# 2 数据类型系统

本段阐述有关此规范所定义的数据类型系统背后的概念性框架，此框架受到了 [[ISO 11404]](http://www.w3.org/TR/xmlschema11-2/#ISO11404) 标准（关于独立于语言的数据类型）以及 [[SQL]](http://www.w3.org/TR/xmlschema11-2/#SQL) 、Java之类的编程and 的类型系统的影响。

此规范所讨论的数据类型绝大部分是众所周知的诸如integer、time之类的抽象概念。此处并不包含完整的此类抽象概念的定义——已经有许多其它出版物提供了优秀的定义，但是，本规范将尝试尽可能好地描述这些抽象概念，以使它们能够容易地被识别与区分于其它易混淆的概念。

注：本规范中仅定义了那些在模式处理过程中需要用到的操作和关系，使用这些类型的应用程序通常应实现一些额外的函数或关系以类型更通用。

## 2.1 数据类型

定义：在本规范中，一个数据类型具有三个属性：

值空间（Value Space），由一系列值构成；

词法空间（Lexical Space），一个字面量的集合，用于描述值；

一个包含类型相关操作、关系及程序的集合，定义值空间上的等于关系和顺序关系，以及一个词法映射（Lexical Mapping）——用于将词法空间映射到值空间。

对于词法映射，有一个对应的提供值到标准“词法描述”（Lexical Representation）的逆映射会很有用，虽然这在模式处理过程中不需要，但本规范还是对其进行了描述。对于一些数据类型（尤其是QName和NOTATION），其词法描述到值和映射是上下文相关的，因此并没有标准映射（Canonical Mapping）。

注：标准映射定义在原生（Primitive）数据类型上，如果通过直接约束值空间来派生一个新类型，则对于排除在值空间外的所有值，其对应的词法描述也排除在词法空间之外。对这些类型的标准映射是其原始类型的标准映射的子集。正则面（Facet，以后直接用Facet吧）以及其它由实现定义的词法Facet，直接约束词法空间，当一个给定值具有多个词法描述时，这些Facet可能会禁用其标准描述同时允许其它词法描述，在这种情况下，这些值依然在值空间中但没有标准描述，本规范没有为这种情况提供Recourse，应用程序可以以自认为合适的方式自由处理。

注：本规范有时使用“类型”短语来代替“数据类型”一词，尤其是在某些能够确切预期它即指“数据类型”的场合（例如在”Union Type”、“List Type”段落中）。这两个术语没有本质区分，可互换使用。相比之下，“类型”与“简单类型定义”（Simple Type Definition）就有所差别：“类型”以值空间、词法空间和词法映射为特征，它独立于在“简单类型定义”中用于描述特定的值空间、词法空间的Facet及其它机制。一个类型可由不同的“简单类型定义”用不同的Facet策略来描述。

## 2.2 值空间

类型的值空间可通过以下方式来定义：

在其它地方已明确的基本观念（内涵定义）；

定义一个已定义类型的值枚举；

通过限定，定义一个已定义类型的值空间的子集；

组合已定义类型的值空间（见List和Union）。

每个值空间都必须有标识（Identity）和（equality）关系，部分值空间具有顺序关系，少数值空间具有其它一些关系和操作。

### 2.2.1 标识（Identity）

值空间天生具有identity关系，只有当两个事实上就是同一个东西——即没有任何办法来区分它们的时候，两个东西就是identical的。

### 2.2.2 相等（equality）

所有原生类型的值空间上都规定了相等关系，大部分类型的相等关系就是identity关系。在少数的例外中，相等关系总是根据类型相关的操作进行小心地定义，如果两个值在所有的操作中都可以互相替换，那就说明它们是相等的。

另一方面，相等关系并不需要覆盖整个值空间（虽然大部分时候会），特殊地，在float和double类型中，NaN与自己不相等。

## 2.4 数据类型差异

从不同维度来对数据类型进行分类，使用不同术语来描述类型及类型定义的特征是有用的。

### 2.4.1 Atomic、List与Union类型

原子（Atomic）值是指一个基本的值，在本规范中不存在任何更简单的类型来构成它。

原子类型指值空间仅包含原子值的类型，包括anyAtomic及由其派生的类型；

列表（List）类型指值由有限个原子值的序列组成的类型，列表中的值来自列表的项类型参数（item type）指定的类型（可以是原子类型，或仅包含原子类型的Union类型）；

联合（Union）类型指多个其它类型的值空间、词法空间和词法映射合并得到的新值空间、词法空间和词法空间对应的类型。

#### 2.4.1.2 List

列表类型的项类型可以是词法空间允许空格的普通类型、原生、原子类型，以及它们的联合，这是因为列表中的项会先根据空格进行分割，然后才映射到值空间。这意味着，列表项的词法描述中不会出现空格——即使它是允许的。

列表类型的词法空间由项类型的词法描述序列用空格拼接而成，从列表类型派生并应用于派生类型上的pattern约束应用于列表类型的词法空间上，而不是项类型的词法空间，类似地，枚举约束也是应用于列表类型而不是项类型的词法空间上。

列表类型的等于关系：当且仅当两个列表值的项数、项值及其顺利完全一致时，列表值相等。

当通过约束（restrict）从一个列表类型派生新类型时，可以应用的约束Facet有：

|  |  |
| --- | --- |
| Facet | Description |
| length | 列表中允许的项数 |
| maxLength | 列表中最多允许的项数 |
| minLength | 列表中至少要包含的项数 |
| enumeration | 应用于列表类型的词法空间上 |
| pattern | 应用于列表类型的词法空间上 |
| whiteSpace | 固定为collapse |
| assertions |  |

#### 2.4.1.3 Union

在联合中，一个给定的词法描述可能同时存在于不同成员类型的词法空间中，因而可以被映射为多种类型的值，在特定上下文中，它的取值可能难以确定。当在【XSD1.1 Part 1:Structures】的上下文中时，xsi:type属性可用于决定映射到哪个值。而在其它的上下文中，可以借助其它规则（如type coercion rules）。

当一个联合类型的成员类型是另一个联合类型时，它的值空间、词法空间和词法映射是基类型的成员类型和当前类型的其它成员类型的并集。

目前没有内置联合类型。

### 2.4.2 Special、Primitive与Ordinary类型

特殊（special）类型包括anySimpleType和anyAtomicType，它们因在类型继承体系中的地位而特殊。

原生（primitive）类型指那些即不是特殊类型又不是从其它类型定义而来的类型，所有原生类型都以anyAtomicType为基类，但它们的值和词法空间必须不同（given in prose）,它们无法由应用程序通过facet约束来描述。

普通（ordinary）类型指特殊类型与原生类型之外的所有其它类型，普通类型可以根据其类型定义及构造它们的类型（原始类型）来理解。

使用Facet约束创建新类型时，其基类必须是原生类型或普通类型。Facet只能将基类的值空间、词法空间约束到一个子集。

List类型以anySimpleType为基类。

Union类型以anySimpleType为基类。

### 2.4.3 Definition、Derivation、Restriction与Construction

定义（Definition）指对类型相关属性的确切认定，尤其是类型的值空间、词法空间及词法映射。

特殊类型和原生类型的属性由本规范定义，这些类型的简单类型定义存在于每个有效的模式中，它们作为这些类型的代表，但并不携带相关的信息，不足以用于定义类型。

对于其它类型，它们的简单类型定义是足料的，普通类型的属性可以根据类型的定义以及其基类型、项类型或成员类型的属性来推定。

派生（Derivation）定义类型与其基类型的关系。除anySimpleType外，所有类型都具有基类，基类可以是特殊类型、原生类型或普通类型。

类型不可以以自身或其派生类为基类，派生关系不可以构成环。除anySimpleType外，所有类型都直接或间接派生于anySimpleType。

约束（Restriction）意味着取基类的值空间、词法空间以及词法映射的一个子集作为一个新的类型定义。

三种构造类型的形式都会产生对基类的约束：facet约束直接产生相关facet的约束，List和Union则产生对anySimpleType的约束（它们以anySimpleType为基类）。本规范无法产生一种词法空间或值空间大于其基类的类型。

构造（Construction）指使用其它类型来定义新的类型。所有普通类型都通过约束、列表、联合其它类型的值空间、词法空间构造而来。

### 2.4.4 Built-in与User-Defined类型

内置（Built-in）类型是本规范已定义的类型，包括特殊类型、原生类型以及普通类型；

自定义（User-Defined）类型指用户通过模式设计器定义的类型；

# 4 数据类型组件

## 4.1 简单类型定义

### 4.1.1 简单类型定义模式的组成

|  |  |
| --- | --- |
| 组件 | 说明 |
| annotations | 注释 |
| name | xs:NCName类型值，可选 |
| target namespace | xs:anyURL类型值，可选 |
| final | {restriction,extension,list,union}的子集，表示样式不可以哪些方式继承 |
| context | 与name互斥，必须且只能出现两者之一；用于匿名类型 |
| base type definition | 简单类型的基类必须也是简单类型——例外：anySimpleType的基类是anyType |
| facets | 一组约束面 |
| fundamental facets | 一组基础约束面 |
| variety | {atomic,list,union}之一，必须；例外：对anySimpleType为absent |
| primitive type definition | 原始类型，对于所有variety值不为atomic的类型都是必须的，否则为absent；例外：对anyAtomicType为absent。除absent外，原始类型必须为原始内置类型（primitive built in definition）之一。 |
| item type definition | 用于variety为list的简单类型定义，否则为absent。取值范围：原始类型，variety为atomic的简单类型，variety为union且所有基类均为atomic的联合类型。 |
| member type definitions | 用于variety为union的简单类型定义，否则为absent。为一个原始类型或普通简单类型列表，不允许包含任何特殊类型（special type）。 |

### 4.3 Constraining Facets

约束Facet指在继承时可以设置或更改（用于控制派生类型的各个方面）的模式组件。

本规范刚约束Facet分为三类：

* 用于格式化初始的字面量——在检查它是否是一个类型的词法空间成员之前，称为pre-lixical Facet。仅包含WhiteSpace。
* 用于约束类型的词法空间，称为Lexical Facet。仅包括Pattern。
* 用于约束类型的值空间，称为Value-based Facet。

验证器必须支持本部分定义的所有Facet，另外可以自由决定支持其它的约束Facet，不支持的Facet可视为unknow（行为不确定）。

### 4.3.1 长度系列（Length，minLength，maxLength）

取值为正整数，定义限制的长度值，长度的单位：

* string及其派生类型、anyUrl：以字符为单位；
* hexBinary、base64Binary及其派生类型：二制制数据长度（以字节为单位）；
* list：列表中的项数。

验证规则：数据长度分别等于、大于等于、小于等于指定值时验证通过。

限制：

* 此三者不应相互冲突；
* minLength的值不能小于父类已定义的minLength值；
* maxLength的值不能大于父类已定义的maxLength值；
* 如果父类定义了length，则当前指定的length值只能与之相等。

注1：length上的fixed标记只是出于与其它Facet保持一致以及与1.0版规范兼容，实际上Length是无法改变的，无论其Fixed值为何。

注2：QName、NOTATION及其派生类上的此类 Facet已被弃用，在规范的后续版本中可能将被移除。

### 4.3.2 pattern

值为一组正则表达式。当一个restriction元素中包含多个pattern子元素时，它们之间为“或”关系；但是，它们与类层次中的父类或派生类中定义的pattern属于“与”关系。

限制：It is an [·error·](http://www.w3.org/TR/xmlschema11-2/#dt-error) if there is any member of the [{value}](http://www.w3.org/TR/xmlschema11-2/#f-p-value) of the [pattern](http://www.w3.org/TR/xmlschema11-2/#f-p) facet on the [{base type definition}](http://www.w3.org/TR/xmlschema11-2/#std-base_type_definition) which is not also a member of the [{value}](http://www.w3.org/TR/xmlschema11-2/#f-p-value).

### 4.3.3 enumeration

将值空间约束为一组指定值。enumeration在其值空间上未定义顺序关系，其中值的order属性与基类保持一致。

限制：派生类定义的枚举值必须属于父类的值空间。

### 4.3.4 whiteSpace

可选值：

preserve：保持不变；

replace：所有#x9(tab)、#xA(line feed)、#xD(carriage return)替换为#x20(space)；

collapse：先进行replace，然后将连续的#x20替换为单个#x20，并移除头尾的#x20。

whiteSpace应用于所有原子类型和list类型。对除string外的所有原子类型（包括其派生类），值固定为collapse，不允许更改；对string类型，值为preserve，但可在派生类中更改；对list，值固定为collapse，不允许更改。whiteSpace不直接应用于union，但当值被成功验证为某个成员类型时，将应用该类型的whilteSpace策略。

限制：当父类定义为collapse时，派生类不可以定义为replace或preserve；当父类定义为replace时，派生类不可以定义为preserve。

空注：可见，即使是whiteSpace这样的约束，它也不能从“覆盖基类的定义”这样的角度来理解，它同样必须遵循值空间收缩的原则。

### 4.3.5 值区间相关（maxInclusive、minInclusive、maxExclusive、minExclusive）

用于限定一个有序的值空间到其一个子集。此类Facet的取值必须为基类值空间中的值。

maxInclusive/minInclusive和maxExclusive/minExlusive均用于将值域限定为一个区间，Inclusive与Exclusive的差别在一个在于创建闭区间，一个创建开区间。

max与min不必成对出现。

空注：Exclusive不是指排除一个空间，若要定义一个值域——从基类的值域中排除一个区域，只能使用Union实现。

限制：不允许规定无效的区域（不包含任何值）；定义的区间不允许超出父类中相应Facet定义的区间。

### 4.3.6 十进制值位数相关（totalDigits与fractionDigits）

约束值空间，用于限定Decimal及其派生类的算术精度及量级，分别用于限定十进制数值的总位数及小数位数。前者取值为正数，后者取值为非负数。

当totalDigits值为t时，所限定的值空间中的值满足：i/(10 ^ n)，其中|i| < (10 ^ t), 0 <= n <= t。

当fractionDigits值为f时，所限定的值空间中的值满足：i/(10 ^ n)，其中i为任意整数，0 <= n <= f。

totalDigits所限定的位数可以是十进制计数法的总位数，或科学计数法的系数位数。totalDigits不直接约束词法空间，加上前置符号和未尾零位后总位数超出指定值的词法表示不违反totalDigits。totalDigits也不影响NaN、INF、-INF。

fractionDigits所限定的位数是值的十进制计数法中，小数点右边可以允许的最大位数。它同样不直接约束词法空间。

限制：指定的值不允许大于基类指定的相应值，totalDigits的值必须大于等于fractionDigits的值。

### 4.3.7 Assertions

限定值空间中的值必须满足指定的XPath表达式，可由一组Assertion组件构成。

### 4.3.8 explicitTimezone

用于要求或禁止对date、time类型的时区偏移。可选值为required、prohibited和optional。

限制：如果在基类中已经指定了非optional的值，则在派生类中不允许更改。