

ICS 75 - 010

E 07

备案号: 29445—2010



# 中华人民共和国石油天然气行业标准

SY/T 6782—2010

---

## 石油行业 XML 应用指南

Guide of XML application for petroleum industry

2010—05—01 发布

2010—10—01 实施

---

国家能源局 发布

## 目 次

|                                       |    |
|---------------------------------------|----|
| 前言 .....                              | II |
| 1 范围 .....                            | 1  |
| 2 规范性引用文件 .....                       | 1  |
| 3 术语和定义 .....                         | 1  |
| 4 XML 语法 .....                        | 2  |
| 4.1 XML 文档结构 .....                    | 2  |
| 4.2 XML 文档的基本语法 .....                 | 4  |
| 5 文档类型定义 .....                        | 7  |
| 5.1 DTD 概述 .....                      | 7  |
| 5.2 DTD 元素声明 .....                    | 8  |
| 5.3 DTD 属性声明 .....                    | 9  |
| 6 XML Schema .....                    | 10 |
| 6.1 导言 .....                          | 10 |
| 6.2 类型定义 .....                        | 10 |
| 6.3 元素声明 .....                        | 11 |
| 6.4 属性声明 .....                        | 12 |
| 6.5 模式注释 .....                        | 12 |
| 附录 A (资料性附录) 标记语言 .....               | 14 |
| 附录 B (资料性附录) 基于 XML 数据交换原理 .....      | 16 |
| 附录 C (资料性附录) XML 模式与关系数据库模式映射方法 ..... | 17 |
| 附录 D (资料性附录) 采出井日数据模型映射 .....         | 21 |

## 前 言

随着石油工业信息化的逐步深入，从企业集成的角度来看，企业已由最初的数据集成向专业集成再到部门集成直至最后到企业集成方向发展。从这个发展趋势可以看出，无论是专业内部集成还是部门间的集成或者是企业间的集成都离不开数据交换。对于石油工业来讲，如何对所交换的数据有规范可循是一个很重要的问题。近几年，随着 W3C 组织 XML 系列规范发布与在各领域中的应用，作为一种通用的置标语言很快得到各个领域的重视，这些组织纷纷基于 XML 置标语言来构建企业数据交换标准。像世界著名的石油开放软件联盟（POSC）近些年也发布了基于 XML 的系列规范，其中的通用井场信息传输规范（WITSML）、生产优化数据交换规范（PRODML）等就是典型的规范。制定这些规范的目的是将钻井信息、完井信息以及生产优化信息在应用之间实现实时交换。可以看出，XML 置标语言必将作为石油行业未来实现数据交换的一个重要规范，因此提出本标准。

本标准的附录 A、附录 B、附录 C 和附录 D 是资料性附录。

本标准由石油信息与计算机应用专业标准化技术委员会提出并归口。

本标准负责起草单位：中国石油勘探开发研究院、大庆石油学院。

本标准主要起草人：袁满、高雪、李鹏飞。

# 石油行业 XML 应用指南

## 1 范围

本标准规定了构建 XML 文档的一些概念、规则与方法。  
本标准适用于石油行业各级信息系统数据集成与数据交换。

## 2 规范性引用文件

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

Extensible Markup Language (XML) 1.0 (fourth edition)

W3C Recommendation 16 august 2006: <http://www.w3.org/TR/xml/> 规范

W3C XML Specification DTD: <http://www.w3.org/TR/xhtml1/DTDs.html> 规范

井场信息传输规范 WITS: *Wellsite information transfer specification*

生产数据交换标准 *Production XML standard*: (<http://www.prodml.org>)

XMLSchema 推荐标准: <http://www.w3.org/TR/xmlschema-0/>

<http://www.w3.org/TR/xmlschema-1/>

<http://www.w3.org/TR/xmlschema-2/>

## 3 术语和定义

下列术语和定义适用于本标准。

### 3.1

**数据对象 data object**

一个文档。

### 3.2

**解析实体 parse entity**

由 XML 文字（字符数据、卷标或两者皆有）所组成的实体。

### 3.3

**模式 schema**

一个基于 XML 的语法或 Schema 规范，负责定义和描述 XML 文档的结构和内容模式。它可以定义 XML 文档中存在哪些元素和元素之间的关系，并且可以定义元素和属性的数据类型。

### 3.4

**标记 tag**

用于表示元素的记号。

### 3.5

**元素 element**

XML 文档内容的基本单元。从语法上讲，一个元素由一个开始标记、一个结尾标记以及括在标记之间的文本字符数据组成。

### 3.6

**文档实体 document entity**

也称为文档体，是指文档实例的集合。

### 3.7

**标头 prolog**

XML 或 XML Schema 的前导说明部分。

### 3.8

**名称空间 name space**

W3C 推荐标准提供的一种统一命名 XML 文档中的元素和属性的机制。使用名称空间可以明确标识和组合 XML 文档中来自不同标记词汇表的元素和属性，避免了名称之间的冲突。

### 3.9

**文档类型定义 document type definition (DTD)**

一套关于标记符的语法规则。

### 3.10

**XML 解析器 XML parse**

一个可以读入一个文档并分析其结构的软件包。

### 3.11

**样式表 cascading style sheet**

用于（增强）控制网页样式并允许将样式信息与网页内容分离的一种标记性语言。

### 3.12

**数据交换 data exchange**

一个或多个计算机软件系统之间传递数据的过程。

## 4 XML 语法

一个数据对象（指文档）要成为规范的 XML 文档必须满足两个约束，即规范和有效性，规范应满足三个条件：

- 整体上是一个标记文档，即文档中包含一个以上元素，并且根元素必须包含其他元素。
- 满足 [www.w3.org/TR/REC-XML](http://www.w3.org/TR/REC-XML) 中的所有规范约束。
- 文档中直接或间接应用的解析实体都是规范的。

有效性应满足两个条件：

- 文档和一个文档类型定义 [document type definition (DTD)] 或者模式 (Schema) 相关联。
- 文档遵守其中定义的规则。

### 4.1 XML 文档结构

XML 文档语法比较简单，XML 文档由许多不同作用的标记构成。有些标记起声明作用，有些起注释作用，有些起标记数据的作用。

#### 4.1.1 XML 的逻辑结构

一个 XML 文档由五个部分组成：第一部分是 XML 必要声明部分，第二部分是 DTD 声明（如果存在），第三部分是处理指令，第四部分是文档实体，最后一部分是树状结构。

##### 4.1.1.1 XML 必要声明部分

该部分也称为 XML 标头信息 (prolog)，它是 XML 规范中规定的每一份 XML 文档中所必须的一项声明，并且必须放在 XML 文档的第一行。

示例：

```
<? XML version = " 1.0" encoding = " gb2312" standalone = " yes" >
```

#### 4.1.1.2 DTD 声明

这部分主要用于验证 XML 文档的合法性。在 DTD 中，对 XML 文档中出现的每一个标记与其属性加以定义，明确文档中标记之间的关系、标记数据类型以及标记数量等。有两种 DTD，一种是内部 DTD，另一种是外部 DTD，下面给出内部 DTD 示例。

示例：

```
<? XML version = " 1.0" encoding = " gb2312" standalone = " yes" >
<! DOCTYPE DOCUMENT [
.....
] >
```

#### 4.1.1.3 处理指令部分

该部分是 XML 处理程序必须扫描的指令，被称为处理指令。处理指令含有应用程序所使用的标记信息，处理指令格式以“<”开始，并以“? >”结束。

示例：

```
<? XML stylesheet type = " text/xsl" href = " mystyle. xsl"? >
```

该示例表示用样式表文件 mystyle. xsl 来显示 XML 文档，其中 type 属性用于选择样式，而 href 属性表示样式表文件的路径。

XML 必要声明部分、DTD 声明部分和处理指令部分统称为文档头部。

#### 4.1.1.4 文档实体部分

文档实体也称文档体，它实际上是文档实例的集合。这是 XML 文档中真正存放数据的位置，它是 XML 文档的数据部分。文档实体是 XML 文档的主体，在应用之间传输 XML 文档的目的就是要交换这部分数据。

#### 4.1.1.5 树状结构部分

逻辑结构是指概念上的 XML 文档，可以将 XML 文档看作是由包含大量信息的数据按照层次关系组织起来的结构。其中的数据可能会作为元素或者属性出现在 XML 文档中，形成一个树状结构，其中树状结构的顶端是顶层元素，也称根元素。

### 4.1.2 XML 的物理结构

就物理结构而言，可以将 XML 文档看成是由许多独立的物理文件组成，这些文件在 XML 中被称为“实体”，因此，一个 XML 文档实际上是由一组“实体”构成。

#### 4.1.2.1 XML 实体

和逻辑结构中的根元素类似，在物理结构中所有的 XML 文档也存在一个“文档实体”或称根实体，这个实体封装了整个 XML 文档。也就是说，这个由 XML 本身给出的根实体指出整个 XML 文档的内容。在这个结构中，一个实体可以包含对其他实体的引用，而被引用的实体又可以包含对另外一些实体的引用。

在 DTD 中定义的实体包含两种类型：通用实体和参数实体。通用实体即是在 XML 文档中用于存储文本数据的实体。

它在 DTD 中定义的格式为：

```
<! ENTITY entity _ name SYSTEM file _ path >
```

实体名由用户来命名，它表示要替换的文本数据的名称，而文本内容则是用户所指定的实体名要替换的具体文本数据。如果是对外部文件的引用，那么该外部文件的路径由参数 file \_ path 指定为外部文件的磁盘路径，或者是该文件的 URL。

参数实体定义格式为：

```
<! ENTITY %entity _ name " text _ content" >
```

#### 4.1.2.2 实体的使用方法

实体是一个被命名的标记数据块，它可以是一个数据串，也可以是一个完整的文件。实体可以包含已解析的数据或未解析的数据。已解析的数据由字符组成，其中一些字符组成文本数据，另一些字符组成标记。未解析数据则是那些不进行语法解析的数据。

实体使用包含两部分：实体声明和实体引用。实体声明在 DTD 中完成，实体引用则是引用一个已在实体声明中声明过的实体，其引用形式为 &entity\_name。

```

    示例：
<? XML version = " 1.0" encoding = " UTF - 8" standalone = " yes" >
<! DOCTYPE DOCUMENT [
! ELEMENT DOCUMENT (wellNotice)
<! ELEMENT wellNotice (name, kind, permitID, refilling, permitAssociate, wellIdentification,
wellConstruction, noticeAttachment) >
.....
<! ENTITY OPEN _ CLOSE " active" >
] >
<DOCUMENT>
    <wellIdentification>
        <name>05 - 24A</name>
        <numAPI>50 - 029 - 22204 - 01 - 00</numAPI>
        <kind>development</kind>
        <purpose>oil</purpose>
        <originalWellPurpose>oil</originalWellPurpose>
        <wellStatus>&OPEN _ CLOSE; </wellStatus>
        <fieldName>ADL - 028325</fieldName>
    </wellIdentification>
</DOCUMENT>
```

解析器在对 XML 文档解析时，就会直接将 “&OPEN \_ CLOSE;” 替换成 “active”。

4.1.2.3 XML 中定义的实体引用

XML 已经预先定义了 5 个实体引用，用来代表在 XML 文档字符数据中出现的特定符号，这些实体引用见表 1。

表 1 XML 文档字符数据中使用的特定符号

| 实体引用   | 所代表的字符 |
|--------|--------|
| &amp;  | &      |
| &lt;   | <      |
| &gt;   | >      |
| &apos; | '      |
| &quot  | "      |

4.2 XML 文档的基本语法

XML 文档由字符数据和标记组成，其编写格式类似于 HTML，但是语法要求比 HTML 更为严格。

4.2.1 XML 元素

4.2.1.1 标记

XML 文档中基本的数据单位就是标记 (tag), XML 中标记可分为开始标记、结尾标记和空标记。标记在 XML 文档中以 “<” 符号开始, 以 “>” 符号结束, “<” 和 “>” 符号称为分隔符, 用于将标记从文档的字符数据中区分出来。开始标记以 “<” 符号开始, 后面跟有标记名, 其语法格式为:

```
<tag _ name attribute _ name = " attribute _ value" >
```

各参数说明如下:

tag \_ name: 标记名, 即为 XML 元素名, 如果标记名是英文, 必须注意区分大小写。

attribute \_ name: 属性名称。

attribute \_ value: 属性值, 需要用单引号或双引号包起来, 若有多个属性, 使用空格分隔。

结尾标记以 “</” 符号开始, 后面也跟标记名, 遇到第一个 “>” 符号标记结束。其语法格式为:

```
</tag _ name>
```

tag \_ name 为结尾标记名称, 它需要和开始标记相同, 而且要和开始标记成对出现。

示例:

```
<wellIdentification>
<name>05 - 24A </name>
<numAPI>50 - 029 - 22204 - 01 - 00</numAPI>
<kind> development </kind>
<purpose>oil</purpose>
<originalWellPurpose>oil</originalWellPurpose>
<wellStatus>active</wellStatus>
<fieldName> ADL - 028325</fieldName>
</wellIdentification>
```

#### 4.2.1.2 XML 元素

元素 (element) 是 XML 文档内容的基本单元。从语法上讲, 一个元素由一个开始标记、一个结束标记以及在标记间的文档字符数据组成。其语法格式为:

```
<tag _ name>data _ content</tag _ name>
```

XML 标记只是 XML 元素的一部分, data \_ content 是元素的内容, 它可以是字符数据, 也可以包含其他 XML 元素。data \_ content 可以没有内容, 这类元素称为空元素。其语法格式为:

```
<tag _ name/>或
```

```
<tag _ name></tag _ name>
```

在创建元素时还要遵循如下基本规则, 只有符合这些规则的元素才是符合 XML 语法的元素, 规则如下:

- a) 一份 XML 文档中至少要有一个元素。
- b) 一份 XML 文档中有且只有一个根元素。由于 XML 文档本身的结构是树状的, 所以一棵树只能有一个根元素, 其他元素都属于该根元素的子元素。

示例:

```
<wellIdentification>
<name>05 - 24A </name>
<numAPI>50 - 029 - 22204 - 01 - 00</numAPI>
<kind>development</kind>
<purpose>oil</purpose>
<originalWellPurpose>oil</originalWellPurpose>
```

```
<wellStatus>active</wellStatus>
<fieldName> ADL - 028325</fieldName>
</wellIdentification>
```

此处，wellIdentification 是根元素，name，numAPI，kind，purpose，originalWellPurpose，wellStatus 以及 fieldName 均是 wellIdentification 的子元素。

#### c) XML 命名规则：

- 1) XML 元素命名可使用英文大小写字符（a~z，A~Z）、数字（0~9）、下划线字符、句点字符以及短横线字符，整个名称中不能有空格。
- 2) 元素名称不能以数字字符开头。
- 3) XML 区分大小写。

#### 4.2.1.3 XML 置标规则

XML 置标规则如下：

- 标记不可缺少。
- XML 文档中至少要有有一个元素。
- 标记区分大小写。
- 标记必须严格配对，所有元素要有结尾标记。
- 标记要正确嵌套，标记之间不能交叉。

#### 4.2.2 XML 属性

XML 属性是对标记的进一步描述和说明，属性的目的是用于在元素中提供额外的信息。

- a) XML 元素可以有属性，元素的每个属性是一个名称—数值对。
- b) XML 元素拥有的属性可以为一个到多个，属性的名称不能重名。需要注意的是元素内可以嵌套子元素，属性则不允许。

XML 中属性设置的语法格式为：

```
<tag _ name attribute _ name1 = " attribute _ value1 attribute _ name1 = " attribute _ value1" >
element _ content
</tag _ name>
```

示例：

```
<wellNotice uid = " 304 - 088" >
<name>Application for Sundry Approval</name>
<kind>amend</kind>
<subKind>abandon</subKind>
<permitID>Form 10 - 403</permitID>
.....
</wellNotice>
```

#### 4.2.3 XML 语法的其他内容

##### 4.2.3.1 CDATA

XML 解析器对字符“<”和“&”非常敏感，如果有一大段文本，其中有很多“<”和“&”，但又不用做标记，可用 CDATA 标记标出，这样 XML 解析器就可不对这段文本解析。其语法格式为：

```
<![CDATA [
There are some characters that the parser do not parse them in CDATA tag
]] >
```

##### 4.2.3.2 XML 注释

注释为文件和数据提供说明，注释中的内容被 XML 解析器所忽略。XML 文档注释以 `<!-- comment _ content -->` 来标识。注释需要注意以下几点：

- 注释语句不能出现在 XML 声明之前。
- 注释不允许出现在标记中，而是单独的语句。
- 注释不能嵌套使用。

## 5 文档类型定义

### 5.1 DTD 概述

使用 XML 结构化的组织数据，只保证良好的格式是不够的，必须保证文档的有效性。文档类型定义 [document type definitions (DTD)] 就是为保证 XML 文档的有效性而引入的约束机制。

DTD 文件的声明语法不同于 XML 的语法，DTD 文件开头必须使用专用的关键字以告诉解析器这个区段属于 DTD 声明。其声明语法为：

```
<! DOCTYPE root _ node [  
<! DTD 定义的内容>  
>
```

root \_ node 表示与 XML 文档中对应的根元素名称。DTD 定义的内容包括：

- DTD 元素声明。
- DTD 属性声明。
- DTD 实体声明。
- DTD 标记声明。

XML 中使用的 DTD 有三种方式：一是直接将 DTD 插放在 XML 文件中与 XML 元素放在一起；二是使用外部独立的 DTD 文件；三是使用混合的方式，XML 文档同时使用上述两种方式，用以建立更复杂的 DTD。

#### 5.1.1 内部 DTD

DTD 指令直接写在 XML 文档中，其所定义的约束范围只能应用于此 XML 文档。内部 DTD 语法如下：

```
<! DOCTYPE root _ node [  
.....  
>
```

root \_ node 为 XML 根元素名称。

示例：

```
<? XML version = " 1.0" encoding = " UTF-8" standalone = " yes"? >  
<! DOCTYPE DOCUMENT [  
<! ELEMENT DOCUMENT (wellNotice)  
<! ELEMENT wellNotice (name, kind, permitID, refilling, permitAssociate, wellIdentification,  
wellConstruction, noticeAttachment) >  
<! ELEMENT name (#PCDATA) >  
<! ELEMENT kind (#PCDATA) >  
<! ELEMENT permitID (#PCDATA) >  
.....  
>  
<DOCUMENT>  
<wellNotice>
```

```

<name>Report of Sundry Well Operations</name>
<kind>unknown</kind>
<permitID>Form 10 - 457</permitID>
.....
</wellNotice>
</DOCUMENT>

```

### 5.1.2 外部 DTD

内部 DTD 是将 DTD 指令直接插入到 XML 文档中。另一种方法是将 DTD 的指令独立地存储成一个文件，然后在 XML 文档中指定所使用的 DTD 文件，这就是外部 DTD。

引用外部 DTD 文件时，要在 XML 文档的前导区声明所要使用的 DTD 文件。外部 DTD 文件在 XML 文档中引用的语法格式为：

```
<! DOCTYPE root _ node SYSTEM " filename. dtd" >
```

参数说明如下：

SYSTEM：是 XML 保留字，代表所引用的外部 DTD 文件是由个人创建，而不是国际标准。

root \_ name：XML 文档的根标记。

filename. dtd：外部 DTD 文件的路径。

除了使用 SYSTEM 外，还可以使用 PUBLIC 关键字，它代表所引用的 DTD 是国际标准的，其引用语法如下：

```
<! DOCTYPE root _ node PUBLIC " + | - //组织名称//所引用 DTD 文件的描述//编写 DTD
所使用的语系" " URL" >
```

其中，“+”表示该组织获得 ISO 的认证，“-”表示该组织没有获得 ISO 的认证。

例如：

```

<! DOCTYPE DOCUMENT PUBLIC " - //CNPC//Custom XML Version 1.0//CN"
" http: //www. cnpc. com. cn/drillng/ welloperation/>

```

## 5.2 DTD 元素声明

DTD 元素（element）声明是声明 XML 元素的语法，包含元素的标记、内含的子元素和元素内容数据，它同时也声明了 XML 文件的元素架构。

### 5.2.1 DTD 元素声明的语法

XML 文档按照树状结构进行数据的组织，在树状结构中，除末端的叶子元素之外，中间是由不断分支的枝干元素所组织，所以 DTD 中元素声明对应为叶子元素声明和枝干元素声明。

#### 5.2.1.1 叶子元素声明

语法格式为：

```

<! ELEMENT element _ name (data _ type) >
element _ name：XML 文档中对应元素名称。
data _ type：元素数据内容的类型。

```

#### 5.2.1.2 枝干元素声明

语法格式为：

```

<! ELEMENT element _ name (child _ element1, child _ element2, ...) >
<! ELEMENT child _ element1 (data _ type) >
<! ELEMENT child _ element2 (data _ type) >

```

其中：

element \_ name：XML 文档中对应元素名称。

child \_ element1, child \_ element2：element \_ name 元素的子元素。

data \_ type: 元素数据内容的类型, 在 5.2.2.2 中说明。

示例:

```
<? XML version = " 1.0" encoding = " UTF - 8" standalone = " yes" >
<! DOCTYPE DOCUMENT [
<! ELEMENT DOCUMENT (wellNotice)
<! ELEMENT wellNotice (name, kind, permitID, refilling, permitAssociate, wellIdentification,
wellConstruction, noticeAttachment) >
<! ELEMENT name (#PCDATA) >
<! ELEMENT kind (#PCDATA) >
<! ELEMENT permitID (#PCDATA) >
<! ELEMENT refilling (#PCDATA) >
.....
```

5.2.1.3 控制子元素出现的次数

当相同子元素重复出现若干次时, 在 DTD 中需要使用特定的符号来声明子元素出现的次数, 见表 2。

表 2 控制子元素出现次数的符号表

| 所用符号 | 含 义                 |
|------|---------------------|
| +    | 一个元素可出现 1 次或 1 次以上  |
| *    | 一个元素可出现 0 次或 0 次以上  |
| ?    | 一个元素可出现 0 次或 1 次    |
|      | 只能出现符号作用范围的任一元素 1 次 |
| ( )  | 将括号内的数据或元素合并为一个单元   |

5.2.2 DTD 元素的数据

如果元素没有子元素, 此时 DTD 定义的就是元素内标记包围的数据内容。

5.2.2.1 空元素 (EMPTY)

空元素是指在标记间没有任何数据, 这时该元素必须使用 EMPTY 关键字:

```
<! ELEMENT kind (EMPTY) >
```

以上定义的 kind 元素就是一个空元素, 在 XML 文档中元素对应为:

```
<kind/>
```

5.2.2.2 元素数据

如果元素内有数据, 此时数据内容有两种, 可以使用 #PCDATA 和 ANY 来定义。语法格式:

```
<! ELEMENT element _ name (#PCDATA) >
```

```
<! ELEMENT element _ name (ANY) >
```

示例:

```
<! ELEMENT permitID (#PCDATA) >
```

```
<! ELEMENT permitID (ANY) >
```

5.3 DTD 属性声明

XML 文档中元素通常具有多个属性, 和元素必须要在 DTD 中声明一样, 元素的属性也需要在 DTD 中进行声明。

语法格式如下:

```
<! ATTLIST element _ name attribute _ name attribute _ type default _ value>
```

参数说明如下：  
element\_name: 属性所属的 XML 元素名称。  
attribute\_name: 属性名称。  
attribute\_type: 属性值的类型，如表 3 所示。  
default\_value: 属性的默认类型。

表 3 DTD 中属性值类型

| 属性值类型      | 说 明                                         |
|------------|---------------------------------------------|
| CDATA      | 文本数据字符串，解析器不会进一步解析是否有元素，XML 保留字和大部分属性使用这种类型 |
| ID         | 唯一，不能被文档中其他任何 ID 属性值共享的数字，需要字母开头            |
| IDREF      | 其他元素的 ID 属性值                                |
| IDREFS     | 其他元素的 ID 属性值列表，使用空格分隔                       |
| ENTITY     | 在 DTD 中声明的实体                                |
| ENTITIES   | DTD 中声明的多个实体，使用空格分隔                         |
| NMTOKEN    | XML 名称，包含字母、数字、[·]，[-]，[_] 和 [:] 组合的名称      |
| NMTOKENS   | 使用空格分隔的 NMTOKEN                             |
| NOTATION   | 在 DTD 中声明的注释名                               |
| Enumerated | 枚举类型的属性，其值只能从已声明的选项中选取，不能填入新项目              |

6 XML Schema

一个 schema 由导言、若干个类型定义和元素声明组成。

6.1 导言

导言放在根元素里面，根元素的名字必须为“schema”。

示例：

```
<xsd: schema targetNamespace=" http: //www. witsml. org/schemas/120"
  xmlns: xsd=" http: //www. w3. org/2001/XMLSchema"
  xmlns: witsml=" http: //www. witsml. org/schemas/120"
  xmlns=" http: //www. witsml. org/schemas/120" >
```

参数说明如下：

xmlns: xsd: 声明命名空间，前缀为 xsd，xsd 前缀是 W3C 模式元素前缀。  
targetNamespace: 指定模式的目标名称空间。  
xmlns: witsml: 声明命名空间，前缀为 witsml。  
xmlns: 指明缺省的命名空间，即元素、属性等名称前无前缀时，使用 xmlns 所指定的命名空间。

6.2 类型定义

包括简单类型创建和复杂类型创建两种。

6.2.1 简单类型定义

当 XML 中内建的简单类型不能满足应用的需要时，可以自己定义简单类型。

示例：

```
<xsd: simpleType name=" catalogID" base=" xsd: string" >
  <xsd: pattern value=" \d {3} - \d {4} - \d {3}" />
```

</xsd:simpleType>

参数说明如下：

name：简单类型名称。

base：创建的简单类型所基于 XML 内建的简单类型。

pattern：指定正规表达式，使创建的简单类型值符合指定的格式。

使用平面创建简单类型：使用平面（faces）可以对简单类型存储的数据施加限制。

a) 限制简单类型存储数据的范围（minInclusive 和 maxInclusive）。

示例：

```
<xsd:simpleType name=" dayOfMonth" base=" xsd: integer" >
  <xsd: minInclusive value=" 1" />
  <xsd: maxInclusive value=" 31" />
```

</xsd:simpleType>

b) 指定正规表达式（pattern）。

示例：见简单类型定义。

c) 枚举值列表（enumeration）。

示例：

```
<xsd:simpleType name=" weekday" base=" xsd: string" >
  <xsd: enumeration value=" Sunday" />
  <xsd: enumeration value=" Monday" />
  .....
  <xsd: enumeration value=" Saturday" />
```

### 6.2.2 复合类型创建

简单类型的元素里面只有内容，不再包括属性或者其他元素。而当元素里面包含属性和其他元素时，就为复合类型。复合类型定义本身通常包含元素声明、对其他元素引用和属性声明。

示例：

```
<xsd:complexType name=" address" >
  <xsd:element name=" name" type=" xsd: string" />
  <xsd:element name=" street" type=" xsd: string" />
  <xsd:element name=" city" type=" xsd: string" />
  <xsd:element name=" province" type=" xsd: string" />
  <xsd:element ref=" note" minOccurs=" 0" />
  <xsd:attribute name=" phone" type=" xsd: string" use=" optional" />
</xsd:complexType>
```

### 6.2.3 匿名类型定义

当一个类型只使用一次时，可以定义匿名类型。匿名类型可以是简单匿名类型和复合匿名类型。

示例：

```
<xsd:simpleType base=" xsd: integer" >
  <xsd: maxExclusive value=" 20" />
</xsd:simpleType>
```

## 6.3 元素声明

### 6.3.1 声明元素

语法格式：

```
<xsd:element name=" element_name" type=" type_name" />
```

示例：

```
<xsd: element name = " commonData" type = " witsml: cs _ commonData" />
```

### 6.3.2 声明空元素

不包含任何内容，可以包含属性。

示例：

```
<xsd: element name = " image" >
  <xsd: complexType content = " empty" >
    <xsd: attribute name = " source" type = " xsd: string" />
    <xsd: attribute name = " width" type = " xsd: float" />
  <xsd: attribute name = " height" type = " xsd: float" />
  </xsd: complexType>
</xsd: element>
```

### 6.3.3 指定元素出现的次数

示例：

```
<xsd: element name = " commonData" type = " witsml: cs _ commonData" minOccurs = " 0" max-Occurs = " 1" />
```

参数说明如下：

minOccurs：指定元素出现的最小次数。

maxOccurs：指定元素出现的最大次数。

### 6.3.4 指定元素的默认值

示例：

```
<xsd: element name = " commonData" type = " xsd: integer" fixed = " 100" />
```

## 6.4 属性声明

### 6.4.1 属性声明

语法格式：

```
<xsd: attribute name = " attribute _ name" type = " type _ name" />
```

示例：

```
<xsd: attribute name = " version" type = " xsd: string" />
```

### 6.4.2 指定属性的约束和缺省值

语法格式：

```
<xsd: attribute name = " attribute _ name" type = " type _ name" use = " use _ value" />
```

其中

use \_ value = optional：可选的。

use \_ value = required：必须的。

use \_ value = fixed：属性固定，可用 value 属性设置。

use \_ value = default：如果属性未出现，使用 value 属性设置的缺省值；如果属性出现，它的值就是文档中赋给它的值。

use \_ value = prohibited：属性禁止出现。

示例：

```
<xsd: attribute name = " version" type = " xsd: string" use = " optional" />
```

## 6.5 模式注释

XML 定义了三种新元素来为模式添加注释：<xsd: annotation>，<xsd: document>和<xsd: appInfo>。

示例:

```

<? xml version = " 1.0" encoding = " UTF - 8"? >
  <xsd: schema elementFormDefault = " qualified" attributeFormDefault = " unqualified"
targetNamespace = " http: //www. witsml. org/schemas/120"
xmlns: xsd = " http: //www. w3. org/2001/XMLSchema"
xmlns: witsml = " http: //www. witsml. org/schemas/120"
xmlns = " http: //www. witsml. org/schemas/120" >
  <xsd: include schemaLocation = " cs _ logHeader. xsd" />
  <xsd: include schemaLocation = " cs _ logData. xsd" />
  <xsd: include schemaLocation = " cs _ dataTypes. xsd" />
  <xsd: include schemaLocation = " cs _ catalog. xsd" />
  <xsd: include schemaLocation = " cs _ commonData. xsd" />
  <xsd: include schemaLocation = " cs _ customData. xsd" />
  <xsd: annotation>
    <xsd: documentation>
      Log Information that supports the WITSML standard.
    </xsd: documentation>
  </xsd: annotation>

```

附录 A  
(资料性附录)  
标记语言

标记语言 (markup language) 采用一系列约定的标记来对电子文档进行标记, 从而实现对电子文档的语义、结构及格式的定义。这些标记必须能够容易地与内容相区分, 并且易于识别。标记语言必须定义什么样的标记是允许的, 什么样的标记是必须的。标记是如何与文档的内容相区别的, 以及标记的含义是什么。例如, HTML (hypertext markup language, 超文本标记语言) 就是目前网络中网页文档格式定义的一种重要的标记语言, 它与 SGML 和 XML 都属于一个大家族, 标记语言家族见图 A. 1。

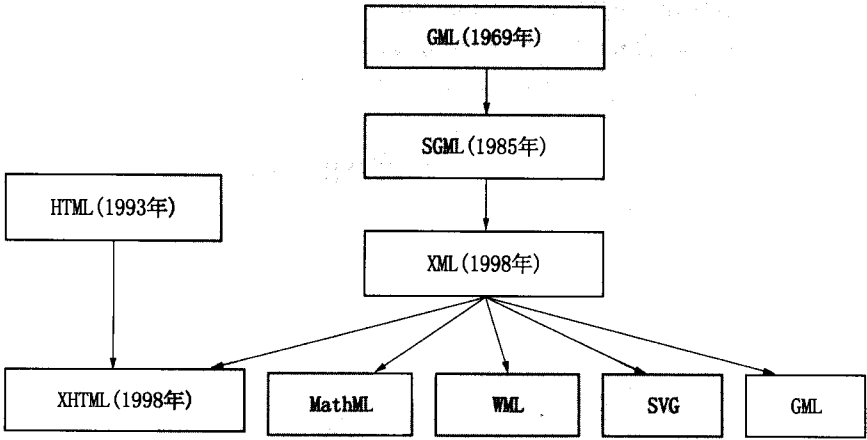


图 A. 1 标记语言家族

1996 年, 人们开始致力于描述一个新的标记语言, 它是一种在 Web 中应用 SGML 的灵活性和强大功能的方法, W3C (world wide web consortium) 专门成立专家小组以从事这项工作。1998 年 2 月, W3C 批准了 XML1.0 规范, 标志着一种崭新的、重要的语言 XML (extensible markup language, 可扩展的标记语言) 的产生。XML 具备 SGML 的核心特性之一。

A. 1 通用标记语言

IBM 公司的研究人员 Ed Mosher, Ray Lorie 和 Charles F. Goldfarb 在 20 世纪 60 年代开发了一种文档描述语言, 用来解决不同系统中文档格式不同的问题。IBM 把这种标识语言称为通用标记语言 (generalized markup language, GML), GML 是第一种现代标记语言, 它是一种自参考的语言, 可以用于标记任何数据集合的结构, 同时, 它也是一种元语言 (meta-language), 即能够描述其他语言及其语法和词汇表的语言。

A. 2 标准通用标记语言

标准通用标记语言 (standard generalized markup language, SGML), 是一种元语言 (meta-language), 也就是说, SGML 本身并不是一种标记语言, 而是定义了一种生成标记语言的方法, 是定义新的标记语言的基础。SGML 的实质是强调描述性标记, 引入文档类型定义 (DTD) 概念, 具有平台独立性, 所以, SGML 具有结构化、确认性和可扩展性三个特点, 而且可以跨平台。但是, 庞大、复杂且严格定义的规范说明令人对 SGML 望而却步, 这使得 SGML 的使用和推广受到很大

的限制。

### A.3 超文本标记语言

超文本标记语言 (hypertext markup language, HTML) 起源于标准通用标记语言, 是一种国际标准文本排版标记语言。HTML 是专门用于网络超文本描述的语言, 它最大的优势就是其格式简单, 但随着 WEB 应用的不断发展, HTML 的局限性也越来越明显地体现出来了, 尤其在以下四个方面:

- a) 扩展性。
- b) 数据结构性。
- c) 数据确认方面。
- d) 受限于浏览器。

### A.4 XML 技术背景

由于标准通用标记语言及超文本标记语言的缺点, 为适应互联网应用发展的需求, 特别是网络数据交互和业务集成的需求, 1996 年, 人们开始致力于描述一个新的标记语言, 它既具备了 SGML 的强大功能和可扩展性, 同时又具有 HTML 的简单性。W3C 专门成立了一个专家组从事此项工作, 至 1998 年 2 月, W3C 批准了 XML1.0 规范 V1 版本。

XML 即可扩展标记语言 (extensible markup Language), 是一种与平台无关的表示数据的方法。简单地说, 使用 XML 创建的数据可以被任何应用程序在任何平台上读取, 甚至可以通过手动编码来编辑和创建 XML 文档。基于 XML 的特点, 它主要应用在如下四个方面:

- a) 元数据内容定义。
- b) 数据交换。
- c) 制作多媒体文档。

### A.5 应用程序接口 DOM 和 SAX

在进行数据交换过程中, 如何访问 XML 文档中的内容, 这是开发者所关心的一个问题。W3C 为此制定了一套访问 XML 文档内容的标准接口规范, 即 DOM (document object model) 规范。

除了 W3C 制定的 DOM 接口规范外, XML\_DEV 邮件列表中的成员根据应用的需求也自发地定义了一套对 XML 文档进行操作的接口规范——SAX (simple APIs for XML)。在应用中可根据应用的需求来具体选取哪个作为开发接口, DOM 和 SAX 应用程序接口在程序开发中的地位参见图 A.2。

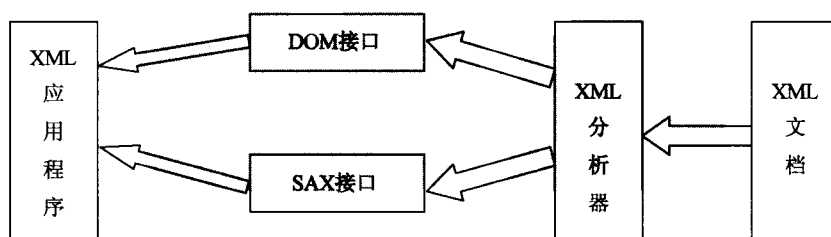


图 A.2 DOM 和 SAX 在应用程序开发过程中所处的地位

从图 A.2 中可以看出, 应用程序接口 DOM 和 SAX 都是由 XML 分析器提供的、对 XML 文档进行直接操作的接口标准。而应用程序不是直接对 XML 文档进行访问和操作的。如果应用程序要对 XML 文档进行访问和操作, 首先要通过 XML 分析器对 XML 文档进行分析, 然后, 应用程序通过 XML 分析器提供的 DOM 接口或 SAX 接口对分析结果操作, 从而间接地实现对 XML 文档的访问与操作。

## 附 录 B

### (资料性附录)

#### 基于 XML 数据交换原理

随着信息社会的快速发展以及互联网在企业中的深入应用，企业中的各种业务数据在不同的应用系统之间进行传递。对于数据交换的方法有很多，而 XML 正在逐渐成为互联网上数据表示和交换的标准。

基于 XML 的数据交换原理是使用 XML 来描述不同应用系统所传递的数据的结构（包括模式约束等）。XML 数据交换可以分为三层结构，即表现层（CSS，XSL，DOM，SAX）、数据组织层（DTD，Schema，XML 语法规则）以及数据交换层。

#### B.1 数据发布

数据发布是应用层叠样式表 CSS（cascading style sheets）和可扩展样式语言 XSL（extensible styles language）技术，通过给 XML 数据赋予一定的样式信息使得 XML 数据能够在浏览器上进行显示。基于 XML 的数据发布是服务器—浏览器模式的数据交换。

#### B.2 数据集成

数据集成是把不同来源、格式、特点性质的数据在逻辑上或物理上有机地集中，从而为企业提供全面的数据共享。在企业数据集成领域，已经有了很多成熟的框架可以利用。目前通常采用联邦式、基于中间件模型和数据仓库等方法来构造集成的系统，而基于 XML 的数据集成就是基于中间件模型的方式。这些技术在不同的着重点和应用上解决数据共享和为企业提供决策支持。

#### B.3 交易自动化

由于 XML 的平台无关性，并且 XML 文档适合于在互联网上进行数据传输，所以 XML 有助于提高应用的自动化程度。

## 附录 C

## (资料性附录)

## XML 模式与关系数据库模式映射方法

按照企业数据交换的作用范围来分,一种是企业内部的数据交换,它主要表现在企业内部不同应用系统之间的数据集成;一种是企业与外部商业伙伴之间的数据交换,这就是企业的电子商务交易。该数据交换标准是针对企业内部的数据集成。

由于企业的业务数据一般均存放在关系数据库中,为了实现企业内部的数据交换,必须以某种方法,在 XML 文档和关系数据库模式之间相互进行映射,并尽量保留原模式的语义。这样使 XML 成为异构应用之间数据交换的载体。

这种与数据库模式进行相互映射的 XML 文档一般来说是以数据为中心的 XML 文档。它将 XML 用作数据的传输载体,供机器读取。映射的方法有多种,本标准采用基于相关对象映射机制。

## C.1 XML 模式 (DTD) 到关系数据库模式的映射

相关对象的映射机制是,它将 XML 文档建模为一棵对象树,对象树中存放着 XML 文档数据,然后再将对象树映射到数据库中,或者反向映射。因此对象树是映射的中间纽带。首先明确两个概念:简单元素类型,只有 pcdat 内容的元素类型称为简单元素类型,这从 XML 模式 (XML schema) 中借鉴过来。简单元素类型存放一个单一的数据值,相当于面向对象编程语言中的标量数据类型。XML 中的属性类型 (attribute) 也是简单的类型。复杂数据类型,元素类型中还有元素、混合内容和属性称为复杂数据类型,这也是从 XML 模式中借鉴的。复杂类型相当于面向对象编程语言中的类。XML 模式到关系数据库模式映射分两步。

## a) DTD 到对象模式映射:

- 1) 将复杂数据类型映射为类。
- 2) 将简单元素类型映射为类的标量数据类型。
- 3) 将元素属性 (attribute) 映射为类的域 (property),类域的数据类型由元素属性类型决定。
- 4) 将子元素也映射为类的域。

示例:

```
<! ELEMENT pc_pro_well_vol_daily (prod_date, liq_prod_daily, oil_prod_daily, water_prod_daily, gas_prod_daily, cd_well_source *) >
<! ELEMENT prod_date (#PCDATA) >
<! ELEMENT liq_prod_daily (#PCDATA) >
<! ELEMENT oil_prod_daily (#PCDATA) >
<! ELEMENT water_prod_daily (#PCDATA) >
<! ELEMENT gas_prod_daily (#PCDATA) >
<! ELEMENT cd_well_source (well_id, org_id, well_common_name) >
<! ELEMENT well_id (#PCDATA) >
<! ELEMENT org_id (#PCDATA) >
<! ELEMENT well_common_name (#PCDATA) >
```

按照上述的映射规则把 DTD 模式映射为对象模式:

```
class pc_pro_well_vol_daily {
date prod_date;
float liq_prod_daily;
```

```

float oil _ prod _ daily;
float water _ prod _ daily;
CD _ WELL _ SOURCE cd _ well _ source;
}
Class CD _ WELL _ SOURCE {
String well _ id;
String org _ id;
String well _ common _ name;
}

```

b) 将对象模式映射为数据库模型式。

相关对象映射的第二步是将类映射为表，数量属性映射为列，指针/参考属性映射为关系中主键/外键。

|                                                                                                                                                                                              |   |                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> class pc _ pro _ well _ vol _ daily { date prod _ date float liq _ prod _ daily; float oil _ prod _ daily; float water _ prod _ daily; CD _ WELL _ SOURCE cd _ well _ source; } </pre> | ⇒ | <pre> table pc _ pro _ well _ vol _ daily: column prod _ date column liq _ prod _ daily column oil _ prod _ daily column water _ prod _ daily column well _ id _ fk </pre> |
| <pre> class cd _ well _ source { String well _ id; String org _ id; String well _ common _ name; } </pre>                                                                                    | ⇒ | <pre> table cd _ well _ source: column well _ id column org _ id column well _ common _ name column well _ id _ pk </pre>                                                  |

## C.2 关系数据库模式到 XML 模式 (DTD) 的映射

把要转换的所有数据库模式看成一片森林，每棵树都是由一个或多个相关的表构成，一棵树只有一个树根，把处于树根的表称为根表，其他的称为枝表。显然，根表一定要映射为复杂元素类型，枝表可以映射为简单或复杂元素类型，并成为其他复杂模型的引用。那么，分别对每一棵树进行处理，从根表开始，处理规则如下：

- a) 根表生成一个复杂元素类型，内容模型为空的序列。
- b) 每个数据列（不是键列）生成 PCDATA 类型的简单元素类型，并在序列中增加一个引用，可以为空的列生成可选的引用。
- c) 处理枝表，方法如根表。
- d) 在根表的内容模型中增加一个对枝表生成的元素类型的引用。
- e) 如果根表的键是主键，对枝表生成的元素类型的引用是可选和重复的 (\*)。
- f) 如果根表的键是主键且含有数据，为键中的每个列生成 PCDATA 的元素类型，并在根内容模型中增加引用，引用同样是可选和重复的 (\*)。
- g) 如果根表的键是外键且可以为空，引用可选 (?)。
- h) 枝表除了生成元素类型嵌套于复杂元素类型内，还可以生成 PCDATA 文本，作为复杂元素类型的 PCDATA 内容。

示例：

table cd\_well\_source  
 column well\_id (主键)  
 column org\_id  
 column well\_common\_name

table pc\_pro\_well\_vol\_daily  
 column prod\_date;  
 column liq\_prod\_daily;  
 column oil\_prod\_daily;  
 column water\_prod\_daily;  
 column well\_id (主外键)

为根表 pc\_pro\_well\_vol\_daily 的数据列 liq\_prod\_daily, oil\_prod\_daily, water\_prod\_daily 生成 PCDATA 类型的元素, 并引用:

```
<! ELEMENT pc_pro_well_vol_daily (liq_prod_daily, oil_prod_daily, water_prod_daily) >
<! ELEMENT liq_prod_daily (#PCDATA) >
<! ELEMENT oil_prod_daily (#PCDATA) >
<! ELEMENT water_prod_daily (#PCDATA) >
```

再为主键 well\_id 和 prod\_date 生成 PCDATA 类型的元素并增加引用:

```
<! ELEMENT
pc_pro_well_vol_daily (well_id, prod_date, liq_prod_daily, oil_prod_daily, water_
prod_daily) >
<! ELEMENT liq_prod_daily (#PCDATA) >
<! ELEMENT oil_prod_daily (#PCDATA) >
<! ELEMENT water_prod_daily (#PCDATA) >
<! ELEMENT well_id (#PCDATA) >
<! ELEMENT prod_date (#PCDATA) >
```

为主(外)键(well\_id)输出的表(cd\_well\_source)生成元素, 并在 pc\_pro\_well\_vol\_daily 中生成对 cd\_well\_source 的引用:

```
<! ELEMENT
pc_pro_well_vol_daily (well_id, prod_date, liq_prod_daily, oil_prod_daily, water_
prod_daily, cd_well_source*) >
<! ELEMENT well_id (#PCDATA) >
<! ELEMENT prod_date (#PCDATA) >
<! ELEMENT liq_prod_daily (#PCDATA) >
<! ELEMENT oil_prod_daily (#PCDATA) >
<! ELEMENT water_prod_daily (#PCDATA) >
<! ELEMENT cd_well_source () >
```

为枝表 cd\_well\_source 的数据列和主键列生成元素类型和引用:

```
<! ELEMENT
pc_pro_well_vol_daily (well_id, prod_date, liq_prod_daily, oil_prod_daily, water_
prod_daily, cd_well_source*) >
<! ELEMENT well_id (#PCDATA) >
<! ELEMENT prod_date (#PCDATA) >
<! ELEMENT liq_prod_daily (#PCDATA) >
<! ELEMENT oil_prod_daily (#PCDATA) >
<! ELEMENT water_prod_daily (#PCDATA) >
<! ELEMENT cd_well_source (well_id, org_id, well_common_name) >
```

**SY/T 6782—2010**

<! ELEMENT well\_id ( #PCDATA) >  
<! ELEMENT org\_id ( #PCDATA) >  
<! ELEMENT well\_common\_name ( #PCDATA) >

附 录 D  
(资料性附录)  
采出井日数据模型映射

### D.1 油井日数据的数据库模式到 XML DTD 模式映射示例

将油井日数据模型的关系模式描述为 DTD 文档, 根据 DTD 文档可以手动或由某种程序自动生成关系模式和 XML 文档间的映射, 再根据映射将数据库中数据存储在 XML 文档中, 这样, 存储数据的 XML 文档就可以成为企业内部 (或商业伙伴) 间数据交换的载体。

DTD 映射模式如下:

```
<XML version=" 1.0" standalone=" yes" >
<! DOCTYPE DBA01 [
<! ELEMENT
DBA01 (JH, RQ, CYFS, SCSJ, BJ, PL, YZ, CC, CC1, YY, TY, HY, SXDL, XXDL, DB-
DLC, DBDY, RCYL1, RCYL, RCSL, RCQL, QYHS, HS, HS1, JKWD, RCYL2, RCYHS,
CSYL, CSWD, HYWD, HYJHWND, CCJHWND, CCJND, CCBHJND, BZDM1, BZDM2, JC-
DM, LYRCYL, XYLYRCYL, LYFS, RXFBZ, RCBZ, BX, QYB, BZ, JHWFZL, CCJH-
WND1, DAA01*) >
<! ELEMENT JH (#PCDATA) >
<! ELEMENT RQ (#PCDATA) >
<! ELEMENT CYFS (#PCDATA) >
<! ELEMENT SCSJ (#PCDATA) >
<! ELEMENT BJ (#PCDATA) >
<! ELEMENT PL (#PCDATA) >
<! ELEMENT YZ (#PCDATA) >
<! ELEMENT CC (#PCDATA) >
<! ELEMENT CC1 (#PCDATA) >
<! ELEMENT YY (#PCDATA) >
<! ELEMENT TY (#PCDATA) >
<! ELEMENT HY (#PCDATA) >
<! ELEMENT SXDL (#PCDATA) >
<! ELEMENT XXDL (#PCDATA) >
<! ELEMENT DBDLC (#PCDATA) >
<! ELEMENT DBDY (#PCDATA) >
<! ELEMENT RCYL1 (#PCDATA) >
<! ELEMENT RCYL (#PCDATA) >
<! ELEMENT RCSL (#PCDATA) >
<! ELEMENT RCQL (#PCDATA) >
<! ELEMENT QYHS (#PCDATA) >
<! ELEMENT HS (#PCDATA) >
<! ELEMENT HS1 (#PCDATA) >
<! ELEMENT JKWD (#PCDATA) >
```

<! ELEMENT RCYL2 (#PCDATA) >  
 <! ELEMENT RCYHS (#PCDATA) >  
 <! ELEMENT CSYL (#PCDATA) >  
 <! ELEMENT CSWD (#PCDATA) >  
 <! ELEMENT HYWD (#PCDATA) >  
 <! ELEMENT HYJHWND (#PCDATA) >  
 <! ELEMENT CCJHWND (#PCDATA) >  
 <! ELEMENT CCJND (#PCDATA) >  
 <! ELEMENT CCBHJND (#PCDATA) >  
 <! ELEMENT BZDM1 (#PCDATA) >  
 <! ELEMENT BZDM2 (#PCDATA) >  
 <! ELEMENT JCDM (#PCDATA) >  
 <! ELEMENT LYRCYL (#PCDATA) >  
 <! ELEMENT XYLYRCYL (#PCDATA) >  
 <! ELEMENT LYFS (#PCDATA) >  
 <! ELEMENT RXFBZ (#PCDATA) >  
 <! ELEMENT RCBZ (#PCDATA) >  
 <! ELEMENT BX (#PCDATA) >  
 <! ELEMENT QYB (#PCDATA) >  
 <! ELEMENT BZ (#PCDATA) >  
 <! ELEMENT JHWFZL (#PCDATA) >  
 <! ELEMENT CCJHWND1 (#PCDATA) >  
 <! ELEMENT

DAA01 (JH, CYJH, YT, QKDY, XQKDM, QCDS1, QCDS2, YCDBS1, YCDBS2, SCDS,  
 CW, SKJDDS1, SKJDDS2, CS, SKYXHD, SKSYHD, YLSYHD, ELSYHD, MQJB, SJJB,  
 TCJB, TCRQ, CM, KM, DM, DH, JLZH, YPFL, ZSRQ, JSRQ1, CQRQ, ZQRQ, YC-  
 ZBSD, YSDCYL, YSBHYL, YSDCWD, PLYL, BGRQ, SCBZ, SCLX, SCZRRQ, SCJXRQ,  
 HXSQRQ, FZRQ, ZCRQ, BFRQ, BFYY, BFFS, JWBZ, CXDM, BZ, QKDYDM, SKYCDS1,  
 SKYCDS2, JSHD, CYRQ, HSRQ)

<! ELEMENT JH (#PCDATA) >  
 <! ELEMENT CYJH (#PCDATA) >  
 <! ELEMENT YT (#PCDATA) >  
 <! ELEMENT QKDY (#PCDATA) >  
 <! ELEMENT XQKDM (#PCDATA) >  
 <! ELEMENT QCDS1 (#PCDATA) >  
 <! ELEMENT QCDS2 (#PCDATA) >  
 <! ELEMENT YCDBS1 (#PCDATA) >  
 <! ELEMENT YCDBS2 (#PCDATA) >  
 <! ELEMENT SCDS (#PCDATA) >  
 <! ELEMENT CW (#PCDATA) >  
 <! ELEMENT SKJDDS1 (#PCDATA) >  
 <! ELEMENT SKJDDS2 (#PCDATA) >  
 <! ELEMENT CS (#PCDATA) >

<! ELEMENT SKYXHD (#PCDATA) >  
 <! ELEMENT SKSYHD (#PCDATA) >  
 <! ELEMENT YLSYHD (#PCDATA) >  
 <! ELEMENT ELSYHD (#PCDATA) >  
 <! ELEMENT MQJB (#PCDATA) >  
 <! ELEMENT SJJB (#PCDATA) >  
 <! ELEMENT TCJB (#PCDATA) >  
 <! ELEMENT TCRQ (#PCDATA) >  
 <! ELEMENT CM (#PCDATA) >  
 <! ELEMENT KM (#PCDATA) >  
 <! ELEMENT DM (#PCDATA) >  
 <! ELEMENT DH (#PCDATA) >  
 <! ELEMENT JLZH (#PCDATA) >  
 <! ELEMENT YPFL (#PCDATA) >  
 <! ELEMENT ZSRQ (#PCDATA) >  
 <! ELEMENT JSRQ1 (#PCDATA) >  
 <! ELEMENT CQRQ (#PCDATA) >  
 <! ELEMENT ZQRQ (#PCDATA) >  
 <! ELEMENT YCZBSD (#PCDATA) >  
 <! ELEMENT YSDCYL (#PCDATA) >  
 <! ELEMENT YSBHYL (#PCDATA) >  
 <! ELEMENT YSDCWD (#PCDATA) >  
 <! ELEMENT PLYL (#PCDATA) >  
 <! ELEMENT BGRQ (#PCDATA) >  
 <! ELEMENT SCBZ (#PCDATA) >  
 <! ELEMENT SCLX (#PCDATA) >  
 <! ELEMENT SCZRRQ (#PCDATA) >  
 <! ELEMENT SCJXRQ (#PCDATA) >  
 <! ELEMENT HXSQRQ (#PCDATA) >  
 <! ELEMENT FZRQ (#PCDATA) >  
 <! ELEMENT ZCRQ (#PCDATA) >  
 <! ELEMENT BFRQ (#PCDATA) >  
 <! ELEMENT BFYY (#PCDATA) >  
 <! ELEMENT BFFS (#PCDATA) >  
 <! ELEMENT JWBZ (#PCDATA) >  
 <! ELEMENT CXDM (#PCDATA) >  
 <! ELEMENT BZ (#PCDATA) >  
 <! ELEMENT QKDYDM (#PCDATA) >  
 <! ELEMENT SKYCDS1 (#PCDATA) >  
 <! ELEMENT SKYCDS2 (#PCDATA) >  
 <! ELEMENT JSHD (#PCDATA) >  
 <! ELEMENT CYRQ (#PCDATA) >  
 <! ELEMENT HSRQ (PCDATA) >

] &gt;

**D.2 油井日数据的数据库模式到 XML Schema 模式映射示例**

```

<? xml version = " 1.0" encoding = " UTF - 8"? > ?
  <xsd: schema? xmlns: xsd = " http: //www. w3. org/2001/XMLSchema"
    elementFormDefault = " unqualified" attributeFormDefault = " unqualified" >
    <xsd: annotation>
      <xsd: document>
        oil well daily data schema
      </xsd: document>
    </xsd: annotation>
    <xsd: element name = " DBA01" >
      <xsd: complexType>
        <xsd: element name = " JH" type = " xsd: string" />
        <xsd: element name = " RQ" type = " xsd: date" />
        <xsd: element name = " CYFS" type = " xsd: string" />
        <xsd: element name = " SCSJ" type = " xsd: float" />
        <xsd: element name = " BJ" type = " xsd: float" />
        <xsd: element name = " PL" type = " xsd: float" />
        <xsd: element name = " YZ" type = " xsd: float" />
        <xsd: element name = " CC" type = " xsd: float" />
        <xsd: element name = " CC1" type = " xsd: float" />
        <xsd: element name = " YY" type = " xsd: float" />
        <xsd: element name = " TY" type = " xsd: float" />
        <xsd: element name = " HY" type = " xsd: float" />
        <xsd: element name = " SXDL" type = " xsd: float" />
        <xsd: element name = " XXDL" type = " xsd: float" />
        <xsd: element name = " DBDLC" type = " xsd: float" />
        <xsd: element name = " DBDY" type = " xsd: float" />
        <xsd: element name = " RCYL1" type = " xsd: float" />
        <xsd: element name = " RCYL" type = " xsd: float" />
        <xsd: element name = " RCSL" type = " xsd: float" />
        <xsd: element name = " RCQL" type = " xsd: float" />
        <xsd: element name = " QYHS" type = " xsd: float" />
        <xsd: element name = " HS" type = " xsd: float" />
        <xsd: element name = " HS1" type = " xsd: float" />
        <xsd: element name = " JKWD" type = " xsd: float" />
        <xsd: element name = " RCYL2" type = " xsd: float" />
        <xsd: element name = " RCYHS" type = " xsd: float" />
        <xsd: element name = " CSYL" type = " float" />
        <xsd: element name = " CSWD" type = " xsd: float" />
        <xsd: element name = " HYWD" type = " xsd: float" />
        <xsd: element name = " HYJHWND" type = " xsd: float" />
      </xsd: complexType>
    </xsd: element>
  </xsd: schema>

```

```

    <xsd: element name = " CCJHWND" type = " xsd: float" />
    <xsd: element name = " CCJND" type = " xsd: float" />
    <xsd: element name = " CCBHJND" type = " xsd: float" />
    <xsd: element name = " BZDM1" type = " xsd: string" />
    <xsd: element name = " BZDM2" type = " xsd: string" />
    <xsd: element name = " JCDM" type = " xsd: string" />
    <xsd: element name = " LYRCYL" type = " xsd: float" />
    <xsd: element name = " XYLYRCYL" type = " xsd: float" />
    <xsd: element name = " LYFS" type = " xsd: string" />
    <xsd: element name = " RXFBZ" type = " xsd: string" />
    <xsd: element name = " RCBZ" type = " xsd: string" />
    <xsd: element name = " BX" type = " xsd: float" />
    <xsd: element name = " QYB" type = " xsd: float" />
    <xsd: element name = " BZ" type = " xsd: string" />
    <xsd: element name = " JHWFZL" type = " xsd: float" />
    <xsd: element name = " CCJHWND1" type = "xsd: float" />
</xsd: complexType>
<xsd: key name = " JHPK" >
  <xsd: selector xpath = " DBA01" />
  <xsd: field xpath = " @JH" />
</xsd: key>
<xsd: key name = " RQPK" >
  <xsd: selector xpath = " DBA01" />
  <xsd: field xpath = " @RQ" />
</xsd: key>
<xsd: keyref name = " JHFK" refer = " JHPK" >
  <xsd: selector xpath = " DBA01" />
  <xsd: field xpath = " @JH" />
</xsd: keyref>
</xsd: element>
<xsd: element name = " DAA01" >
  <xsd: complexType>
    <xsd: element name = " JH" type = " xsd: string" />
    <xsd: element name = " CYJH" type = " xsd: string" />
    <xsd: element name = " YT" type = " xsd: string" />
    <xsd: element name = " QKDY" type = " xsd: string" />
    <xsd: element name = " XQKDM" type = " xsd: float" />
    <xsd: element name = " QCDS1" type = " xsd: float" />
    <xsd: element name = " QCDS2" type = " xsd: float" />
    <xsd: element name = " YCDBS1" type = " xsd: float" />
    <xsd: element name = " YCDBS2" type = " xsd: float" />
    <xsd: element name = " SCDS" type = " xsd: float" />
    <xsd: element name = " CW" type = " xsd: string" />
  </xsd: complexType>
</xsd: element>

```

```

<xsd: element name = " SKJDDS1" type = " xsd: float" />
<xsd: element name = " SKJDDS2" type = " xsd: float" />
<xsd: element name = " CS" type = " xsd: float" />
<xsd: element name = " SKYXHD" type = " xsd: float" />
<xsd: element name = " SKSYHD" type = " xsd: float" />
<xsd: element name = " YLSYHD" type = " xsd: float" />
<xsd: element name = " ELSYHD" type = " xsd: float" />
<xsd: element name = " MQJB" type = " xsd: string" />
<xsd: element name = " SJB" type = " xsd: string" />
<xsd: element name = " TCJB" type = " xsd: string" />
<xsd: element name = " TCRQ" type = " xsd: date" />
<xsd: element name = " CM" type = " xsd: string" />
<xsd: element name = " KM" type = " xsd: string" />
<xsd: element name = " DM" type = " xsd: string" />
<xsd: element name = " DH" type = " xsd: integer" />
<xsd: element name = " JLZH" type = " xsd: string" />
<xsd: element name = " YPFL" type = " xsd: string" />
<xsd: element name = " ZSRQ" type = " xsd: date" />
<xsd: element name = " JSRQ1" type = " xsd: date" />
<xsd: element name = " CQRQ" type = " xsd: date" />
<xsd: element name = " ZQRQ" type = " xsd: date" />
<xsd: element name = " YCZBSD" type = " xsd: float" />
<xsd: element name = " YSDCYL" type = " xsd: float" />
<xsd: element name = " YSBHYL" type = " xsd: float" />
<xsd: element name = " YSDCWD" type = " xsd: float" />
<xsd: element name = " PLYL" type = " xsd: float" />
<xsd: element name = " BGRQ" type = " xsd: date" />
<xsd: element name = " SCBZ" type = " xsd: string" />
<xsd: element name = " SCLX" type = " xsd: string" />
<xsd: element name = " SCZRRQ" type = " xsd: date" />
<xsd: element name = " SCJXRQ" type = " xsd: date" />
<xsd: element name = " HXSQRQ" type = " date" />
<xsd: element name = " FZRQ" type = " xsd: date" />
<xsd: element name = " ZCRQ" type = " xsd: date" />
<xsd: element name = " BFRQ" type = " xsd: date" />
<xsd: element name = " BFYY" type = " xsd: string" />
<xsd: element name = " BFFS" type = " xsd: string" />
<xsd: element name = " JWBZ" type = " xsd: string" />
<xsd: element name = " CXDM" type = " xsd: string" />
<xsd: element name = " BZ" type = " xsd: string" />
<xsd: element name = " QKDYDM" type = " xsd: string" />
<xsd: element name = " SKYCDS1" type = " xsd: float" />
<xsd: element name = " SKYCDS2" type = " xsd: float" />

```

```
<xsd: element name = " JSHD" type = " xsd: float" />
<xsd: element name = " CYRQ" type = " xsd: date" />
<xsd: element name = " HSRQ" type = " xsd: date" />
</xsd: complexType>
  <xsd: key name = " JHPK" >
    <xsd: selector xpath = " DAA01" />
    <xsd: field xpath = " @JH" />
  </xsd: element>
</xsd: schema>
```

---

中华人民共和国  
石油天然气行业标准  
石油行业 XML 应用指南  
SY/T 6782—2010

\*

石油工业出版社出版  
(北京安定门外安华里二区一号楼)  
石油工业出版社印刷厂排版印刷  
新华书店北京发行所发行

\*

880×1230 毫米 16 开本 2 印张 59 千字 印 1—1500  
2010 年 8 月北京第 1 版 2010 年 8 月北京第 1 次印刷  
书号: 155021·6478 定价: 16.00 元  
版权专有 不得翻印