Core technology since 1996

XML 의 이해

Understanding of XML

Release 1.0

2010/10

novo networks

목 차 (Table of Contents)

1. AUDIENCE 4

2. XML 문법 – 시작하기 5

2.1 TAG, CONTENT, ELEMENT 5
2.2 NESTED ELEMENT 7
2.3 CONTENT 의 종류 9
2.4 XML IS TREE 10
2.5 ATTRIBUTE 12

3. XML 문서 검색하기 14

3.1 XPATH 14
3.2 E4X 16

4. XML 문법 – 좀 더 깊게 17

4.1 ESCAPING CHARACTER, CDATA, COMMENT 17
4.2 XML DECLARATION 20
4.3 NAMESPACE 21

5. XML SCHEMA LANGUAGE 24

5.1 DTD 25
5.2 XML SCHEMA DEFINITION(XSD) 28
5.3 RELAX NG 32

6. XML API 34

6.1 DOM 34
6.2 SAX 35

APPENDIX A. 문서정보 37

A.1 문서이력 37

APPENDIX B. REFERENCES 38

1. Audience

이 문서는 XML(Extensible Markup Language)를 처음 접하는 사람을 위해 작성되었으며, XML 문법과 XML 문서 내용에 접근하는 방식 등 기초적이고 일반적인 내용을 담고 있습니다. 추가적으로 XML 라이브러리 사용법이나 XML 문서를 display 하는 방법(XSL, CSS 와 같은) 등이 궁금하신 분들께서는 reference 의 문서나 기타 다른 문서를 참고하시길 바랍니다.

2. XML 문법 – 시작하기

■ Valid XML document

XML 문법에 대해 설명하기 전에 'well-formed XML 문서'에 대해 정의할 필요가 있습니다. well-formed XML 문서란 W3C 라는 단체에서 1998 년에 제정한 XML 1.0 권고안을 따르는 XML 문서를 말합니다. 2004 년에 최신 버전인 1.1 버전이 배포되었는데, 두 버전은 큰 차이가 없으므로 여기서는 좀 더 널리 쓰이는 XML 1.0 버전을 기준으로 삼겠습니다. 통상적으로 'XML 문서'라고 하면 앞에 'well-formed'가 생략되었다고 간주됩니다.

well-formed XML 문서를 작성하기 위한 규칙을 만든 이유는 XML 문서를 읽어서 쉽게 markup 데이터를 구별해 낼 수 있는 프로그램을 만들기 위해서 입니다. 이렇게 XML 문법을 이해할 수 있고, XML 문서를 읽어 어플리케이션이 필요로 하는 정보를 제공하는 component 를 XML parser 라고 합니다. XML 1.0 규격을 따르는 XML parser 가 well-formed XML 이 아닌 문서를 받아들이면 에러를 표시합니다. 다음 절부터는 XML 문서를 작성하는 과정을 통해 XML 1.0 에서 정의하는 문법 규칙을 살펴보도록 하겠습니다.

2.1 Tag, content, element

게임을 좋아하는 친구와 함께 여러 가지 게임에 관한 정보를 종합하는 웹 페이지를 만들려고 합니다. 친구와 상의한 결과 우선 게임에 관한 정보를 모아 XML 문서로 만들어 놓으면 좋겠다는 결론을 내렸습니다. 주제가 정해졌으니, 이제 첫 번째 XML 문서를 작성해 봅시다.

```
<games></games>
```

부등호 기호(<>)로 둘러싸여 있는 글자들을 **tag(태그)**라고 합니다. '<games>'는 여는 태그, '</games>'는 닫는 태그라고 부릅니다. 한 쌍의 여는 태그와 닫는 태그는 슬래시('/')가 있는지 없는지를 제외하면 모양(대·소문자 포함)이 같아야 합니다. 여기에 내용을 좀 더 추가하면 다음과 같습니다.

```
<games>StarCraft</games>
```

태그 사이에 'StarCraft'라는 데이터를 적어 넣었습니다. 태그 사이에 적힌 데이터를 **content(내용)**라고 합니다. 이 content 가 XML 을 통해 저장할 실제 데이터입니다. 그렇다면 태그 안에 적혀있는 'games'는 무슨 의미일까요? 이것은 content 가 무엇에 관한 데이터인지를 알려주는 역할을 합니다. 따라서 위의 XML 문서를 보고 'StarCraft 는 game 이구나' 라고 추측할 수 있습니다. 이렇게 XML 문서는 한 쌍의 태그가 content 를 둘러 싸고 있고, 태그는 그 content 에 관한 정보를 제공하는 구조를 가지고 있습니다.

한 쌍의 태그와 그 태그 안의 content 를 모두 통틀어 **element(요소)**라고 부릅니다. 즉 '<games>StarCraft</games>'가 전부 element 인 것입니다. 태그 안의 글자인 'games'는 element 의 이름이라고 볼 수 있습니다. 보통 '<games>StarCraft</games>' 라는 element 를 가리켜 'games element'라고 부릅니다.

element 이름은 숫자나 구두점, 혹은 'xml'이라는 문자열(대소문자 상관 없이)로 시작하면 안 되고, 공백이 없어야 한다는 것을 제외하면 마음대로 지을 수 있습니다. <game>StarCraft</game>, <product>StarCraft</product>는 물론 심지어 <게임>스타크래프트</게임>과 같이 한글로도 쓸 수 있습니다. 이러한 자유로움은 XML 을 'Extensible'하게 만들어 주는 특성 중 하나입니다.

2.2 Nested element

```
<games>StarCraft</games>
```

위 문서는 간단해 보이긴 하지만, 너무 간단해서 별 쓸모가 없어 보입니다. 이제 좀 더 쓸만한 XML 문서가 되도록 확장해 봅시다. 게임 중에서도 비디오 게임에 관한 데이터를 쓰려고 합니다. 어떻게 해야 할까요?

```
<games><video_game></video_game></games>
```

위와 같이, element 의 content 자리에 또 다른 element 를 집어 넣을 수 있습니다. 이렇게 element 안에 element 가 들어있는 구조를 가리켜 **nested element(중첩된 요소)**라고 합니다. element 를 중첩시킬 수 있다는 점은 XML 을 강력하게 만들어 주는 특성입니다.
이번에는 비디오 게임의 이름을 추가해 봅시다.

```
<games><video_game><title></title></video_game></games>
```

title element 를 video_game element 안에 추가하였습니다. 그런데 문서가 점점 길어질수록 읽기도 점점 힘들어집니다. 위 문서를 좀 더 읽기 쉽게 바꾸어 봅시다.

```
<games>
        <video_game>
                <title></title>
        </video_game>
</games>
```

indent(들여쓰기)를 적용하였습니다. 한결 알아보기가 쉬워 보입니다. 계속해서 title element 에 내용을 채워 봅시다.

```
<games>
        <video_game>
                <title>StarCraft</title>
        </video_game>
</games>
```

게임의 이름은 StarCraft 로군요. 이 게임의 개발사에 관한 정보도 추가하려면 어디에 추가해야 할까요?

```
<games>
        <video_game>
                <title>StarCraft</title>
                <developer></developer>
        </video_game>
</games>
```

video_game element 안, title element 바로 다음에 개발사에 관한 정보를 추가하였습니다. 이렇게 한 element 안에는 여러 개의 element 가 중첩되어 있을 수 있습니다. element 의 중첩에는 아무런 제한이 없으므로 원하는 대로 데이터를 가공할 수 있습니다.

```
<games>
        <video_game>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
</games>
```

개발사의 이름도 적어보았습니다. 이제 이 XML 문서는 게임 중에서 Blizzard Entertainment 에서 개발한 StarCraft 라는 이름의 비디오 게임에 관한 데이터를 뜻하게 되었습니다.

2.3 Content 의 종류

content 의 종류로는 element content, simple content, mixed content 세 가지가 있습니다. XML 문서를 통해 각각의 뜻을 살펴봅시다.

```
<games>
        <video_game>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
                StarCraft has a professional competition.
        </video_game>
</games>
```

이전 절의 마지막 문서에 굵은 글씨의 데이터(StarCraft 라는 게임에 관한 특기사항)를 추가한 문서입니다.

games element 는 content 로 video_game 이라는 element 만을 가지고 있습니다. content 자리에 element 만 있을 경우(element 의 개수는 상관 없음) 이를 **element content** 라고 합니다.

title element 와 developer element 는 content 로 평범한 text 만을 가지고 있습니다. 이것을 **simple content**(혹은 text content)라고 합니다.

그렇다면 video_game element 는 어떨까요? video_game element 는 content 로 title element, developer element 와 함께 "StarCraft has a professional competition." 라는 내용의 text 도 가지고 있습니다. 이렇게 content 자리에 element 와 text 가 둘 다 있는 것을 가리켜 **mixed content** 라 합니다.

2.4 XML is Tree

이전 절의 문서에 다른 종류의 비디오 게임에 관한 정보를 추가하려 합니다.

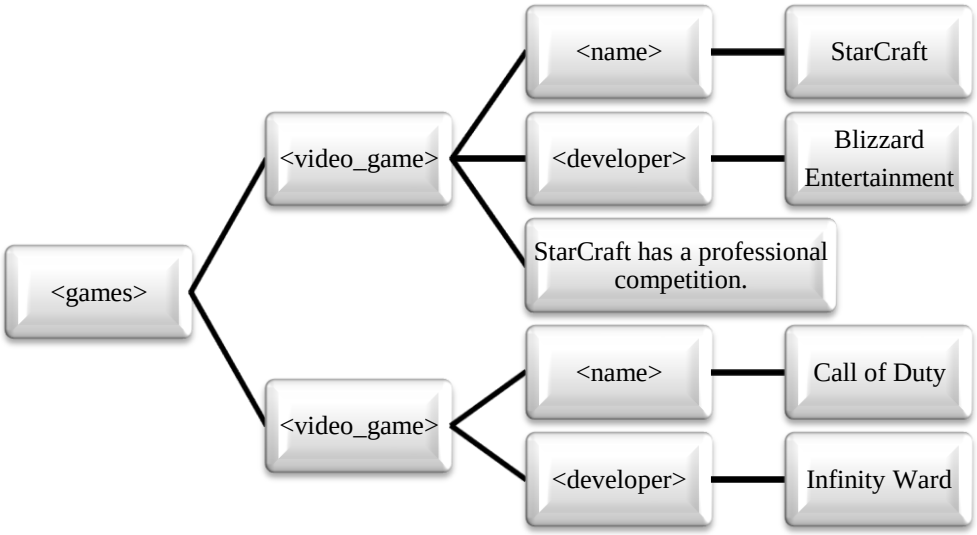
```
<games>
        <video_game>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
                StarCraft has a professional competition.
        </video_game>
        <video_game></video_game>
</games>
```

games element 안에 video_game 이라는 이름의 element 를 하나 더 추가하였습니다. 참고로 <video_game></video_game> 과 같이 content 자리에 아무것도 없는 element 를 가리켜 empty(빈) element 라고 부르는데, <video_game/> 이라고 축약하여 쓸 수도 있습니다.

두 번째 video_game element 안에도 첫 번째 video_game element 와 같이 title 과 developer element 를 추가하여 줍시다.

```
<games>
        <video_game>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
                StarCraft has a professional competition.
        </video_game>
        <video_game>
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

새 element 들을 추가하면서, XML 이 tree 구조를 가지고 있다는 것을 눈치채신 분이 있으실 것 같군요. 위의 예제를 tree 로 표현하면 다음과 같습니다.



XML 이 tree 구조를 가지고 있다는 것은 큰 장점입니다. table 형태의 데이터만 저장할 수 있는 database 에 비해 XML 은 좀 더 다양한 구조의 데이터들을 저장할 수 있습니다. 다음은 XML 의 tree 구조를 나타내기 위한 몇 가지 용어에 대한 설명입니다.

tree 가 하나의 root node 를 갖듯이, XML 문서는 반드시 단 하나의 **root element(최상위 요소)**를 가져야 합니다. 위의 XML 문서의 root element 는 games element 입니다.

element 와 그 element 의 content 사이의 관계를 부모-자식 관계라고 합니다. element 는 content 의 **parent(부모)**라고 하고, content 는 element 의 **child(자식)**라고 합니다. 위의 XML 문서에서 games element 는 video_game element 의 parent 이고, title element 는 video_game element 의 child 입니다. root element 는 parent 가 없는 element 라고도 할 수 있습니다.

parent 의 parent, parent 의 parent 의 parent…와 같은 관계를 **ancestor(조상)**라고 하고, child 의 child, child 의 child 의 child…와 같은 관계를 **descendant(자손)**라고 합니다. games element 는 developer element 의 ancestor 이고, developer element 는 games element 의 descendant 입니다.

parent 가 같은 content 끼리의 관계를 **sibling(형제)**이라 합니다. 첫 번째 video_game element 와 두 번째 video_game element 는 서로 sibling 관계입니다.

2.5 Attribute

element 에 content 를 추가하는 것 말고도, XML 문서에 데이터를 추가하는 방법이 한 가지가 더 있습니다. 바로 attribute(속성)입니다. attribute 를 이용해서 이전 절의 문서에 게임이 발매된 연도에 관한 정보를 추가할 수 있습니다.

```
<games>
        <video_game release="1998">
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <video_game>
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

굵게 표시된 글자가 새로 작성된 attribute 입니다. attribute 는 element 의 여는 태그 이름 옆에 attribute_name="value"와 같은 형태로 적히며, 태그의 이름과 attribute 는 공백으로 구분됩니다. attribute 는 그 attribute 가 적혀있는 element 에 관한 데이터를 뜻하게 됩니다. 위의 문서는 "첫 번째 video_game element 에는 release 라는 속성이 있는데, 그 값은 1998 이다"와 같은 의미를 가집니다.

한 element 는 여러 개의 attribute 를 가질 수도 있습니다. 이번에는 게임의 장르에 관한 정보를 추가해 봅시다.

```
<games>
        <video_game release="1998" genre='Real-time strategy'>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <video_game>
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

여러 개의 attribute 는 공백을 이용하여 서로 구분됩니다. 위의 문서에서와 같이 큰 따옴표(" ")외에 작은 따옴표(' ')도 쓸 수 있습니다. 한 가지 주의할 점은 한 element 안에는 같은 이름을 가진 attribute 가 여러 개 있어서는 안 된다는 것입니다. 이제 두 번째 video_game element 에도 attribute 를 추가해 봅시다.

```
<games>
        <video_game release="1998" genre="Real-time strategy">
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <video_game release="2003" genre="First Person Shooter">
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

두 번째 비디오 게임의 발매 연도는 2003 년이고, 장르는 1 인칭 슈팅 게임이라는 것을 알 수 있습니다.

3. XML 문서 검색하기

여태까지는 새로운 XML 문서를 작성하는 데에만 집중하였으니, 이제 관심을 이미 존재하는 XML 문서를 다루는 쪽으로 돌려봅시다. 친구가 게임에 관한 데이터를 모두 정리한 XML 문서를 건네주었습니다. 그 문서에서 개발사가 Blizzard Entertainment 인 게임들의 이름을 얻어내고 싶습니다. 그런데 문서가 이전 장에서 본 것처럼 몇 줄 안 된다면 한눈에 봐도 이름을 찾을 수 있겠지만, 만약 문서가 수백, 수천 줄이라면 어떨까요? 어떻게 그 많은 내용 중 내가 알고 싶은 데이터만 뽑아낼 수 있을까요? 이번 장에서는 XML 문서를 '검색'할 수 있게 해주는 두 가지 방법에 대해서 알아보고자 합니다.

3.1 XPath

XPath 는 XML Path Language 의 약자로, XML 문서에서 특정한 부분을 선택하는 언어입니다. W3C 에서 1999 년에 1.0 버전을 배포하였고 2007 년에 2.0 버전을 배포하였는데, 여기서는 좀 더 널리 쓰이는 1.0 버전을 기준으로 설명하겠습니다.

XPath 는 절대적 경로와 상대적 경로, 축약되지 않은 문법과 축약된 문법을 제공하는데, 지금은 절대적 경로와 축약된 문법에 대해서만 알아보도록 하겠습니다. 자세한 내용이 궁금하신 분은 reference 의 문서들을 보시기 바랍니다.

```
<games>
        <video_game release="1998" genre='Real-time strategy'>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <video_game release="2003" genre="First Person Shooter">
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

위의 문서는 이전 장에서 보았던 문서입니다. 이제 이 문서에서 원하는 정보를 얻기 위해 어떠한 표현을 써야 하는지에 대해 알아봅시다.

모든 비디오 게임에 관한 정보를 찾고 싶어요!

비디오 게임에 관한 정보를 얻으려면 video_game element 를 찾아야 합니다. video_game element 는 games element 의 자식이므로, **/games/video_game** 이라는 표현으로 문서를 검색할 수 있습니다. slash(/)의 왼쪽은 parent element, 오른쪽은 child element 를 뜻합니다. games element 의 왼쪽에는 아무것도 없으므로 root element 임을 알 수 있습니다. 문서에는 video_game element 가 두 개 이므로, 검색 결과가 두 개 나오겠군요.

문서 전체에서 게임의 이름들만 찾고 싶어요!

게임의 이름은 title element 가 가지고 있습니다. 특정 element 의 자식이 아니라 문서 전체에서 찾으려 하므로, **//title** 이라는 표현을 쓸 수 있습니다. '//'는 오른쪽 element 가 왼쪽 element 의 descendant 임을 뜻합니다. 여기서는 왼쪽에 아무것도 없으니 root element 부터 모든 descendant 를 검색하겠다는 뜻입니다.

두 번째 비디오 게임에 대한 정보를 찾고 싶어요!

문서를 보니 video_game element 가 두 개 있군요. 이렇게 같은 이름을 가진 sibling element 가 여러 개 있을 경우, 위에서부터 순서대로 index 를 붙여 검색할 수 있습니다. 이때의 index 는 1 부터 시작합니다. 따라서 **/games/video_game[2]** 라고 검색하면 원하는 결과를 얻을 수 있습니다.

첫 번째 비디오 게임의 발매 연도를 알고 싶어요!

발매 연도는 release 라는 이름의 attribute 로 저장되어 있습니다. attribute 를 검색할 때는 이름 앞에 '@'를 붙여주면 됩니다. **/games/video_game[1]/@release** 라고 검색하면 원하는 결과를 얻을 수 있습니다.

두 번째 비디오 게임의 이름과 개발사를 알고 싶어요!

비디오 게임의 이름과 개발사 정보는 각각 title 과 developer element 가 가지고 있습니다. 두 element 를 각각 검색해도 되지만, 한꺼번에 검색하는 방법도 있습니다. **/games/video_game[2]/*** 라고 검색하면, 두 번째 video_game element 의 모든 child element 를 찾아줍니다. 여기서 '*'(asterisk)는 'all'을 뜻합니다. 살짝 응용해서 /games/video_game[2]/@* 라고 검색하면, 두 번째 video_game element 의 모든 attribute 를 찾아줄 것입니다.

개발사가 Blizzard Entertainment 인 게임의 이름을 알고 싶어요!

일단 "Blizzard Entertainment" 라는 내용을 가진 developer element 를 child 로 가지는

video_game element 를 찾아야 합니다. 그리고 찾는 것이 '게임의 이름'이니 그 video_game element 의 child 중 title element 를 찾아야 합니다. 이를 XPath 표현으로 나타내면 **/games/video_game[developer="Blizzard Entertainment"]/title** 이라고 할 수 있습니다. 이와 같이 각괄호([])안에 조건식을 넣을 수 있습니다.

3.2 E4X

E4X(ECMAScript for XML)는 ActionScript 등의 언어에서 쓸 수 있는 확장 언어로, XML 을 int 나 char 같은 primitive type 으로 취급하여 XML 을 쉽게 다룰 수 있게 해줍니다. XML 문서를 추가, 수정, 편집하는 등 다양한 기능을 갖고 있지만 여기서는 E4X 의 검색 기능에 대해서만 간략하게 소개하도록 하겠습니다. 아래 표는 이전 절에서 소개한 XPath expression 과 동일한 기능을 하는 E4X expression 을 나타냅니다.

XPath expression	E4X expression
/games/video_game	**games.video_game**
//title	**games..title**
/games/video_game[2]	**games.video_game[1]**
/games/video_game[1]/@release	**games.video_game[0].@release**
/games/video_game[2]/*	**games.video_game[1].***
/games/video_game[developer="Blizzard Entertainment"]/title	**games.video_game.(developer=="Blizzard Entertainment").title**

E4X 에 대해 자세한 내용이 궁금하신 분들은 reference 의 문서나 기타 다른 문서들을 참고하시기 바랍니다.

4. XML 문법 – 좀 더 깊게

2 장에서 소개한 기초 XML 문법만 알아도 XML 문서를 이해하고 사용하는데 큰 어려움은 없지만, 특수한 경우를 처리하거나 부가적인 정보를 추가하기 위해 몇 가지 더 알아야 할 문법들이 존재합니다. 이번 장은 그러한 문법들에 대해 다루고 있습니다.

4.1 Escaping character, CDATA, Comment

여태까지 봐왔던 games 문서에 'Command & Conquer' 라는 이름을 가진 게임에 관한 정보를 넣으려 합니다. 이제 여러분들은 다음과 같이 XML 문서를 작성할 수 있을 것입니다.

```
<games>
        <video_game release="1995" genre="Real-time strategy">
                <title>Command & Conquer</title>
                <developer>Westwood Studios</developer>
        </video_game>
</games>
```

그러나 애석하게도 위 문서에는 한 가지 문제가 존재합니다. XML 에서 ampersand('&') 문자는 특수한 용도를 위해 예약되어 있는 기호이기 때문에 content 자리에 그냥 써서는 안되기 때문입니다. 이렇게 예약되어 있는 기호를 content 자리에 쓰기 위해 **escaping character** 라는 것이 존재합니다. 다음은 XML 에서 예약되어 있는 기호들과 escaping character 들입니다.

기호	Escaping character
& (ampersand)	&amp;
< (less-than sign)	&lt;
> (greater-than sign)	&gt;
' (apostrophe)	&apos;
" (double quotation mark)	&quot;

escaping character 를 이용하여 위의 문서를 바르게 고치면 다음과 같습니다.

```
<games>
        <video_game release="1995" genre="Real-time strategy">
                <title>Command &amp;amp; Conquer</title>
                <developer>Westwood Studios</developer>
        </video_game>
</games>
```

XML parser 는 content 를 읽어가다가 '&' 나 '<' 같은 예약어를 만나면 그 다음에 나오는 문자는 content 가 아닌 다른 문자라고 해석합니다. 그래서 simple content 를 'Parsed Character DATA', 줄여서 PCDATA 라고 부릅니다. parser 가 content 를 일일이 해석하지 않고 문자 그대로 받아들이게 하기 위해 **CDATA(Character DATA) section** 이라는 것을 사용할 수 있습니다.

```
<games>
        <video_game release="1995" genre="Real-time strategy">
                <title><![CDATA[Command & Conquer]]></title>
                <developer>Westwood Studios</developer>
        </video_game>
</games>
```

이전 문서를 CDATA section 을 사용하여 고친 것입니다. 굵게 표시한 부분이 CDATA section 인데, "<!CDATA[" 로 시작하여 "]]>" 로 끝납니다. XML parser 는 CDATA section 안에 있는 글자를 해석하지 않고 그냥 문자 그대로 받아들입니다. 따라서 CDATA section 안에 예약어를 그대로 적을 수 있습니다. escaping character 를 너무 많이 써야 할 경우에 유용합니다.

XML 문서에 주석을 적는 것도 가능합니다. 다음은 위의 문서에 주석을 추가한 것입니다.

```
<games>
        <video_game release="1995" genre="Real-time strategy">
<!-- Date of modifying: 2010/10/4 -->
                <title><![CDATA[Command & Conquer]]></title>
                <developer>Westwood Studios</developer>
        </video_game>
</games>
```

굵은 글씨가 주석문입니다. 주석은 "<!--"로 시작해서 "-->"로 끝납니다. 태그 안에 주석을 쓸 수는 없으며, 주석의 내용 안에 "--" 문자열이 존재해서는 안됩니다.

4.2 XML Declaration

친구로부터 건네 받은 XML 문서를 XML parser 에 입력하여 parsing 해보고자 합니다. 이 때 XML parser 는 XML 문서 자체에 대한 몇 가지 정보를 알아야 할 필요가 있습니다. 이때 쓰이는 것이 XML declaration 입니다. 다음은 이전 절의 문서에 XML declaration 을 추가한 것입니다. 제일 윗줄이 XML declaration 의 한 예입니다.

```
<?xml version='1.0' encoding='UTF-8' standalone='yes'?>
<games>
        <video_game release="1995" genre="Real-time strategy">
                <title>Command &amp;amp; Conquer</title>
                <developer>Westwood Studios</developer>
        </video_game>
</games>
```

XML declaration 을 작성하는 규칙은 다음과 같습니다.

- "<?xml"로 시작해서 "?>"로 끝나야 합니다.
- version 속성은 필수이지만 encoding 과 standalone 속성은 안 써도 됩니다.
- 속성은 반드시 version, encoding, standalone 의 순서대로 쓰여야 합니다.
- XML declaration 은 반드시 XML 문서의 가장 첫 부분에 쓰여야 합니다.

이제 XML declaration 의 세 가지 속성이 무엇을 뜻하는지 살펴보겠습니다. 먼저 version 속성은 XML 권고안의 버전을 뜻합니다. version 속성의 값이 1.1 인 문서를 XML 1.0 parser 가 읽어 들이면 parser 는 에러를 표시합니다.

encoding 속성은 XML parser 가 XML 문서를 읽을 때 어떠한 encoding 을 사용할 것인가에 관한 속성입니다. encoding 속성을 작성하지 않으면 XML parser 는 문서를 기본적으로 UTF-8 이나 UTF-16 으로 encoding 하게 됩니다.

standalone 속성은 해당 XML 문서가 다른 파일에 있는 element 나 그림, 개체 등을 참조하고 있는지 아닌지를 나타냅니다. 이 속성의 값은 반드시 "yes", "no" 둘 중 하나여야 합니다. 값이 "yes"라면 문서가 외부 파일에 의존하고 있지 않다는 것을 의미하고, "no"라면 외부 파일에 의존하고 있음을 의미합니다.

4.3 Namespace

여태까지의 XML 문서에서는 비디오 게임에 관한 데이터만 다루었는데, 이번에는 아케이드 게임에 관한 데이터를 추가하려고 합니다. 다음은 'Arkanoid' 라는 이름의 아케이드 게임에 대한 데이터를 추가한 XML 문서입니다.

```
<games>
        <video_game release="1998" genre="Real-time strategy">
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <arcade_game release="1986" genre="Breakout clone">
                <title>Arkanoid</title>
                <developer>Taito</developer>
        </arcade_game>
</games>
```

여러분은 이 문서를 친구에게 주었고, 친구는 이 문서에서 게임의 이름에 대한 정보만 얻으려 한다고 가정해 봅시다. 문서 전체에서 'title' 이라는 이름을 가진 element 를 검색하면 <title>StarCraft</title>, <title>Arkanoid</title> 라는 두 개의 결과를 얻을 수 있을 것입니다. 그런데 어떤 것이 비디오 게임의 이름이고, 어떤 것이 아케이드 게임의 이름인지 어떻게 알 수 있을까요?

불행히도 위의 문서에서는 두 title element 를 서로 구분할 수 없습니다. element 의 이름을 작성자가 마음대로 지을 수 있다는 것은 XML 의 장점 중 하나이지만, 그 때문에 위와 같이 element 이름은 같지만 그 의미가 다른 경우가 발생하기도 합니다. 이러한 문제를 해결하기 위한 것이 바로 **XML Namespace** 입니다.

다음은 XML namespace 를 이용하여 비디오 게임과 아케이드 게임을 서로 구별할 수 있게 작성한 XML 문서입니다.

```
<games xmlns:video="http://en.wikipedia.org/wiki/Video_game"
xmlns:arcade="http://en.wikipedia.org/wiki/Arcade_game">
    <video:video_game video:release="1998" video:genre="Real-time strategy">
        <video:title>StarCraft</video:title>
        <video:developer>Blizzard Entertainment</video:developer>
    </video:video_game>
    <arcade:arcade_game arcade:release="1986" arcade:genre="Breakout clone">
        <arcade:title>Arkanoid</arcade:title>
        <arcade:developer>Taito</arcade:developer>
    < arcade:arcade_game>
</games>
```

element 와 attribute 이름 앞에 'video'와 'arcade'라는 접두사(prefix)를 각각 붙여두었습니다. 접두사와 이름 사이에는 colon(':')을 적어 서로 구분합니다.

'접두사를 이용하여 element 를 서로 구분한다는 것 까지는 알겠는데, games element 의 attribute 자리에 적혀있는 xmlns:video="http://en.wikipedia.org/wiki/Video_game" 같은 건 대체 뭐지?' 라는 의문이 들 수 있습니다. 바로 이 부분이 namespace 에서 가장 중요한 부분입니다.

element 들을 서로 구분하기 위해 접두사를 사용하였는데, 그 접두사마저 똑같다면 어떻게 될까요? 이런 경우를 방지하기 위해 접두사로 이미 존재하는 고유한 인터넷 주소인 URL(Uniform Resource Locator)을 사용할 수 있습니다. 그런데 URL 은 그대로 접두사로 사용하기에는 너무 길고 번거로우므로, xmlns attribute 를 이용하여 글자 수를 줄이는 것입니다.

xmlns:video="http://en.wikipedia.org/wiki/Video_game" 은 '지금부터 namespace 접두사로 video 라는 것을 쓸 것인데, 사실 video 는 http://en.wikipedia.org/wiki/Video_game 과 같은 뜻이다' 라는 의미를 가집니다. 마치 C 언어의 typedef 문과 같은 역할을 합니다. 이렇게 선언된 namespace 접두사들은 접두사가 선언된 element 와 그 element 의 descendant 들에서만 쓸 수 있습니다.

namespace 는 어떤 element 가 어떤 그룹에 속해있는지를 알려주는 것 이외의 의미를 가지지 않습니다. <arcade:title> 이라는 표현은 그 element 의 이름이 arcade:title 이라는 것을 뜻하는 것이 아니라, title 이라는 이름을 가진 element 가 arcade 라는

그룹에 포함되어 있다는 것을 의미합니다. 이것은 C:\Users 에 있는 myfile.txt 와 C:\Program Files 에 있는 myfile.txt 는 같은 이름을 가졌지만 전혀 다른 파일이라는 것과 비슷한 개념입니다.

구분을 위해 접두사를 쓰는 것 까지는 좋은데, 문서가 약간 지저분해 보일 수 있습니다. 이런 경우에 **default namespace** 를 사용할 수 있습니다

```
<games>
        <video_game release="1998" genre="Real-time strategy"
        xmlns="http://en.wikipedia.org/wiki/Video_game">
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <arcade_game release="1986" genre="Breakout clone"
        xmlns="http://en.wikipedia.org/wiki/Arcade_game">
                <title>Arkanoid</title>
                <developer>Taito</developer>
        <arcade_game>
</games>
```

xmlns attribute 뒤에 아무 접두사도 적지 않으면 해당 namespace 는 그것이 선언된 element 와 그 element 의 모든 descendant 들의 기본 namespace 로 인식됩니다. 위의 문서는 직전의 문서와 같은 의미를 가진다고 볼 수 있습니다.

그런데 "이름이 같은 element 를 구분하기 위해 namespace 를 도입했는데, default namespace 를 써서 접두사를 없애면 결국 element 의 이름은 여전히 같은 것 아닌가?" 하는 의문이 들 수 있습니다. 실제로 XPath 는 default namespace 를 제대로 처리하지 못합니다. 그러므로 namespace 를 본래의 의미에 충실하게 사용하려면 default namespace 를 사용하지 않는 것이 좋습니다. 대신에 default namespace 는 XML Schema Definition(XSD) 등의 응용 기술을 위한 XML 문서에서 유용하게 쓰입니다. 5.2 절의 XML Schema Definition(XSD) 설명에서 default namespace 의 쓰임새를 볼 수 있습니다.

5. XML Schema Language

■ Valid XML document

2 장과 4 장에서는 XML 문법에 대해 알아보았습니다. 이제 우리는 어떤 XML 문서가 XML 문법에 맞는지 컴퓨터를 이용해 판별할 수 있습니다. 그렇다면 그 XML 문서가 무엇에 관한 데이터를 뜻하는지에 대해서도 컴퓨터가 알아낼 수 있을까요? 사람은 태그의 이름과 내용을 보고 XML 문서가 무엇에 관한 것인지 짐작할 수 있지만, 컴퓨터가 그렇게 하기는 쉽지 않습니다. 그 대신에 컴퓨터는 **XML schema language** 라는 것을 사용합니다.

XML schema language 는 XML 문서가 어떤 구조로 되어있는지, element 는 몇 개를 가지고 있는지, element 의 이름은 무엇인지, element 의 attribute 와 content 로는 어떤 것이 들어가야 하는지 등을 규정하는 데에 쓰입니다. XML schema 를 이용하여 어떤 XML 문서를 검사한 결과 규정된 구조에 맞게 되어있다고 판별되면 그 문서를 **valid (유효한) XML 문서**라고 부릅니다.

이 장에서는 가장 많이 쓰이는 XML schema language 세 가지에 대해서 예제를 통해 간략하게 소개하도록 하겠습니다. 자세한 내용이 궁금하시면 reference 문서들을 참고하시기 바랍니다.

5.1 DTD

DTD 는 Document Type Definition 의 약자로, W3C 에서 발표한 XML 1.0 버전 권고안에 포함되어 있는 XML schema language 입니다.

```
<games>
        <video_game release="1998" genre='Real-time strategy'>
                <title>StarCraft</title>
                <developer>Blizzard Entertainment</developer>
        </video_game>
        <video_game release="2003" genre="First Person Shooter">
                <title>Call of Duty</title>
                <developer>Infinity Ward</developer>
        </video_game>
</games>
```

위의 XML 문서를 'valid' 하다고 판별하는 DTD 문서를 작성하면서 DTD 문법에 대해 알아봅시다.

```
<!DOCTYPE games []>
```

가장 먼저 적어야 할 것은 DOCTYPE 선언입니다. 위의 DOCTYPE 선언은 root element 의 이름이 games 인 XML 문서에 대한 DTD 를 작성한다는 의미를 가집니다. 이후에 적을 모든 DTD 내용은 DOCTYPE 선언의 대괄호('[]') 안에 들어가게 됩니다.

이제 첫 번째로 root element 인 games element 를 정의하는 문장을 적어봅시다. XML 문서에서 games element 는 두 개의 video_game element 를 가지고 있습니다. 애석하게도 DTD 는 이름이 같은 element 의 개수를 지정하는 문법을 가지고 있지 않으므로, games element 는 0 개 이상의 video_game element 를 가질 수 있다고 가정합시다. 다음은 이를 DTD 로 표현한 것입니다.

```
<!DOCTYPE games [
        <!ELEMENT games (video_game)*>
]>
```

element 정의는 <!ELEMENT element_name (content)> 와 같은 형태로 적습니다. '*'(asterisk)는 그 앞의 내용이 0 번 혹은 그 이상 반복될 수 있음을 뜻합니다.

이제 video_game element 가 어떤 모양을 가져야 하는지 정의해 봅시다. video_game element 는 content 로 title 과 developer element 를 가지고 있습니다.

```
<!DOCTYPE games [
        <!ELEMENT games (video_game)*>
        <!ELEMENT video_game (title, developer)>
]>
```

video_game element 의 모양을 정의하였습니다. 그런데 XML 문서를 보니 video_game element 는 attribute 도 가지고 있군요. attribute 정의는 다음과 같은 방법으로 할 수 있습니다.

```
<!DOCTYPE games [
        <!ELEMENT games (video_game)*>
        <!ELEMENT video_game (title, developer)>
        <!ATTLIST video_game release CDATA #REQUIRED>
]>
```

video_game element 의 두 attribute 중 release attribute 에 대한 정의입니다. attribute 정의는 <!ATTLIST element_name attribute_name type optional> 과 같은 모양으로 적습니다. 위의 DTD 예제에서 'CDATA'는 release attribute 의 값은 그냥 string type 임을 의미하고, '#REQUIRED'는 video_game element 는 꼭 release attribute 를 가지고 있어야 한다는 것을 의미합니다.

video_game element 의 두 번째 attribute 인 genre attribute 에 대한 정의를 적으려면 어떻게 해야 할까요? attribute 정의가 <!ATTRIBUTE… 가 아닌 <!ATT**LIST**… 로 시작한다는 것을 눈치채신 분들이라면 짐작이 가능하실 것 같군요.

```
<!DOCTYPE games [
        <!ELEMENT games (video_game)*>
        <!ELEMENT video_game (title, developer)>
        <!ATTLIST video_game release CDATA #REQUIRED
                genre CDATA #REQUIRED>
]>
```

genre attribute 를 추가한 모습입니다. release attribute 정의 바로 다음에 이어서 쓸 수 있습니다.

마지막으로 title element 와 developer element 의 정의를 적어봅시다. 두 element 모두 content 로는 simple content 만을 가지고 있습니다.

```
<!DOCTYPE games [
        <!ELEMENT games (video_game)*>
        <!ELEMENT video_game (title, developer)>
        <!ATTLIST video_game release CDATA #REQUIRED
                genre CDATA #REQUIRED>
        <!ELEMENT title (#PCDATA)>
        <!ELEMENT developer (#PCDATA)>
]>
```

5 장에서 simple content 는 PCDATA 라고도 한다고 했었습니다. 따라서 위와 같이 element 의 정의를 적을 수 있습니다.

이제 위의 DTD 선언을 XML 문서 파일의 맨 위(XML declaration 이 있을 경우 XML declaration 과 root element 의 사이)에 적어 넣고, 그 문서를 DTD 로 validation check 가 가능한 XML parser 에 입력하면 XML 문서가 valid 한지 아닌지 판별할 수가 있습니다.

5.2 XML Schema Definition(XSD)

DTD 는 XML 권고안에 유일하게 포함되어 있는 XML schema language 이기 때문에, 이전 절에서 보았듯이 별도의 파일이 아닌 XML 문서 자체에 포함시킬 수 있습니다. DTD 는 이런 점 이외에도 몇 가지 장점으로 인해 지금까지도 널리 쓰여왔지만, 몇 가지 해결 불가능한 단점도 가지고 있습니다. 다음은 DTD 의 단점들입니다.

▪ DTD 문법은 XML 문법과 다릅니다. 따라서 아주 작고 가볍게 구현된 XML parser 는 XML 문서의 DTD 부분을 그냥 무시해버릴 수도 있습니다.
▪ 매우 복잡한 XML 문서의 구조를 나타낼 경우, DTD 문서를 한눈에 봐서는 어떤 구조인지 알아보기가 힘들 수 있습니다. 이전 절에서 보았던 DTD 문서 예제를 떠올려 봅시다. DTD 문서가 그 예제의 열 배 정도 되는 길이라면, 그 문서가 어떤 구조를 나타내고 있는지 빠르게 알아내기란 쉽지 않을 것입니다.
▪ DTD 는 다소 간단한 문법을 가지고 있기 때문에, 구체적인 구조를 나타내기가 힘듭니다.
▪ 데이터 형식에 대한 표현이 부족합니다. year 라는 속성이 있다면 이 속성은 그 값으로 숫자만 가져야 하지만, 이전 절에서 보았듯이 DTD 는 이러한 경우를 표현할 수가 없습니다.
▪ DTD 는 XML namespace 에 관한 spec 이 완성되기 이전에 만들어졌기 때문에 namespace 를 제대로 처리하지 못합니다.

이러한 단점들을 해결한 XML schema language 가 바로 XML Schema Definition, 약자로는 XSD 입니다(여기서의 "Schema"는 항상 대문자로 시작해야 합니다). 이전 절에서 본 XML 문서를 valid 하다고 판별하는 XSD 문서 예제를 작성하면서 XSD 문법에 대해 알아봅시다.

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

위의 예제는 가장 먼저 써야 할 부분으로, DTD 의 DOCTYPE 선언과 비슷한 역할을 합니다. xmlns 는 namespace 를 선언할 때 쓰는 속성이라는 것을 4 장에서 보았습니다. 그러므로 위 예제는 schema 라는 root element 가 있고 그 element 는 "http://www.w3.org/2001/XMLSchema" 라는 default namespace 에 속해있다는 것을 나

타냄을 알 수 있습니다. 이는 이 XML 문서가 XSD 문서라는 것을 알려줍니다. elementFormDefault="qualified" 속성은 이 XSD 문서로 판별할 XML 문서의 element 와 attribute 들이 namespace 와 결합되어 있다는 것을 뜻합니다. 지금은 XSD 문서에 필수적으로 써야 하는 속성이라는 정도로만 알고 넘어갑시다.

element 와 attribute 들의 정의는 모두 이 schema element 내부에 적힙니다. 역시 root element 인 games element 에 대한 정의부터 적어봅시다.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <element name="games">
      <complexType></complexType>
   </element>
</schema>
```

XSD 문서 구조는 XML 문법을 그대로 따르기 때문에 좀 더 직관적입니다. 위의 예제에서 굵게 표시한 부분을 읽어보면 'games 라는 이름을 가진 element 에 대한 정의구나' 라고 쉽게 추측할 수 있습니다. games element 의 content 에 대한 정의는 'element' element 의 content 자리에 적히게 됩니다. complexType element 는 games element 가 content 로 element 를 가지고 있음을 뜻합니다.

XML 문서에서 games element 는 video_game element 를 두 개 가지고 있는데, XSD 에서는 element 개수를 자유롭게 지정할 수 있으므로, video_game element 가 최소 0 번에서 최대 2 번까지 나타날 수 있다고 가정해 봅시다.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <element name="games">
      <complexType>
         <element name="video_game" minOccurs="0"
maxOccurs="2">
            <complexType></complexType>
         </element>
      </complexType>
   </element>
</schema>
```

minOccurs attribute 는 해당 element 가 최소 몇 번 이상 나타나야 하는지를 지정하고, maxOccurs attribute 는 최대 몇 번까지 나타날 수 있는지를 지정합니다. attribute 를 적지 않을 경우 기본값은 1 입니다. video_game element 도 content 로 element 를 가지고 있으므로 complexType element 를 적어줍니다.

video_game element 는 title 과 developer element 를 content 로 가지고 있습니다. 이를 XSD 로 표현하면 다음과 같습니다.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <element name="games">
      <complexType>
         <element name="video_game" minOccurs="0"
maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="title" type="string"/>
                  <element name="developer" type="string"/>
               </sequence>
            </complexType>
         </element>
      </complexType>
   </element>
</schema>
```

sequence element 는 그 안에 적혀있는 element 들이 순서대로 XML 문서에 등장해야 한다는 것을 뜻합니다. element 정의에 적혀있는 type attribute 는 element 가 simple content 만을 가질 경우 그 content 가 어떤 형식을 가지고 있는지를 지정합니다. 따라서 title 과 developer element 의 content 는 string 형식입니다. 형식 지정에 관한 문법이 거의 없는 DTD 와는 달리 XSD 는 정수형, 실수형은 물론 날짜, 시간 등 여러 형식을 사용하여 content 모양에 대해 제한을 둘 수 있습니다.

마지막으로 video_game element 의 두 attribute 에 관한 정의를 해봅시다.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <element name="games">
      <complexType>
         <element name="video_game" minOccurs="0"
maxOccurs="unbounded">
            <complexType>
               <sequence>
                  <element name="title" type="string"/>
                  <element name="developer" type="string"/>
               </sequence>
               <attribute name="release" type="gYear" use="required"/>
               <attribute name="genre" type="string" use="required"/>
            </complexType>
         </element>
      </complexType>
   </element>
</schema>
```

type attribute 는 위에서 설명한 것과 같이 attribute 값에 대한 형식을 지정합니다. release attribute 는 그 값으로 연도만을 가지기 때문에, XSD 권고안에서 정의하는 연도 형식인 'gYear'를 type 으로 적어두었습니다. use attribute 는 이 attribute 가 꼭 존재해야 하는지 아닌지를 나타냅니다. 값이 required 이므로 video_game element 는 release 와 genre attribute 를 반드시 가지고 있어야 합니다.

이제 이 XSD 문서 이름을 games.xsd 라 하고, 이를 이용하여 XML 문서를 검증하려면 XML 문서의 root element 에 다음과 같이 attribute 를 추가해 주어야 합니다.

```
<games  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="games.xsd">…</games>
```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" attribute 는 이 XML 문서가 XSD 로 검증할 수 있는 문서임을 알려줍니다.
xsi:noNamespaceSchemaLocation="games.xsd" attribute 는 이 문서가 games.xsd 파일에 의해 검증될 것임을 뜻합니다.

5.3 RELAX NG

XSD 는 많은 기능을 가지고 있지만, 그만큼 배우고 사용하기가 어렵습니다. 이전 절에서 봤듯이 같은 내용을 가지고 있는 XML schema 문서임에도 불구하고 DTD 로는 몇 줄 안 되는 내용이 XSD 로는 상당히 길어집니다. DTD 처럼 짧고 간단하게 쓸 수 있으면서 DTD 보다 많은 기능을 가진 XML schema language 가 없을까 하는 생각이 든다면, RELAX NG 를 써볼 수 있습니다.

RELAX NG 는 Regular Language for XML Next Generation 의 약자로, XSD 와 흡사한 모양의 문법을 제공하는 동시에, 훨씬 직관적이고 간단한 compact syntax 도 함께 제공합니다. 여기서는 compact syntax 를 이용하여 이전 절의 XML 문서를 valid 하다고 판단하는 RELAX NG 문서를 작성해 보도록 하겠습니다.

```
element games {}
```

RELAX NG compact syntax 에서는 DTD 의 DOCTYPE 선언이나 XSD 의 schema element 같은 선언이 필요 없습니다. 그냥 바로 root element 정의부터 작성하면 됩니다. 위의 예제는 games 라는 element 가 존재해야 한다는 것을 뜻합니다. games element 의 content 에 대한 정의는 중괄호({})안에 적습니다.

```
element games {
        element video_game {}*
}
```

RELAX NG compact syntax 는 상당히 직관적입니다. 위의 예제를 보면 'games element 안에 video_game element 가 있구나' 라고 어렵지 않게 추측할 수 있습니다. '*'(asterisk)는 DTD 에서와 같이 video_game element 가 0 번 혹은 그 이상 반복될 수 있음을 뜻합니다.

이제 video_game element 에 attribute 정의를 추가해 봅시다. attribute 정의는 element 정의와 거의 흡사한 모양을 가지고 있습니다.

```
element games {
        element video_game {
                attribute release { xsd:gYear },
                attribute genre { text }
        }*
}
```

video_game element 정의에 release 와 genre attribute 정의를 추가하였습니다. attribute 정의의 중괄호('{}')안에는 attribute 값의 형식이 들어갑니다. release attribute 는 연도를 값으로 가지므로 중괄호 안에 xsd:gYear 라고 적었고, genre attribute 는 일반 문자열을 값으로 가지므로 text 라고 적었습니다. xsd:gYear 형식은 XSD 설명에서 봤듯이 XSD 권고안에서 정의하는 데이터 타입입니다. RELAX NG 에서는 XSD 의 데이터 타입을 사용할 수 있습니다.

```
element games {
        element video_game {
                attribute release { xsd:gYear },
                attribute genre { text },
                element title { text },
                element developer { text }
        }*
}
```

video_game element 안에 title 과 developer element 정의를 추가하면 schema 문서가 완성됩니다. XSD 는 물론 DTD 보다도 짧고 쉬운 문법을 가지고 있음을 알 수 있습니다.

RELAX NG 는 DTD 나 XSD 와는 달리 XML 문서와 RELAX NG 문서를 연결시키는 방법에 대한 문법이나 제한이 없습니다. 대신 RELAX NG 를 지원하는 parser 나 프로그램에서 문서 연결에 대한 설정을 한 후 XML 문서의 validation 을 체크해볼 수 있습니다.

6. XML API

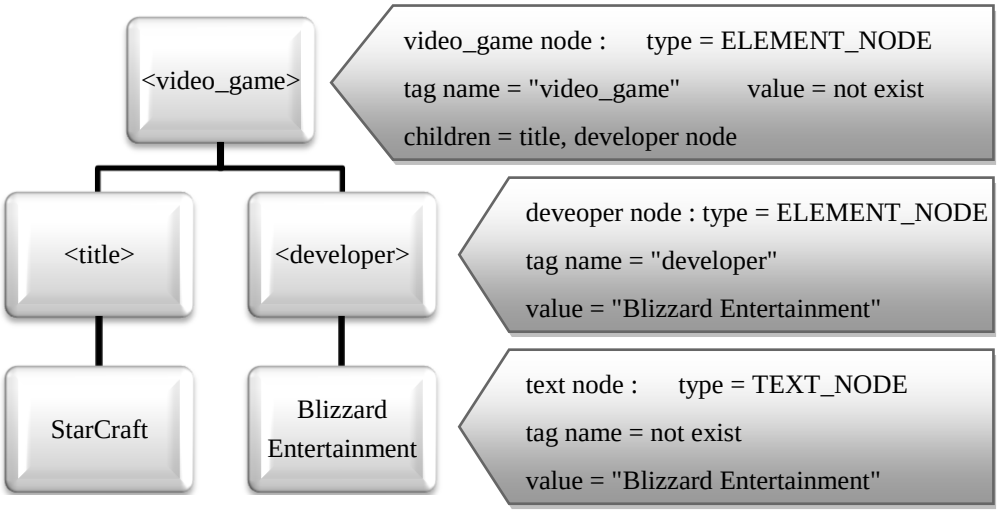
XML 은 인간과 어플리케이션, 혹은 어플리케이션과 어플리케이션 사이의 데이터 교환을 쉽게 하기 위해 만들어졌습니다. 이는 인간뿐만 아니라 컴퓨터도 XML 문서를 읽고 해석할 수 있어야 한다는 뜻입니다. 컴퓨터가 XML 문서를 이해할 수 있게 도와주는 API(Application Programming Interface)가 몇 종류 있는데, 그 중에서 가장 유명한 DOM 과 SAX 에 대해서 간단히 소개하도록 하겠습니다.

6.1 DOM

DOM 은 Document Object Model 의 약자입니다. DOM 이 XML 문서를 어떻게 처리하는지 예제를 통해 알아봅시다.

```
<video_game>
        <title>StarCraft</title>
        <developer>Blizzard Entertainment</developer>
</video_game>
```

위의 XML 문서를 DOM 방식으로 만들어진 XML parser 가 읽어 들였다고 가정해 봅시다. parser 는 위의 문서를 다음과 같은 tree 구조의 객체로 만들어 메모리에 저장합니다.



이제 이 XML 문서의 데이터를 이용할 어플리케이션은 DOM parser 에게 node 단위의 데이터 객체를 달라고 요청할 수 있습니다(예를 들면 'developer element node 에 대한 정보를 달라' 라는 식으로). 이렇게 어플리케이션이 parser 에게 직접 어떤 데이터를 달라고 요청한다는 뜻에서 DOM parser 를 pull parser 라고도 합니다.

DOM 은 XML 문서를 전부 메모리에 올려놓고 작업을 시작하기 때문에, 원하는 데이터가 어떤 부분에 있든지 바로 접근이 가능하고, 데이터를 추가, 수정하거나 삭제하는 것도 가능합니다. 반면, 맨 처음에 XML 문서를 전부 읽어서 메모리에 올려야 하기 때문에 초기 구동 속도가 느리고, XML 문서가 아주 큰 경우 메모리 공간이 부족할 수도 있습니다.

6.2 SAX

SAX 는 Simple API for XML 의 약자로, XML 문서를 순차적으로 읽어서 처리하는 방식입니다. 이전 절의 XML 문서 예제를 SAX 방식의 parser 로 처리한다면 다음과 같습니다.

start_element : video_game
start_element : title
character : StarCraft
end_element : title
start_element : developer
character : Blizzard Entertainment
end_element : developer
end_element : video_game

SAX 는 XML 문서를 하나의 string 으로 인식하여 처음부터 쭉 읽어나갑니다. 그러다가 여는 태그를 만나면 '여기 여는 태그가 있다!' 라고 어플리케이션에 알려주고, character(simple content)를 만나면 '여기 character 가 있다!' 라고 알려주고, 닫는 태그를 만나면 '여기 닫는 태그가 있다!' 라고 알려주는 방식으로 작업을 진행합니다. 이렇게 차례대로 일을 진행하다가 뭔가 특별한 일이 생기면 다른 component 에 이 일을 보고하는 방식을 event-driven 방식이라고 합니다.
parser 가 알아서 어플리케이션에게 데이터를 밀어주기 때문에 SAX parser 를 push parser 라고도 합니다.

SAX 는 문서를 처음부터 끝까지 읽어나가기만 하면 되기 때문에 메모리를 사용할 필요가 없습니다. 따라서 문서를 처리하는 속도가 빠르고, 매우 큰 XML 문서를 처리하는 데에도 어려움이 없습니다. 하지만 문서를 차례대로 읽기 때문에 내가 원하는 부분의 데이터만 얻어오는 일이 불가능하고, 읽기 이외의 작업은 불가능하기 때문에 문서를 수정할 수가 없습니다. 또한 사용 방법이 DOM 에 비해 다소 어렵습니다.

DOM 과 SAX 는 각각 상반되는 장단점을 가지고 있기 때문에, 원하는 작업의 특성을 잘 생각하여 어떤 방식의 parser 를 고를 것인지 판단해야 합니다.

지금까지 XML 이 어떤 기술이고, 어떻게 다룰 수 있는지에 대해 이야기해보았습니다. 이 문서가 여러분들이 XML 을 이해하는 데에 조금이나마 보탬이 되었기를 바랍니다.

Appendix A. 문서정보

A.1 문서이력

■ 문서목적

XML 에 대해 이해하는 것을 돕는다.

■ 작성자

이혜린/ salad@novonetworks.com

■ 작성이력

최초작성 : 2010-10

Appendix B. References

Beginning XML 2nd, 3rd Edition 한국어판

Hunter, Watt, Rafter, Duckett, Ayers, Chase, Fawcett, Gaven, Patterson

XML 의 개념과 다양한 응용 기술을 소개하는 책입니다.

Extensible Markup Language (XML) 1.0 (Fifth Edition)

W3C Recommendation 26 November 2008

http://www.w3.org/TR/xml/

W3C 의 XML 1.0 권고안입니다.

Namespaces in XML 1.0 (Third Edition)

W3C Recommendation 8 December 2009

http://www.w3.org/TR/xml-names/

W3C 의 XML 1.0 namespace 권고안입니다.

Extensible Markup Language (XML) 1.1 (Second Edition)

W3C Recommendation 16 August 2006, edited in place 29 September 2006

http://www.w3.org/TR/xml11/

W3C 의 XML 1.1 권고안입니다.

XML Tutorial

http://www.w3schools.com/xml/default.asp

XML 문서 작성법을 알 수 있는 tutorial 입니다.

XML Path Language (XPath)

W3C Recommendation 16 November 1999

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

W3C 의 XPath 1.0 권고안입니다.

E4X Tutorial

https://developer.mozilla.org/en/E4X_Tutorial

mozilla developer center 의 E4X tutorial 페이지입니다.

DTD Tutorial

http://www.w3schools.com/dtd/default.asp

DTD 문서 작성법을 알 수 있는 tutorial 입니다.

XML Schema Part 0: Primer Second Edition

W3C Recommendation 28 October 2004

http://www.w3.org/TR/xmlschema-0/

W3C 의 XSD 권고안입니다.

XML Schema Tutorial

http://www.w3schools.com/schema/default.asp

XSD 문서 작성법을 알 수 있는 tutorial 입니다.

Interactive XML tutorials

http://xmlzoo.net/

DTD 와 XSD 를 이용하여 직접 XML 문서의 validation 을 체크해볼 수 있습니다.

RELAX NG home page

http://relaxng.org/

RELAX NG 의 홈페이지입니다. specification 문서와 문법 tutorial 등을 볼 수 있습니다.

다음은 이 문서에서는 소개하고 있지 않은 XML 관련 기술에 대해 궁금해 하실 분들을 위한 문서입니다.

The Extensible Stylesheet Language Family (XSL)

http://www.w3.org/Style/XSL/

XSL(Extensible Stylesheet Language)의 홈페이지입니다. 권고안, tutorial 등 다양한 문서들을 볼 수 있습니다. 읽기 전에 HTML 에 대해 미리 익혀두시는 것이 좋습니다.

Cascading Style Sheets - Learning CSS

http://www.w3.org/Style/CSS/learning

CSS(Cascading Style Sheets)의 tutorial 페이지입니다. 역시 읽기 전에 HTML 에 대해 미리 익혀두시는 것이 좋습니다.