XML Schema

sprievodca nielen pre tvorcov SOAP služieb

Obsah

1 Čo je XML Schema?

Formát XML je dlhoročný štandard na uchovávanie a prenos štruktúrovaných dát. Je textový, je univerzálny, je primerane čitateľný nielen strojom, ale aj človekom, a spája sa s ním množstvo technológií, napríklad **XHTML** (jazyk pre webové stránky), **XSLT** (jazyk na transformácie XML do XML, či iných formátov), či **SOAP** (webové služby).

XML je "jazyk na tvorbu jazykov", a jednotlivé dokumenty musia spĺňať konkrétne pravidlá:

- *"Ktoré elementy sú povolené na tomto mieste?* môžem v elemente `<b>` mať tabuľkový element `<table>`? Ak nie, ktoré elementy tam môžu byť?
- *V akom poradí musia ísť konkrétne elementy?* Som povinný mať najprv meno a potom priezvisko, alebo môžu ísť v ľubovoľnom poradí?
- *Aký dátový typ má obsah elementu?* Je to číslo? Dátum? Zložený element?
- *Aké sú pokročilé pravidlá, napríklad pre jedinečnosť?* Koľkokrát sa v texte môže vyskytnúť element s identifikátorom 2435?

Important
XML Schema je jazyk, ktorý určuje pravidlá, ktoré musia spĺňať jednotlivé elementy dokumentu XML. Inými slovami, definuje gramatiku pre XML dokumenty.

2 Kde sa používa XML Schema?

XML Schema sa používa na viacerých miestach:

- **SOAP** služby definujú štruktúru správ pomocou XML schémy.
- Java technológia **JAXB** umožňuje konvertovať XML na objekty a späť. *XML Schema* určuje pravidlá prevodu.
- vývojárske prostredia pre prácu XML umožňujú použiť *XML Schemu* na automatické dopĺňanie (*autocomplete*) povolených elementov

3 Minimalistická schéma

XML schéma sa píše v jazyku XML. To je síce zvláštne, ale umožňuje to mať jednotný formát pre dokumenty i ich gramatiky.

Minimalistická schéma, ba priam jej kostra, vyzerajú nasledovne:

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar">
</schema>
```

- Schéma má koreňový element `<schema>`.
- Atribút `xmlns` udáva implicitný menný priestor pre elementy. Keďže tento dokument popisuje XML schému, jeho elementy budú patriť do menného priestoru špecifikácie *XML Schema*, ktorým je `http://www.w3.org/2001/XMLSchema`.
- Atribút `targetNamespace` (*cieľový menný priestor*) hovorí, do ktorého menného priestoru budú patriť elementy dokumentu popisovaného schémou.

3.1 Varianty minimalistickej schémy

V tomto prípade sme element `schema` zaviedli do implicitného menného priestoru (*default namespace*). V niektorých príkladoch sa uvádza explicitný prefix menného priestoru a elementy schémy majú dohodnutý prefix `xsd:`, či `xs:`.

```
<?xml version="1.0"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="urn:example:calendar">
</xsd:schema>
```

4 Jednoduchá schéma

Predstavme si jednoduchý bežný document:

```
<calendar xmlns="urn:example:calendar"> ❶
    3 events ❷
</calendar>
```

1. Dokument má jediný (koreňový) element `<calendar>`, ktorý patrí do menného priestoru (*namespace*) `urn:example:calendar`.
2. Obsahom elementu je text (reťazec, *string*).

Schéma, ktorá popisuje uvedený dokument, vyzerá nasledovne:

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar">
    <element name="calendar" type="string" />
</schema>
```

Schéma popisuje dokumenty s jediným koreňovým elementom `<calendar>`, ktorý má povolený len reťazcový obsah, bez elementov. Reťazce sú indikované dátovým typom v atribúte `type`, kde `string` reprezentuje *reťazce* v jazyku XML schémy.

Note
XML Schema má zabudovaných štyridsaťštyri (!) jednoduchých dátových typov. Medzi najbežnejšie patrí `string` (reťazce), `double` (desatinné čísla), `int` pre celé čísla, či `dateTime` pre dátumy a časy. Úplný zoznam je v špecifikácii, konkrétne zozname primitívnych dátových typov a v dodatočnom zozname odvodených dátových typov.

4.1 Validácia XML schémy

Na validáciu XML schémy môžeme použiť viacero nástrojov, napríklad `xmllint`:

```
xmllint calendar.xml --schema calendar.xsd
```

Dokument, ktorý spĺňa všetky pravidlá danej XML schémy, je **validný**. Hovoríme tiež, že takýto dokument je **inštanciou** príslušnej XML schémy.

5 Schéma s vnorenými elementami

Vylepšime dokument o kalendár, ktorý obsahuje vnorené elementy:

```
<calendar xmlns="urn:example:calendar">
    <event>Conference Intro at 17:00</event>
    <event>On XML Schemas at 20:00</event>
    <event>Conference outro</event>
</calendar>
```

Schéma má pravidlá:

1. Koreňový element je `<calendar>`.
2. Kalendár obsahuje jeden a viac elementov `<event>` pre konkrétne udalosti.
3. Každá udalosť obsahuje len text.

Schéma bude vyzerať nasledovne:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"> (1)
    <element name="calendar"> (2)
        <complexType> (3)
            <sequence> (4)
                <element name="event" maxOccurs="unbounded" type ↩
                    ="string" /> (5)
            </sequence>
        </complexType>
    </element>
</schema>
```

(2) Element `<calendar>` v schéme je jediný povolený koreňový element. (Pravidlo 1.)

(3) Element `<complexType>` hovorí, že `<calendar>` bude obsahovať podelementy. Ide o **komplexný typ**, na rozdiel od predošlej verzie, ktorá bola *simple type*, jednoduchý typ.

(4) Element `<sequence>` hovorí, že elementy v kalendári musia ísť v takom poradí, v akom sú uvedené v schéme. V tomto prípade to nezaváži, pretože `<calendar>` obsahuje výhradne elementy rovnakého typu `<event>`, kde na poradí aj tak nezáleží.

(5) Element `<event>` je jednoduchý element, ktorý obsahuje reťazce (typ `string`). Element `maxOccurs` nastavuje neobmedzený počet opakovaní, teda neobmedzený počet udalostí v kalendári. Minimálny počet opakovaní je jedna, čo možno voliteľne nastaviť v elemente `minOccurs`.

(1) Atribút `elementFormDefault` s hodnotou `qualified` hovorí, že všetky lokálne elementy musia byť kvalifikované. Podrobnosti si vysvetlíme nižšie, ale bez tohto nastavenia by sme narážali na nelogické vlastnosti validácie.

5.1 Pokročilý dizajn schémy: kvalifikované elementy a lokálne elementy

Caution
XML schéma v štandardnom správaní hovorí, že žiadny z lokálnych elementov nesmie byť kvalifikovaný.

Rozoberme si toto pravidlo postupne pojem za pojmom.

5.1.1 Kvalifikované elementy

Kvalifikovaný element (*qualified element*) patrí do nejakého menného priestoru.

Example 5.1 Príklady kvalifikovaných elementov

Všetky elementy našej inštancie patria do menného priestoru `urn:example:calendar`, ktorý sme zároveň vyhlásili za implicitný. Vďaka pravidlu o dedení menných priestorov v hierarchii je tento menný priestor implicitný nielen pre `<calendar>`, ale aj pre udalosti `<event>`.
Kvalifikovaný názov pre elementy z našej inštancie:

- `{urn:example:calendar}calendar` pre koreňový element,
- `{urn:example:calendar}event` pre udalosťové elementy.

Naša inštancia je ekvivalentná explicitnej verzii, kde každý element vyfasuje explicitný prefix.

```
<cal:calendar xmlns:cal="urn:example:calendar">
    <cal:event>Conference Intro at 17:00</cal:event>
    <cal:event>On XML Schemas at 20:00</cal:event>
    <cal:event>Conference outro</cal:event>
</cal:calendar>
```

5.1.2 Globálne a lokálne elementy

Globálny element (*global element*) je taký, ktorý je priamym potomkom elementu `<schema>` v XML schéme. Všetky ostatné elementy deklarované v schéme sú **lokálne**.

Example 5.2 Príklady globálnych a lokálnych elementov

V našej schéme máme:

- jeden globálny element `<calendar>`
- a jeden lokálny element `<event>`.

5.1.3 Atribút `elementFormDefault`

Atribút `elementFormDefault` s hodnotou `qualified` hovorí, že všetky lokálne elementy v inštancii *musia* byť kvalifikované. Toto správanie, hoci ho musíme uviesť explicitne, dáva pre inštancie logickejšie správanie než keď ho vynecháme.

Ak by sme atribút vynechali, je to ekvivalentné hodnote `unqualified`.

5.1.3.1 Atribút `elementFormDefault` s hodnotou `unqualified`

Pripomeňme si pravidlo zhora.

> XML schéma v štandardnom správaní (`elementFormDefault` s hodnotou `unqualified`) zakazuje kvalifikovanie lokálnych elementov.

Inštancia, ktorá spĺňa schému bez atribútu `elementFormDefault`, musí vyzerať takto:

```
1 <cal:calendar xmlns:cal="urn:example:calendar"> ❶
2     <event>Conference Intro at 17:00</event> ❷
3     <event>On XML Schemas at 20:00</event> ❸
4     <event>Conference outro</event>
5 </cal:calendar>
```

❶ Element `calendar` je kvalifikovaný: `{urn:example:calendar}calendar`. Podľa schémy ide o globálny element.

❷, ❸ Lokálne elementy `event` nepatria do žiadneho menného priestoru: nemajú žiaden prefix menného priestoru a dokument XML nedeklaruje žiaden implicitný menný priestor.

Ak by sme použili klasický dokument s implicitným menným priestorom a pokúsili sa ho zvalidovať, dostaneme chybu.

```
<calendar xmlns="urn:example:calendar">
    <event>Conference Intro at 17:00</event>
    <event>On XML Schemas at 20:00</event>
    <event>Conference outro</event>
</calendar>
```

Chybová hláška hovorí o porušení pravidla XML schémy. Element `<event>`, ktorý je v schéme lokálny, je v XML inštancii kvalifikovaný (patrí do menného priestoru `urn:example:calendar`), čo nie je povolené.

```
Schemas validity error : Element '{urn:example:calendar}event':  ↩
    This element is not expected. Expected is ( event ).
```

Validátor jasne hovorí, že element s kvalifikovaným menom `{urn:example:calendar}event` v elemente kalendára nie je povolený. Namiesto neho je očakávaný nekvalifikovaný element `event`, ktorý nepatrí do žiadneho menného priestoru.

Ako z toho von?

Možnosti sú dve:

1. Buď upravíme XML schému a zavedieme pravidlo `elementFormDefault` s hodnotou `qualified`.
2. Alebo upravíme dokument tak, aby zodpovedal schéme a „odkvalifikujeme“ elementy udalostí tak, ako je to v príklade s nekvalifikovanými elementami pre udalosti.

6 Schéma s komplexnými vnorenými elementami

Predstavme si teraz ešte zložitejšiu inštanciu:

```
<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
    </event>
</calendar>
```

XML schéma, ktorá popisuje tento dokument:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace ↩
    ="urn:example:calendar" elementFormDefault="qualified"> ❶
  <element name="calendar">
```

```
    <complexType>
      <sequence>
        <element name="event" minOccurs="0" maxOccurs="unbounded ↩
            "> ❷
          <complexType>
            <sequence>
              <element name="date" type="dateTime" /> ❸
              <element name="description" type="string" /> ❹
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

❶ Schéma už rovno zapína pravidlo o povinnosti kvalifikovať všetky elementy, aj globálne, aj lokálne.

❷ Lokálny element `<event>` je po novom *komplexným*, pretože môže obsahovať dva podelementy pre dátum (`<date>`) a popis (`<description>`). Sekvencia hovorí, že elementy musia ísť v presnom poradí, najprv dátum a potom popis, pričom výmena nie je povolená.

❸ Element `<date>` pre dátum má dátový typ `dateTime`, čo je zabudovaný dátový typ pre dátumy a časy.

❹ Element `<description>` pre popis je reťazcový.

Keďže elementov typu `<event>` môže byť nula až nekonečno, povolené sú aj extrémne varianty.

Prázdny kalendár:

```
<calendar xmlns="urn:example:calendar" />
```

Kalendár s dvoma udalosťami.

```
<?xml version="1.0"?>

<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
    </event>
    <event>
        <date>2019-05-30T10:00:00</date>
```

```
        <description>Conference</description>
    </event>
</calendar>
```

7 Schéma s viacnásobne vnorenými elementami

Elementy môžeme vnárať aj viacnásobne. Pridajme ku každej udalosti aj zoznam účastníkov.

```
<calendar xmlns="urn:example:calendar">
    <event>
        <date>2019-05-30T09:00:00</date>
        <description>Welcome Drink</description>
        <participants> (1)
            <participant>John Doe</participant> (2)
            <participant>Jane Doe</participant> (3)
        </participants>
    </event>
    <event>
        <date>2019-05-30T10:00:00</date>
        <description>Conference</description> (4)
    </event>
</calendar>
```

(1) Všimnime si, že prvá udalosť má dvoch účastníkov uvedených v rámci elementu `<participants>`.

(2), (3) Každý účastník má svoj vlastný element, kde uvedieme jeho meno.

(4) Udalosť nemusí mať žiadnych potvrdených účastníkov.

Schéma následne zopakuje trik s vnáraním elementov:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified">
  <element name="calendar">
    <complexType>
      <sequence>
        <element name="event" minOccurs="0" maxOccurs="unbounded ↩
            ">
          <complexType>
```

```
            <sequence>
              <element name="date" type="dateTime" />
              <element name="description" type="string" />
              <element name="participants" minOccurs="0"> (1)
                <complexType>
                  <sequence>
                    <element name="participant"
                             type="string"
                             maxOccurs="unbounded" /> (2)
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

(1) Pribudol jeden lokálny element `<participants>`, ktorý má minimálny počet výskytov nastavený na nulu, čo je ekvivalent nepovinného elementu. Ide o komplexný element so sekvenciou vnorených elementov rovnakého typu.

(2) Každý účastník je reťazcového typu.

Štýl matrioška
V schéme máme jeden globálny element `<calendar>` a viacero lokálnych elementov: `<event>`, v ňom `<date>`, `<description>` a `<participants>`, a v rámci neho účastníka `<participant>`.
Tento štýl vnárania elementov sa niekde nazýva **matrioška** podľa slávnej ruskej bábiky, ktorá obsahuje bábiky, ktoré obsahujú bábiky.

---

8 Schéma s viacerými koreňovými elementami

Globálne elementy schémy určujú povolené koreňové elementy. Doposiaľ sme mali povolený jediný koreňový element `<calendar>`, ale sú situácie, keď jedna schéma popisuje viacero možných inštancií s rozličnými koreňmi.

Medzi príklady z praxe patrí:

- jazyk **DocBook** pre písanie dokumentácie, ktorý povoľuje knihy `<book>`, ale aj články `<article>`
- formát správ vo webových službách **SOAP**, ktorý povoľuje vlastnú definíciu správ pre požiadavky a odpovede. Príkladom môže byť `<CalendarResponse>` pre odpoveď a `<CalendarRequest>` pre požiadavku.

Pridajme si do schémy ďalší koreňový element pre dokument reprezentujúci jednu udalosť.

```
<event xmlns="urn:example:calendar">
    <date>2019-05-30T09:00:00</date>
    <description>Welcome Drink</description>
    <participants>
        <participant>John Doe</participant>
        <participant>Jane Doe</participant>
    </participants>
</event>
```

Schéma, ktorá zvládne aj kalendár, aj jednu udalosť vyzerá nasledovne. Nie je to vonkoncom optimálna schéma, pretože sa v ňom opakujú definície elementu `<event>`, ale to opravíme neskôr.

```
<?xml version="1.0"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified">
```

```
  <element name="event"> (1)
    <complexType>
      <sequence>
        <element name="date" type="dateTime" />
        <element name="description" type="string" />
        <element name="participants" minOccurs="0">
          <complexType>
            <sequence>
              <element name="participant"
                       type="string"
                       maxOccurs="unbounded" />
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>

  <element name="calendar">
    <complexType>
```

```
      <sequence>
        <element name="event" minOccurs="0" maxOccurs="unbounded ↩
            "> ❷
          <complexType>
            <sequence>
              <element name="date" type="dateTime" />
              <element name="description" type="string" />
              <element name="participants" minOccurs="0">
                <complexType>
                  <sequence>
                    <element name="participant"
                             type="string"
                             maxOccurs="unbounded" />
                  </sequence>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>
</schema>
```

❶, ❷ Definícia elementu sa opakuje, pretože je naozaj rovnaká v samostatnom dokumente i v zozname udalostí v kalendári.

9 Schéma s opakujúcimi sa elementami

Element `<event>` v predošlom príklade sa vyskytuje na dvoch rozličných miestach: buď ako koreňový element alebo ako súčasť kalendára.

Ak chceme zrecyklovať, či znovupoužiť definíciu bez jej opakovania, vytiahnime definíciu štruktúry tohto elementu von, mimo elementov, a následne sa na ňu odkážeme z oboch miest.

Podobne ako v bežnom programovaní tried, či štruktúr `struct` môžeme definovať štruktúru elementu ako samostatný pomenovaný typ.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar"> ❶

    <complexType name="Event"> ❷
```

```
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
            <element name="participants" minOccurs="0">
                <complexType>
                    <sequence>
                        <element name="participant" type="string ↩
                            " maxOccurs="unbounded" />
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>

    <element name="calendar">
        <complexType>
            <sequence>
                ❸
                <element name="event" type="cal:Event"
                         minOccurs="0"
                         maxOccurs="unbounded" />
            </sequence>
        </complexType>
    </element>

    <element name="event" type="cal:Event" />  ❹
</schema>
```

❷ Deklaráciu sme presunuli do elementu `<complexType>`, ktorý sme pomenovali `Event` a dali sme mu rovnakú štruktúru udalosti kalendára, ako v predošlých príkladoch.

Dôležitá je jedna vec: typ `Event` patrí do menného priestoru `urn:example:calendar`, čo je určené atribútom `targetNamespace`.

❸ Element vo vnútri kalendára, teda `<event>` už neuvádza svoju vnútornú štruktúru explicitne, ale odkazom na komplexný typ.

Takýto odkaz však musíme urobiť nepriamo, okľukou cez *prefix menného priestoru*. Spomenuli sme, že typ `Event` patrí do menného priestoru `urn:example:calendar` (jeho kvalifikované meno je `{urn:example:calendar}Event`). Keďže menné priestory môžu byť mimoriadne dlhé — napríklad `http://www.w3.org/2001/XMLSchema` — musíme použiť ich zástupné mená (aliasy), teda **prefixy**. V atribúte `type` sme sa rozhodli použiť prefix `cal:`. Musíme však ešte určiť, že `cal:` je prefix pre `urn:example:calendar`.

❶ Mapovanie medzi menným priestorom `urn:example:calendar` a jeho prefixom `cal` urobíme v koreňovom elemente pomocou klasického me-

chanizmu menných priestorov. Prefix menného priestoru je ľubovoľný, my sme sa rozhodli pre krátky a úderný `cal`.

4 Voľne stojaci element `<event>` ako koreňový element je tiež typu `cal:Event`. Platí podobná filozofia: použijeme odkaz na komplexný typ `Event`, pričom jeho menný priestor je určený prefixom.

Note
Takýto štýl schémy sa nazýva žalúzia (Venetian Blind). Koreňové elementy sú globálne, všetky ostatné elementy sú lokálne. Viacnásobne používané štruktúry sú deklarované cez pomenované komplexné či jednoduché typy.

---

10 Schéma s vlastnými jednoduchými typmi

Jednoduché typy podporujú rozličnú sadu špeciálnych obmedzení. Reťazce s dĺžkou v danom rozsahu, čísla v danom intervale, iné reťazce spĺňajúce formát v tvare regulárneho výrazu, či hodnoty z daných možností.

XML Schema umožňuje definovať typy pomocou reštrikcií (*restrictions*) a faziet (*facets*).

Predstavme si, že chceme obmedziť popis udalosti na 32 znakov. V schéme dodáme vlastný jednoduchý dátový typ. Vložíme ho priamo pod element `<schema>`:

```
<simpleType name="Description">
    <restriction base="string">
        <maxLength value="32" />
    </restriction>
</simpleType>
```

Jednoduchý typ (*simple type*), ktorý určuje formát hodnôt v elemente, vznikol:

- ako reštrikcia zabudovaného dátového typu `string` (reťazec). Reštrikcie deklarujeme v elemente `<restriction>`
- s jednou fazetou, ktorá obmedzí dĺžku na 32 znakov. Tá je uvedené v elemente `<maxLength>`.

Následne vieme upraviť dátový typ v elemente `<description>`. Namiesto reťazcového typu použijeme odkaz na typ `Description`:

```
<element name="description" type="cal:Description" />
```

Podobne ako v prípade zložených dátových typov použijeme plne kvalifikovaný odkaz, kde menný priestor uvedieme pomocou prefixu `cal`.

Obmedzenie začne platiť pre ľubovoľný element `<description>`, bez ohľadu na to, či je v samostatnom dokumente `<event>` alebo v rámci udalosti kalendára.

Nasledovný dokument prestane zodpovedať schéme:

```
<event xmlns="urn:example:calendar">
    <date>2019-05-30T10:00:00</date>
    <description>A Very Long Conference Name With More Than 32  ↩
        Characters</description>
</event>
```

Pri pokuse o validáciu uvidíme chyby indikujúce porušenie facetov a reštrikcí:

```
Element '{urn:example:calendar}description': [facet 'maxLength'] ↩
     The value has a length of '56'; this exceeds the allowed  ↩
    maximum length of '32'.
Element '{urn:example:calendar}description': 'A Very Long  ↩
    Conference Name With More Than 32 Characters' is not a valid ↩
     value of the atomic type '{urn:example:calendar}Description ↩
    '.
```

11 Inštancie a automatické priradenie schémy

Každá inštancia dokumentu môže mať implicitne priradenú schému, oproti ktorej sa dá zvalidovať.

```
---
<calendar xmlns="urn:example:calendar"
          xsi:schemaLocation="urn:example:calendar calendar.xsd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
```

Koreňovému elementu môžeme priradiť atribút `schemaLocation`. Jeho hodnota pozostáva z dvojíc oddelených medzerami, napr. `urn:example:calendar calendar.xsd`

- prvá časť dvojice reprezentuje **menný priestor**, ku ktorému priradíme XML schému
- druhá časť dvojice predstavuje adresu **URL**, na ktorej sa nachádza XML schéma k predošlému mennému priestoru.

11.1 Lokácie XML schém dokumentu

Lokácia XML schémy je adresa URL, ktorá musí byť dohľadateľná a stiahnuteľná validátorom. Validátor získa súbor schémy z danej adresy a použije ju pri validácii inštancie XML.

Adresa musí byť:

- **absolútna**, napr. `https://www.w3.org/2009/XMLSchema/XMLSchema.xsd`
- **relatívna**, kde sa očakáva, že cesta k schéme je uvedená vzhľadom k lokácii inštancie. V príklade čakáme, že schéma `calendar.xsd` je v rovnakom „adresári“ ako dokument XML.

11.2 Atribút `schemaLocation`

Samotný atribút `schemaLocation` je plne kvalifikovaný a patrí do menného priestoru `http://www.w3.org/2001/XMLSchema-instance`. Ak ho chceme použiť v dokumente, musíme sa naňho odkázať cez prefix menného priestoru, ktorý je podľa konvencie `xsi`. To je dôvod, prečo musíme deklarovať mapovanie medzi prefixom a menným priestorom:

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

12 Alternatívne zápisy schémy

Alternatívny obvyklý zápis XML schémy využíva explicitný prefix menného priestoru:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:cal="urn:example:calendar"
           targetNamespace="urn:example:calendar"
           elementFormDefault="qualified">
    <!-- ... -->

    <xs:simpleType name="Description">
        <xs:restriction base="xs:string">
            <xs:maxLength value="32" />
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

Doposiaľ sme mali zavedený implicitný menný priestor `http://www.w3.org/2001/XMLSchema`, čo znamenalo, že všetky elementy z jazyka **XML Schema** sme

mohli uviesť bez prefixu menného priestoru. Mnoho XML schém však používa explicitný prefix, ktorým je obvykle `xs`, či `xsd`.

V takom prípade musíme:

1. Zaviesť mapovanie prefixu na menný priestor. V koreňovom elemente uvedieme:

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

2. Všetky elementy patriace do menného priestoru jazyka *XML Schema* musia byť uvedené s prefixom: napríklad `<xs:schema>`, či `<xs:complexType>`.
3. Všetky dátové typy z jazyka *XML Schema* musia byť uvedené s prefixom, napríklad `xs:string`, či `xs:dateTime`.

V ukážke vidíme, ako sa element reštrikcie `<xs:restriction>` odvodil od zabudovaného elementu reťazec (*string*), na ktorý sa odkážeme pomocou prefixu, teda `xs:string`.

Note
XML schéma, kde konštrukčné elementy (`element`, `complexType` atď.) sú v mennom priestore s prefixom `xs`, resp. `xsd`, je ekvivalentná schéme, kde sú konštrukčné elementy v implicitnom mennom priestore. Jediný rozdiel uvidíme v prípade, že chceme konštruovať schému pre dokumenty, u ktorých elementy nepatria do žiadneho menného priestoru (atribút `targetNamespace` vynecháme). To je veľmi okrajová situácia, ktorá sa neodporúča použiť a ak áno, konštrukčné elementy musia mať explicitný prefix.

---

13 Schéma s elementami, ktoré majú atribúty

13.1 Komplexné elementy s atribútmi

Elementy popisované schémou môžu mať svoje vlastné atribúty. Tie často popisujú *metadáta*, teda dáta o dátach reprezentovaných v dokumente.

Zoberme si dokument, kde atribút app hovorí o aplikácii, ktorá vytvorila príslušný kalendár.

```
<calendar xmlns="urn:example:calendar" app="TurboCalendar">
    ...
</calendar>
```

Atribút app je reťazcový. Poďme ho teraz zareprezentovať v schéme.

Important
Element s atribútami musí byť vždy komplexný (`complexType`). Ak chceme jednoduchý typ (simple type) s atribútom, musíme ho deklarovať ako komplexný typ.

```
<element name="calendar">
    <complexType>
        <sequence>
            <element name="event" type="cal:Event" minOccurs="0" ↩
                maxOccurs="unbounded" />
        </sequence>
        <attribute name="app" type="string" use="required"/> (1)
    </complexType>
</element>
```

(1) V elemente `<calendar>` deklarujeme atribút app typu reťazec (*string*).

Atribút `use` určuje povinnosť atribútu. Hodnota `required` vraví, že atribút je povinný. (Ďalšie možnosti sú: implicitný `optional` pre nepovinné atribúty a `prohibited` pre zakázaný atribút.)

Important
Napriek tomu, že v XML dokumente sú atribúty uvedené pred vnorenými elementami, v XML schéme najprv uvádzame podelementy (v príklade `<sequence>`) a až následne uvádzame atribúty.

13.2 Jednoduché elementy s atribútmi – rozšírenia typov pomocou extensions

Ak chceme ukázať jednoduché elementy (bez vnorených elementov) s atribútmi, musíme sa vrátiť k úplne prvému príkladu. Dodajme doňho atribút app.

```
<calendar xmlns="urn:example:calendar" app="iCal">3 events</ ↩
    calendar>
```

Schéma, ktorá popisuje uvedený jednoduchý dokument, musí zadeklarovať `<calendar>` ako komplexný typ, a to i napriek tomu, že obsah je jednoduchý.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace ↩
    ="urn:example:calendar">
    <element name="calendar">
        <complexType>
            <simpleContent> (1)
                <extension base="string"> (2)
                    <attribute name="app" type="string" use=" ↩
                        required"/> (3)
                </extension>
            </simpleContent>
        </complexType>
    </element>
</schema>
```

(1) Element zadeklarujeme ako komplexný typ, ale s jednoduchým obsahom (*simple content*), ktorý zakazuje podelementy.

(2) Element následne použijeme ako rozšírenie (**extension**) existujúceho jednoduchého typu — v našom prípade reťazca — ktorému dodáme ďalšie atribúty.

(3) Elementu `<calendar>` dodáme povinný (*required*) reťazcový (*string*) atribút s názvom app.

Note
Rozšírenie (**extension**) plní v XML schéme podobnú funkciu ako dedičnosť v objektovo orientovanom programovaní. Typ, ktorý rozširujeme („od ktorého dedíme“) predstavuje akúsi šablónu, ktorú obohatíme o nové atribúty, či podelementy.

13.2.1 Jednoduché typy s atribútmi a facetmi: použitie reštrikcie a extenzie pre jeden element

Niekedy chceme, aby element obsahoval naraz aj atribúty, ale podliehal špeciálnym obmedzeniam (napríklad na dĺžku, či formát).

V taktom prípade použijeme trik:

1. Vytvoríme vlastný dátový typ s reštrikciou.
2. Použijeme ho ako dátový typ elementu, ktorý rozšírime o atribúty.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        xmlns:cal="urn:example:calendar">

    <simpleType name="CalendarSummary"> (1)
        <restriction base="string"> (2)
            <pattern value="\d+ event(s)?" /> (3)
        </restriction>
    </simpleType>

    <element name="calendarSummary">
        <complexType>
            <simpleContent> (4)
                <extension base="cal:CalendarSummary"> (5)
                    <attribute name="app" type="string" use="
                        required"/>
                </extension>
            </simpleContent>
        </complexType>
    </element>
</schema>
```

(1) Deklarujeme vlastný dátový typ `CalendarSummary` reprezentujúci element s jednoduchým textovým obsahom.

(2) Obsah elementu vytvoríme reštrikciou typu reťazec.

(3), (4) Fazetou reštrikcie bude regulárny výraz.

(5) Sumár kalendára (koreňový element) definujeme ako komplexný typ s jednoduchým obsahom, ktorý založíme na dátovom type `CalendarSummary`. Extenziou nášho vlastného dátového typu získame element s formátom vyhovujúcim regulárnemu výrazu, ktorému vieme pridať dodatočné atribúty.

Inštancia dokumentu XML, ktorý vyhovuje schéme:

```
<calendarSummary xmlns="urn:example:calendar" app="iCal">3
    events</calendarSummary>
```

Takýto dokument s jedným elementom podporuje i atribút app, i predpis na obsah zodpovedajúci regulárnemu výrazu.

14 Kompletná XML schéma

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar"
>
    <element name="event" type="cal:Event" />

    <element name="calendar">
        <complexType>
            <sequence>
                <element name="event" type="cal:Event" minOccurs ↩
                    ="0" maxOccurs="unbounded" />
            </sequence>
            <attribute name="app" type="string" use="required"/>
        </complexType>
    </element>

    <complexType name="Event">
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="cal:Description" / ↩
                >
            <element name="participants" minOccurs="0">
                <complexType>
                    <sequence>
                        <element name="participant" type="string ↩
                            " maxOccurs="unbounded" />
                    </sequence>
                </complexType>
            </element>
        </sequence>
    </complexType>

    <simpleType name="Description">
        <restriction base="string">
            <maxLength value="32" />
        </restriction>
    </simpleType>
</schema>
```

15 Práca s externými schémami

XML Schema podporuje dva spôsoby práce s externými schémami:

- **include**, kde vieme do existujúcej schémy „vložiť“ definície z inej schémy, ale rovnakého menného priestoru tak, ako keby boli v nej uvedené priamo.
- **import**, kde vieme do existujúcej schémy dotiahnuť definície z inej schémy a iného menného priestoru.

15.1 Inklúzia externých schém

Inklúzia schémy „skopíruje“ obsah externej schémy do aktuálnej schémy. Inklúdovať môžeme len elementy z rovnakého menného priestoru ako má cieľový menný priestor (*target namespace*) aktuálnej schémy.

Predstavme si základnú schému pre udalosti:

event.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar">
    <complexType name="Event">
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
        </sequence>
    </complexType>
    <element name="event" type="cal:Event" />
</schema>
```

Schéma má cieľový menný priestor `urn:example:calendar` a deklaruje v ňom jediný globálny element `<event>`.

A teraz si vytvorme druhú schému, `calendar.xsd`, ktorá chce využiť existujúce deklarácie z externej schémy `event.xsd`.

calendar.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:calendar"
        elementFormDefault="qualified"
        xmlns:cal="urn:example:calendar">

    <include schemaLocation="event.xsd" /> ❶

    <element name="calendar"> ❷
        <complexType>
            <sequence>
                <element name="event" type="cal:Event" maxOccurs ↩
                    ="unbounded"/> ❸
```

```
            </sequence>
        </complexType>
    </element>
</schema>
```

1. Schéma `calendar.xsd` má cieľový menný priestor zhodný s cieľovým menným priestorom schémy `event.xsd` (ide o priestor `urn:example:calendar`). Môžeme teda do nej priamo vložiť (*include*) obsah externej schémy. Inklúziu zrealizujeme elementom `<include>` a uvedením absolútnej adresy URL alebo relatívnej adresy k externej schéme.
2. V schéme si deklarujeme vlastný element `<calendar>`, ktorý bude obsahovať zoznam udalostí `<event>`.
3. Štruktúra každého elementu sa riadi komplexným typom `Event` deklarovaným v schéme `event.xsd`

Caution
Zo schémy môžeme sa môžeme odkazovať len na globálne typy a elementy.

15.1.1 Referencie na elementy z externej schémy

Elementy pre udalosti môžeme použiť aj iným spôsobom, odkazom. Namiesto deklarácie elementu a odkazu na jeho typ môžeme uviesť **referenciu**:

```
<element ref="cal:event" maxOccurs="unbounded"/> (1)
```

1. Referenciu na globálny element realizujeme atribútom `ref`, kde uvedieme kvalifikovaný názov elementu použiteľného na príslušnom mieste. (Kvalifikovaný názov uvedieme pomocou prefixu, ktorý musí byť namapovaný na príslušný menný priestor, ideálne v koreňovom elemente schémy.)

15.2 Import externých schém

Import schémy vezme elementy a typy z menného priestoru externej schémy a sprístupní ich v aktuálnej schéme.

Warning
Importované súčasti musia byť z iného menného priestoru ako má cieľový menný priestor aktuálnej schémy!

Predstavme si základnú schému pre udalosti:

event.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="urn:example:event"
        xmlns:e="urn:example:event"
        elementFormDefault="qualified"> (1)

    <complexType name="Event"> (2)
        <sequence>
            <element name="date" type="dateTime" />
            <element name="description" type="string" />
        </sequence>
    </complexType>
    <element name="event" type="e:Event" /> (3)
</schema>
```

(1) Schéma `event.xsd` má cieľový menný priestor `urn:example:event` namapovaný na prefix e.

(2) V schéme deklarujeme komplexný typ `Event` s dvoma lokálnymi podelementami pre dátum a popis.

(3) Deklarujeme globálny element `<event>`, ktorého štruktúra sa riadi komplexným typom `Event`. Tento element má kvalifikované meno `{urn:example:event}event`

Vytvorme teraz druhú schému pre kalendár:

calendar.xsd

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        elementFormDefault="qualified"
        targetNamespace="urn:example:calendar"
        xmlns:e="urn:example:event"> (1)

    <import namespace="urn:example:event" schemaLocation="event.xsd" /> (2)

    <element name="calendar">
        <complexType>
```

```
            <sequence>
                <element ref="e:event" maxOccurs="unbounded"/> (3)
            </sequence>
        </complexType>
    </element>
</schema>
```

(1) Schéma pre kalendár `calendar.xsd` deklaruje elementy do cieľového menného priestoru `urn:example:calendar`. Zároveň deklarujeme mapovanie prefixu e na menný priestor udalostí z importovanej schémy.

(2) Schému pre udalosti `event.xsd` zavedieme do aktuálnej schémy. Keďže menný priestor tejto externej schémy je odlišný od cieľového menného priestoru aktuálnej schémy, použijeme `<import>`. Uvedieme adresu schémy (`schemaLocation`) a menný priestor, do ktorého importneme jej prvky.

(3) Elementy kalendára znovupoužijeme zo schémy. Použijeme *referenciu* na globálny element `event` zo schémy pre udalosti. Odkaz samozrejme uvedieme v kvalifikovanom tvare, s použitím prefixu e.

Pozrime sa teraz na vzhľad inštancií:

calendar.xml

```
<calendar xmlns="urn:example:calendar"
    xmlns:e="urn:example:event"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:example:calendar calendar.xsd">(1)

    <e:event> (2)
        <e:date>2019-05-30T09:00:00</e:date>
        <e:description>Test</e:description>
    </e:event>
</calendar>
```

(1) Kalendár má koreňový element `<calendar>` s celým kvalifikovaným menom `{urn:example:calendar}calendar`. Validácia sa bude riadiť schémou `calendar.xsd`.

(2) Vnorené elementy však patria do ëxternej schémy"pre udalosti, ktorá má odlišný menný priestor. Keďže podľa schémy `event.xsd` patrí element `<event>` do menného priestoru `urn:example:event`, musíme ho uviesť s korektným prefixom (v našom príklade e). Nezabudnime na to, že import zlučuje elementy z dvoch odlišných menných priestorov, čo je dôvod, prečo sa prefixy líšia.

Note
V prípade, že sa v inštancii zídu dva menné priestory, môže pomôcť explicitné uvedenie prefixov. Dokument potom môže vyzerať nasledovne:
calendar.xml

```
<cal:calendar xmlns:cal="urn:example:calendar"
              xmlns:e="urn:example:event"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema- ↩
                  instance"
              xsi:schemaLocation="urn:example:calendar calendar ↩
                  .xsd">
    <e:event>
        <e:date>2019-05-30T09:00:00</e:date>
        <e:description>Test</e:description>
    </e:event>
</cal:calendar>
```

- Kalendárový element má kvalifikované meno `{urn:example:calendar}calendar` a má prefix `c`.
- Udalostný element má kvalifikované meno `{urn:example:event}event` a má prefix `e`. Oba prefixy sme namapovali na príslušné menné priestory v koreňovom elemente inštancie.

16 Literatúra a zdroje

- XML Schema Part 0: Primer Second Edition. Jednoduchý úvod do XML schém od autorov špecifikácie.
- What does `elementFormDefault` do in XSD?. Vysvetlenie pravidla o kvalifikovaní elementov.
- Introducing Design Patterns in XML Schemas. Návrhové vzory pri tvorbe XML schém.