Chakra®

DB 감시 및 성능 분석 툴의 혁신 ㈜웨어밸리의 Chakra

글 | **김범** ㈜웨어밸리 수석연구원 bkim@warevalley.com

원활한 데이타베이스 운영을 위해 전문 모니터링 툴을 사용하는 것은 필수적이다. 그러나, 이런 모니터링 툴의 연결과 작동이 DB 시스템에 또 다른 큰 부담이 된다면?

㈜웨어밸리의 Chakra는 데이타베이스 시스템이나 그 어떠한 시스템에 접속하거나 데이타를 요청하는 일 없이 성능 관리와 SQL 감사 작업을 성공적으로 수행할 수 있다. 특히, 웨어밸리의 또 다른 개발품 Orange for Oracle과 연동하면, 오라클 개발자와 DBA에게 최상의 DB 모니터링 툴이 될 것이다.

DB 시스템의 또 다른 부담, 모니터링 툴

데이타베이스를 감시하기 위해 많은 기업이나 전산 담당자들이 사용하고 있는 모니터링 툴들은 모니터링을 위해 데이타베이스 시스템에 직접 혹은 간접 연결을 통하여 성능 자료를 검출하고 있다. 그러나, 이러한 성능 정보 검출 형태 자체가 업무 시스템에 영향을 주게 되는데, 시스템 운영자들은 당연히 그럴 수밖에 없을 것이라 생각하며 그에 순응하게 된다.

한편, DBMS에 대한 전문지식이 날로 깊어지면서 이러한 모니터링 툴은 더 많은 전문적인 기능을 보유하여 수많은 성능 정보를 운영자들에게 제공하게 된다. 그러나, 모니터링 툴이 성능 정보를 많이 제공하면 할수록 데이타베이스 시스템은 빌링이나 고객관리, 금융 데이타 처리 등 고유의 업무 외에 성능 정보 제공이라는 일을 더 많이 함으로써 자신의 성능을 100% 발휘하지 못할 뿐 아니라 모니터링 툴에 대한 부담으로 자체 운영 정보에 대한 성능 정보가 정확하지 않게 된다. 데이타베이스 시스템의 정밀한 오류분석을 위해서는 모니터링 툴의 사용이 불가피하지만, 실제 데이타베이스 시스템의 성능을 좌우하는 것의 90% 이상이 데이타베이스 시스템으로 요청되는 SQL의 품질이며 나머지가 기타 운영 파라미터나 환경적 요소라 할 수 있다.

SQL 감사, 고객들이 바라는 또 하나의 요구사항

개발중인 시스템이나 이미 개발이 완료되어 운영중인 시스템을 관리하기 위해 중요한 두 가지 항목을 얘기한다면, 하나는 앞서 얘기한 성능 관리이고 다른 하나는 감사(auditing)의 역할이다. 고객의 시스템에 치명적인 영향을 줄 수 있는 데이타베이스 변경을 추적할 수 있어야 하며, 비정상적인 단계로 접근하거나 고객의 데이타를 조회하는 등 보안상의 이유, 혹은 데이타를 처리함에 필요한 인증 절차를 거치지 않은 경우에 대한 사후 추적이 고객들이 요구하는 중요한 사항이라 할 수 있다.

하지만, 기존의 모니터링 툴들이 직접 혹은 간접적인 연결을 통하여 초당 수백, 수천의 트랜잭션이 처리되는 데이타베이스 시스템에 요청되는 모든 SQL에 대한 100% 감사를 수행할 수 있는가? 대답은 No일 것이다. 만일 기존의 방식처럼 그런 기능을 제공하고 있다면, 고유 업무에 대한 성능을 저버리는 일이 될 것이다.

시스템에 대한 부담을 꺼리는 고객을 위하여...

실제 OLTP 성격의 운영계에는 수천, 수만 건의 트랜잭션들이 처리되고 있으며, 이러한 모든 처리는 당연 SQL을 통해 이루어진다고 볼 수 있다. 이 중에 자주 쓰이는 단 몇 개의 SQL만이라도 악성 SQL로 자주 사용된다면 시스템의 전체적인 성능은 당연지사 추락하게 된다. 혹은 예전에 좋은 성능을 발휘하던, 전문가에 의해 작성된 SQL도 매번 변하는 데이타의 분

포나 통계에 따라 성능이 언제 저하될지 모르는 일이다. 사실 이러한 문제는 고객이 운영하는 시스템에서 지금도 발생하며, 이러한 이유로 주기적인 컨설팅 의뢰가 이루어지고 있는 것이다.

그렇다면, 기존 방식처럼 데이타베이스 시스템에 직접 혹은 간접 연결을 함으로써 초당 수백, 수천 건의 트랜잭션들이 처리되는 업무 시스템들에 '모니터링'이라는 명분으로 또 다른 부하를 주는 것은 고객들이 제일 안타까워 하는 부분이 아닐까?. 고객의 시스템이 과중한 업무나 비정상적인 경우로 인해 힘들어 할 때 모니터링 그 자체가 시스템에 대한 부담을 가중시키는 안타까운 현실이 계속되고 있다.

필자가 어느 회사의 DBA 업무를 맡고 있는 고객으로부터 다음과 같은 얘기를 들은 적이 있다. "데이타베이스 시스템이 이상할 때는 일단 먼저 모니터링 툴부터 다운시킵니다." 모니터링 툴이 전문화될수록 많은 성능 정보를 데이타베이스 시스템에 요청함으로써 더 많은 부담을 안긴다는 것이다. 웃지 못할 고객의 사정이 아닌가.

고객 시스템에 부담을 주는 모니터링 툴은 가라

데이타베이스 시스템의 성능을 결정하는 최대 요소는 요청되는 SQL의 품질이며, 모든 SQL에 대한 성능을 하나도 빠짐없이 측정할 수 있을 때 어떠한 SQL이 부하를 많이 발생시키는지 혹은 전체적으로 언제 데이타를 가장 바쁘게 처리하고 무엇이 문제의 원인이 되는지를 파악할 수 있다. 이렇게 추적된 SQL의 성능 평가는 기존의 모니터링 툴이 제공하는 어떠한 문제 원인보다 운영자가 훨씬 쉽게 접근할 수 있는 자료가 된다.

그리고, 데이타베이스 시스템에 요청되는 모든 SQL을 100% 기록, 보관함으로써 데이타베이스 변경에 대한 이력이나 보안에 대한 완충적인 역할을 수행하게 된다.

웨어밸리의 Chakra는 이 모든 기능을 수행할 수 있다. 그리고, 중요한 것은 고객이 운영하는 데이타베이스 시스템은 Chakra의 존재를 모른다는 것이다. 즉, Chakra는 데이타베이스 시스템이나 그 어떠한 시스템에 접속하거나 데이타를 요청하는 일이 없다는 것이다. 이것은 기존의 모니터링 툴이 데이타베이스 시스템에 부하를 주는 것과는 전혀 다른 개념이며 다른 접근 방법이다. 지금부터 이 Chakra에 대한 소개를 하고자 한다. 또한, 그 어떠한 시스템에도 접속하지 않기에 Chakra가 가지는 장점 이외에 기능적 한계를 웨어밸리에서 개발한 성능 관리 및 개발 툴인 Orange와의 연계를 통하여 극복하며 고객이 아쉬워하는 '성능 관리와 SQL 감사' 라는 두 마리 토끼를 고객들과 함께 잡을 수 있다.

Chakra
: DB 성능에 미치는 영향 제로, SQL 감사와 성능 분석은 100%

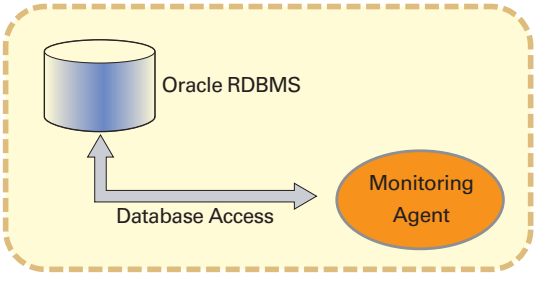
기존 모니터링 툴의 문제점

기존의 모니터링 툴이 가지는 방식은 크게 두 가지로 볼 수 있으며, 이들 문제점은 다음과 같이 시스템이나 네트워크 부하를 가중시켰다.

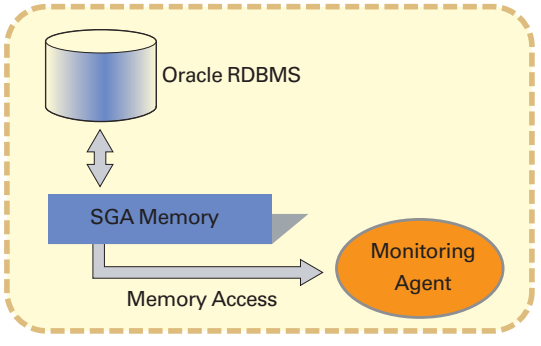
첫째, 데이타베이스가 존재하는 시스템에 에이전트를 운영하여 에이전트를 통해 데이타베이스 시스템의 성능 정보를 추출하는 방식인데, 이 경우 에이전트가 데이타베이스에 직접적으로 접속하는 방식〈그림 1〉 혹은 SGA 메모리에 접근하여 간접적으로 성능 정보를 읽는 방식〈그림 2〉이 있다.

둘째, SQL*Net을 통하여 일반 클라이언트들과 동일한 방식으로 원격으로 데이타베이스에 직접 접속하여 성능 정보를 추출하는 방식이다〈그림 3〉.

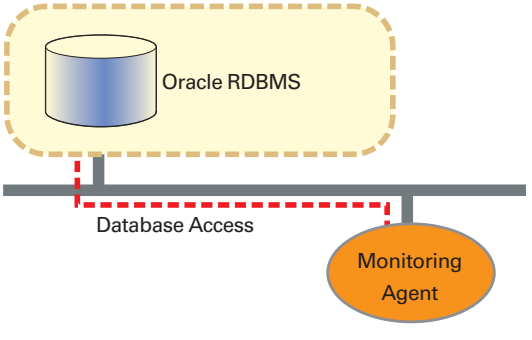
그러나, 이들 3가지 방식 모두 다음과 같은 문제를 가지고 있다.



〈그림 1〉 데이타베이스에 대한 직접 접속 방식



〈그림 2〉 SGA 메모리를 이용한 간접 접속 방식



〈그림 3〉 SQL*Net을 통한 직접 접속 방식

◉ **직접 접속 방식**
- Oracle Database 시스템에 대한 부하 증가
- CPU 부하 증가
- 메모리 사용 증가
- 에이전트 설치에 따른 운용자 부담 증가
- 에이전트의 비정상적인 활동으로 인한 위험 증가

◉ **SGA 메모리를 이용한 간접 접속 방식**
- CPU 부하 증가
- 메모리 사용 증가
- 에이전트 설치에 따른 운용자 부담 증가
- 에이전트의 비정상적인 활동으로 인한 위험 증가

◉ **SQL*Net을 통한 직접 접속 방식**
- Oracle Database 시스템에 대한 부하 증가
- CPU 부하 증가
- 메모리 사용 증가
- 네트워크 사용량 증가

이상은 기존의 모든 모니터링 툴이 가지는 일반적인 형태이다. 어떠한 방식이든 데이타베이스 시스템에 부하를 주게 되고 더 나아가 데이타베이스에 접속하기 위해 필요한 네트워크 용량에 많은 부담을 주게 된다. 훌륭한 모니터링 기능을 제공하고자 더욱더 많은 성능 정보를 수집할수록 그 부담은 증가하게 되며, 결국 그 부담은 고객이 운영하는 업무 시스템으로 돌아간다. 또한, 구조상 요청되는 SQL을 모두 다 잡을 수 없게 되어 있어 데이타베이스 변경에 대한 이력이나 비인증 절차를 거친 처리 과정을 추적할 수 없게 된다.

네트워크 패킷 분석을 통한 DB 성능 관리 및 SQL 감사

Chakra는 네트워크 패킷 분석을 통한 DB 성능 관리 도구로서, 네트워크상에서 데이타베이스에 접근하는 모든 패킷들을 관리/분석하여 모든 성능 이슈를 조기에 발견하며 다양한 분석 정보를 제공한다〈그림 4〉. 기존의 모니터링 툴이 서버에 직접/간접으로 접속하여 시스템에 상당한 부하를 주는 반면, Chakra는 네트워크상에서 흘러다니는 패킷 정보만을 수집하여 고객이 운영하는 시스템에 전혀 부하가 없다.

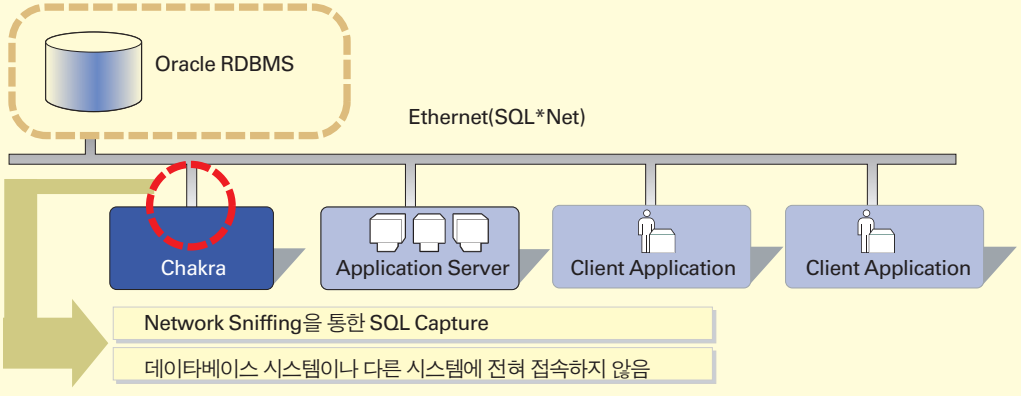
Chakra는 데이타베이스에 요청되는 모든 SQL과 그와 관련된 정보를 모두 저장하여 분석할 수 있는데, 세션정보, 사용자 응답시간의 최소/최대값 및 평균값 등의 통계자료, 작업처리율, 네트워크상에서 전송된 패킷 개수/패킷량 또는 에러, 데이타베이스 접속 정보 등을 분석하여 자주 실행되거나 자원을 과도하게 사용하는 악성 SQL을 추출해낸다. 또한 데이타베이스 성능 튜닝을 위한 주요 정보 및 자료를 제공하는 것뿐만 아니라 누가, 언제, 어디서, 어떠한 SQL을 요청했는지 등에 대한 모든 SQL 정보를 관리하므로 오류나 위반 행위를 조치하고 신속히 역추적할 수 있는 감사 역할을 수행할 수 있다.

Chakra가 제공하는 특징은 다음과 같다.

◉ 에이전트 형태로도 원격 접속 형태로도 DB에 접속하는 경우가 전혀 없다.
◉ 모든 SQL들을 감시하고 기록할 수 있다.
◉ 데이타베이스 시스템이나 네트워크에 전혀 부하를 주지 않는다
◉ 실시간으로 성능을 분석하고 보고한다.
◉ 성능상의 문제나 감사상의 문제 발생시 즉시 경고 조치할 수 있다
◉ Orange for Oracle과의 자동연계를 통하여 자세한 모니터링 및 문제점을 파악할 수 있다.

Chakra

◉ No Database Connectivity
◉ No Server Side Agent :
◉ No Middleware to connect other systems
◉ No Communication between CHAKRA and other systems
◉ 0% Impact



〈그림 4〉 네트워크 패킷 분석을 통한 DB 성능 관리 및 SQL 감사 - Chakra

- 100 % SQL Log

Chakra가 제공하는 대표적인 기능은 다음과 같다.

- SQL Audit
- SQL Monitor
- Alert Monitor
- Report
- Session Monitor
- Statistics Monitor
- Log Analyzer

Chakra가 감시할 수 있는 오라클 버전 및 시스템은 다음과 같다.

- ORACLE 7, 8, 8*i*, 9*i*, 9*i* R2
- Unix Series : SUN, HP, IBM AIX, Alpha Server
- Linux
- Windows Series

Chakra 운영 플랫폼은 다음과 같다.

- 하드웨어 : Pentium III 이상의 CPU 장착 서버(2CPU 추천)
- 플랫폼 : Windows NT, 2k Server
- 네트워크 : Ethernet
- SQL 캡처 : 1~10,000 SQL per second
- 디스크 용량 : 수십 KB ~ 20GB (운영시스템의 TPS나 로깅 정책에 따라 다름)
- 모니터링 인스턴스 수 : 1 Chakra : N 인스턴스

오라클 개발자와 DBA를 위한 도구, Orange for ORACLE과의 연동

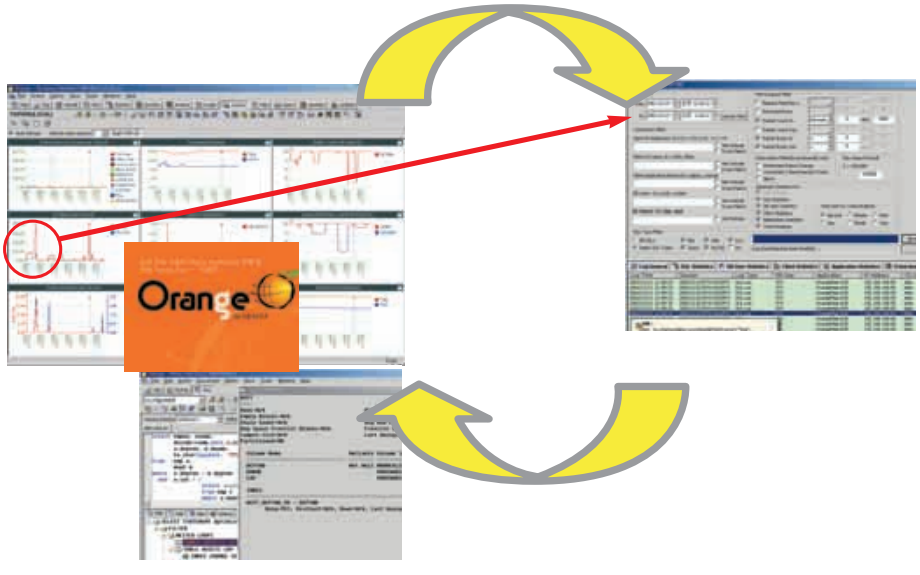
문제가 발생했을 때 가끔씩 해결 원인을 쉽게 찾지 못하고 이를 해결하기 위해 데이타베이스의 안쪽을 일일이 들여보아야 할 경우가 생길 수도 있다. 하지만 이러한 경우가 성능 향상을 위해 주기적으로 해야 할 일은 아닐 것이다.

쉽게 얘기해서 데이타베이스의 성능이 좋다 나쁘다 이렇게 판단할 수 있는 최초의 판단 기준은 고객들에 대한 처리 응답시간이 중요할 것이다. 데이타베이스 내부에서 일어나는 여러 가지 성능 이벤트들이나 데이타 처리 단위, 경로, 단계 이런 것들을 따지는 것은 문제점 추적시 차후의 좀더 세분화된 과정일지 모르는 성능 분석 단계일 뿐이며, 이를 주기적으로 기존의 모니터링 방식으로 처리하는 것은 지금까지 고객이 어쩔 수 없이 시스템에 부하를 주면서 당연하게 여겼던 부분이다.

Chakra는 모든 SQL들을 기록 보관하고 데이타베이스에 접속하지 않기 때문에 데이타베이스 내부의 내용을 볼 수 없다. 그러나, 자사의 Orange For Oracle과의 자동연계를 통하여 기존의 모니터링 툴이 문제점을 파악하는 것보다 훨씬 손쉽게 찾을 수 있다.

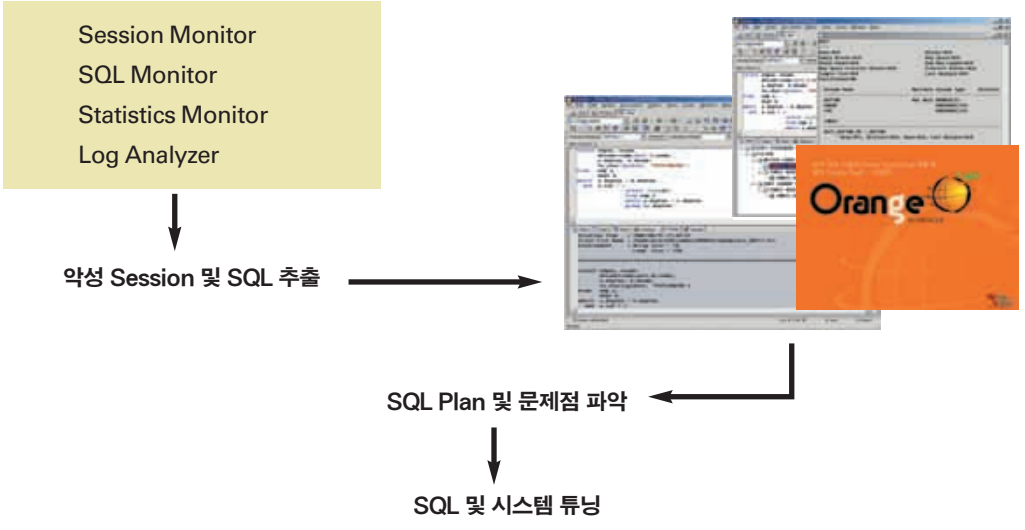
따라서 Chakra를 운영하다가 데이타베이스 시스템에 문제를 발생시키는 SQL을 지적하고 이에 대한 보다 정밀한 분석을 위해 Orange와 연동할 수 있다. 혹은, 반대로 Orange로 데이타베이스 내부나 성능추이 상황을 분석하다가 문제가 되는 시점에 요청되었던 모든 SQL을 Chakra를 통해 역추적할 수 있다. 따라서, 아무런 부하 없이 데이타베이스를 기록, 감시하다가 필요시에 Orange와 연동하여 정밀 분석을 수행할 수 있다<그림 5>, <그림 6>.

요청된 SQL 역 추적 과정



<그림 5> Chakra와 Orange의 연동 – 악성 SQL 세부 분석

악성 SQL 세부 분석 과정



〈그림 6〉 Chakra와 Orange의 연동 – 요청된 SQL 역추적 과정

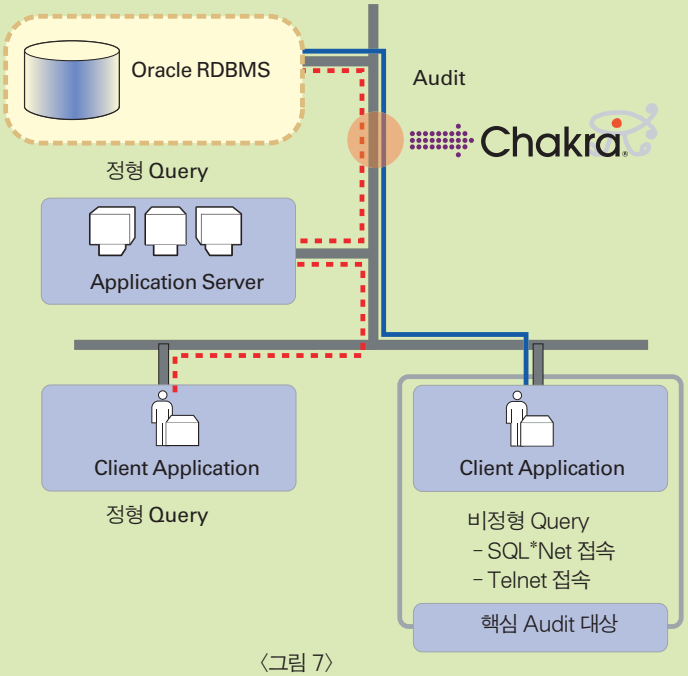
Chakra의 기능별 이용 사례

다음은 실제 고객이 Chakra를 이용하는 사례를 들어보도록 하겠다.

사례 1 : 감사

고객들이 가장 아쉬워하는 부분 중의 하나가 감사(auditing)일 것이다. 시스템에 아무런 부하 없이 요청되는 SQL들을 감사하기를 원하는데, Chakra는 앞서 말한 것과 같이 아무런 부하 없이 모든 SQL을 기록, 분석한다.

고객의 요구에 따라 시스템으로 요청되는 모든 SQL을 기록할 경우도 발생하고, 운영의 여러 가지 상황에 따라 실제 필요한 부분만 기록할 필요가 발생할 것이다. 〈그림 7〉처럼 운영체제에 애플리케이션 서버나 정형(이미 애플리케이션에 의해 정해진) 질의를 모두 기록하는 경우에 불필요한 기록 데이타가 쌓이므로 비정형 질의만을 기록함으로써 운영체제에 대한 감사 기능을 충실히 수행할 수 있다.



〈그림 7〉

사례 2 : SLA 지원

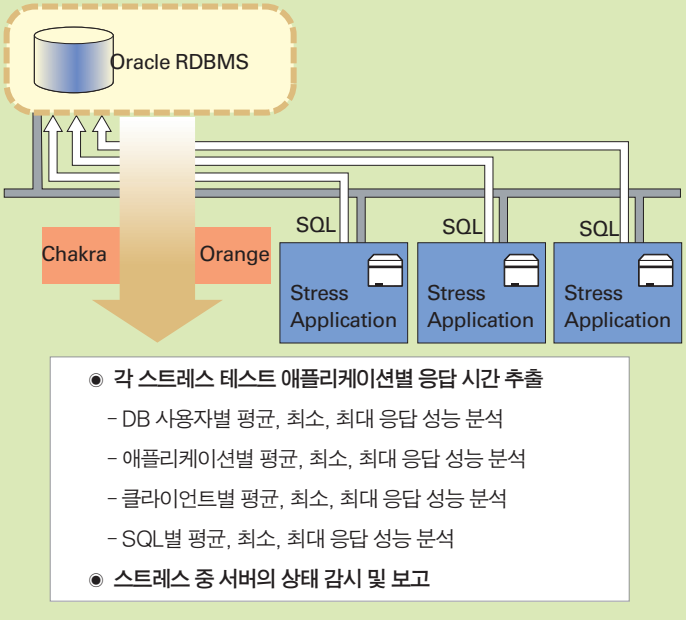
Chakra가 제공하는 Session Monitor, SQL Monitor, Alert Monitor, Statistics Monitor와 온라인 모니터나 Log Analyzer와 같은 오프라인 데이타 분석을 통해 언제 어느 시점에 문제가 되었던 요소를 빠른 시간 내에 도출할 수 있다.

데이타베이스 시스템이 개발중인 것도 있으며 이미 개발이 완료되어 운영중인 시스템도 존재한다. 각 경우에 시스템의 상황이나 문제점의 대상이 틀리게 된다. 관리자의 경우 이미 운용중인 시스템이 얼마나 잘 실행되고 있는지 성능상의 문제가 없는지 이러한 것들이 주요 관리 이슈가 될 것이며, 개발자의 경우 어떠한 SQL이 문제가 되고 영향을 주는지 등 실제 운영 전에 개발 및 성능 향상에 관련된 요소가 주요 이슈가 될 것이다. Chakra는 이러한 요구사항을 받아 들이기에 충분한 도구이다〈그림 8〉.

〈그림 8〉 Chakra의 SLA 지원 사례

사례 3 : 스트레스 테스트 지원

많은 시스템들은 시스템을 오픈하기 전에 실제 운용과 비슷한 경우나 최악의 상황을 고려하기 위해 스트레스 테스트를 하곤 한다. 이러한 스트레스 테스트 없이 오픈한다면, 실제 운용시에 많은 지장을 초래할 수 있으므로 고객들은 각 요소별로 성능 수치를 얻고자 한다. Chakra는 접속 유저별, 애플리케이션별, 요청된 SQL별, 접속 호스트별 성능과 추이 상황을 분석해 낼 수 있다. 이는 순수하게 데이타베이스 시스템과 데이타베이스에 접근하는 클라이언트나 애플리케이션 서버와의 환경간의 성능을 측정해야 하며, 측정시 순수 업무 외에 다른 요소가 성능에 지장을 주어서는 안 된다. Chakra는 이와 같은 상황에서 아무런 부하 없이 어느 시스템에도 접속하지 않고 고객이 오픈하고자 하는 시스템이 얼만큼의 성능 발휘를 하는지에 대한 자료를 제공해 준다〈그림 9〉.



〈그림 9〉 Chakra스트레스 테스트 지원

Chakra Functional List

:: Session Monitor
- 세션 정보 표시(호스트, 사용자, 클라이언트 IP, 애플리케이션 프로그램, 사용 시간...)
- SQL(Query, DML, DDL, DCL, PLSQL) Log List 필터링 검색
- 각 SQL별 상세 정보(returned rows, bytes, response time, server processing time...)
- DML, DDL 등 사용량(개수) 변화 표시
- 평균 응답 시간 및 기타 처리 시간
- 패킷 인/아웃(패킷 카운트, 바이트) 추이 분석
- Top Session Monitor

:: SQL Monitor
- 각 SQL별 상세 정보(returned rows, bytes, response time, server processing time..)
- Top SQL Monitor
- Running SQL/Top SQL(요청 개수별, 응답 시간별, 리턴 로우 수 별 등)

:: Statistics monitor
- 인스턴스별 사용자 접속 수 추이 분석
- 인스턴스의 전체/사용자별/시간대별 패킷 인/아웃(패킷 카운트, 바이트) 추이 분석
- 인스턴스의 전체/사용자별/시간대별 응답 시간/서버 처리 시간 추이 분석
- 피크(비정상적 성능 상태) 타임 분석(Used SQL별 상세정보)
- 세션 카운트의 추이 분석
- LOB Data Usage 흐름 분석

:: Alert monitor
- User-Define Severities
- Alert Colored Text and Image
- E-Mail List (alert 발생시 통보 대상)
- 사운드 : 가청 경보 음 설정
- 액션 : 경보 발생시 조치할 스크립트 등록
- Alert Range Define
- Alert Notification(E-Mail, Sound, Visual)
- Instance Level Alert
- Session Level Alert
- SQL Level Alert

:: Log & Etc
- 사용자 IP, DB 사용자명, 호스트, 인스턴스별 SQL(Query, DML, DDL, DCL, PLSQL) Log
- 항목(응답시간, 세션 수, SQL 요청 수 등)별 피크 타임 분석
- 객체별 접근 사용자 탐색
- 인스턴스별 현재 사용자/세션/SQL 상태 표시.
- Alert & Report, Notification
- Orange Plan & Trace Tool 연동

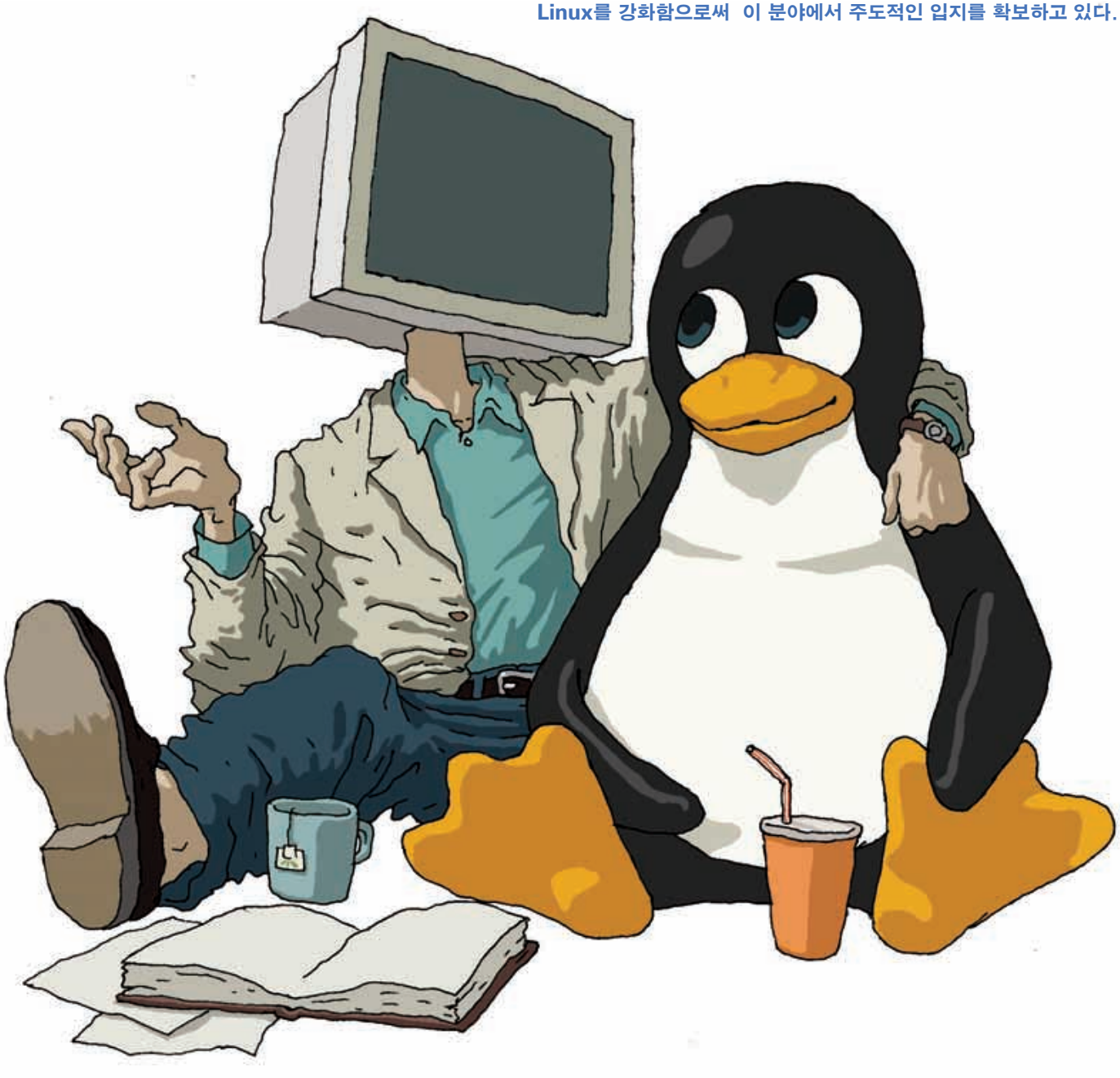
CHAKRA is Powerful with 0 % impact

실제 운용되는 데이타베이스 시스템의 성능은 시스템적인 환경 요소보다 그에게 요청되는 SQL에 따라 많은 차이를 보인다. 다시 말해 90% 이상의 성능 향상 부분을 차지하는 요소, 즉 SQL을 잘 관리할 수 있도록 하는데 Chakra는 그 목적을 둔다. Chakra는 기타 모니터링 툴과는 접근방식에서부터 많은 차이를 보인다. 물론 이들간의 많은 기능적인 차이점이 보이지만 장단을 구별할 성격이 아니다. Chakra는 현재 운용중인 시스템에 부하를 전혀 주지 않으면서 기타 모니터링 툴을 통하여 꾀할 수 있는 시스템의 성능 향상보다 더 좋은 결과를 가져다 준다. 또한, 기존의 모니터링 툴들이 감히 해낼 수 없는 초당 수천, 수만 번의 트랜잭션이 이루어지는 데이타베이스 시스템에 대한 SQL 로그와 성능 측정을 0%의 부하로써 해결한다.

Linux의 중요성과 오라클의 지원

글 | **Ken Jacobs** (오라클 서버 기술 사업부 제품전략 담당 부사장) ken.Jacobs@oracle.com

Linux 발전을 위한 오라클의 여러 가지 시도로 Linux의 범용성이 더욱 향상되고 있다. 오라클은 1999년에 Linux 플랫폼을 위한 최초의 상용 데이타베이스를 출시한 이래, 가용성 및 확장성 측면에서 대기업의 요구사항을 만족시킬 수 있도록 Linux를 강화함으로써 이 분야에서 주도적인 입지를 확보하고 있다.



왜 Linux인가?

최근 오라클은 Linux 클러스터링을 대폭 향상시킬 뿐만 아니라 개발자들에게 Linux 및 파일 시스템과 운용될 수 있는 새로운 기능을 제공하는 일련의 Linux용 라이브러리 및 툴킷을 발표했다. 이러한 새로운 툴은 Linux의 실용성 및 유용성을 현저히 향상시켜 줄 것이다. 그리고, 오라클은 Red Hat Linux Advanced Server 2.1에 Oracle9*i*를 설치한 고객들에게 기술 지원을 제공하고 있다. 이러한 서비스 지원은 기업이 Linux상에서 애플리케이션을 자신 있게 론치할 수 있도록 지원함으로써 중대한 이정표를 수립하게 되었다.

이러한 새로운 기능 및 지원에 대해 세부적으로 논의하기에 앞서, '왜 Linux인가?' 또한 '왜 오라클과 Linux인가?' 라는 몇 가지 근본적인 질문을 던져보고자 한다. 오라클은 주요(및 비주류) 상용 운영체제에 수십 년간의 시간을 투자해 왔다. 그렇다면, 개방형 소스 운영체제를 지원하는 데 이렇게 많은 노력을 기울이는 이유는 무엇일까?

이러한 질문에 대한 답변은 분명하다. 간단히 말해, 시장이 Linux를 원하고 있기 때문이다. 예를 들면, IDC는 Linux 환경에 대한 지출이 2001년 8,000만 달러에서 2006년 28% CAGR(Compound Annual Growth Rate)인 2억 8,000만 달러로 증가할 것으로 예측하고 있다. Gartner Group도 2007년까지 Linux 시장의 매출 규모가 초기 구매 기준으로 보았을 때 총 선적 매출의 18%에 달하는 90억 달러를 웃돌 것이라고 전망하고 있다. 이러한 전망들이 현실로 나타나고 있으며, 오라클이 Linux를 전적으로 지원하는 데 납득할 만한 의미를 부여하고 있다(2002년 6월까지 Oracle9*i* Database Release 2 for Linux 제품이 68,000회 이상 다운로드됐다.)

Linux의 토대이자 오픈 소스의 토대를 이루는 근본 철학이 매우 매력적이라는 점도 간과할 수 없다. 오픈 소스의 이점은 널리 알려져 있다. 오픈 소스는 소규모 개발자 그룹에 의해 사용이 제한되는 전용 시스템이 아니라 대규모 개발자 및 사용자 그룹이 협력하여 오늘날 기업을 보다 효과적으로 운영하는 데 사용할 수 있는 유연한 시스템을 개발 및 수정하는 모델을 의미한다.

Linux에서 이러한 협업 개발 모델을 채택하고 있는 가장 큰 이유는 코어 UNIX 아키텍처의 모듈형 특성 때문이다. Linux 커널은 모든 종류의 유틸리티 프로그램의 기반을 이루며 인터페이스는 공용으로서 널리 사용되고 있다. 따라서 개발자는 다른 컴포넌트에 부정적인 영향을 미치지 않으면서 Linux용 NIC 드라이버로 작업할 수 있다. 완전한 전용 방식의 운영체제(예 : Windows)의 경우, 다른 팀 소속의 개발자들은 전체 작업에 영향을 미치지 않고 특정 부분만을 조작하는 데 어려움을 겪게 된다.

오라클이 Linux에 전력 투구하는 이유는 대규모 메인프레임에서 서버, 데스크탑에 이르기까지 거의 모든 하드웨어 플랫폼에서 Linux가 실행되기 때문이다. 오늘날 이러한 다기능성(versatility)은 거의 유일무이하다고 할 수 있다. 이러한 다기능성을 통해 클러스터링 환경에서 실행됨으로써 Linux는 독보적인 입지를 확보하게 되며, 널리 사용되는 오픈 소스 플랫폼에서의 애플리케이션 설치시 보다 뛰어난 제어 및 유연성을 기업에게 제공함으로써 Linux상에서 운영되는 애플리케이션의 투자 가치를 높여 준다.

이것이 오라클이 Linux를 지원하는 근본적인 이유이다. 그러면, 최근 오라클이 출시한, 클러스터링을 지원하며 Linux 성능을 향상시킨 Linux 라이브러리 및 툴킷에 대해 살펴보고, Oracle9*i* Database에 대한 기능 강화와 Linux상에서의 오라클의 관리 용이성 및 성능을 향상시킨 Linux 커널에 대해서 알아보자.

Linux 클러스터링의 성능 향상

기업은 가용성 및 확장성을 요구하고 있으며, 클러스터링은 가용성 및 확장성 측면 모두에서 중요한 역할을 수행한다. Linux 클러스터링의 특징은 상용 하드웨어상에서 운영되는 클러스터를 처리하는 것으로서, 소규모 기업 및 개인 사용자에게까지 클러스터 사용이 확대되고 있다. 오라클과 Red Hat이 최근 발표한 Linux의 기능 강화를 통해 클러스터 관리 및 성능은 더욱 향상됐다. 가장 주목할 만한 기능은 다음과 같다.

- **클러스터 파일 시스템** *** 클러스터링과 관련하여 가장 중요한 기능 강화는 Linux를 위한 클러스터 파일 시스템의 출시이다. 클러스터 파일 시스템을 사용하기 전에는 디스크를 사용하여 클러스터 요소를 관리했다. 예를 들면, 이전에는 오라클 데이타베이스가 사용하는 모든 데이타파일에 대한 디스크 파티션을 설정해야 했다. 오라클의 e-business 애플리케이션용 데이타베이스를 생각해 보자. 일반적으로 이러한 데이타베이스는 약 230개의 데이타파일을 보유하고 있기 때문에, 230개의 파티션을 생성해야만 했다.

  이와 반대로, 클러스터 파일 시스템은 전체 클러스터의 관리를 대폭 간소화시켰다. 클러스터 파일 시스템을 사용함으로써 클러스터의 모든 디스크를 총괄하여 하나의 대형 파티션을 생성할 수 있으며, 이러한 모든 노드는 간단히 파일 시스템에 액세스할 수 있다. Oracle9*i* Real Application Cluster를 위해 특별히 개발된 이 공유 파일 시스템은 각 노드가 자체 로컬 복사본을 확보할 필요 없이 클러스터 내 모든 노드가 단일의 오라클 홈을 공유할 수 있도록 지원한다. 또한 오라클 홈 파일에서 변경된 사항은 가용성을 저하시키지 않고 클러스터를 통해 동적으로 복사된다.

- **NIC 페일오버** *** NFT(Network Fault Tolerance) 또는 NIC 이중화로 불리는 페일오버 지원을 통해 2개 이상의 서버는 항상 링크를 활성화 상태로 유지할 수 있게 된다. NIC 페일오버는 각 시스템에 호환 가능한 보조 NIC를 설치할 수 있다는 것을 의미하며, 주요 네트워크 커넥션에 장애가 발생한 경우 백업 NIC가 자동적으로 네트워크 커넥션을 인수하여 유지하게 된다. 이제 오라클을 Linux상에서 운영함과 동시에 보조 NIC를 설치함으로써 고가용성을 보장할 수 있게 되었다.

- **I/O 펜싱** *** STOMITH(Shoot The Other Machine In The Head)라 불리는 I/O 펜싱(Fencing)은 여러 노드에 의해 공유되는 파일

시스템의 무결성을 유지할 수 있도록 지원한다. 예를 들면, 하나의 노드가 다른 노드에 '상태'를 알려주는 신호인 '하트비트(heartbeat)'를 보내는 작업을 중단한다고 하자. 이 중단된 노드는 온라인 상태가 되면 파일 시스템을 손상시키거나 클러스터에 영향을 미치게 될 것이다. I/O 펜싱은 하나의 노드에 장애가 발생하면 다른 노드가 장애가 발생한 노드를 파일 시스템으로부터 '차단'하도록 한다. 특히 오라클 제품을 위해 특화된 오라클 I/O 펜스는 장애가 발생한 데이타베이스 노드의 나머지 쓰기 작업이 파일 시스템으로 작성되지 않도록 지원한다.

◉ **와치도그 드라이버 패치** *** 와치도그(watchdog)는 I/O 펜싱을 대체할 만한 로우 레벨 대안이다. 와치도그 프로그램은 각 노드를 모니터링하며 운영체제 또는 다른 프로세스가 응답을 정지한 경우, 시스템을 재시작한다. 오라클은 와치도그 드라이버를 위해 Linux 2.4 커널에 대한 패치를 제공함으로써 이제 와치도그 드라이버는 Linux와 운용될 수 있게 되었다.

◉ **파이어와이어 패치** *** 이들 패치는 Linux상에서의 방화벽 문제를 수정하며 공유 디스크가 방화벽 드라이버상에서 운영될 수 있도록 지원한다. 방화벽을 사용하게 되면 보다 저렴하게 클러스터 시스템을 공유 디스크에 구축할 수 있으며 클러스터링된 애플리케이션을 테스트할 수 있다.

Next Steps >>>>>>>>>>

▹ OTN 가입
otn.oracle.com

▹ 새로운 Linux 라이브러리 액세스
linux.otncast.otnxchange.oracle.com

▹ Oracle9*i* Database 다운로드
otn.oracle.com/software/products/oracle9*i*

자세한 정보

▹ 클러스터 파일 시스템
ocfs.otncast.otnxchange.oracle.com/servlets/ProjectHome

▹ I/O 펜싱
iofence.otncast.otnxchange.oracle.com/servlets/ProjectHome

▹ 와치도그 프로세스
Watchdog.otncast.otnxchange.oracle.com/servlets/ProjectHome

▹ 방화벽
linux1394.otncast.otnxchange.oracle.com/servlets/ProjectHome

▹ Oracle9*i* 설명서
otn.oracle.com/docs/products/oracle9i

관리의 용이성 향상

오라클은 Red Hat과 협력하여 추가 툴 및 퍼실리티를 개발함으로써 Linux에서 운영되는 제품의 관리를 손쉽게 하고자 노력했다. Linux를 위한 관리의 용이성 기능 강화는 다음과 같다.

◉ **Cluster Manager** *** 앞에서 언급한 강화된 클러스터링 기능을 적극 활용하기 위해서 오라클은 Cluster Manager라는 Oracle9*i* Real Application Cluster를 위한 새로운 컴포넌트를 개발했다. Cluster Manager는 모든 클러스터링 서비스를 모니터링하는 데몬이다. 이 데몬을 구성하여 시작하면, 필요한 경우 오라클이 데몬에 액세스할 수 있다.

◉ **Israid 유틸리티** *** Israid는 오라클이 개발한 유틸리티로서, RAID 스토리지 관리를 지원한다. 이 프로그램을 사용함으로써 Linux md 디바이스에 대해 질의하여 디바이스 및 그가 속해 있는 블록에 대한 정보를 얻을 수 있다. Israid를 사용하게 되면 raidtab 파일을 복구할 수 있고 파일 정보의 정확성을 확인할 수 있으며, 수동으로 또는 Israid를 사용하여 디스크가 이름을 변경한 경우 md 구성을 재실행할 수 있다.

◉ **Network Console and Crash Dump Facility** *** Red Hat의 Advanced Server 2.1과 함께 번들로 제공되는 이 툴은 Linux 충돌 신호 메시지를 포함하여 모든 커널 메시지를 네트워크를 통해 중앙집중형 서버로 로깅하는 네트워크 콘솔을 제공한다. 이 툴은 시스템 및 커널 로그에 대한 중앙집중화된 일관된 뷰를 제공함으로써 Linux 관련 문제를 신속하게 해결할 수 있도록 지원한다.

OTN의 Linux Center 및 커뮤니티 코드

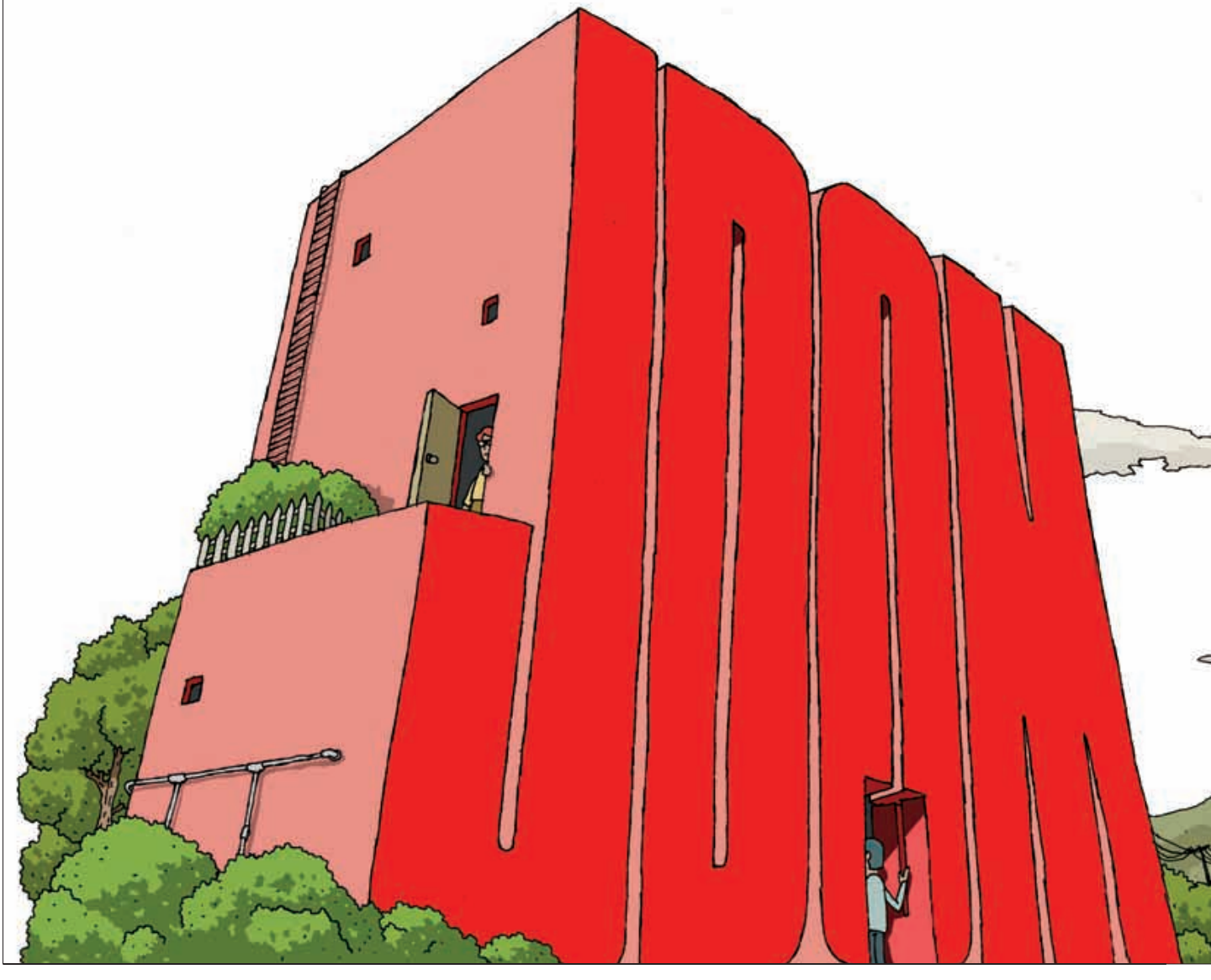
Linux의 기능을 이해하고 활용할 수 있도록 지원하고, 오라클이 Linux 개발자들에게 제공하는 다양한 툴 및 자원을 검색할 수 있는 장소를 제공하기 위해 OTN(Oracle Technology Network)은 최근 Linux Center(otn.oracle.com/linux)를 새로 오픈했다. 이 센터에서는 설치 및 구성 힌트, 관련 다운로드 및 설명서에 대한 링크, Red Hat 소프트웨어에 대한 특별 할인 정보, Linux 관련 서적에 대한 액세스 등을 제공한다.

또한 OTN의 새로운 커뮤니티 코드 섹션에서는 앞서 설명한 Linux용 라이브러리 및 유틸리티에 대한 정보를 제공하며, GPL 개방형 소스 라이선스로 제공되는 모든 라이브러리, 유틸리티 및 코드를 브라우즈하고 다운로드 받을 수 있다.

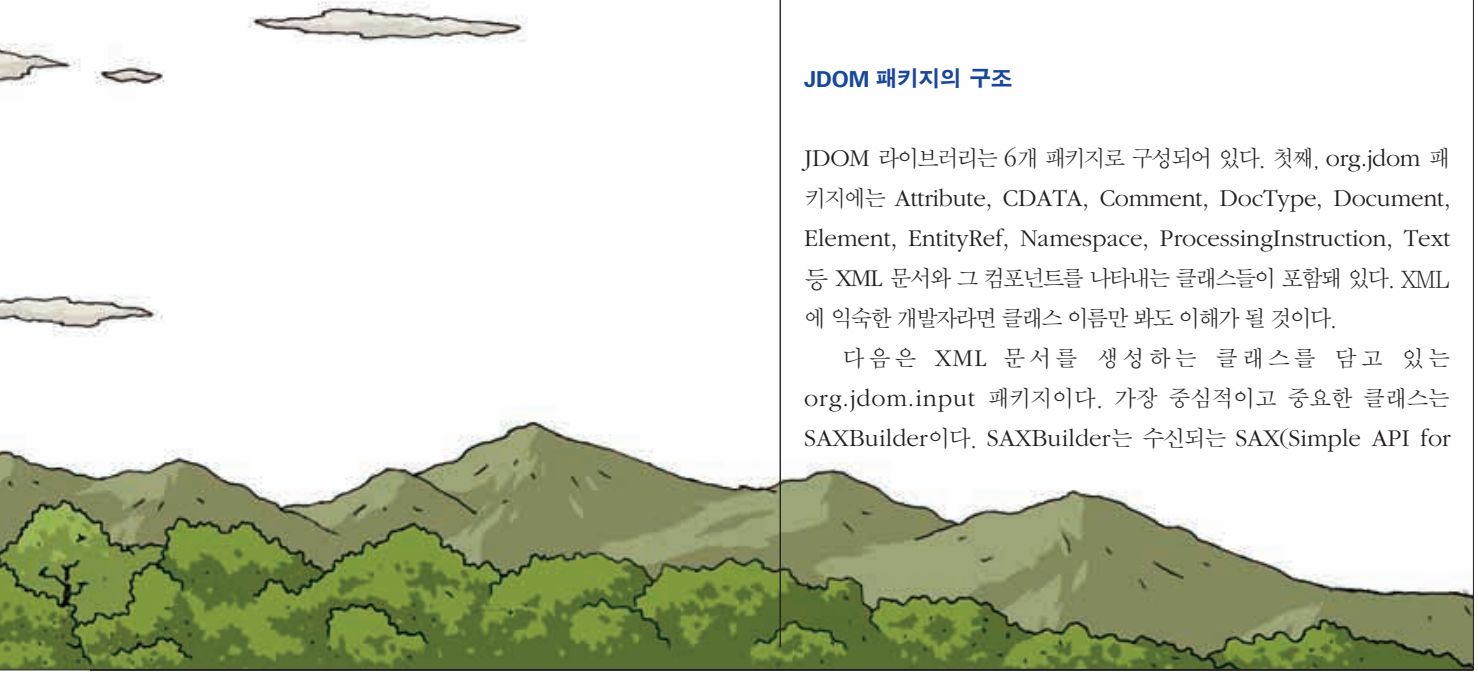
이 글의 원문은 http://otn.oracle.com/oramag/oracle/02-nov/o62dba.html을 참조하기 바랍니다.

JDOM과 XML 파싱

글 | **Jason Hunter** (Servlets.com의 발행인 겸 JCP Executive Committee의 이사) jasonhunter@servlets.com
감수 | **이진호** (DB기술팀) jinho.lee@oracle.com



Java에서의 XML 가공을 간소화해 주는 JDOM은
Java에 최적화된 XML 데이타 가공을 위한 오픈 소스 라이브러리로,
공동 협력을 통해 설계, 개발되었으며
현재 메일링 리스트에 등록된 가입자만 3,000여 명에 이르고 있다.
이 라이브러리는 곧 공식 Java 사양으로 채택될 것으로 전망된다.
JDOM은 무엇이며, 개발자에게 왜 JDOM이 중요한지 알아보자.



대부분의 개발자는 과거 XML 데이타 구조를 가공하기 위해 수많은 Java 라이브러리 중 하나를 이용해본 적이 있을 것이다. 그렇다면 JDOM(Java Document Object Model)은 무엇이며 개발자는 왜 JDOM을 필요로 할까?

JDOM은 Java에 최적화된 XML 데이타 가공을 위한 개방 소스 라이브러리이다. JDOM은 W3C(World Wide Web Consortium) DOM과 유사하기는 하지만, DOM을 기반으로 설계되거나 DOM을 모델링하지 않은 대안적인 문서 객체 모델이다. 가장 큰 차이점은 DOM은 언어 중립적으로 설계되었고 초기에 HTML 페이지의 JavaScript 가공에 주로 이용되었던 반면, JDOM은 Java 전용으로 설계됐기 때문에 메소드 오버로딩(method overloading), 컬렉션(collections), 리플렉션(reflection) 및 친숙한 프로그래밍 환경 등 Java의 기본 기능들을 활용한다는 데 있다. Java 프로그래머에게는 JDOM이 보다 자연스럽고 '알맞게' 느껴질 것이다. 이는 언어 중립적인 CORBA(Common Object Request Broker Architecture)에 비해 Java에 최적화된 RMI(Remote Method Invocation) 라이브러리가 보다 더 자연스럽게 느껴지는 것과 유사하다고 할 수 있다.

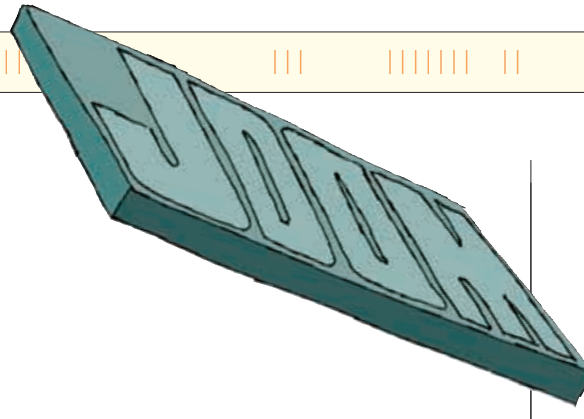
JDOM은 jdom.org에서 구할 수 있으며, 개방 소스로 Apache 스타일(상용 친화적) 라이선스로 제공된다. JDOM은 공동 협력을 통해 설계 및 개발됐으며, 메일링 리스트에 등록된 가입자만도 3,000여 명에 이른다. 또한 이 라이브러리는 Sun의 JCP(Java Community Process)에 Java Specification Request(JSR-102)로 채택됐으며, 곧 공식 Java 사양으로 채택될 것으로 전망된다.

이 글에서는 JDOM의 기술적 측면에 대해 다룰 것이다. 먼저, 주요 클래스에 대한 정보를 소개하고, 이어 Java 프로그램에서 JDOM을 이용하는 방법에 대해 설명할 것이다.

JDOM 패키지의 구조

JDOM 라이브러리는 6개 패키지로 구성되어 있다. 첫째, org.jdom 패키지에는 Attribute, CDATA, Comment, DocType, Document, Element, EntityRef, Namespace, ProcessingInstruction, Text 등 XML 문서와 그 컴포넌트를 나타내는 클래스들이 포함돼 있다. XML에 익숙한 개발자라면 클래스 이름만 봐도 이해가 될 것이다.

다음은 XML 문서를 생성하는 클래스를 담고 있는 org.jdom.input 패키지이다. 가장 중심적이고 중요한 클래스는 SAXBuilder이다. SAXBuilder는 수신되는 SAX(Simple API for



XML) 이벤트를 참조해 이에 대응하는 문서를 구성함으로써 문서를 생성한다. 파일이나 다른 스트림으로부터 문서를 생성하고자 한다면 SAXBuilder를 이용해야 한다. SAXBuilder는 SAX 파서를 이용해 스트림을 읽은 뒤 SAX 파서 콜백에 따라 문서를 생성한다. 이 설계의 좋은 점은 SAX 파서의 속도가 빨라질수록 SAXBuilder도 빨라진다는 것이다. 그밖에 주요 입력 클래스는 DOMBuilder이다. DOMBuilder는 DOM 트리를 통해 문서를 생성한다. 이 클래스는 이미 존재하는 DOM 트리를 JDOM 버전으로 대신 사용하고자 할 경우 편리하다.

이러한 빌더의 잠재성에는 아무런 제한이 없다. 예를 들어, Xerces에는 SAX보다 더 낮은 수준에서 운용되는 XNI(Xerces Native Interface)가 있으므로 SAX를 통해 노출되지 않는 일부 파서 정보를 다루기 위해서 XNIBuilder를 사용하는 것이 적합할 수도 있다. JDOM 프로젝트를 지원해온 한 가지 대중적인 빌더는 ResultSetBuilder이다. 이 빌더는 JDBC ResultSet을 통해 SQL 결과를 다양한 구성의 요소(element)와 속성(attribute)을 가지는 XML 문서를 표현한다.

org.jdom.output 패키지에는 XML 문서를 출력하는 클래스가 포함돼 있다. 가장 중요한 클래스는 XMLOutputter이다. XMLOutputter는 파일, 스트림, 소켓으로 출력할 수 있도록 문서를 바이트 스트림으로 변환한다. XMLOutputter는 원시 출력, 가공 출력, 압축 출력 등을 지원하는 다수의 특별 구성 옵션을 가지고 있다. 이 클래스는 상당히 복잡하다. DOM Level 2에 아직도 이런 기능이 없는 것은 바로 이런 이유 때문일 것이다.

그 밖에 문서의 컨텐트를 기반으로 SAX 이벤트를 생성하는 SAXOutputter가 있다. 이 클래스는 모호해 보이기는 하지만 XSLT 변환시 매우 유용한데, 이는 문서 데이타를 엔진으로 전송하는 데 있어 SAX 이벤트가 바이트 스트림보다 훨씬 효율적인 방식이기 때문이다. 또한 문서를 DOM 트리 형식으로 표현하는 DOMOutputter도 있다. 그 밖에 수십 라인의 코드만으로 문서를 JTree로 보여주는 JTreeOutputter도 있는데, JTreeOutputter를 ResultSetBuilder와 함께 사용할 경우 코드 몇 라인만 추가하는 것만으로도 SQL 질의 결과를 트리 뷰로 나타낼 수 있다.

DOM과는 달리, JDOM에서는 해당 문서가 빌더에 구속되지 않는다는 점에 주목해야 한다. 따라서 데이타를 담는 클래스와 데이타를 구조화하는 다양한 클래스, 이 데이타를 사용하는 그 밖의 여러 클래스가 포함된 세련된 모델이 생성된다. 원하는 만큼 자유롭게 혼합해 사용할 수 있다.

org.jdom.transform 및 org.jdom.XPath 패키지에는 기본 XSLT 변환과 XPath 조회를 지원하는 클래스가 포함돼 있다.

마지막으로, org.jdom.adapters 패키지는 DOM 상호작용의 라이브러리를 지원하는 클래스를 포함하고 있는데, 이 패키지의 클래스를 호출할 필요가 전혀 없다. 이들 클래스가 존재하는 이유는 각 DOM의 구현 방식이 각각의 부트 스트래핑 작업 방식별로 서로 다른 함수 이름을 사용하기 때문이며, 이에 따라서 각 어댑터 클래스가 표준 콜을 파서 전용 콜로 번역한다. JAXP(Java API for XML Processing)는 어댑터 클래스가 과도하게 사용될 때의 문제점에 대한 대안으로서, 실제로 이들 클래스에 대한 요구를 감소시키는 역할을 한다. 그러나 모든 파서가 JAXP를 지원하는 것은 아니고, 또한 라이선스 문제 때문에 어디나 JAXP가 설치돼 있는 것도 아니기 때문에, 이러한 클래스들에 대한 필요성은 여전히 남아 있다.

문서의 생성

문서는 org.jdom.Documentclass에 의해 표현된다. 다음은 완전히 새로운 문서를 생성하는 경우이다.

```
// This builds: <root/>
Document doc = new Document(new Element("root"));
```

또한 파일이나 스트림, 시스템 ID, URL 등을 통해 문서를 생성할 수도 있다.

```
// This builds a document of whatever's in the given resource
SAXBuilder builder = new SAXBuilder();
Document doc = builder.build(url);
```

소수의 콜을 조합함으로써 간단한 JDOM 문서를 생성할 수도 있다.

```
// This builds: <root>This is the root</root>
Document doc = new Document();
Element e = new Element("root");
e.setText("This is the root");
doc.addContent(e);
```

파워유저라면 다양한 방법을 연속적으로 호출하는 'method chaining'을 선호할 것이다. 이 방식을 통해 여러 개의 메소드를 한 번에 호출할 수 있다. 다음은 method chaining의 예이다.

```
Document doc = new Document(
    new Element("root").setText("This is the root"));
```

다음은 JAXP/DOM를 이용해 동일한 문서를 생성하는 예이다.

```
// JAXP/DOM
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.newDocument();
Element root = doc.createElement("root");
Text text = doc.createText("This is the root");
root.appendChild(text);
doc.appendChild(root);
```

SAXBuilder 이용하기

앞서 설명했듯이 SAXBuilder는 모든 바이트 근간 자원으로부터 문서를 생성하는 간단한 메커니즘을 제공한다. 입력 변수가 없는 기본 SAXBuilder( ) 생성자는 내부적으로 JAXP를 이용하여 SAX 파서를 선택한다. 파서를 변경하고자 할 때는 javax.xml.parsers.SAXParserFactory 시스템 속성을 파서가 제공하는 SAXParser Factory를 가리키도록 설정하면 된다. Oracle9*i* Release2 XML 파서의 경우 다음과 같이 실행하면 된다.

```
java -Djavax.xml.parsers.SAXParserFactory=
oracle.xml.jaxp.JXSAXParserFactory YourApp
```

Xerces 파서의 경우 다음과 같이 실행한다.

```
java -Djavax.xml.parsers.SAXParserFactory=
org.apache.xerces.jaxp.SAXParserFactoryImpl YourApp
```

만약 JAXP가 설치돼 있지 않다면 SAXBuilder는 Apache Xerces를 기본값으로 이용한다. SAXBuilder 인스턴스가 생성된 뒤에는 다음과 같은 몇 가지 속성을 빌더에 설정할 수 있다.

```
setValidation(boolean validate)
```

이 메소드는 문서 생성 중 DTD(Document Type Definition)에 대해 검증할 것인지 여부를 파서에 알려준다. 기본으로 설정된 값은 false이다. 사용된 DTD는 문서의 DocType 내에서 참조된 것이다. 다른 DTD에 대해 검증하는 것은 아직 불가능한데, 이 기능을 지원하는 파서가 아직 없기 때문이다.

```
setIgnoringElementContentWhitespace(boolean ignoring)
```

Oracle XML Tools

이 XDK(XML Developer Kit)는 오라클이 개발자를 위해 제공하는 무료 XML 툴 라이브러리이다. 이 라이브러리에서는 JDOM과 함께 이용할 수 있는 XML 파서와 XSLT 변환 엔진이 제공된다. 이들 툴에 대한 자세한 정보는 Oracle XML 홈페이지인 oracle.com/xml에서 제공된다.

파서를 다운로드 받으려면 'XDK for Java'로 명명된 XML Developer Kit를 찾은 뒤 좌측의 '소프트웨어' 항목을 클릭해 다운로드를 시작하면 된다. 다운로드 받은 파일을 열면 xalparserv2.jar 파일에 파서가 들어 있다.

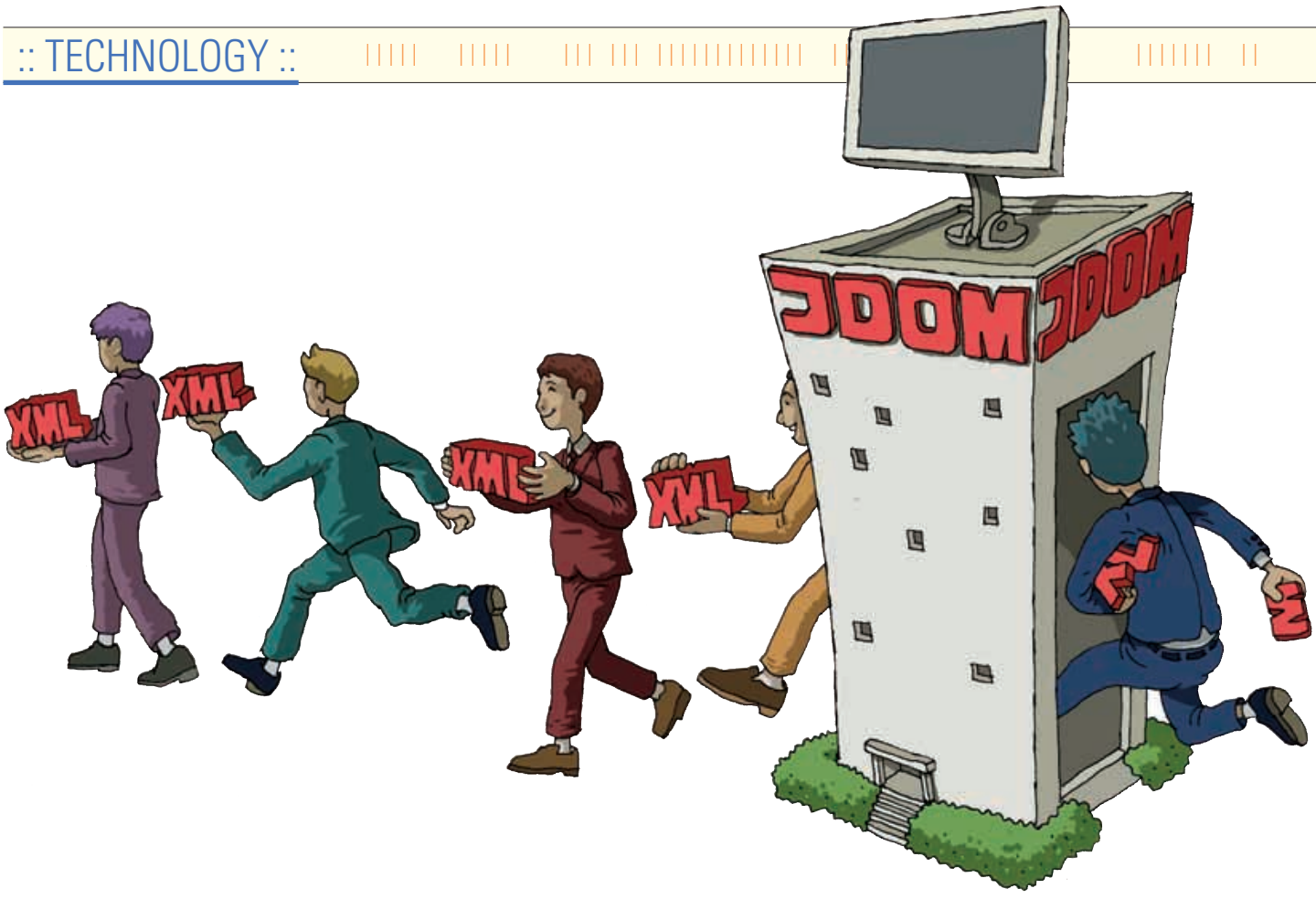
JDOM 및 기타 소프트웨어가 기본값으로 오라클 파서를 이용하도록 구성하려면 JAXP javax.xml.parsers.SAXParserFactory 시스템 속성을 oracle.xml.jax.JXSAXParserFactory으로 설정해야 한다. 이는 JAXP에 오라클 파서를 이용하겠다고 밝히는 것이다. 가장 쉬운 방법은 다음과 같은 명령을 이용하는 것이다.

```
java -Djavax.xml.parsers.SAXParserFactory=
oracle.xml.jaxp.JXSAXParserFactory
```

혹은 다음과 같이 프로그래밍 방식을 이용할 수도 있다.

```
System.setProperty("jaxax.xml.parsers .SAXParserFactory",
"oracle.xml.jaxp.JXSAXParserFactory");
```

오라클은 XDK 외에도 Oracle9*i* Database Release 2에 Native XML 리포지토리를 제공하고 있다. Oracle9*i* XML Database(XDB)는 고성능의 네이티브 XML 저장 및 검색 기술이다. XDB는 W3C XML 데이타 모델을 Oracle9*i* Database로 완벽하게 수용하며 XML의 네비게이션 및 질의를 위한 새로운 표준 액세스 방법을 제공한다. XDB를 이용할 경우 관계형 데이타베이스 기술의 모든 이점과 함께 XML 기술의 장점을 활용할 수 있다.



위 메소드는 요소 컨텐트에서 '무시할 수 있는 여백(ignorable whitespace)' 을 무시할 것인지 여부를 파서에 알려준다. XML 1.0 사양에 의하면, 요소 컨텐트의 여백은 파서에 의해 보존돼야 하지만 DTD에 대해 검증할 경우 문서의 특정 부분이 여백을 지원하지 않는다는 사실을 파서가 인식할 수 있기 때문에 이 영역의 여백은 '무시할 수' 있다. 기본값으로는 해제 상태이다. 문서를 입력 때와 동일한 컨텐트를 출력하고자 할 때가 아니라면 '무시 가능' 으로 사용하는 것이 일반적으로 성능상 바람직하다. 단, 이 플래그는 DTD 검증이 수행될 때만 유효하며 이때는 이미 검증 과정을 통한 성능 저하가 발생한 것이기 때문에 결국 이 메소드는 검증이 이미 이용되고 있을 경우에만 유용하다는 점을 유의해야 한다.

```
setFeature(String name, String value)
```

위 메소드는 기본 SAX 파서상에 기능을 설정하는 방법이다. 이 방법은 원시적인 호출 방식이기 때문에 이 방법을 이용할 때는 매우 신중해야 한다. 왜냐하면 특정 기능(예 : 네임스페이스 변경)을 잘못 설정할 경우 JDOM 작업이 중단될 수도 있기 때문이다. 게다가 파서 전용 기능에 의존할 경우 이식성을 제한할 위험이 있다. 이 콜은 스키마 검증을 선택할 때 가장 유용하다.

```
setProperty(String name, Object value)
```

위 메소드는 기본 SAX 파서상에 속성을 설정하는 방법이다. 이 방법 역시 원시 호출 방식으로, 위험한 동시에 특히 스키마 검증시 파워유저에게 유용한 방법이다.

다음 코드는 이 방법들을 조합해 검증 기능을 선택하고 여백 무시 가능 기능으로 설정한 뒤 JAXP 선택 파서를 이용해 로컬 파일을 읽게 된다.

```
SAXBuilder builder = new SAXBuilder();
builder.setValidation(true);
builder.setIgnoringElementContentWhitespace(true);
Document doc = builder.build(new File("/tmp/foo.xml"));
```

XMLOutputter를 이용한 문서 출력

문서는 다양한 포맷으로 출력될 수 있지만 가장 흔한 포맷은 바이트 스트림이다. JDOM에서는 XMLOutputter 클래스가 이 기능을 제공한다. 이 클래스의 기본 생성자는 문서에 저장된 원문 그대로 문서를 출력하려 한다. 아래 코드는 원문 그대로 문서의 내용을 파일에 출력하는 코드이다.

```
// Raw output
XMLOutputter outp = new XMLOutputter();
```

```
outp.output(doc, fileStream);
```

여백에 신경 쓰지 않아도 된다면 텍스트 트리밍을 선택해 약간의 공간을 절약할 수 있다.

```
// Compressed output
outp.setTextTrim(true);
outp.output(doc, socketStream);
```

사람 눈에 맞춰 문서의 인쇄 상태를 보기 좋게 만들려면 들여쓰기와 줄 바꿔쓰기를 추가하면 된다.

```
outp.setTextTrim(true);
outp.setIndent("  ");
outp.setNewlines(true);
outp.output(doc, System.out);
```

이미 여백을 통해 포맷된 문서에 위의 가공 기능을 다시 적용할 경우 트리밍을 선택해야 한다. 그렇지 않으면 이미 포맷된 상태에서 또다른 포매팅을 가하는 것이 돼 최종 출력 상태가 보기 흉하게 된다.

요소 트리의 네비게이션

JDOM은 요소 트리(element tree)의 네비게이션을 간편하게 해준다. 루트 요소를 호출하려면 다음 코드를 이용한다.

```
Element root = doc.getRootElement();
```

모든 자식 요소 리스트를 불러오는 방법은 다음과 같다.

```
List allChildren = root.getChildren();
```

주어진 이름의 요소만을 호출하려면,

```
List namedChildren = root.getChildren("name");
```

주어진 이름의 요소 중 첫 번째 요소만을 호출하려면 다음을 이용한다.

```
Element child = root.getChild("name");
```

getChildren() 콜을 통해 반환된 리스트는 모든 Java 프로그래머가 알고 있는 리스트 인터페이스의 구현인 java.util.List이다. 이 리스트에서 특기할 만한 것은 이것이 라이브 리스트라는 점이다. 리스트에 가해진 모든 변경사항은 원본 문서 객체에도 반영된다.

```
// Remove the fourth child
allChildren.remove(3);
// Remove children named "jack"
allChildren.removeAll(root.getChildren("jack"));
// Add a new child, at the tail or at the head
allChildren.add(new Element("jane"));
allChildren.add(0, new Element("jill"));
```

이러한 리스트를 통한 대치 방법을 이용하면, 수많은 별도의 방법들을 과도하게 사용하지 않고도 요소를 다양하게 가공할 수 있다. 그러나, 편의상 주로 이용하는 작업인, 마지막에 요소를 추가하거나 이름이 있는 요소들을 삭제하는 경우 요소 자체에 이미 동일한 메소드가 포함돼 있기 때문에 이 작업을 실행할 때는 리스트를 우선 호출할 필요가 없다.

```
root.removeChildren("jill");
root.addContent(new Element("jenny"));
```

JDOM의 또 다른 장점은 한 문서 내에서 혹은 여러 문서 사이에서 요소들을 이동하는 작업이 간편하다는 것이다. 이 때 몇 개의 문서간에 이동하든 관계없이 동일한 코드를 사용할 수 있다.

```
Element movable = new Element("movable");
parent1.addContent(movable);   // place
parent1.removeContent(movable); // remove
parent2.addContent(movable);   // add
```

DOM의 경우 요소의 이동이 JDOM에서만큼 쉽지 않은데, 이는 DOM에서는 요소들이 그들을 생성한 객체에 강하게 묶여 있기 때문이다. 따라서 문서간 이동시에는 DOM 요소가 직접 '임포트' 되어야 한다.

JDOM에서 한 가지 유념할 사항은, 요소를 다른 데 추가하기 전에 제거해야 한다는 점이다. 이렇게 해야 트리 내에서 순환이 발생하는 것을 막을 수 있다. detach() 메소드를 이용하면 분리/추가 작업을 라인 하나로 처리할 수 있다.

```
parent3.addContent(movable.detach());
```

요소를 다른 부모에 추가하기 전에 먼저 분리하지 않았을 경우, 해당 라이브러리는 Exception을 떨어트릴 것이다(정확하고 도움이 되는 오류 메시지와 함께). 또한 라이브러리는 요소에 스페이스와 같은 부적절한 문자가 포함되지 않도록 요소의 이름과 컨텐트를 확인한다. 또한 단일 루

트 요소의 포함 여부, 일관적인 네임스페이스 선언 여부 및 주석과 CDATA 섹션에 금지된 문자열이 없는지 등 기타 여러 규칙도 검증한다. 이를 통해 가능한 한 프로세스 초기 단계에서 문서가 'well-formed' 인지 확인하는 과정이 이루어지게 되는 것이다.

요소 속성의 처리

요소 속성의 예를 들면 다음과 같다.

```
<table width="100%" border="0"> ... </table>
```

요소 참조를 통해, 어떤 이름의 속성 값이든 요소에 요청할 수 있다.

```
String val = table.getAttributeValue("width");
```

또한 타입 변환과 같은 특별 가공 작업을 위해 속성을 객체로 불러올 수도 있다.

```
Attribute border = table.getAttribute("border");
int size = border.getIntValue();
```

속성을 설정하거나 변경하려면 setAttribute( )를 사용한다.

```
table.setAttribute("vspace", "0");
```

속성을 삭제하려면 removeAttribute( )를 사용한다.

```
table.removeAttribute("vspace");
```

요소의 텍스트 컨텐트 작업

텍스트 컨텐트를 가진 요소의 예를 들면 다음과 같다.

```
<description>
 A cool demo
</description>
```

JDOM에서는 호출을 통해 텍스트를 직접 이용할 수 있다.

```
String desc = description.getText();
```

한 가지 유의할 점은, XML 1.0 사양에서는 여백의 보존이 요구되기 때문에 이 경우 '\n A cool demo\n'이 반환된다는 것이다. 그러나 실제 환경에서는 여백의 포매팅에 대해 크게 유념할 필요가 없으므로 가장자리의 여백을 무시하고 텍스트를 불러오는 편리한 방법이 있다.

```
String betterDesc = description.getTextTrim();
```

여백을 아예 없애고 싶다면 스페이스를 이용해 내부의 여백을 표준화하는 getTextNormalize( ) 메소드를 이용하면 된다. 이 메소드는 다음과 같은 텍스트 컨텐트에 이용할 때 편리한다.

```
<description>
 Sometimes you have text content with formatting
 space within the string.
</description>
```

텍스트 컨텐트를 변경하고자 할 때는 setText( ) 메소드를 이용한다.

```
description.setText("A new description");
```

텍스트에 포함된 특수 문자는 모두 문자로서 올바르게 해석되어 출력시에 적절한 의미를 유지하게 된다. 다음 콜을 예로 들어보자.

```
element.setText("<xml/> content");
```

내부 저장 영역은 이 문자열을 그대로 문자로 저장할 것이다. 이 컨텐트에 대한 함축적 파싱은 이루어지지 않는다. 출력시에는 다음과 같이 표현된다.

```
&lt;xml/&gt; content
```

이는 이전 setText( ) 콜의 의미론적 내용을 보존하기 위한 것이다. 따라서, 요소 내에 XML 컨텐트를 포함하고자 한다면 적절한 JDOM 자식 요소 객체를 추가해야 할 것이다.

JDOM에서는 CDATA 섹션을 처리할 수도 있다. CDATA 섹션은 파싱되어서는 안 될 텍스트 블록을 지시한다. 원래 CDATA 섹션은 <와 > 같은 에스케이프 문자열을 과도하게 사용하지 않고도 HTML이나 XML을 손쉽게 포함시킬 수 있게 해주는 문법적인 용어이다. 그러나 JDOM에서는 이를 객체화하여 사용한다. CDATA 섹션을 생성하려면 이 문자열을 CDATA 객체로 래핑하면 된다.

```
element.addContent(new CDATA("<xml/> content"));
```

JDOM의 장점은 getText( ) 콜이 문자열을 CDATA 섹션으로 나타낼 것인지 일일이 물어보지 않고 문자열을 반환한다는 점이다.

혼합 컨텐트의 처리

어떤 요소에는 여백, 주석, 텍스트, 자식 요소 등 수많은 항목들이 포함되어 있다.

```
<table>
 <!-- Some comment -->
 Some text
 <tr>Some child element</tr>
</table>
```

어떤 요소에 텍스트와 자식 요소가 모두 들어 있을 경우 '혼합 컨텐트'를 포함하고 있다고 한다. 혼합 컨텐트를 처리하는 것은 어려울 수도 있지만, JDOM을 이용하면 쉽게 처리할 수 있다. 텍스트 컨텐트를 불러오고 자식 요소를 네비게이션하는 기본 이용 사례는 간단하게 처리할 수 있다.

```
String text = table.getTextTrim();  // "Some text"
Element tr = table.getChild("tr");  // A straight reference
```

주석, 여백 블록, 프로세싱 명령어, 엔티티 참조 등이 필요한 고급 응용의 경우, 혼합 컨텐트를 리스트로서 직접 불러올 수 있다.

```
List mixedCo = table.getContent();
Iterator itr = mixedCo.iterator();
while (itr.hasNext()) {
 Object o = i.next();
 if (o instanceof Comment) {
  ...
 }
 // Types include Comment, Element, CDATA, DocType,
 // ProcessingInstruction, EntityRef, and Text
}
```

자식 요소 리스트와 마찬가지로 혼합 컨텐트 리스트를 변경할 경우 원래 문서에도 영향을 미치게 된다.

```
// Remove the Comment.  It's "1" because "0" is a whitespace block.
mixedCo.remove(1);
```

자세히 살펴보면, 여기에 Text 클래스가 포함돼 있다는 사실을 알 수 있다. JDOM은 내부적으로 Text 클래스를 이용해 문자열 컨텐트를 저장하는데, 이는 이 문자열이 부모를 갖도록 하고 XPath 액세스를 보다 쉽게 지원하도록 하기 위한 것이다. 원시 컨텐트 리스트에 액세스할 때 텍스트만을 불러오거나 설정할 경우라면 이 클래스에 대해 염려할 필요가 없다.

DocType, ProcessingInstruction, EntityRef 클래스에 대해 보다 자세한 정보는 jdom.org의 API 설명서를 참조하기 바란다.

Namespaces로 작업하기

JDOM은 XML Namespaces를 확실히 지원한다. JDOM은 네임스페이스 권고사항이 발표된 이후에 만들어졌기 때문에, 다른 API와 달리 네임스페이스 이전 API나 deprecation된 것들이 없다. JDOM에서 네임스페이스는 다음과 같은 Namespace 클래스로 표현된다.

```
Namespace xhtml = Namespace.getNamespace(
 "xhtml", "http://www.w3.org/1999/xhtml");
```

모든 XML 객체는 생성시에 이름을 부여 받으며, 동시에 옵션으로 다음과 같은 네임스페이스를 받을 수 있다.

```
elt.addContent(new Element("table", xhtml));
```

네임스페이스를 받지 않는 경우에는 해당 요소가 'no namespace'임을 의미한다. 요소의 네임스페이스는 요소 유형의 고유 부분이기 때문에, JDOM은 네임스페이스가 문서에서 이동할 때 바뀌지 않는지를 확인한다. 네임스페이스를 가지고 있지 않는 요소가 네임스페이스를 가지고 있는 요소 아래로 이동한다면, 이 요소는 명시적으로 네임스페이스를 상속하지 않는다. 이러한 의미론적 구조와 실제 텍스트 표기법을 구분하지 못한다면 이 부분에서 많은 혼동이 생길 수 있다.

XMLOutputter 클래스는 여러 문서 요소가 섞인 상황에도 네임스페이스 항목을 정렬하고 모든 'xmlns'가 적당한 위치에 선언되었는지를 확인한다. 기본적으로 클래스는 가장 먼저 필요한 곳에 네임스페이스를 선언하는데, 트리에서 보다 위쪽에 선언하고자 하는 경우(즉, 루트에 대한 모든 선언)는 해당 명령을 제공하는 element.addNamespaceDeclaration( ) 메소드를 사용한다.

모든 JDOM 요소나 속성 액세서 메소드(Accessor Method)는 대상 네임스페이스를 지정하는 namespace 인수를 옵션으로 지원한다. 아래의 예제는 xhtml 네임스페이스를 사용하도록 지정한다.

```
List kids = html.getChildren("title", xhtml);
Element kid = html.getChild("title", xhtml);
```

XML NAMESPACES

Namespaces는 XML에서 사용하기 매우 까다로운 부분일 수 있다. Namespaces의 목표는 Java 패키지 처럼 동일한 이름을 갖는 두 개의 클래스들을 각각 패키지에 따라 구분하는 것과 유사하다. 하지만, 다음의 의미를 이해할 수 있는 사람은 실제로 많지 않다.

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml">
 <xhtml:title>Home Page</xhtml:title>
</xhtml:html>
```

<html> 요소에서 "xmlns:xhtml=" 속성 이름이 갖는 의미는 특별하다. 이것은 네임스페이스 선언으로서, "xhtml"("xmlns:" 뒤에 무엇이 오건)은 속성 값으로 사용된 URI의 별칭이어야 한다는 것이다. 보다 명확하게 말하면, 네임스페이스가 선언된 해당 요소와 이에 속한 하부 컨텐트에서 이름에 "xhtml" 접두어가 사용되는 경우 "http://www.w3.org/1999/xhtml" 네임스페이스에 있는 항목이라는 것을 의미한다.

이 네임스페이스는 웹 주소처럼 보이지만 웹 주소가 아닌 단순한 문자열이다. 여기서 중요한 점은 "xhtml"은 별칭일 뿐이고 네임스페이스 그 자체는 아니라는 점이다. 네임스페이스에는 디폴트 네임스페이스가 있는데. 접두어가 없는 네임스페이스가 그것이다. 디폴트 네임스페이스를 사용하여 초기 xhtml 컨텐트를 다음과 같이 작성할 수 있다.

```
<html xmlns="http://www.w3.org/1999/xhtml">
 <title>Home Page</title>
</html>
```

"xmlns=" 선언이 XML에 없는 경우, 모든 요소들은 'no namespace'로 간주되며 이는 여러 가지 면에서 Java의 기본 패키지의 개념과 유사하다.

```
Attribute attr = kid.getAttribute("id", xhtml);
```

액세서 메소드(Accessor Method)를 호출할 때 유의할 것은, 정확한 URI(Uniform Resource Identifier)를 사용하는 것이다. XML Namespaces가 이를 통해 운용되기 때문이다.

액세서 메소드(Accessor Method)에 네임스페이스 인스턴스가 제공되지 않는 경우에는 네임스페이스가 없는 요소를 탐색하게 된다. JDOM에서 말하는 'no namespace'는 문자 표현 그대로 네임스페이스가 없는 것이며, 이를 상위 객체의 네임스페이스나 기타 어떤 것으로도 대체하지 않음으로써 추후에 발생 가능한 버그를 미연에 방지한다.

ResultSetBuilder에 대한 추가 정보

ResultSetBuilder는 SQL 결과를 XML 문서로 처리하지 않아도 되는 사람들을 위해 더욱 확장된 JDOM 빌더이다. 이에 대해서는 org.jdom.contrib.input 패키지의 jdom-contrib 리포지토리에 있는 ResultSetBuilder를 참고하기 바란다.

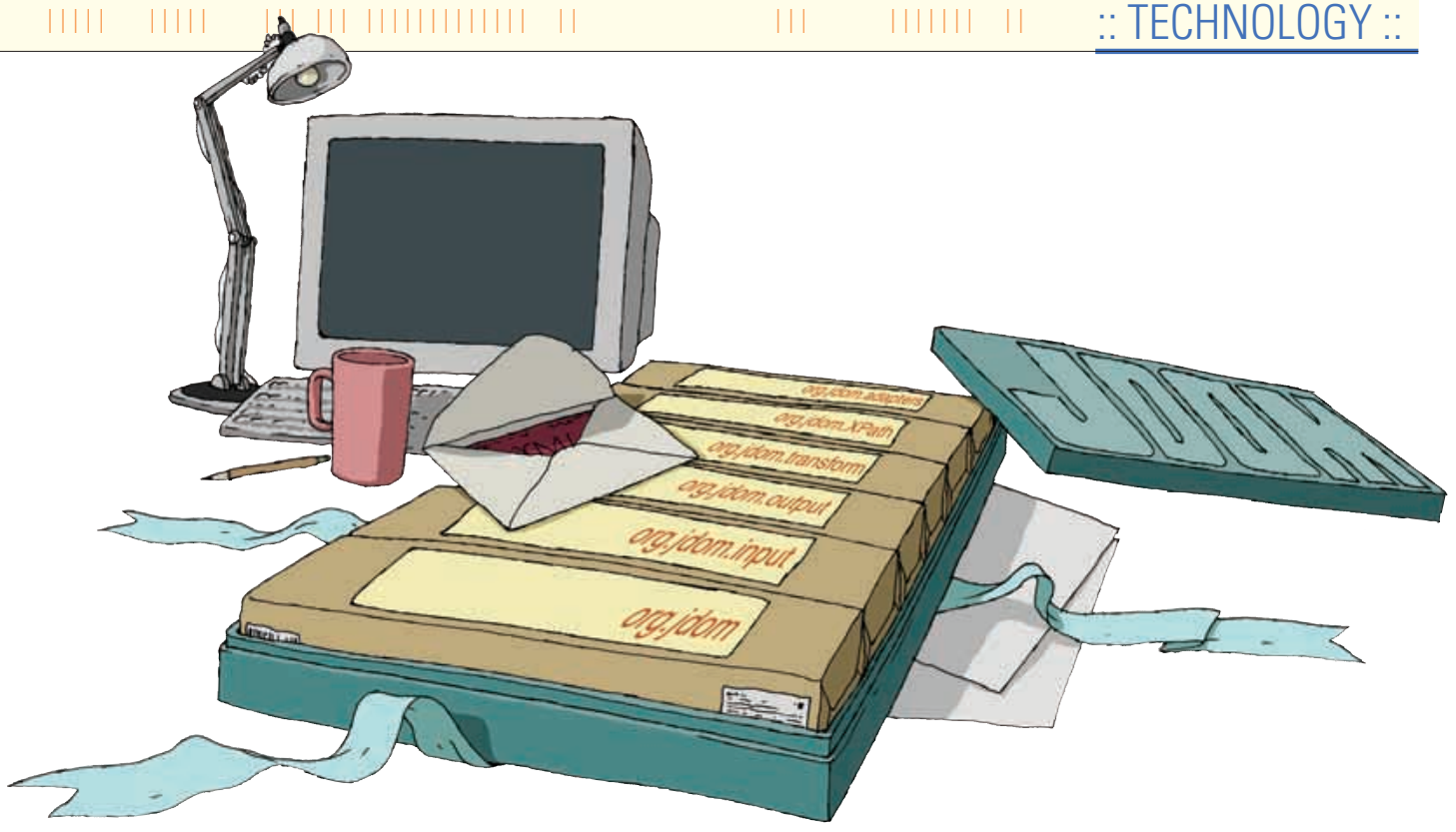
ResultSetBuilder 생성자(constructor)는 java.sql.ResultSet을 입력으로 받아들이고 build( ) 메소드를 통해 org.jdom.Document를 반환한다.

```
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery("select id, name from registry");
ResultSetBuilder builder = new ResultSetBuilder(rs);
Document doc = builder.build();
```

특정 구성 정보를 제공하지 않는 경우, 위의 코드는 다음과 유사한 문서를 생성한다.

```
<result>
 <entry>
  <id>1</id>
  <name>Alice</name>
 </entry>
 <entry>
  <id>2</id>
  <name>Bob</name>
 </entry>
</result>
```

ResultSetBuilder 클래스는 질의의 ResultSetMetaData를 사용해 컬럼명을 결정하고, 이 컬럼명을 요소 이름으로 사용한다. 기본적으로 루트 요소는 'result'라는 이름을 가지며, 각 로우는 'entry'라는 이름을 갖지만, 각각 setRootName(Stringname), setRowName(Stringname)으로 바꿀 수도 있다. 그리고 setNamespace(Namespaces)를 통해 문

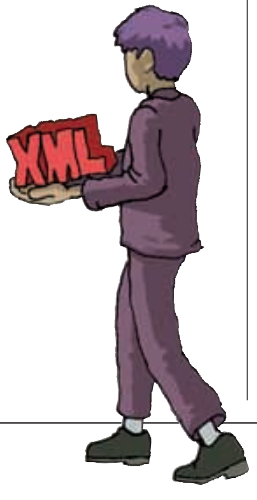


서에 네임스페이스를 할당할 수도 있다.

ResultSet 컬럼을 요소 대신 XML 속성으로 표현하고자 하는 경우에는, setAsAttribute(StringcolumnName)나 setAsAttribute (StringcolumnName, StringattribName)을 호출하면 된다. 단 setAsAttribute(StringcolumnName, StringattribName) 메소드를 사용하여 속성 이름을 바꿀 수 있으며, setAsElement (StringcolumnName, StringelemName)을 사용하여 요소 이름을 바꿀 수 있다는 점을 알아두자. 앞의 두 가지 메소드는 원하는 경우 이름 대신 순차적인 인덱스를 사용할 수도 있다. 다음은 이미 작성된 문서를 위의 메소드를 통해 재포맷하는 과정과 그 결과를 보여준다.

```
ResultSetBuilder builder = new ResultSetBuilder(rs);
builder.setAsAttribute("id");
builder.setAsElement("name", "fname");
Document doc = builder.build();
```

```
<result>
 <entry id="1">
  <fname>Alice</fname>
 </entry>
 <entry id="2">
  <fname>Bob</fname>
 </entry>
</result>
```



이 클래스는 나중의 검색을 위해 데이타베이스에 XML 문서를 저장할 수 있는 방법 같은 것은 제공하지 않으며, 데이타베이스의 데이타를 XQuery 호출할 수 있는 방법 또한 제공하지 않는다. 이러한 작업을 실행하려면, Oracle9*i*의 기능인 XMLDB 같은 고유한 XML 데이타베이스가 있어야 한다.

내장형 XSLT

이제까지 핵심 라이브러리에 대한 기본 사항을 살펴보았다. 지금부터는 좀더 높은 수준의 기능인 XSLT(eXtensible Stylesheet Language Transformation) 언어에 대해 살펴보도록 한다.

XSLT는 XML 파일을 사용하여 변환을 처리함으로써 한 포맷으로부터 다른 포맷으로 XML 컨텐트를 전환하는 표준적인 방법을 제공한다. XSLT는 일반적으로 XHTML 웹 페이지로 XML을 보내거나, 한 스키마에서 다른 스키마로 XML을 바꾸는 데 사용된다. JDOM은 XSLT 엔진에 대한 JAXP 표준 인터페이스를 사용함으로써 인-메모리 XSLT 변형에 대해 내장형 지원을 제공한다. 중요한 클래스로는 org.jdom.transform 패키지의 JDOMSource 및 JDOMResult가 있다. JDOMSource는 변환을 위한 입력으로 JDOM 문서를 제공하며, JDOMResult는 결과를 JDOM 문서로 처리한다. 〈리스트 1〉은 인-메

〈리스트 1〉

XSL Transform

```
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.transform.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
public class XSLTransform {
 public static void main(String[] args) throws Exception {
  // Build the base document, just assume we get 2 params
  String docname = args[0];
  String sheetname = args[1];
  SAXBuilder builder = new SAXBuilder();
  Document doc = builder.build(docname);

  // Use JAXP to get a transformer
  Transformer transformer = TransformerFactory.newInstance()
   .newTransformer(new StreamSource(sheetname));

  // Run the transformation
  JDOMSource source = new JDOMSource(doc);
  JDOMResult result = new JDOMResult();
  transformer.transform(source, result);
  Document doc2 = result.getDocument();

  // Display the results
  XMLOutputter outp = new XMLOutputter();
  outp.setTextNormalize(true);
  outp.setIndent("  ");
  outp.setNewlines(true);
  outp.output(doc2, System.out);
 }
}
```

모리 변환을 실행하는 완성된 프로그램이다.

독자들은 여러 가지 소스와 결과를 혼합하고 일치시킬 수 있다. 예를 들어, 문서를 출력하고자 하나 인-메모리 JDOM 표현식이 필요 없는 경우에는, 대신 javax.xml.transform.stream.StreamResult를 임포트하여 사용할 수 있다.

```
JDOMSource source = new JDOMSource(doc);
StreamResult result = new StreamResult(System.out);
transformer.transform(source, result);
```

여기 하단부에서는 제대로 된 XMLOutputter를 통한 재포맷이 제외되었다.

XPath 포함

XPath는 문자열 조회 경로를 사용하여 XML 문서 일부를 참조할 수 있는 방법을 제공한다. XPath를 사용하면 간단한 경로 표현식에 근거하여 필요한 정보만 발췌할 수 있기 때문에 문서 전체를 검색하지 않아도 된다. 다음과 같은 XHTML 문서를 보자.

```
<table border="1">
 <tr>
  <th> </th>
  <th>Open</th>
  <th>Close</th>
 </tr>
 <tr>
  <td>Sunday</td>
  <td>11am</td>
  <td>4pm</td>
 </tr>
 <tr>
  <td>Monday</td>
  <td>9am</td>
  <td>5pm</td>
 </tr>
</table>
```

여기서 Monday가 속한 <tr>절에서 시작 시간인 '9am'을 선택하는 Xpath 표현식은 /table/tr[td= "Monday"]/td[2]. 이다. Beta 8 이후 코드 버전에서 JDOM은 org.jdom.xpath.XPath 클래스를 내장하여 이를 지원한다. 이를 사용하려면 먼저 다음과 같이 XPath.newInstance( )를 호출하여 XPath 인스턴스를 생성해야 한다.

〈리스트 2〉

XPath를 이용한 정보 얻기

```
import java.io.*;
import java.util.*;
import org.jdom.*;
import org.jdom.input.*;
import org.jdom.output.*;
import org.jdom.xpath.*;

public class XPathReader {

public static void main(String[] args) throws IOException,
JDOMException {
  if (args.length != 1) {
   System.err.println("Usage: samples.XPathReader [web.xml]");
   return;
  }
  String filename = args[0];
  PrintStream out = System.out;

  SAXBuilder builder = new SAXBuilder();
  Document doc = builder.build(new File(filename));

  // Print servlet information
  XPath servletPath = XPath.newInstance("//servlet");
  List servlets = servletPath.selectNodes(doc);

  out.println("This WAR has "+ servlets.size() +" registered servlets:");
  Iterator i = servlets.iterator();
  while (i.hasNext()) {
   Element servlet = (Element) i.next();
   out.print("\t" + servlet.getChild("servlet-name")
                .getTextTrim() +
        " for " + servlet.getChild("servlet-class")
                .getTextTrim());
   List initParams = servlet.getChildren("init-param");
   out.println(" (it has " + initParams.size() + " init params)");
  }

  // Print security role information
  // Notice how we're directly fetching Text content
  XPath rolePath = XPath.newInstance("//security-role/role-name/text()");
  List roleNames = rolePath.selectNodes(doc);

  if (roleNames.size() == 0) {
   out.println("This WAR contains no roles");
  }
  else {
   out.println("This WAR contains " + roleNames.size() + " roles:");
   i = roleNames.iterator();
   while (i.hasNext()) {
    out.println("\t" + ((Text)i.next()).getTextTrim());
   }
  }
 }
```

```
XPath xpath = XPath.newInstance("/some/xpath");
```

그런 다음 selectNodes()를 호출하여 주어진 XPath 컨텍스트에 대한 응답 목록을 얻는다. 그 컨텍스트는 문서가 될 수도 있고 문서 내 요소가 될 수도 있다.

```
List results = xpath.selectNodes(doc);
```

이 외에도 단일 노드, 번호 값, 문자열 값 등을 검색하는 데 사용되는 메소드가 여러 개 있으며, 기본 XPath 구현은 http://jaxen.org의 Jaxen을 사용한다.

〈리스트 2〉는 XPath를 사용하여 서블릿 배포 설정 파일인 web.xml 파일에서 정보를 얻는 것이다. 〈리스트 3〉의 web.xml 파일을 고려할 때 출력은 다음과 유사하게 나타날 것이다.

```
This WAR has 2 registered servlets:
    snoop for SnoopServlet (it has 0 init params)
    file for ViewFile (it has 1 init params)
This WAR contains 3 roles:
    manager
    director
    president
```

〈리스트 3〉

예제 web.xml 파일(〈리스트 2〉의 XPath 프로세스와 함께 사용)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app>
  <servlet>
    <servlet-name>
      snoop
    </servlet-name>
    <servlet-class>
      SnoopServlet
    </servlet-class>
  </servlet>
  <servlet>
    <servlet-name>
      file
    </servlet-name>
    <servlet-class>
      ViewFile
    </servlet-class>
    <init-param>
   <param-name>
        initial
      </param-name>
      <param-value>
        1000
      </param-value>
      <description>
        The initial value for the counter  <!-- optional -->
      </description>
    </init-param>
  </servlet>

  <distributed/>

  <security-role>
   <role-name>
     manager
   </role-name>
   <role-name>
     director
   </role-name>
   <role-name>
     president
   </role-name>
  </security-role>
</web-app>
```

JDOM Beta 9의 새로운 기능

현재 JDOM은 Beta 8이며, 다음 버전은 거의 최종 API를 결정하는 버전이 될 것으로 보이다. Beta 9 버전에서 나타나는 API 일부 변경 사항은 다음과 같다.

* 전체 문서 트리를 쉽게 반복하기 위한 방법을 제시한다.
* 프로그래머가 특정 프로그램 필터 규칙을 만족시키는 트리만을 사용하도록 해주는 Filter 인터페이스(현재 내부적으로 사용하고 있는)를 제시한다.
* 횡단(traversal) 모델을 지원할 기본 인터페이스를 포함시켜, 이를 통해 정확한 유형을 모르는 경우에도 컨텐트를 분리할 수 있도록 한다.
* XSLT 변형을 간소화하고 JAXP 호출을 숨길 수 있는 클래스를 제공한다.
* 프로그래머가 선택한 출력에서 실제 사용할 수 없는 문자들의 출력을 처리할 방법을 완성한다.

JDOM은 Java 프로그래머가 배우기 쉽도록 생성된 강력하고 자연스러운 제품으로서, Java 프로그래머는 JDOM을 사용하여 효율적으로 XML과 상호 작용할 수 있다. 이 제품은 쉽게 이용할 수 있는 라이선스를 제공하는 오픈 소스로 출시되었기 때문에 무료로 사용할 수 있고, JAXP나 DOM, SAX와 잘 통합되며 Java Community Process의 감독 하에 있는 공식 JSR이다.

이 글의 원문은 http://otn.oracle.com/oramag/oracle/02-sep/o52jdom.html과 http://otn.oracle.com/oramag/oracle/02-nov/o62odev_jdom.html을 참조하기 바랍니다.