息模型如图 4：

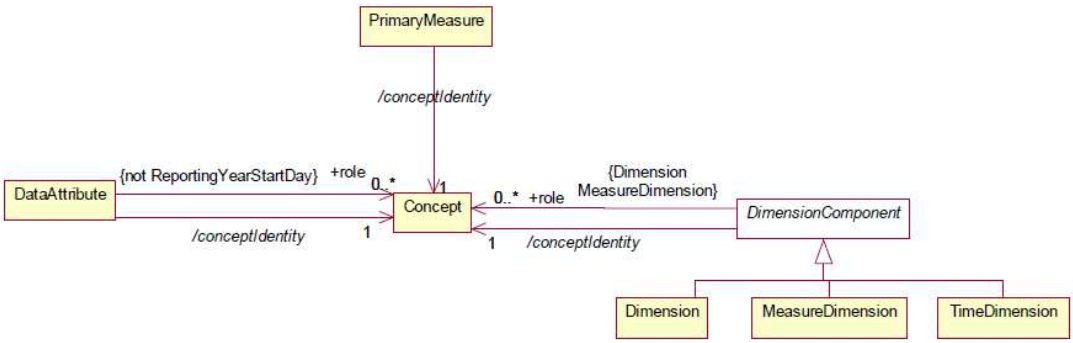

图 4 概念角色提取的信息模型

对一个维度、度量维度和数据属性（非 ReportingYearStartDay）指定 0 或多个概念角色是可能的。时间维度、主要度量以及属性“ReportingYearStartDay”已经明确的定义了角色，因此不能用其他概念角色再进一步指定。

7.3 技术机制

维护和使用概念角色的机制如下：

a) 任何注册机构可以有一种概念语法，此语法包含众多标识概念角色的概念。实际上，从技术角度来讲，任何机构可以有多种这样的语法，虽然我们不推荐这样做；

b) 包含“角色”概念的概念语法可以包含没有发挥角色作用的概念；

c) 概念本身是否是一个角色概念并没有明确的指示；

d) 因而，一种概念语法中的任何概念都可以是一个“角色”概念；

e) 机构有责任告知其用户团体，哪些概念在哪种语法概念中扮演“角色”，以及这个角色的意义和说明。换句话说，这些概念必须被应用所知，没有技术机制可以告知一个应用如何处理这样一个角色；

f) 如果一个 DSD 元素中概念标识所附概念（维度、测量维度、属性）包含在同时包含着概念角色的概念语法中，那么 DSD 元素就可以扮演概念所暗示的角色，如果这些可以被应用所理解；

g) 如果一个 DSD 元素中概念标识所附概念（维度、度量维度、属性）没有包含在同时包含着概念角色的概念语法中，同时 DSD 元素正在扮演一个角色，那么该概念角色被语法中的概念角色所标识。

7.4 SDMX-ML 实例

SDMX 维护的跨域概念语法包含了一些角色概念（FREQ 被作为一个实例）。

```
<structure:Dimension id="FREQ">
  <structure:ConceptIdentity>
    <URN>
urn:sdmx:org.sdmx.infomodel.conceptscheme.Concept=SDMX:CROSS_DOMAIN_CONCEPTS[1.0].FREQ
</URN>
  </structure:ConceptIdentity>
</structure:Dimension>
```

这是否是一个角色，取决于有关跨域概念语法中 FREQ 作为一种频率角色的应用理解。

使用一种非跨域概念语法，需要使用跨域概念语法指定一个角色。下面是有关 FREQ 的又一个实例。

```
<structure:Dimension id="FREQ">
  <structure:ConceptIdentity>
    <URN>
  urn:sdmx:org.sdmx.infomodel.conceptscheme.Concept=JBG:MY_CONCEPTS[1.0].FREQ
    </URN>
  </structure:ConceptIdentity>
  <structure:ConceptRole>
    <URN>
    urn:sdmx:org.sdmx.infomodel.conceptscheme.Concept=SDMX:CROSS_DOMAIN_CONCEPTS[1.0].FREQ
    </URN>
  </structure: ConceptRole >
</structure:Dimension>
```

明确说明，这个维度正在承担一个被跨域概念语法中的 FREQ 概念所标识的角色。同样，此应用需要理解跨域概念语法中的 FREQ 作为一个角色意味着什么。

这是在社区中实现互操作性所需的全部。重要的一点在于一个社区必须指定一个专门的机构，该机构有权在概念语法中定义概念角色并维护这些“角色”，同时还维护任意相关处理工具的使用文档。这会确定系统之间的互操作性，有助于保理解这些概念的使用。

注：每一个元素（数据属性、主要度量、维度、度量维度、时间维度）都有一个强制的识别关联（概念标识），并且如果这个概念同时还表示角色，那么它也可能通过某种方式声明这一点。

7.5 SDMX 跨域概念语法

SDMX 跨域概念语法中的所有概念都可以承担一个角色，这种语法将会包含版本 2.0 中所有的角色，或者基于与 2.0 版本兼容的考虑，或者出于对 2.1 版本中已被接受的额外角色需求，下面表格列示了这一语法中所需的一些概念。

注：每一个元素（数据属性、主要度量、维度、度量维度、时间维度）都有一个强制性的识别关联（概念标识），如果这个概念同时也标识了角色，那么可能需要通过属性 isRole（isRole=True）来声明，其他额外识别角色的

概念可以通过+role 关联来指定。

8 约束

8.1 引言

在此版本的 SDMX 中，约束是一个可维护的工具，并可以与以下各点相关联：

——数据结构定义；

——元数据结构定义；

——数据流；

——元数据流；

——提供许可；

——数据提供者（这里可限制到发布日历约束）；

——简单或可查询的数据源。

注：不论约束与何种对象关联，其都对与所约束对象关联的 DSD 中代码表的内容形成约束。当然，对于数据提供者来说，这一点并不适用，因为数据提供者可以通过数据提供协议与许多 DSDs 相关关联。这也是约束类型上的约束可被附加至一个数据提供者的原因。

8.2 约束类型

约束可以是以下两种之一：

——内容约束；

——可附加约束。

可附加约束被用来定义“cube slices”，其用来识别序列关键字或维度数值形式的数据子集。这样做的目的是允许元数据附加到约束中，以及约束定义的“cube slices”中。元数据可以通过“参考元数据”机制——MSD 和元数据集附加，或通过一个 DSD 中的组附加。图 5 是一个 DSD 的语法片段，显示了允许一个 DSD 中的组中参照的一个约束。

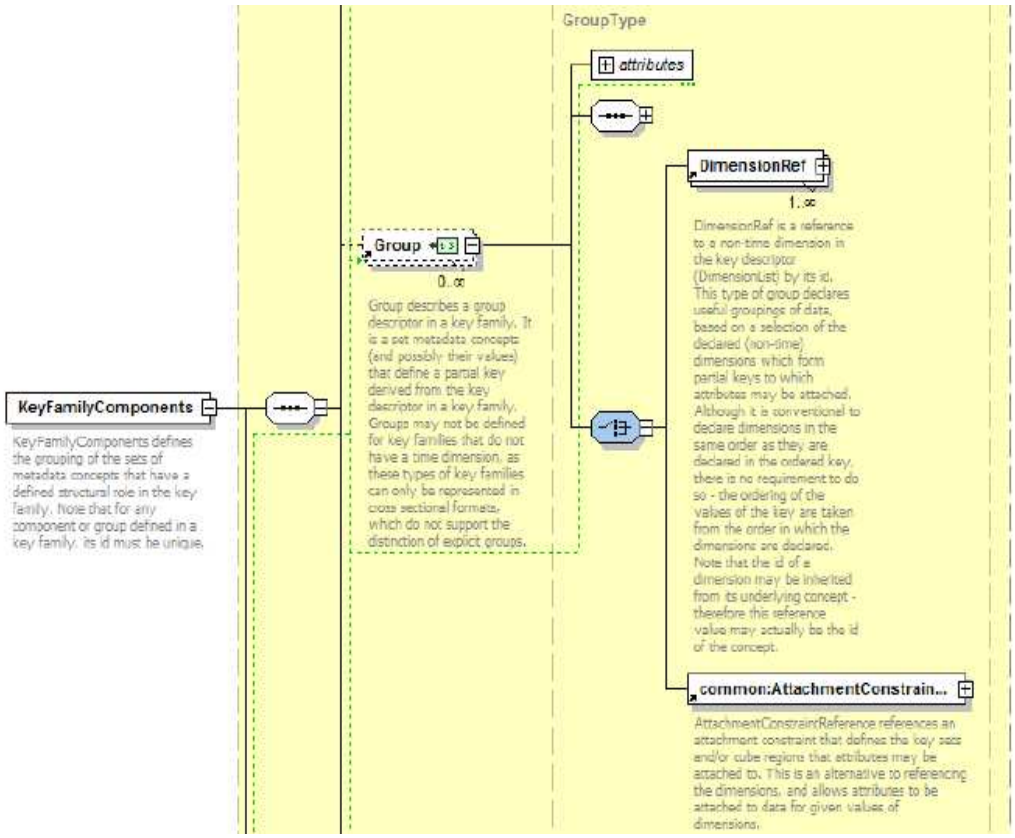

图 5 来自显示参照附加约束的 SDMX-ML 中的摘录

对于内容约束，特别是继承规则的应用，将在下面内容详述。

8.3 内容约束规则

8.3.1 内容约束范围

一个内容约束被用来指定一个数据或元数据源的内容，以元素数值或关键字的形式表示。

以数据形式表示的元素包括：

——维度；

——度量维度；

——时间维度；

——数据属性；

——主要测量。

同时关键字是“KeyDesciptor”的内容，例如，对于每一个关键字，序列的关键字是由每一个值的维度和度量维度组成。

以参考元数据形式表示的元素包括：

——目标对象是以下之一：

- 关键描述者数值；
- 数据集；
- 报告时期；
- 可识别对象。

——元数据属性。

因此，“关键字”是目标对象的综合，这些目标对象为了元数据对象而定义。

对于一个基于 DSD 的约束，内容约束可以参照一个或多个以下内容：

——数据结构定义；

——数据流；

——提供协议。

对于一个基于 MSD 的约束，约束的内容可以参照一个或多个以下内容：

——元数据结构定义；

——元数据流；

——提供协议。

此外，对某一对象可能会有多个内容约束，例如：对于某一个 DSD 的多个约束。

关于约束附件的灵活性，清晰的使用规则是必需的。这些将在以下详细说明。

8.3.2 多内容约束

对任一种可约束工具（例如，DSD）都有许多内容约束，并遵守以下限制：

8.3.2.1 立方区域

相关内容如下：

a) 约束可以包括多个 Member Selection（例如，维度）；

b) 对任意一个附加对象，Member Selection（例如，维度 FREQ）只能被包含在一个内容约束中（例如，某个 DSD 或某个数据流）。

8.3.2.2 关键字集

关键字集将按照它们在约束中出现的顺序加以处理，而且可以使用通配符（例如，任意并不明确参照的关键字位置被认为是“所有数值”）。由于关键集可以是“包含的”或“不包含的”，这里建议拥有通配符的关键字集在拥有特殊序列关键字的关键字集之前声明。这将大大减小关键字被随意包含或排除的风险。

8.3.3 内容约束的继承

8.3.3.1 内容约束的附件级别

总共有三个层次的附件约束，它们适用于以下继承规则：

DSD/MSD-最高级

数据流/元数据流-第二级

提供协议-第三级

注：这些规则不适用于简单数据源或可查询数据源：附加于这些对象的内容约束只是用来解决本对象，而不会考虑附加在其它对象上的约束（例如，提供协议，数据流，DSD）

一个内容约束没有必要附加到更高级别的对象之中，例如，如果没有附加相关的数据流或 DSD 约束，一个提供协议的内容约束是有效的。

8.3.3.2 过程处理中的级联规则

在数据流/元数据流或者提供协议的约束处理过程中必须考虑更高层次上的声明约束。低层次约束的规则下面将详述。

注：有一种情形，一个约束在较低层级上被指定之前需要在一个更高层级上被指定。因此，一个高层级的约束可能会使一个较低层级的约束无效。SDMX 对这样一个冲突发生时应如何处理没有规定。但是，评估约束的级联规则是清晰的——更高层级的约束在任何冲突中具有优先权，会使得低层级上的限制更小。

8.3.3.3 立体区域

a) 没有必要在更高层级产物中建立一个约束（例如，被数据流参照的 DSD），但是如果没有这样一个更高层级的约束，那么：
   1) 更低层级的约束需要比高一级的同样成员（例如：维度）更加严格（例如，如果维度 FREQ 被限制到一个 DSD 中的 A、Q，则数据流或提供协议中的约束不可以是 A、Q，M，甚至仅是 M，其只能进一步限制 A、Q）；
   2) 对任一成员低层级的约束进一步限制相同成员在更高层级上的内容。
b) 在内容约束中没有参照的任一成员按照更高层级上的内容约束被认定；
c) 如果在解决低层级约束不如高层级约束严格时发生冲突，那么高层级的约束被使用。

注：一个更高层级的约束在一个内容约束中可能约束四个维度，一个低层级的约束在两、三或四个内容约束中可能约束相同的四个维度。

8.3.3.4 关键集

a) 没有必要在更高层级上建立一个约束（例如，被数据流参照的 DSD），但如果有这样一个高层级的约束：
   1)低层级的约束需要比高层级的严格；
   2)任一成员在低层级上的约束进一步约束着在高层级上的关键字。
b) 在内容约束中没有参照的任一成员按照更高层级上的内容约束被认定；
c) 如果在解决低层级约束不如高层级约束严格时发生冲突，那么低层级的关键字不被认为是约束的一部分。

注：关键集的一个关键字有通配元素。例如，可能约束中简单地将维度 FREQ 约束到“A”，则所有的符合 FREQ=A 的关键字都是有效的。

以下说明了继承机制是如何工作的。注意这是一个概念性的逻辑，实际系统可能会与下述运行路径有所差异。

a) 决定所有可能在更高层级上有效的关键字；
b) 这些关键字被认为是从低层级约束对象中继承的，遵守低层级指定的约束；
c) 决定所有可能使用低层级约束的关键字；
d) 在低层级上继承所有与高层级约束匹配的关键字；
e) 如果低层级约束中关键字不是继承得到，那么他们是无效的如果低层级约束不是继承得来的，那么它是无效的。

8.3.4 约束实例

以下脚本被使用（见图 6）。

DSD

包含以下维度：

GEO — 地理

SEX — 性别

AGE — 年龄

CAS — 当前活动状态

在 DSD 中，常用的代码表被使用，并且需要在不同指定实际代码的层级上限制这些代码表，这些代码对于附加在内容约束上的对象是有效的。

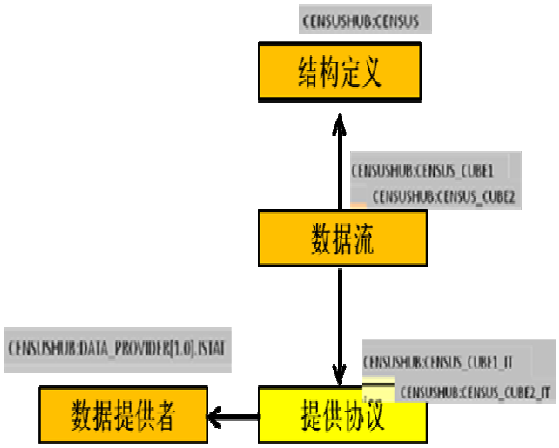

图 6 约束脚本实例

约束声明如图 7：

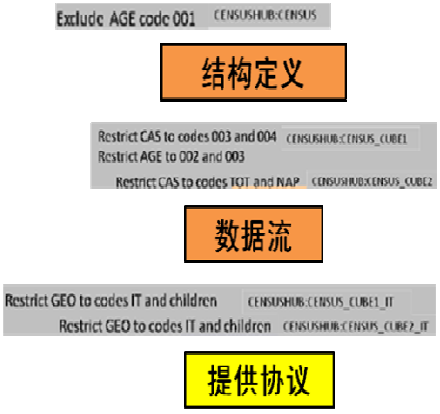

图 7 内容约束实例

注：

1. AGE 为 DSD 而约束，并为数据流 CENSUS_CUBE1 受到进一步限制。

2. 同样的约束同样适用于提供协议。

以上级联规则的脚本运行结果如下：

DSD

1.通过在维度 AGE 代码表中剔除代码 001 被约束。

数据流 CENSUS_CUBE1

a) 通过将维度 AGE 代码表限制到代码 002 和 003 被约束（注意，比起指定除代码 001 以外所有代码的 DSD 声明，这是一个更为严格的约束）。

b) 将 CAS 代码限制至 003 和 004。

数据流 CENSUS_CUBE2

a) 将 CAS 维度代码表限制至代码 TOT 和 NAP。

b) 继承应用于 DSD 层级上的 AGE 约束。

提供协议 CENSUS_CUBE1_IT

a) 将维度 GEO 代码表限制至 IT 及其下级。

b) 从有关维度 AGE 和 CAS 的数据流 CENSUS_CUBE1 中继承约束。

提供协议 CENSUS_CUBE2_IT

a) 将维度 GEO 代码限制至 IT 及其下级。

b) 从有关维度 CAS 的数据流 CENSUS_CUBE2 中继承约束。

c) 继承应用于 DSD 层级上的 AGE 约束。

约束定义如下：

DSD 约束

```
<structure:ContentConstraint id="CONSTRAINT1" agencyID="CENSUSHUB" type="Allowed" >
  <common:Name>name</common:Name>
  <structure:ConstraintAttachment>
    <structure:DataStructure>
    <Ref agencyID="CENSUSHUB" id="CENSUS"></Ref>
    </structure:DataStructure>
  </structure:ConstraintAttachment>
  <structure:CubeRegion include="true">
      <!-- note uses the ability ot exclude values - i.e all values valid except this one -->
  <common:KeyValue id="AGE" include="false">
    <common:Value>001</common:Value>
  </common:KeyValue>
  </structure:CubeRegion>
</structure:ContentConstraint>
```

数据流约束

```
<structure:ContentConstraint id="CONSTRAINT2" agencyID="CENSUSHUB" type="Allowed" >
  <common:Name>name</common:Name>
  <structure:ConstraintAttachment>
  <structure:Dataflow>
    <Ref agencyID="CENSUSHUB" id="CENSUS_CUBE1"></Ref>
  </structure:Dataflow>
  </structure:ConstraintAttachment>
  <structure:CubeRegion include="true">
  <common:KeyValue id="AGE" include="true">
    <common:Value>002</common:Value>
    <common:Value>003</common:Value>
  </common:KeyValue>
  <common:KeyValue id="CAS">
    <common:Value>003</common:Value>
    <common:Value>004</common:Value>
  </common:KeyValue>
  </structure:CubeRegion>
</structure:ContentConstraint>

<structure:ContentConstraint id="CONSTRAINT3" agencyID="CENSUSHUB" type="Allowed" >
  <common:Name>name</common:Name>
  <structure:ConstraintAttachment>
  <structure:Dataflow>
    <Ref agencyID="CENSUSHUB" id="CENSUS_CUBE2"></Ref>
  </structure:Dataflow>
  </structure:ConstraintAttachment>
  <structure:CubeRegion include="true">
  <common:KeyValue id="CAS" include="true">
    <common:Value>TOT</common:Value>
    <common:Value>NAP</common:Value>
  </common:KeyValue>
  </structure:CubeRegion>
</structure:ContentConstraint>
```

提供协议约束

```
<structure:ContentConstraint id="CONSTRAINT4" agencyID="CENSUSHUB" type="Allowed" >
  <common:Name>name</common:Name>
  <structure:ConstraintAttachment>
  <structure:ProvisionAgreement>
    <Ref agencyID="CENSUSHUB" id="CENSUS_CUBE1_IT"></Ref>
  </structure:ProvisionAgreement>
  <structure:ProvisionAgreement>
    <Ref agencyID="CENSUSHUB" id="CENSUS_CUBE2_IT"></Ref>
  </structure:ProvisionAgreement>
  </structure:ConstraintAttachment>
  <structure:CubeRegion include="true">
  <common:KeyValue id="GEO" include="true">
    <common:Value cascadeValues="true">IT</common:Value>
  </common:KeyValue>
  </structure:CubeRegion>
</structure:ContentConstraint>
```

关于“如何消除.NET SDMX Web Service 中额外元素”的相关资料，见附录A。

附 录 A
（资料性附录）
如何消除.NET SDMX Web Service 中额外元素

A.1 问题说明

为了实施一个遵守 Web 服务的 SDMX，需要使用标准化的 WSDL 文件来描述预期的请求/响应结构。操作的请求报文包含了一个封装元素（例如，“GetGenericData”），封装了一个称之为“GenericDataQuery”的标签，它其实是实际的 SDMX 查询 XML 报文，包含了需要被 Web 服务处理的查询。同样，响应也是通过一个封装元素“GetGenericDataResponse”来格式化。

由于在 SOAP 规范中已定义，一个 SOAP 报文的根元素是封套，包含了一个可选的的 Header（头）和一个强制的 Body（体）。以下并列是基于 WSDL 的 Body 内容。

```xml
<SOAP-ENV:Envelope
   <SOAP-ENV:Body>
      <GetGenericData>
         <sdmx:GenericDataQuery>
            ...
         </sdmx:GenericDataQuery>
      </GetGenericData>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

当前分析 SOAP 请求方式的区别，基于此产生的问题是何时去实施前面提及的在.net 框架下的 web 服务。

使用.NET 架构建立这样一个 Web 服务已经通过一种带有一个 XML 文档说明（称之为“Query”）的方法完成（例如，实例中的 getGenericData）。在 Microsoft .NET 中实施的区别在于 SDMX GenericDataQuery 周围需要有一个额外的 XML 容器。自框架被允许作为一个远程过程呼叫来自动发布 Web 服务，这是意料之中的动作，因此，将每一个参数封装进一个额外元素。.NET 请求说明如下：

```xml
<SOAP-ENV:Envelope
  <SOAP-ENV:Body>
    <GetGenericData>
      <Query>        <!-- MS .Net implementation -->
        <GenericDataQuery>
          ...
        </GenericDataQuery>
      </Query>       <!-- MS .Net implementation -->
    </GetGenericData>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

进一步来看,这个额外元素也被插入由框架自动生成的WSDL中。因此,这一特性需要Web服务的.NET客户端，这并不是可互操性的方案。

A.2 解决方法

使用.NET执行预期SOAP请求的解决方法与对XML有效负载的序列化和非序列化的人工干预有关。由于它是一个已被备的XML报文Web服务，因此是一种指示性的方式，可以对XML信息全控制。这也是所采用的SDMX Web服务的Java执行（使用Apache Axis）方式。

关于.NET平台，这与.NET Web方法参数XmlAnyElement的使用有关。

Web方法在.NET中使用了XmlSerializer来实现方法并建立响应（见图A.1）

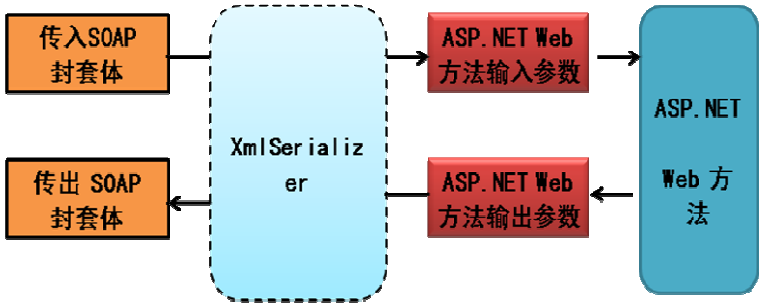

图 A.1 XmlSerializer 实现方法示意图

XML被传递到XmlSerializer，使其反序列化为托管代码中类的实例，代码映射到web方法的输入参数。同样的，为了建立SOAP响应报文的body，web方法的输出参数及返回值被序列化到XML中。

如果开发者希望在序列化和反序列化过程方面形成更多的控制，一个解决办法是使用参数XmlElment。这就提供了在反序列化之前使XML生效的机会，避免首先将其反序列化，对XML进行分析来决定如何反序列化，或者使用各种可直接用于处理XML的强大 XML API。这也给了开发者提供了控制差错的特定方式，而不是使用XmlSerializer可能产生的错误进行差错控制。

对一个Web方法XmlSerializer的反序列化过程进行控制，使用XmlAnyElement是一个简单的解决方式。

为了说明属性 XmlAnyElement 如何工作，我们提供了两种 Web 方法：

```
C#
// Simple Web method using XmlElement parameter

[WebMethod]

public void SubmitXml(XmlElement input)

{ return; }
```

在此方法中，输入参数由 XmlAnyElement 授予。这是暗示这一参数将从一个 xsd：any 元素被反序列化。由于任何参数都不传递属性，这意味着 SOAP 报文中这一参数的整个 XML 元素将会在参数 XmlElement 表示的信息集中。

```
C#
// Simple Web method...using the XmlAnyElement attribute

[WebMethod]

public void SubmitXmlAny([XmlAnyElement] XmlElement input)

{ return; }
```

两者之间的区别在于，对于第一种方法 SubmitXml，XmlSerializer 将会在 SOAP 主体中预期一个称之为 input 的元素作为一个 SubmitXml 元素的直接下级。第二种方法 SubmitXmlAny，则不考虑 SubmitXmlAny 元素的下级名称。它将在输入参数中插入 XML 的全部内容。两种方法中 ASP.NET Help 的信息形式如下所示。首先，看一下未带有 XmlAnyElement 属性的方法报文。

```
XML
<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>

    <SubmitXml xmlns="http://msdn.microsoft.com/AYS/XEService">

      <input>xml</input>

    </SubmitXml>

  </soap:Body>

</soap:Envelope>
```

下面，再看一下带有 XmlAnyElement 属性的方法的报文。

```xml
<?xml version="1.0" encoding="utf-8"?>

<!-- SOAP message for method using XmlAnyElement -->

<soap:Envelope

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>

    <SubmitXmlAny xmlns="http://msdn.microsoft.com/AYS/XEService">

      Xml

    </SubmitXmlAny>

  </soap:Body>

</soap:Envelope>
```

带有 XmlAnyElement 属性的方法缺少一个封装元素。仅有一个带有方法名称的元素对被传送到输入参数的信息进行封装。

更多信息请参见：

http://msdn.microsoft.com/en-us/library/aa480498.aspx

另外，这一点上有关不同请求的问题已被解决。但是，在需要解决的产出 WSDL 上仍然存在着区别。自动生成的 WSDL 不插入额外的元素，但将操作封装元素的内容定义为“xsd：any”类型。

```xml
<xs:element name="GetGenericData">
  <xs:complexType>
    <xs:sequence>
      <xs:any minOccurs="0" maxOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

由于没有一个通用的 WSDL，因此，解决方法仍不具有可互操作性。有两种方法可修正 WSDL。第一种是对产生过程进行干预。相比第二种方法，这一种较为复杂，对产生过程进行覆写，并且为 SDMX 的 Web 服务返回一个预想的 WSDL。

这个可以通过将对“/Service?WSDL”的请求重新定向至预期的 WSDL 来完成，后者被本地储存在应用中。为了达到这一点，从项目中添加一个“全局应用类”项目（.asax 文件），同时复写“Application_BeginRequest”方法中的请求。这在下一部分中将会详细阐述。

这个方法有一点不足，就是每一个 WSDL 的部署结点都需要被重定向到当前的 URL。但如果一个开发者使用一个简单的复写模块，将结点更改至一个当前的部署中，这一不便则可以容易被消除，

A.3 解决方法应用

在 SDMX Web 服务的环境中，对上述解决方法的应用可进行以下翻译：

```C#
[return: XmlAnyElement]

public XmlDocument GetGenericData([XmlAnyElement]XmlDocument Query)

{ return; }
```

SOAP 请求/响应如下所示：

GenericData 请求

```XML
<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>

    <GetGenericData xmlns="http://www.sdmx.org/resources/webservices">

      Xml

    </GetGenericData>

  </soap:Body>

</soap:Envelope>
```

GenericData 响应

```XML
<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

  <soap:Body>

    <GetGenericDataResponse
xmlns="http://www.sdmx.org/resources/webservices">

      Xml

    </GetGEnericDataResponse>

  </soap:Body>

</soap:Envelope>
```

为了覆写自动生成的 WSDL，在解决方法中，浏览器右击项目并选择“ADD”->“New item…”，然后选择“Global Application Class”。这将新建一个“.asax”类文件，在这个类文件中，以下代码将取代现存的空方法：

C#

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
  System.Web.HttpApplication app = (System.Web.HttpApplication)sender;
  if (Request.RawUrl.EndsWith("/Service1.asmx?WSDL"))
  {
    app.Context.RewritePath("/SDMX_WSDL.wsdl", false);
  }
}
```

SDMX_WSDL.wsdl 应该设在应用的根目录下。应用此办法之后，返回的 WSDL 是可预见的。因此，请求报文定义包含：

XML

```
<xs:element name="GetGenericData">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="sdmx:GenericQueryData"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```