Tech Notes(vol.6)

Informix Dynamic 4GL: 4GL의 미래

Informix 4GL 의 영향

Informix 4GL(이하 4GL)은 1980 년대에 처음 출시된 이래 UNIX 환경에서 Informix 데이터베이스에 액세스하는 클라이언트 응용프로그램 개발을 위한 대표적인 프로그래밍 언어가 되었습니다. 4GL 은 개발자들로 하여 금 업계 표준인 SQL 문을 소스 코드에서 작성하거나 아니면 직접 삽입할 수 있게 해주는 체계적이고, 양식 (form)에 기초한, 문자 지향적인, 영어와 유사한 언어입니다. 4GL 은 일부 문(INPUT, MENU 및 다양한 REPORT 기능들)이 "비프로시저 방식(non-procedural)"이라는 점에서 "제 4 세대 언어"라고 할 수 있습니다. 원하는 결과 를 얻기 위해 필요한 모든 단계들을 개발자가 직접 코딩해야 하는 C, COBOL 또는 FORTRAN 같은 제 3 세대 언 어와는 달리, 4GL 개발자는 필요한 결과만 지정하면 4GL 에서 세부 사항을 처리합니다. Informix 4GL 프리프 로세서는, 컴파일 시 4GL 개발자가 선택하여, 4GL 소스 코드로부터 의사-기계 코드(p-code) 또는 C 코드를 생 성할 수 있습니다. 이 두 가지의 런타임 결과물 모두 적절히 만들어진 C 함수와 Informix ESQL/C 함수를 불러 낼 수 있습니다.

Informix 4GL 은 객체 코드로 컴파일할 수 있는 C 와 ESQL/C 코드를 생성합니다. 이러한 4GL 은 단순한 응용프로그램을 원하는 수요자들 뿐 아니라 엔터프라이즈급 데이터베이스 응용프로그램을 요구하는 수요자들 에게, 유지하기 쉽고 값이 저렴한 시리얼 터미널(serial terminal)에서 구현하기 쉬운 데이터베이스 응용프로그 램의 신속한 개발을 지원하는 원숙하고 검증된 제품으로 제공하였습니다. 예를 들어, 문자 터미널의 ESCAPE 키를 눌러 영업 거래를 입력하는 소매점 직원이 있다면 아마도 4GL 응용프로그램을 사용하고 있을 것입니다. 사업체와 업계에서의 대내외적인 요구를 충족시키는 것 이외에도 4GL 은 전세계의 정부 및 교육 기관을 위한 솔 루션도 제공합니다. 10,000 명 이상의 개발자들이 4GL 을 사용하여 2 백만 이상의 런타임 라이센스가 사용된 100,000 개 이상의 응용프로그램을 만들었습니다. 이러한 영업 실적은 Informix 의 성장을 뒷받침해주었으며 많은 Informix 판매업자들(VARs)이 성공할 수 있도록 도와주었습니다. 이는 또한 강력한 4GL 응용프로그램을 개발하고 유지할 수 있는 기술과 경험을 소유한 프로그래머 세대를 창출하는 데에도 일조하였습니다.

GUI 와 구현을 위한 탐색

4GL 이 처음 출시된 이래 수 년간, 많은 4GL 고객들은 4GL 응용프로그램을 문자 터미널에서 그래픽 사용자 인 터페이스(GUI)를 지원하는 플랫폼으로 이전해야 할 필요성을 느껴왔습니다. 이상적으로는, 완전히 다른 프로그 래밍 언어로 코드를 새로 작성하는 것 보다는, 그러한 이전이 신속하고 쉽게, 그리고 4GL 코드와 4GL 개발자들 의 기술에 대한 기업 차원의 투자를 유지하고 활용할 수 있도록 이루어지는 것이 바람직하였습니다.

1990 년대 초 몇몇 Informix 선도자들이 이러한 이전 목표에 착수하려고 시도하였습니다. Informix 4GL/GX for Motif(1992)와 Informix 4GL for Windows(1993)는 4GL 코드를 완벽하게 지원했지만, 4GL 고객들 사이에 서 즉각적으로 받아들여지지는 않았습니다. Informix NewEra(1994)는 4GL 구문을 지원하지만, SuperTables 에서 상응하는 동작을 수행하기 위해서는 전형적인 4GL 프로그램에서 가장 중요한 역할을 하는 화면 대화형(screen-interaction) 문들(CONSTRUCT, DISPLAY ARRAY, INPUT 및 INPUT ARRAY)이 대체되어 야 했습니다. 4GL 구문에 대한 컴파일러 지원이 불완전하고, 실행 프로그램 크기가 더 커지고, 객체 지향적인 프 로그래밍(OOP) 패러다임을 사용할 NewEra 개발자들이 필요해지는 등 다양한 이유 때문에 대부분의 4GL 고객 들은 NewEra 를 사용하여 응용프로그램을 GUI 플랫폼으로 이전하는 작업을 포기했습니다.

4GL 고객들이 이러한 제품들을 이전용 툴로서 사용하는 데에 별다른 관심을 보이지 않음을 알게 되자, 인포믹 스에서는 1997 년부터 GUI 실현을 위한 몇 가지 다른 접근 방법을 모색하기 시작했습니다. Informix 에서는 몇몇 앞선 기술을 주의 깊게 평가한 이후, 4GL 고객을 위한 GUI 실현 솔루션인 Universal Compiler 로 상당히 성공한 한 회사의 솔루션이 최상의 기술 솔루션이라는 결론을 내렸습니다.

4J's 의 힘을 얻은 Informix Dynamic 4GL

1998 년 6 월 Informix 는, Informix 4GL 프로그램을 위한 저렴하면서도 검증된 GUI 실현 솔루션으로서, 신제 품인 Informix Dynamic 4GL 을 발표했습니다. Dynamic 4GL 은 ASCII 윈도우, 윈도우 3.1, 윈도우 95, 윈도우 NT 및 X11/Motif 터미널을 비롯한 다양한 플랫폼에서 운영할 수 있는(소스 코드를 변경하지 않고도) 새로운 엔터프 라이즈급 데이터베이스 응용프로그램 개발을 지원합니다. 궁극적으로 Dynamic 4GL 은 Informix 4GL 을 지원 하는 대부분의 플랫폼에서 이용할

수 있게 될 것입니다.

Informix Dynamic 4GL 은 Informix 4GL 구문을 지원하는 Informix 의 제품이지만, 컴파일러와 런타임 기술은 프랑스 파리에 있는 소프트웨어 회사인 4J's Development Tools(이하 4J's)에 의해 개발되었습니다. 1995 년 이래 4J's 는 Universal Compiler 를 광범위한 GUI 및 문자 기반 환경에서 4GL 응용프로그램 구현을 위한 성공적인 제품으로 발전시켜 왔습니다. 이 기술은 4GL 고객들이 추구해온 기능, 성능 및 유용성을 가지고 있기 때문에 인포믹스는 4J's 와 협력관계를 맺었습니다.

Informix Dynamic 4GL 은 Informix 4GL 고객들을 위한 "업그레이드", "중간", 또는 "유지" 차원의 제품이 아님을 분명히 하고자 합니다. Dynamic 4GL 은 Informix 4GL 제품과는 별개로 판매되는 재아키텍처된 신제품입니다. Informix 4GL 은 문자 기반 환경에서의 실행만을 필요로 하는 고객들을 위해 계속해서 판매 및 지원될 것이지만, 앞으로는 Informix Dynamic 4GL 에 우선적으로 개발 및 마케팅 활동이 집중될 것입니다. GUI 플랫폼에서의 구현을 요구하는 Informix 4GL 고객들에게는 최신의 기술을 이용하기 위해 Informix Dynamic 4GL 로 옮겨갈 것을 권장합니다. Informix Dynamic 4GL 의 한 제품 안에는 디버깅 유틸리티 뿐만 아니라 Informix 4GL Compiler 의 기능이 제공되며, 고전적인 Informix 4GL 에 새롭고 향상된 기능들이 추가되어 있습니다.

Informix Dynamic 4GL 의 가장 중요한 기능들을 살펴보면 다음과 같습니다.

- 윈도우 3.11, 윈도우 95 및 윈도우 NT를 비롯한 윈도우 플랫폼에서 4GL 응용프로그램들의 빠른 구현 지원
- GUI 및 문자 기반 플랫폼에서 4GL 응용프로그램의 성능 및 확장성 향상
- 새로운 응용프로그램의 개발 및 활용 비용 절감
- 기존의 4GL 소스 코드와의 완벽한 호환성 제공

Informix Dynamic 4GL 과 Informix 4GL 의 호환성

Informix Dynamic 4GL 은 Informix 4GL 의 모든 구문을 지원합니다. 그러나 이것은 최상의 성능과 보다 작은 크기의 실행 코드를 제공하기 위해 재아키텍처된 새로운 컴파일러이자 런타임입니다. Informix Dynamic 4GL 을 사용하면 문자 기반 또는 GUI 방식의 새로운 데이터베이스

응용프로그램을 만들 수 있을 뿐만 아니라 기존의 4GL 응용프로그램을 GUI 시스템에서 활용할 수 있습니다.

Dynamic 4GL 은 Informix 4GL 버전 6.x 기능들의 논리적 수퍼세트(logical superset)를 지원합니다. Informix 4GL 버전 7.2 의 Global Language Support(GLS) 기능은, 이미 Informix 4GL 버전 7.2 의 DBCENTURY 및 I/O 스트림 기능을 지원하는, Dynamic 4GL 의 다음 버전에서 통합될 것입니다. Interactive Debugger 는 Informix Dynamic 4GL 에서 완벽한 기능의 원시 언어 디버거(native-language debugger)로 대체 되었습니다.

널리 알려진 Informix 의 Informix 4GL 응용프로그램 개발 언어 및 기술에 기초한 Informix Dynamic 4GL 은 개발자에게는 고성능의 견실하고 완벽한 솔루션을 제공하고 사용자에게는 강화된 기능과 시각적으로 친근한 인터페이스를 제공합니다. Informix Dynamic 4GL 은 개발 팀이 기존의 4GL 코드를 이용하여 엔터프라이즈급 응용프로그램을 만들 수 있게 해주며, 또한 완전히 새로운 응용프로그램을 개발할 수도 있게 해줍니다.

쉬운 구현

Informix Dynamic 4GL 은 어떤 시각적 객체도 가리키고 클릭할 수 있도록 마우스를 완벽하게 지원하는 진정한 그래픽 사용자 인터페이스(GUI)를 제공합니다. 윈도우 3.11, 윈도우 95, 윈도우 NT 및 X 윈도우 시스템 클라이언 트에서 모두 같은 실행 프로그램을 사용하여 텍스트 모드나 그래픽 모드, 어디에서나 응용프로그램을 실행할 수 있을 것입니다.

ASCII 기반의 프론트-엔드(front-end) 응용프로그램을 시각적인 그래픽 사용자 인터페이스로 수동 이전하는 것 은, 중요한 코드 변경과 수백 시간의 디자인 및 엔지니어링 노력을 필요로 하는 매우 어려운 작업이 될 수 있습니 다. 그러나 Dynamic 4GL 은, 사용자의 재교육이나 코드 재작성이나 하드웨어를 업그레이드할 필요 없이, 단순 한 재컴파일을 통해 전통적인 4GL 문자 기반의 화면을 그래픽적인 인터페이스로 쉽게 바꾸어 줍니다. GUI 환경 으로 이렇게 "간편하게" 이전함으로써 시장 및 재정 자원에 들일 상당한 시간을 절약할 수 있습니다.

다음 그림은 시리얼 터미널에서 컴파일 및 실행된 Informix 4GL 의 전형적인 화면 양식 윈도우를 보여줍니다.

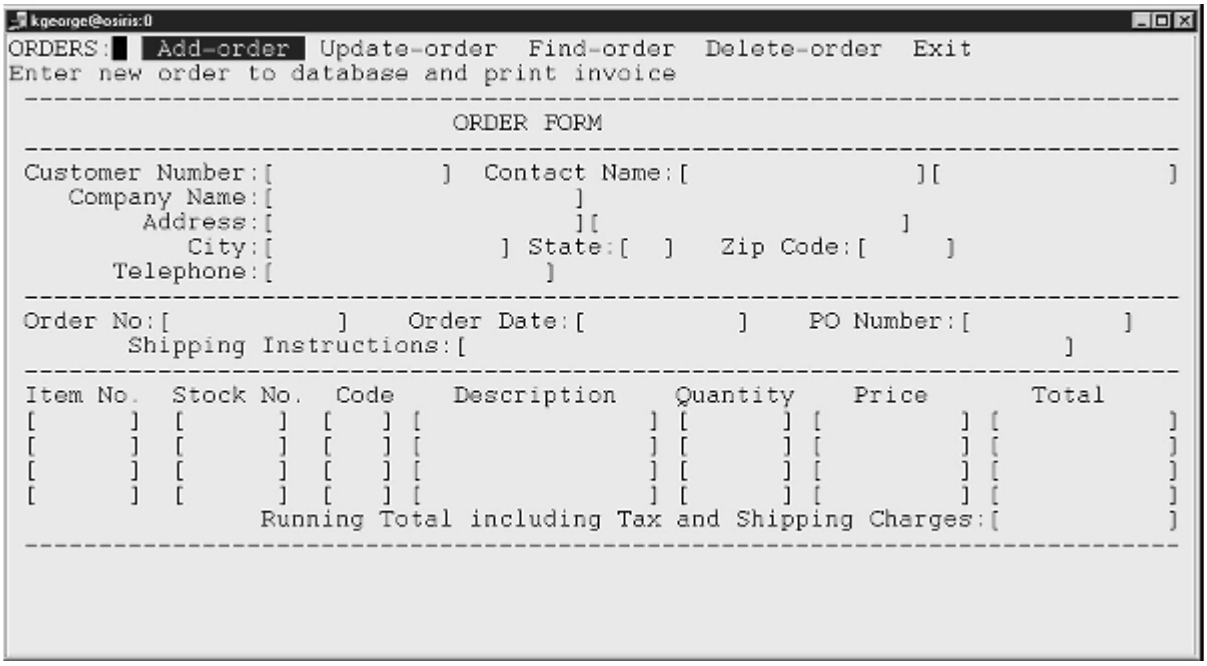


그림 1: 문자 기반 양식의 예

똑같은 4GL 소스 코드를 Informix Dynamic 4GL 로 재컴파일하고, 그 결과로 만들어진 프로그램을 윈도우 NT 시 스템에서 실행했을 때, 똑같은 4GL 소스 코드는 다음과 같이 나타납니다.

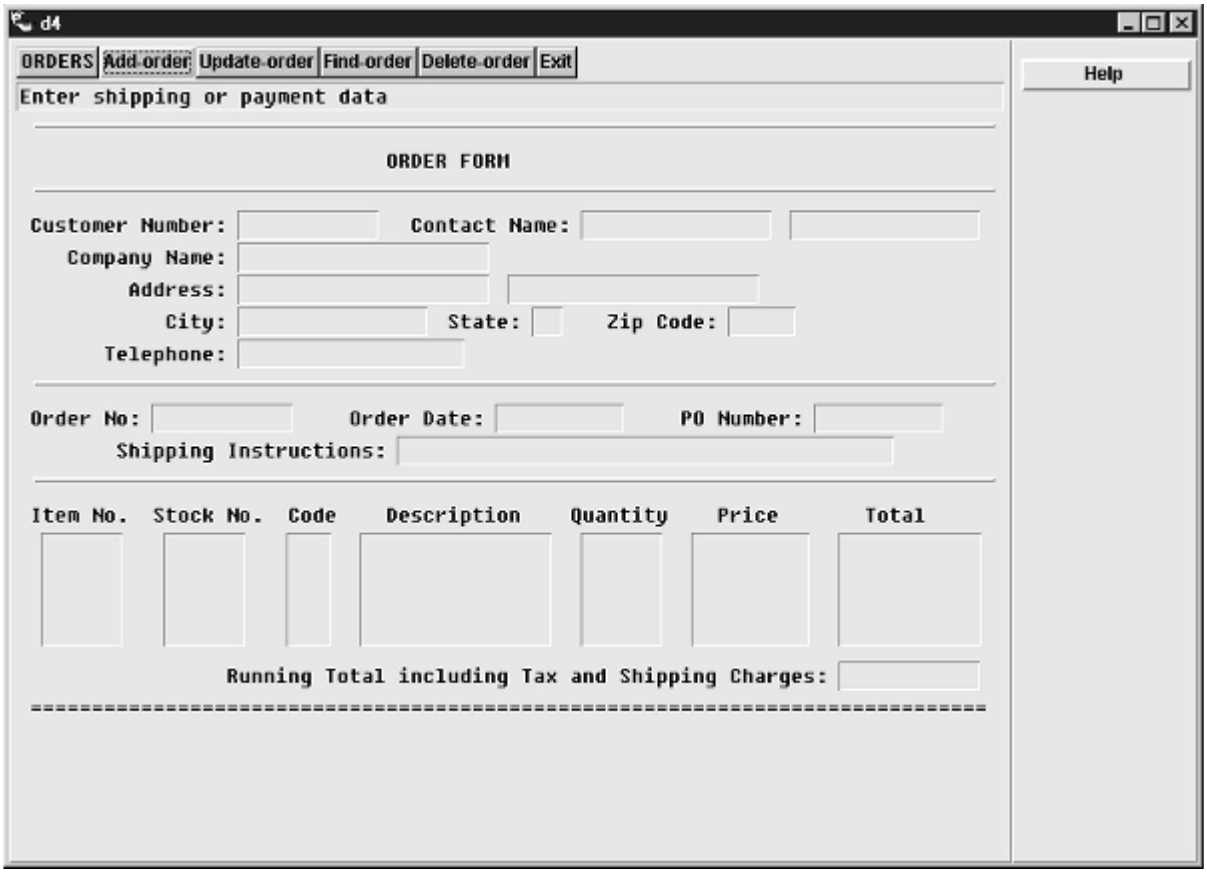


그림 2: 그래픽적인 양식의 예

Dynamic 4GL 은 GUI 기반의 플랫폼에서 4GL 응용프로그램의 외양과 느낌을 향상시키도록 디자인된 추가적인 내장 함수를 Informix 4GL 의 구문에 보충했습니다. 4GL 응용프로그램을 약간만 수정함으로써 팝업 메뉴, 툴 바, 라디오 버튼, 체크 박스, 풍선 도움말을 추가할 수 있으며 막대 그래프, 파이 차트, 사용자 정의 드로잉, 비트 맵 필드 등이 포함된 그래픽적인 리포트 형식을 추가할 수도 있습니다.

최소한의 코딩으로 이러한 기능들을 Dynamic 4GL 프로그램에 추가할 수 있습니다. 원래의 코드를 가지고 항상 작업하므로, "첫 단계"가 필요하지 않습니다. 단순한 재컴파일을 통해 Informix 4GL 화면의 GUI 버전을 만들 어냄으로써, 윈도우 환경에서 기대하는 기능들을 사용자들에게 제공할 수 있습니다. 또한 엔진 컴포넌트의 체계 적 드로잉을 나타내거나 직원들의 사진을 나타내는 필드와 같은 새로운 기능을 응용프로그램에 추가할 수도 있 습니다. 그리고 레이블에서 필드에 이르는 사용자 정의된 시각적 기능(custom visual features)을 추가하기 위 해 Dynamic 4GL 그래픽 화면의 세부 사항들(예를 들면, 크기, 글꼴, 색상 및 위치)을 런타임 시 조절할 수도 있습 니다.

새로운 시각적 객체

화면 양식이 GUI 플랫폼에서 전개될 때, Dynamic 4GL 은 암시적으로(4GL 양식 명세 파일의 전통적인 구문 또 는 화면 대화형 문을 수행하는 과정에서) 또는 새로운 구문이나 새로운 내장 함수를 통해 명시적으로 시각적 객 체를 지원할 수 있습니다. 다음은 Dynamic 4GL 에서 나타낼 수 있는 시각적 객체들입니다.

객체 유형	**객체 명시 방법**
Combo fields	.per 파일의 WIDGET과 CONFIG 문자열
Check buttons	.per 파일의 CHECK BUTTON 키워드
Radio buttons	.per 파일의 RADIO BUTTON 키워드

Radio buttons	.per 파일의 RADIO BUTTON 키워드
Scrolling fields	SCROLL 속성(attribute)
Dialog boxes	FGL_WINBUTTON() 내장 함수
Question boxes	FGL_WINQUESTION() 내장 함수
Message boxes	FGL_WINMESSAGE() 내장 함수
Prompt boxes	FGL_WINPROMPT() 내장 함수

표 1: Informix Dynamic 4GL 에서 표현할 수 있는 시각적 객체

위 객체들과 더불어 GUI 환경에서 사용할 수 있는 기타 시각적 객체들에 대한 자세한 설명은 Dynamic 4GL 문 서에 자세히 기술되어 있습니다.

GUI 와 문자 기반 전개 방식 사이의 호환성

프로세스간 통신: I/O Streams 와 DDE

Informix 4GL 은 RUN 문을 통해 어느 정도 프로세스간 통신을 지원하지만, 이는 윈도우 시스템에서 사용하기가 쉽지 않습니다. Informix Dynamic 4GL 에서는 윈도우 95 와 윈도우 NT 디스플레이 서버에서 특정 윈도우 프로그램을 실행하기 위해 두 개의 새로운 내장 함수, WinExec()와 WinExecWait()를 사용할 수 있습니다. WinExec() 함수는 UNIX 시스템에서 4GL 의 RUN ... WITHOUT WAITING 문과 유사한 효과를 갖습니다. 다른 점이 있다면 WinExec()는 윈도우 프로그램에서도 실행할 수 있다는 것입니다. Informix Dynamic 4GL 은 RUN 문을 지원하지만, 자체 윈도우에서 자식 프로세스(child process)를 실행하는 FGL_SYSTEM()으로 이를 보완합니다.

있습니다. 일부 독자는 이러한 채널 확장 기능들의 이름에서 C++ 스타일의 유효 범위 해결 연산자(scope-resolution operator)를 인식할 수 있을 것입니다.

이러한 새로운 I/O 스트림 기능 이외에도 Informix Dynamic 4GL 은 응용프로그램 간 데이터 교환을 위한 마이크로소프트 통신 프로토콜인 Dynamic Data Exchange(DDE)를 지원합니다. 함께 작동하는 프로그램들은 윈도우 PC 나 UNIX 웍스테이션에서 실행할 수 있지만, 프로그램들 중 적어도 하나는 윈도우 클라이언트에서 실행해야 합니다.

다음의 새로운 함수들은 Dynamic Data Exchange 를 지원합니다.

이름	효과
DDEConnect()	DDE 연결 열기
DDEExecute()	DDE 연결을 사용하여 명령 실행
DDEPoke()	윈도우 프로그램으로 데이터 전송
DDEPeek()	윈도우 프로그램으로부터 값 검색
DDEFinnish()	특정 DDE 연결 종료
DDEFinnishAll()	모든 열려 있는 DDE 연결 종료
DDEGetError()	마지막 DDE 에러 메시지 검색

표 3: Dynamic Data Exchange를 지원하는 새로운 함수들

다음 그림에서 보여주듯이, Informix Dynamic 4GL에서는

DDE에 대한 지원에 네 개의 처리 과정이 포함됩니다.

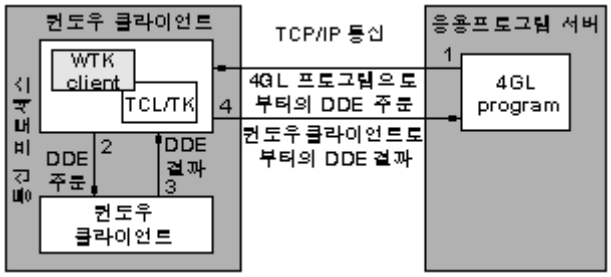


그림 3: DDE 확장 기능을 이용하는 네 부분의 처리 과정

다음은 DDE 확장 기능을 이용하는 네 단계를 설명한 것입니다.

1. 4GL 응용프로그램 서버는 TCP/IP 채널을 사용하여 윈도우 디스플레이 서버("WTK 클라이언트"라고 도 함)로 DDE 요구(request)를 보냅니다.
2. 윈도우 디스플레이 서버는 TCL/TK 함수를 사용하여 DDE 요구를 실행하고, DDE 연결을 통해서 함께 작동하는 윈도우 응용 프로그램으로 데이터를 보냅니다.
3. 함께 작동하는 윈도우 응용프로그램은 요구를 실행하고, DDE 연결을 통해서 그 결과(요구한 데이터일 수도 있고 에러 코드일 수도 있음)를 윈도우 디스플레이 서버로 보냅니다.
4. 윈도우 디스플레이 서버는, TCP/IP 채널을 사용하여 그 결과를 4GL 응용프로그램 서버로 보냅니다.

여타 DDE 구현의 경우와 마찬가지로, Informix Dynamic 4GL 과 함께 작동하는 윈도우 응용프로그램에서도 역시 DDE 를 지원해야 합니다

DISPLAY ARRAY 구문의 향상

Informix Dynamic 4GL 이 Informix 4GL 응용프로그램에 선사한 가장 뛰어난 새로운 기능들은 시각적인 것이지만, Informix Dynamic 4GL 은 그 이외에도 새로운 아키텍처, 이식성, 성능 향상과 같은 기능 추가 및 DISPLAY ARRAY 문의 구문에 대한 향상도 가져왔습니다.

이러한 구문 향상은 4GL 기능들을 보다 체계적으로 만들어 주며, Informix 4GL 이 INPUT ARRAY 에 대해서만 지원하던 일부 컨트롤 블록의 키워드와 액션을 DISPLAY ARRAY 문으로까지 확장함으로써, 개발자와 사용자의 업무를 단순화시켜줍니다.

```
DISPLAY ARRAY record-array TO screen-array

        [ATTRIBUTE (attribute-List)]

        {ON KEY* (key-list)

        statement
```

...

[EXIT DISPLAY]

...

END DISPLAY | [END DISPLAY]}

* 선택적으로 사용할 수 있는 다른 키워드로는 BEFORE DISPLAY, AFTER DISPLAY, BEFORE ROW, 그리고 AFTER ROW 가 있습니다.

Informix Dynamic 4GL 은 또한 DISPLAY ARRAY 문에서 CONTINUE DISPLAY 및 EXIT DISPLAY 키워드에 대한 지원을 새로 추가하였는데, 이들은 다른 4GL 복합문에서의 CONTINUE 및 EXIT 문과 유사한 기능을 합니다. 이러한 구문 향상으로 인해 데이터 디스플레이를 위한 사용자 인터페이스가 단순화되고, 문자 기반 플랫폼과 GUI 플랫폼에서 모두 4GL 프로그램 로직의 기능이 확장됩니다.

이식성과 신속한 개발

Informix Dynamic 4GL 은 위의 이점 이외에도 더 많은 것을 제공합니다. Dynamic 4GL 에서 생성한 P 코드는 c4gl 에서 생성한 객체 코드와 거의 유사한 런타임 성능을 제공합니다. 이것은 고객들이 기대하는 런타임 성능을 희생하지 않고도 개발자들이 개발 기간이 빠르다는 P 코드의 장점을 계속해서 취할 수 있음을 의미합니다. Informix Dynamic 4GL 에서 생성하는 P 코드는 다음과 같은 추가적인 장점을 가지고 있습니다.

- 공간의 절약. 컴파일된 P 코드는 기계 코드(machine code)보다 디스크 공간을 훨씬 덜 차지합니다. P 코드 모듈은 런타임 시 공유된 텍스트 요소들과 연결되며 동적으로 로드됩니다. 그 결과, 생성된 P 코드 응용프로그램은 원본보다 30 배 가량 크기가 작으며, 하드웨어 비용을 상당히 줄여줍니다.
- 필요 메모리의 절감. 실행 시 "동적 가상 기계(Dynamic Virtual Machine)"는 사용 중인 P 코드 모듈만을 동적으로 메모리에 로드합니다. 필요 메모리의 절감은 메모리 사용을 크게 줄임으로써 비용을

절약할 수 있음을 의미합니다.

- 이식성. P 코드로 컴파일된 응용프로그램은 모든 UNIX 및 윈도우 NT 플랫폼에서(동적 가상 기계가 이식된 곳이면 어디서나) 완벽하게 이식이 가능합니다. 이러한 "한 번 작성하고, 모든 곳에서 실행하기" 개념은 다양한 플랫폼에서 응용프로그램을 판매 또는 운영하는 회사들에게 비용과 노력을 상당히 절약할 수 있는 혜택을 제공합니다.

마지막 기능이 Dynamic 4GL 을 이용해야 하는 가장 설득력 있는 이유일 것입니다. 필요 메모리 절감은 향상된 런타임 성능 및 3-티어(tier) 아키텍처와 함께, 80386 또는 80486 기반의 낙후된 PC 에서조차 복잡한 4GL 응용프로그램에 대한 만족할만한 성능을 제공합니다. 해석 가능한 Informix Dynamic 4GL P 코드 모듈에는 프로그램 텍스트와 데이터가 실제로 포함되어 있지 않습니다. 대신 거기에는 프로그램 텍스트와 데이터를 포함하는 P 코드 객체 파일을 가리키는 일련의 포인터 세트만이 있을 뿐입니다. P 코드가 실행되면 동적 가상 기계는 MAIN 프로그램 블록에 대한 코드를 로드하여 이를 실행하기 시작합니다. 현재 실행되고 있는 4GL 함수가 아직 메모리에 로드되지 않은 다른 함수를 호출하면, 동적 가상 기계는 호출된 함수를 포함하는 P 코드 객체 모듈을 로드하고 계속해서 프로그램을 실행합니다.

이렇게 메모리 관리 기능이 향상된 결과, Informix Dynamic 4GL 의 해석 가능한 P 코드 모듈의 크기는, 사용자의 요구에 의해 또는 프로그램 로직 내의 데이터 독립적인 비즈니스 룰에 의해 요구될 것으로 추정되는 모든 P 코드를 포함하는 이전의 코드들보다 작습니다.

3-티어 클라이언트-서버 모델

Informix 4GL 은, 데이터베이스 서버에서는 SQL 요구를 처리하고 클라이언트 시스템에서 실행되는 응용프로그램에서는 업무 로직과 시각적 디스플레이 과제를 처리하는, 고전적인 클라이언트-서버 모델을 사용합니다. 만약 Informix 4GL 응용프로그램에서 디스플레이할 화면 양식이 많거나 복잡한 업무 규칙을 처리해야 한다면, 클라이언트 시스템의 리소스에 부하가 너무 많아서 성능이 저하될 우려가 있습니다. "팻 클라이언트 신드롬(fat client syndrome)"이라고도 종종 불리는 이러한 상황은 클라이언트 응용프로그램이 리소스에 부과하는 요구의 처리 능력과 클라이언트 시스템 성능 사이의 불일치 때문에 생깁니다. 이는 Informix 4GL 만의 문제라기보다는 모든 2-티어 클라이언트-서버 모델이 가지는 문제라고 할 수 있습니다.

이러한 문제의 증상이 나타날 경우 이를 해결할 수 있는 두 가지의 분명한 처리 방법이 있습니다. 원하는

성능이 나오도록 클라이언트 하드웨어를 업그레이드하거나 처리 과정의 일부를 다른 시스템에 할당하여 기존 하드웨어의 부하를 줄이는 것이 그것입니다. Dynamic 4GL 은 "팻 클라이언트" 신드롬을 피할 수 있도록 만들어졌습니다. 다음 그림은 Informix 4GL 의 2-티어 클라이언트/서버 모델과 Informix Dynamic 4GL 의 조정 가능한 3-티어 모델을 비교한 것입니다.

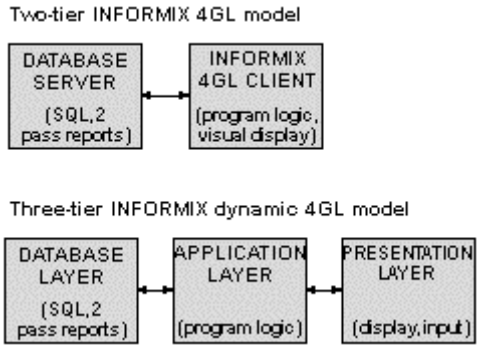


그림 4: Informix 4GL 의 2-티어 클라이언트/서버 모델과 Informix Dynamic 4GL 의 3-티어 모델

Informix Dynamic 4GL은 Informix 4GL에서 클라이언트 응용프로그램에 해당하는 것을 두 개의 레이어, 즉 "응용프로그램 레이어"와 "프리젠테이션 레이어"로 논리적으로 분할하는 3-티어 모델에 기초하고 있습니다. 이렇게 하여, 프로그램 로직을 포함하는 4GL 코드 부분은 시각적 인터페이스를 지원하는 코드와 분리된 별개의 레이어로 처리됩니다. 데이터베이스 서버의 역할은 2-티어 모델과 다르지 않습니다. 하드웨어의 측면에서 보면, 4GL 프로그램 로직은 응용프로그램 레이어를 지원하는 시스템("응용프로그램 서버")에서 실행되며, 프리젠테이션 레이어는 "디스플레이 서버"라고도 부르는 시스템에서 실행됩니다. 이러한 용어를 피하는 사람들도 있는데, 그것은 "서버" 구성 성분이 세 개나 있어서 복잡한 "클라이언트-서버" 모델 내에서 클라이언트 처리에 대한 개념이 모호해질 수 있기 때문입니다. 이것은 세 개의 하드웨어 시스템에서 구현할 수 있는 논리적인 모델이지만, 실제로 응용프로그램 레이어는 프리젠테이션 레이어(또는 데이터베이스 레이어)를 실행하는 시스템과 같은 시스템에서 실행할 수도 있습니다. 영업사원이 랩탑 시스템에서 고객에게 4GL 응용프로그램을 보여주는 것과 같은 특수한 목적이 있는 경우에는, 세 개의 논리적 레이어를 모두 지원하도록 하나의 하드웨어 플랫폼을 구성할 수도 있습니다. 그러나 각각의 레이어를 각기 분리된 플랫폼에서 실행할 경우, 플랫폼들의 운영체제가 모두 같을 필요는 없습니다. 예를 들어, UNIX 응용프로그램 서버에서 실행되는 Informix Dynamic 4GL 프로그램이 매킨토시 X11 디스플레이 서버에서 실행되는 프리젠테이션 레이어를 지원할 수도 있습니다. 3-티어 모델의 경제적인 이점 중 하나는 강력한 PC가 아니라도 디스플레이 서버로 사용할 수 있다는 것입니다. 또 다른 이점은 응용프로그램 서버와 디스플레이 서버 사이의 네트워크 트래픽을 최소화한다는 것입니다.

네트워크를 통해 디스플레이 서버로 전송해야 하는 데이터는, 질의(query)에 대해 데이터베이스 서버에서 돌려주는 전체적인 행위 세트가 아니라, 단순히 텍스트 또는 프리젠테이션 레이어에서 요구하는 이벤트 뿐입니다. 이러한 3-티어 아키텍처는 속도가 느린 다이얼업 라인에서도 상당한 성능을 낼 수 있으므로, 값비싼 네트워크 하드웨어를 업그레이드하지 않아도 됩니다.

소프트웨어 재개발 비용 최소화

Dynamic 4GL의 진가는 이를 사용하여 기존의 4GL 응용프로그램의 수명과 프로그래밍 팀의 집약된 기술을 확장함으로써 소프트웨어 재개발을 피하고 이식 비용을 낮출 때 발휘됩니다. Informix Dynamic 4GL은 4GL 응용프로그램을 자체 개발한 고객들 또는 4GL로 만든 응용프로그램 패키지를 개발한 판매업자들(VARs)에게 놀라운 경제적 혜택을 제공합니다. 이 두 그룹 모두 유사한 문제에 직면해 있습니다. 새로운 기능에 대한 요구사항과, 전형적인 문제로서 윈도우 또는 X11/Motif를 지원해야 하는 과제를 안고 있습니다. 그들이 직면한 문제는, "4GL에 머물러 있어야 하나, 아니면 새로운 개발 툴로 옮겨가야 하나?" 라는 것입니다. 새로운 툴로 옮기려면 데이터베이스 기반을 재평가해야 하기 때문에, 이것은 매우 심각한 문제입니다. 판매업자들은 그들의 4GL 응용프로그램이 문자 기반의 인터페이스를 가지고 있으며 ASCII 터미널이나 TTY 에뮬레이터에서만 실행되므로 경쟁력이 떨어져서 수입이 감소되는 상황에 직면해 있을지도 모릅니다. 그러한 판매업자들은 보통 고객의 기본적인 기능적 요구 사항들을 충족시키는 안정되고 견실하며 기능이 풍부한 응용프로그램을 가지고 있지만, 문자 기반의 인터페이스 때문에 구식이고 비직관적이며 사용하기 어려운 것으로 인식될 것입니다. 내부적으로 사용할 목적으로 4GL 응용프로그램을 개발한 회사들도 내부 사용자들이 보다 직관적이고 시각적으로 보기 좋은 인터페이스를 요구한다면 유사한 문제를 갖고 있을 것입니다. 그들이 갖고 있는 기존의 응용프로그램은 데이터 입력에는 적합할지 몰라도 현대의 사무원들은 보통 그래픽적인 외양과 느낌을 기대합니다. 어느 경우이든, 새로운 응용프로그램을 개발하려면 수 년의 시간과 엄청난 비용이 들 것입니다. 그러한 재개발 노력에는 상당한 위험이 수반되는데, 소프트웨어 리엔지니어링 노력이 크면 클수록 사용할만한 대체물을 얻기 힘들 가능성이 더 커집니다. 이러한 "최악의" 실패를 피할 수 있고 적절한 시간 내에 리엔지니어링 대체물의 개발 및 테스트가 완료되더라도, 그 결과로 얻은 응용프로그램은 원래 4GL 응용프로그램의 견실성이 결여되고 기능이 더 적으며 업무상의 기능성이 떨어질 수 있습니다. 최종 결과물은 종종 "버전 1"이 되는데, 덜 견실하고 덜 안정적이며 기능적인 면에서 떨어집니다.

하드웨어 비용 최소화

비주얼 베이직(Visual Basic)이나 파워빌더(PowerBuilder) 같은 전형적인 2-티어 클라이언트 개발 환경에서는 결과로 얻는 응용프로그램에 대해 실질적인 데스크탑 시스템이 필요합니다. 이 말은 즉 Informix 4GL 응용프로그램을 대체하려면 사용자의 업무 계획에 기반해서가 아니라 응용프로그램의 요구에 기반하여 하부구조(infrastructure)를 강제로 업그레이드해야 한다는 것입니다. 게다가 데이터베이스 서버와 2-티어 응용프로그램용 데스크탑 사이의 네트워크 트래픽은, 인터페이스 명령 및 표시할 데이터만을 데스크탑으로 보내는 Informix Dynamic 4GL 3-티어 시스템의 경우보다 실제로 더 큽니다. 네트워크 설비 확장을 통해 2-티어 시스템의 병목 현상을 피하려는 하드웨어적인 노력은 엄청난 비용의 네트워크 하부구조 업그레이드를 요구합니다.

Informix Dynamic 4GL 의 주요 이점은, 값비싼 소프트웨어 리엔지니어링이나 하부구조 업그레이드 없이도, 기존의 4GL 응용프로그램을 향상시키고, 윈도우 고객의 요구를 충족시키며, UNIX 와 윈도우 NT 서버를 융통성 있게 선택할 수 있는 능력에서 오는 것입니다. Informix Dynamic 4GL 은 기존의 4GL 응용프로그램의 인터페이스를 신속하게 향상시킴으로써, Informix 고객들로 하여금 기존의 응용프로그램, 하부구조 및 기술 인력에 대해 투자한 것을 그대로 활용하게 해줍니다.

맺음말

1998년 초 Informix에서는 Y2K 이슈에 대한 DBCENTURY 솔루션, 대부분의 유럽 및 아시아 언어를 위한 지역 단위의 세계 언어 지원(Global Language Support), 그리고 IDS 데이터베이스와의 향상된 통신 기능이 추가된 Informix 4GL 7.2를 발표했습니다. 문자 터미널용 클라이언트 응용프로그램들을 위해, Informix에서는 Informix 4GL의 판매 및 지원을 계속할 것입니다. 그래픽 사용자 인터페이스를 필요로 하지 않는 고객들과 기존의 응용프로그램에 만족하는 고객들은 계속해서 Informix 4GL을 사용할 수 있으며 현재의 7.2 버전으로 업데이트할 것을 고려해보아야 할 것입니다. 그러나 성능의 향상과 문자 기반 및 GUI 환경 양자에 대한 지원 때문에 Informix Dynamic 4GL은 Informix 4GL의 새로운 미래라고 할 수 있습니다. 4GL 응용프로그램을 위한 새로운 기능 개발은 우선적으로 Informix Dynamic 4GL에 대해 이루어질 것입니다. 현재 진행되고 있는 이후 버전에서는 비영어권에 대한 지원을 확대하고 웹 활용에 대한 지원을 제공할 것입니다.

자바 데이터베이스 연결

자바 프로그래밍 언어

자바를 만든 Sun Microsystems 에서는 자바를 운영 체제 독립적이며 객체 지향적인 프로그래밍 언어라고 설명합니다. "한 번 작성하고, 모든 곳에서 실행하기"라는 슬로건은 널리 대두되고 있는 인터넷 기반의 분산 컴퓨팅 환경에서 주목을 끌만한 제안이라고 할 수 있습니다.

자바로 만든 회사의 정보 시스템(IS)을 개인이나 부서, 그래픽 사용자 인터페이스 지원 여부 또는 하드웨어(PC, 노트북, 팜톱(palmtops), PDA 등)에 상관없이 모든 사원들이 사용할 수 있다고 상상해 보십시오. 그리고 아무 제한 없이 회사 인터넷, 인트라넷 나아가 엑스트라넷 프로젝트를 시작할 수 있다고 상상해 보십시오. 이것이야말로 자바가 그토록 주목을 끌며, 전세계의 IS 기업 경영자들과 개발자 및 사용자들의 관심을 끄는 진정한 이유라고 하겠습니다.

다음으로 주목할만한 제안은 "관리가 필요 없는(Zero Admin)" 자바 응용프로그램과 애플릿의 개발입니다. 회사 IS 시스템의 모든 사용자들에게 IS 의 새로운 버전에 액세스하기 위한 URL(Uniform Resource Locator)을 보내기만 함으로써 시스템 수정이나 향상에 대해 알릴 수 있다고 상상해 보십시오. 이로 인한 비용 절감은 상상을 초월할 것입니다.

자바의 구현과 아키텍처

자바는 가상 기계(VM: virtual machine)에 의해 구현됩니다. VM 이란 호스트 운영 체제 내에서 실행된 이후에 실제 컴퓨터 장치에서 실행되는 소프트웨어적으로 구현된 컴퓨터를 말합니다. 이것은 또한 자바 응용프로그램과 애플릿의 런타임 환경이라고 말할 수도 있습니다.

자바 VM(JVM)은 'C'와 자바로 작성됩니다(일부 어셈블러 코드는 보다 최신 버전의 내용을 취하고 있습니다). 자바 응용프로그램과 애플릿은 개발 과정에서 자바 바이트 코드로 컴파일되며, 바이트 코드 해석기의 역할을 하는 JVM 내에서 실행됩니다. 자바 바이트 코드는 장치 독립적인 중간 기계 코드 포맷이라고 할 수 있습니다. 이러한 기계 코드 포맷은 특정한 장치를 요구하지 않기 때문에 JVM 은 플랫폼에 대해 독립적이라는 장점을 갖습니다.

자바 객체는 자바 클래스들을 사용하여 만드는데, 이러한 클래스들은 자바 클래스 인터페이스의 정의 및 구현(자바 언어 소스 코드 파일에 포함된)의 컴파일된 최종 산물이라고 할 수 있습니다.

"Hello WorldObject" 자바 객체를 위한 자바 객체 어셈블리 단계

다음은 자바 객체(Java Object) 어셈블리 단계입니다:

1. 소스 파일 추출(파일명은 HelloWorldObject.java):
2. Import java.*
3.
4. Class HelloWorldObject {
5.
6. Public static void main(String[ ] args) {
7.
8. System.out.println("Hello World!!!");
9.
10. }
11.
12. }
13.
14. 소스 파일 컴파일(호스트 운영 체제 명령줄로부터):
15. javac HelloWorldObject.java
16.
17. 자바 클래스 파일 생성:
18. HelloWorldObject.class
19.
20. 자바 응용프로그램 실행(객체 구현):
21. java HelloWorldObject
22.

일반적으로 자바 응용프로그램, 애플릿 및 빈(beans)은 객체 메시징 프로토콜(메시징 통신 미들웨어와 상관없음)을 통해 함께 작동하는 수많은 다른 자바 객체들로 구성되어 있습니다. 그리고 보통 자바 개발자는 직접 객체를 만들거나, 다른 사람들이 만든 객체들을 통합합니다. 일단 객체를 만들면 개발자들은 객체들간의

메시징 프로토콜을 구상합니다. 이렇게 하여 최종 사용자들은 자바 응용프로그램, 애플릿 또는 빈을 경험하게 되는 것입니다.

객체 지향적인 프로그래밍 언어와 개념에 대한 추가 정보는 다음 URL을 참고하십시오:

http://www.cgl.uwaterloo.ca/~anicolao/termpaper.html

자바 사용하기

응용프로그램(Apps)

자바 응용프로그램은 명령줄로부터 실행되며, 각 응용프로그램의 호스트 컴퓨터에 설치 또는 이전한 후에, 다음의 명령을 사용하여 컴퓨터의 JVM 내에서 실행된다는 점에서 전통적 응용프로그램 모델과 유사합니다:

```
java myappclass
```

애플릿

애플릿은 일반적으로 생각하는 것처럼, 페이지에 존재하지 않습니다. 실제로 애플릿은 웹 문서 내에서 HTML(HyperText Markup Language) 태그를 통해 인식되는 자바 클래스입니다. 자바 애플릿은 인터넷 또는 회사 인트라넷이나 엑스트라넷 어딘가에 있는 웹 서버로부터 로드됩니다.

자바 빈

미리 만들어진 자바 구성요소를 자바 빈(Beans)이라고 합니다. 자바 빈 API 는 자바 객체들로(자바 빈의 인터페이스 명세를 사용하여) 구성요소 객체들을 조립하는 자세한 내용을 제공합니다. 이러한 구성요소 객체들은 자바 빈 호환 컨테이너 환경에서 사용할 수 있습니다. 그러한 컨테이너 환경의 전형적인 예로 문서, 응용프로그램 어셈블리 환경, 그리고 OpenDOC 과 ActiveX 와 같은 기타 소프트웨어 구성요소 명세를 들 수 있습니다.

자바 빈은 운영 체제 독립적인 조립식 응용프로그램 구성요소의 개발과 구현을 손쉽게 해줍니다. 이러한 구성요소에는 그래픽 요소, 질의(query) 엔진, 프로세스 엔진 그리고 완전한 응용프로그램 등이 포함됩니다. 지금까지는 OpenDOC(주로 매킨토시 OS 와 OS/2 기반)과 ActiveX(주로 마이크로소프트 윈도우 기반)에서 볼 수 있듯이, 주로 운영 체제에서만 경험할 수 있었습니다.

주의

"조립식 구성요소"라는 용어를 사용한다고 해서, 실행 시 소프트웨어 구성요소들이 보여야 할 필요는 없다는 점에 유의하십시오.

자바 클레스 이전 방법

이 문서의 앞부분에서 설명하였듯이, 자바 VM 은 자바 객체의 실행 환경 역할을 합니다. 그러므로 자바 응용프로그램/애플릿/빈을 구현하려면, 우선 서비스를 제공할 해당 VM 에서 구성 클래스들을 실행할 방법이 마련되어야 합니다. 여기에서는 이러한 "자바 클래스 이전" 과정을 소개합니다. 이 과정은 성능과 안전성이라는 의미에서 자바를 인식해 나가기 위한 하나의 중요한 요소입니다. 일반적으로 자바 클래스는 다음과 같은 방법을 사용하여 이전됩니다:

- 웹 서버와 HTTP
- 푸쉬(Push) 기반 복제
- Internet Inter Orb Protocol(IIOP)을 사용하는 CORBA 기반의 Object Request Brokers(ORBs)

웹 서버와 HTTP

HTTP(HyperText Transport Protocol)는 웹 브라우저(HTTP 클라이언트)와 웹 서버(HTTP 서버) 사이에서 파일을 전송할 수 있도록 해주는 인터넷 프로토콜입니다.

자바 애플릿은 본질적으로 HTML 문서와 HTTP 파일 전송 프로토콜에 완전히 연결됩니다. 애플릿은 문서 내에서 HTML 태그를 통해 인식되며, 브라우저가 해당 웹 서버로 자바 클래스 파일을 요청하는 추가 HTTP 메시지를 보내도록 하는 것이 바로 이 HTML 태그이기 때문에 이러한 연결 관계가 가능합니다.

이러한 과정을 다음과 같이 설명할 수 있습니다:

1. 예를 들어, 사용자는 브라우저에 다음과 같이 URL을 입력합니다: http://oplweb.openlinksw.com/demo/JDK1.1/WebScrollDemo.htm
2. 브라우저는 URL을 보내고, "openlinksw.com"이라는 도메인 내에 "www"라는 이름을 가지며 HTTP를 부르는 서버 프로세스(실제 웹 서버)를 실행하는 서버 기계에서, "Demo/JDK1.1"이라는 디렉토리 내에

위치한 "WebScrollDemo.html" 문서를 요구하기 위해 HTTP 프로토콜을 사용하기로 결정합니다.

3. 웹 서버는 브라우저로부터 파일 요구를 받고 "WebScrollDemo.html"이라는 파일을 찾습니다. 일단 파일을 찾으면 서버는 네트워크를 통해 요구한 브라우저로 파일을 보냅니다r.
4. "WebScrollDemo.html" 파일을 받을 때 브라우저는 문서 내에 포함된 다음과 같은 HTML 태그를 처리해 나갑니다.

```
5. <html>
6.
7. <head>
8.
9. <title>WebScrollDemo</title>
10.
11. </head>
12.
13. <body>
14.
15. <hr>
16.
17. <applet
18.
19.    code=WebScrollDemo.class
20.
21.    id=WebScrollDemo
22.
23.    width=600
24.
25.    height=400 >
26.
27. </applet>
```

28.

29. <hr>

30.

31. <A HREF="WebScrollDemo.java" >

32.

33. <I>The Java Source</I></A>

34.

35. </body>

36.

37. </html>

38.

39. 이 문서 내의 "<applet>"이라는 태그는 "openlinksw.com" 도메인 내의 "www" 서버 기계로 또 다른 요구를 보내야 함을 브라우저에 알려주고, "Demo/JDK1.1" 디렉토리 내에 있는 "WebScrollDemo.class"(실제 자바 클래스) 이름의 파일을 요구합니다.
40. 웹 서버는 "WebScrollDemo.class" 파일을 찾아 브라우저로 다시 보냅니다.
41. 자바 클래스 파일을 받으면 브라우저는 자바 가상 기계에 대한 자신의 버전을 사용하여 그것을 실행합니다.
42. 사용자는 자바 애플릿과 상호 작용합니다.

장점

HTTP 의 장점은 다음과 같습니다.

1. HTTP는 웹 브라우저의 일반적인 보급으로 인해 널리 사용됩니다.
2. 자바 객체 및 HTML 문서와 같은 자원들을 쉽게 중앙으로 모을 수 있습니다.

단점

HTTP의 단점은 다음과 같습니다.

1. HTTP는 전송하는 파일 포맷에 민감하지 않기 때문에 모든 파일을 동일하게 취급합니다.
2. HTTP는 국적 없는 프로토콜로, 사용 시 처리 과정에서 사용자의 위치를 알지 못하며 주의를 기울이지도 않습니다.
3. HTTP는 매우 기본적인 사항을 수행하기 위해 만들어진 단순한 프로토콜입니다.
4. HTTP는 각각의 반응에 대해 새로운 연결을 만들기 때문에 본질적으로 비효율적입니다.

푸쉬(Push) 기술

HTTP 의 명백한 결점에 대응하기 위해 최근 "푸쉬 기술"로 널리 알려진 자바 클래스 이전에 대한 접근법이 대안으로 대두되었습니다.

푸쉬 기술(PT)은 지능적인 캐쉬 기능과 파일 복제 구조를 가지고 있어서, 원격 응용프로그램 서버 기계로부터 호스트 기계(이 경우에는 클라이언트)로 자바 바이트 코드를 복사한 다음, 캐쉬되도록 합니다(계속 사용할 수 있도록 복사한 내용이 보관됩니다). 응용프로그램 클래스에 대한 변경 사항이 있을 때에는, 서버에서 JVM 클라이언트에게 이를 알려 주고, 클라이언트 기계로부터 승인을 받은 후에 네트워크를 통해 해당 클라이언트 JVM 으로 갱신된 응용프로그램 클래스가 보내집니다(푸쉬 프로세스).

PT 제품은 항상 클라이언트와 서버 소프트웨어 구성요소를 포함하며, 자바 바이트 코드 이전 메커니즘으로서 자바 응용프로그램의 성능을 실질적으로 높일 수 있습니다. 클라이언트와 서버 사이의 클래스 이전 빈도가 HTTP 에 비해 매우 낮기 때문에 자바 응용프로그램의 성능이 상당히 증가하는 것입니다.

장점

장점은 다음과 같습니다.

1. 자바 응용프로그램의 성능을 향상시킵니다.
2. 자바 객체를 활용하는 매개체로서 웹 브라우저에만 의존하지 않도록 해줍니다.
3. 자바 클래스 이전을 위한 유일한 또는 우세한 프로토콜로서 HTTP에만 의존하지 않도록 해줍니다.
4. "관리 불필요(Zero Admin)"라는 가치 제안을 기본적으로 강화하면서 관리의 부담을 줄여 줍니다.
5. 자바 응용프로그램을 서버에서 집중 관리하며, 요청한 클라이언트에게 전송합니다.

단점

단점은 다음과 같습니다.

1. 대부분의 푸쉬 서버는 자바로 작성되지 않았으므로(이 글을 쓸 당시에는 하나도 없었음), 결과적으로 "한 번 작성하고, 모든 곳에서 실행하기"라는 가치 제안에 합당하지 않습니다. 푸쉬 서버는 현재 솔라리스(Solaris)와 윈도우 NT에서만 사용 가능합니다.
2. 푸쉬 기술 솔루션의 클라이언트 부분이 이 기술을 사용하려는 각 클라이언트에 설치되어야만 합니다. 그러므로 "관리 불필요"라는 가치 제안은 사실상 성취되지 않습니다.
3. 푸쉬 제품은 인터넷 클라이언트로서 PC가 대부분인 오늘날의 현실에서 사용 가능한 제한된 디스크 자원을 너무 빨리 소모할 수 있습니다. 인터넷 클라이언트 장치와 지역 저장 장치는 점점 작아지지 커지지는 않을 것입니다.

Object Request Brokers

몇 년 전에 OMG(Object Management Group: http://www.omg.org)에서는 CORBA(Common Object Request Broker Architecture)라 알려진 표준의 개발 및 정의에 대한 작업을 시작하였습니다. 이것은 객체의 위치, 결합 및 사용에 대한 이슈들과 분산 컴퓨팅 환경에서 객체들이 제공하는 서비스에 대해 기술한 표준안으로, 매우 광범위하고 강력하며 잘 정의되어 있습니다. 누구나 알고 있듯이 인터넷은 분산 컴퓨팅의 중요한 중추가 되었으므로, CORBA 호환 Object Request Brokers(ORBs)에 대한 관심이 높아지고 있습니다.

ORBs 는 지역 또는 네트워크에서 객체들을 찾습니다. ORBs 는 객체들을 사용할 때 각 객체의 물리적 위치가 사용자들에게 별 의미가 없도록 객체들을 결합합니다. ORBs 는 자바 객체를 찾아서 사용하기 위한 가장 본질적이고 응집된 메커니즘입니다.

오늘날에는 웹 사이트를 방문하고 다양한 웹 서버에 보관된 HTML 문서의 내용을 보는 것을 설명할 때 "웹 브라우징"이라는 용어를 사용합니다. 그러나 우리가 보는 문서는 웹 브라우저를 실행하면서 HTTP 를 통하여 PC(또는 기타 인터넷 클라이언트 장치)로 전송된 것이므로, 실제로 이루어지는 것은 "웹 페칭(Web fetching)"이라고 할 수 있습니다. 이는 이러한 HTML 문서에 자바 애플릿이나 멀티미디어 클립 등과 같은 다른 데이터 포맷에 대한 참조가 포함되어 있을 경우에도 마찬가지입니다.

앞으로는 현재 알고 있는 월드 와이드 웹이 "범은하계적 객체 웹(InterGalactic object Web)"으로 통합되어, IIOP(Internet Inter Orb Protocol)를 지원하는 ORBs 를 사용하여 진정한 웹 브라우징을 실행하게 될 것입니다.

웹 사이트는 보다 동적인 다차원적 객체 관련 데이터베이스가 될 것이고, 인류가 사용할 수 있는 모든 정보를 보관할 수 있게 될 것입니다. 더 이상 HTML 문서와 자바 클래스 전체를 클라이언트 인터넷 장치로 이전할 필요가 없어짐으로써, 이러한 객체와 관련된 대부분의 작업을 강력한 서버에서 처리할 수 있게 될 것입니다.

장점

장점은 다음과 같습니다.

1. 자바 클래스 이전과 관련된 네트워크 부하가 줄어들기 때문에 자바 기반 솔루션의 성능과 속도가 향상됩니다.
2. "한 번 작성하고, 모든 곳에서 실행하기"라는 가치 제안이 강화되며, 응용프로그램이 자바로 작성되지 않은 경우에도 모든 중요한 운영 환경에서 구현이 가능합니다. CORBA는 자바보다 훨씬 광범위하며 언어 독립적입니다. 그러므로 CORBA는 다양한 객체 지향적인 프로그래밍 언어들로 만든 객체들을 사용하여 객체 웹을 구성할 수 있도록 해줍니다.
3. IIOP는 HTTP와는 달리 규정 지향적인 프로토콜이므로, 탄력성이나 기능을 훼손하지 않고도 이미 배포된 자바 솔루션을 구현할 수 있습니다.
4. 일부 웹 브라우저 공급자는 브라우저와 함께 ORBs를 제공합니다.

단점

단점은 다음과 같습니다.

1. 비록 IIOP가 ORB 간의 통신에 있어서는 표준으로 정의되었지만, 다른 분야에서는 아직 호환성의 문제가 있습니다.
2. ORBs는 관리를 상당히 필요로 하지만, 이것은 자바 오블릿(orblets: ORB의 클라이언트나 프록시 요소)이 일반화되면 바뀔 것입니다.

자바의 쟁점

Java Development Kit(JDK) 버전 비호환성과 크로스 플랫폼 이용 가능성

자바는 JDK 를 통해 사용할 수 있도록 되어 있습니다. JDK 는 이제 JDBC 를 포함하는 1.1.6 버전(상업용 출시 버전)으로 업그레이드 되었습니 다. 잘 알려져 있듯이, 자바 호환 응용프로그램과 환경은 보조를 맞추어 오지

못했습니다.

"한 번 작성하고, 모든 곳에서 실행하기"라는 가치 제안은 JDK 버전의 특징으로, 현재 가장 널리 사용되고 있는 JDK 버전인 JDK 1.02 에 특 히 적용됩니다. 그러나 자바의 보다 흥미롭고 풍부한 내용은 1.1 이상의 JVM 버전에 들어 있습니다.

자바 클래스 이전

HTTP 는 주로 문서 이전을 처리하기 위해 만들어진 것으로, 일반 응용프로그램을 이용하거나 만들 때 매우 중요한 요소인 응용프로그램 상 태 관리를 처리하지는 않습니다. 그러므로 HTTP 는 기업용 사무 시스템을 구축하기 위한 기초가 아닙니다. 물론, 애플릿은 한 번 작성하면 모든 곳에서 실행되므로 "관리 불필요" 제안에 대해 강한 주장을 할 수 있지만, 일정한 "상태"에 이르면 더 크고 중요한 문제가 대두됩니다.

푸쉬 기술은 일반적으로 각각의 클라이언트와 서버를 필요로 하며 보통 자바로 만들어지지 않으므로, 모든 주요 운영 환경으로 이식할 수 없 다면 자바의 가치 제안이 훼손될 수 있습니다. 더구나 이러한 제품에는 관리가 필요하므로 "관리 불필요"라는 자바의 또 다른 중요한 가치 제 안도 잠재적으로는 훼손됩니다.

TV 의 리모콘처럼 ORBs 도 다른 제조업체의 객체와는 작동하지 않을 수 있습니다. 같은 판매업체로부터 TV 와 리모콘을 모두 사야만 한다고 상상해 보십시오. 이는 판매업자에게는 이익이지만 소비자에게는 손해가 될 것입니다. 다행히 오늘날 가전제품 시장에서 일반적으로 작동 되는 리모콘을 구할 수 있는 것처럼, 역시 일반적으로 작동되는 ORBs 에 대한 개발이 진행되고 있습니다(이것이 기본적으로 IIOP 에서 구현 하는 것입니다).

자바 데이터베이스 연결(JDBC)

자바 데이터베이스 연결(JDBC: Java Database Connectivity)은 데이터베이스 독립적인 API로서, 데이터베이스 독립적인 자바 응용프로그램/애플릿/빈의 개발을 쉽게 해줍니다. 이것은 X/Open SAG CLI 사양에 기반한 자바 추출 층입니다.

JDBC 가치 제안

JDBC 가치 제안은 단순합니다. "데이터베이스 독립적인 자바 응용프로그램/애플릿/빈을 한 번 작성하고, 모든

자바가 우세한 분산 컴퓨팅 환경에서 JDBC 의 잠재력은 실로 무한하다고 할 수 있습니다.

아키텍처

JDBC 드라이버는 JDBC 드라이버 관리자를 통해 JDBC 호환 자바 응용프로그램에 노출됩니다.

JDBC 드라이버 관리자(manager)는 자바 클래스의 구현입니다. 이것은 JDBC 서비스 사용자(응용프로그램/애플릿/빈 개발자)와 서비스 제공자(JDBC 드라이버 개발자)가 모두 사용하는 인터페이스입니다. JDBC 응용프로그램 개발자는 JDBC 드라이버 조합(또는 결합)을 위해 JDBC 드라이버 관리자를 사용합니다. 그리고 JDBC 드라이버 개발자는 JDBC 드라이버 관리자 클래스 구현에 의해 기술된 대로, 명세에 따라 JDBC 클래스를 만듭니다.

드라이버 형식

JDBC 드라이버는 네 개의 주요 범주로 나뉩니다. JDBC 드라이버 유형 1, JDBC 드라이버 유형 2, JDBC 드라이버 유형 3(A와 B 형식), 그리고 JDBC 드라이버 유형 4가 그것입니다

JDBC 드라이버 유형 1

JDBC-ODBC 연결은 대부분의 ODBC 드라이버를 통해 JDBC에 액세스할 수 있게 해줍니다. 일부 ODBC 이진 코드 및 다수의 데이터베이스 클라이언트 코드는 이 드라이버를 사용하는 각각의 클라이언트 기계에 로드되어야 합니다. 이러한 유형의 드라이버는 기업 네트워크에 적합하거나, 3-티어 아키텍처를 사용하여 자바로 만든 응용프로그램 서버 코드용으로 사용할 수 있습니다.

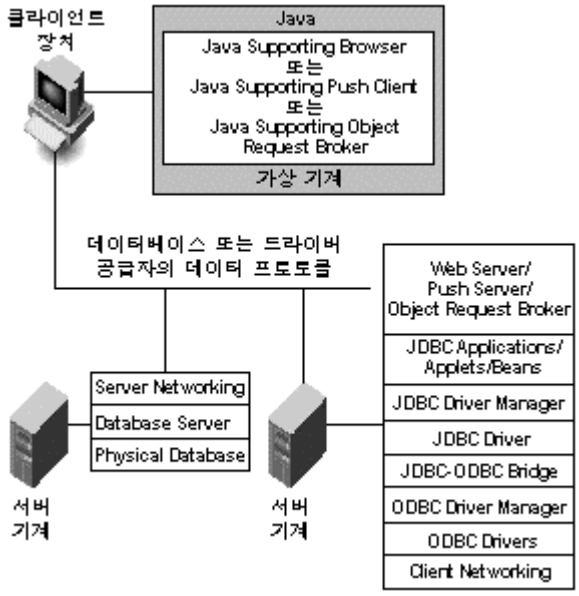


그림 1: JDBC 드라이버 유형 1

장점

장점은 다음과 같습니다.

- 만약 데이터 액세스 표준으로서 기업 차원에서 ODBC를 표준으로 한다면, 이 JDBC 드라이버 형식은 기존의 하부구조에 간단히 적용될 것입니다.

단점

단점은 다음과 같습니다.

- 이것은 "한 번 작성하고, 모든 곳에서 실행하기"라는 가치 제안에 합당하지 않습니다. ODBC는 마이크로소프트 운영 체제 고유의 데이터베이스 연결 표준입니다. JDBC-ODBC 연결은 운영 체제에서 지정하는 네이티브 메소드(native method)를 사용하여 구현됩니다.
- 이러한 접근 방식은, 자바 애플릿으로 솔루션 구현을 시도할 경우, 반드시 윈도우 NT 또는 윈도우 95 기계로 구성된 중간 티어(middle tier)가 있는 3-티어 아키텍처를 강요합니다.
- 또한 JDBC 드라이버 이외에도, ODBC 드라이버와 ODBC-JDBC 연결 소프트웨어(경우에 따라 DBMS 제공자가 추가로 값비싼 네트워크 소프트웨어를 제공합니다)를 각 클라이언트 기계에 설치하고 설정해야 하기 때문에 "관리 불필요"라는 가치 제안도 만족시키지 못합니다.
- JDBC 드라이버 성능과 기능은 그 기반이 되는 ODBC 드라이버의 성능에 전적으로 의존합니다.
- 이것은 결국 유형(네트워크 제품에 대한 실제 및 숨겨진 비용)과 무형(관리, 교육, 구현)의 측면에서 모두 JDBC로 가기 위한 매우 값비싼 접근법입니다.

JDBC 드라이버 유형 2

네이티브 메소드를 통해 JDBC 드라이버 클래스를 구현하는 자체의 API 가 무엇이건 간에(Informix ESQL 이나 Informix CLI 또는 기타 업체의 자바 기반 드라이버 등), 이 메소드들은 일반적으로 해당 데이터베이스 엔진(예를 들어, Informix OnLine Dynamic Server)의 C 기반 Call Level Interface 로 직접 연결됩니다. 따라서 이 메소드들은 ODBC 드라이버에 의해 제공되는 중간 층을 필요로 하지 않습니다.

이 드라이버 형식은 관련 데이터베이스 공급자의 CLI 라이브러리와 자체 연결 라이브러리가 각 클라이언트에

설치될 것과 JDBC 호환 응용프로그램/애플릿/빈을 사용할 것을 요구합니다. 이는 이러한 라이브러리들(예를 들어, 윈도우 기반 운영 환경의 DLL 파일)을 HTTP 로 전송할 수 없기 때문입니다. 각 기계에서 필요로 하는 미들웨어의 예로는 Informix Net 와 Informix Connect 가 있습니다.

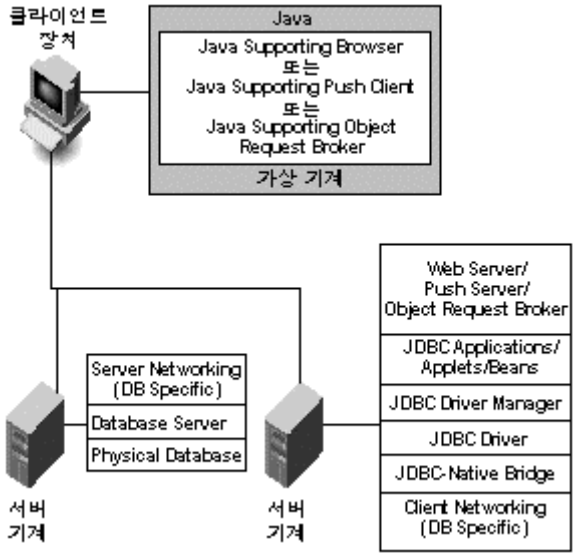


그림 2: JDBC 드라이버 유형 2

장점

장점은 다음과 같습니다.

- 만약 기업 차원에서 단일 데이터베이스 엔진을 사용하며 "ODBC는 느리다"라는 견해에 동의한다면, 이 JDBC 드라이버 형식은 기존의 하부구조(자체 드라이버 기반의)에 간단히 적용될 것입니다.

단점

단점은 다음과 같습니다.

- 데이터베이스 공급자의 CLIs가 독립된 데이터베이스를 제공해 주지 못하기 때문에 "한 번 작성하고, 모든 곳에서 실행하기"라는 가치 제안을 만족시키지 못합니다. JDBC 자체 연결은 네이티브 메소드를 사용하여 구현됩니다.
- 이러한 접근 방식은, 자바 애플릿으로 솔루션 구현을 시도할 경우, 반드시 자바 전용 네트워크 소프트웨어(보이지 않는 고비용 요소)를 실행하는 기계로 구성된 중간 티어(middle tier)가 있는 3-티어 아키텍처를 강요합니다.

- JDBC 드라이버 성능과 기능은 그 기반이 되는 자체 API와 데이터베이스 전용 네트워크 소프트웨어의 성능에 전적으로 의존합니다.
- 이것은 결국 유형(네트워크 제품에 대한 실제 및 숨겨진 비용)과 무형(관리, 교육, 구현)의 측면에서 모두 JDBC로 가기 위한 매우 값비싼 접근법입니다.

JDBC 드라이버 유형 3 (A 형식)

네트-프로토콜 100% 자바(net-protocol all-Java) 드라이버는 JDBC 호출을 DBMS 독립적인 네트워크 프로토콜로 해석하고, 이것은 다시 서버에서 DBMS 프로토콜로 해석됩니다. 이 네트 서버 미들웨어는 모든 자바 클라이언트를 다른 많은 데이터베이스에 연결시킬 수 있습니다. 어떤 프로토콜을 사용할 것인가는 공급자에게 달려 있습니다. 일반적으로 이것은 가장 유연하고 개방된 JDBC 구현의 대안입니다. 이 솔루션의 모든 공급자는 인트라넷 사용에 적합한 제품을 제공할 것입니다. 이러한 제품이 인터넷 접속도 지원하려면, 웹과 관련된 안전성, 방화벽을 통한 액세스 등 추가적인 요구 사항을 처리해야 합니다. 몇몇 공급자들은 기존의 데이터베이스 미들웨어 제품에 JDBC 드라이버를 추가하고 있습니다.

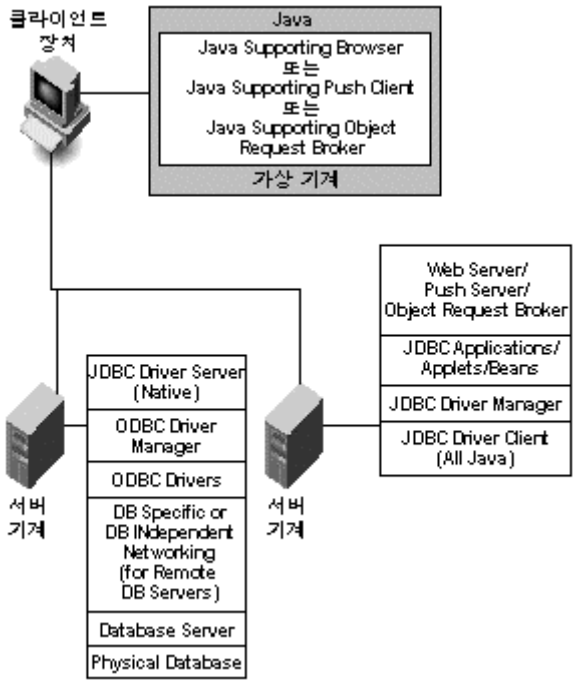


그림 3: JDBC 드라이버 유형 3 (A 형식)

JDBC 드라이버 유형 3 (B 형식)

이 JDBC 형식은 ODBC의 경우와 상반되게 "전선을 통한(across the wire)" 데이터 프로토콜로서 JDBC를 사

용하는 점에서 이전의 아키텍처와 다릅니다.

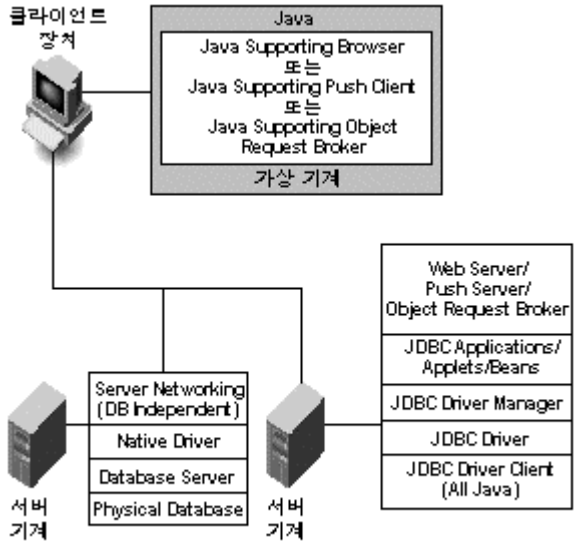


그림 4: JDBC 드라이버 유형 3 (B 형식)

장점

장점은 다음과 같습니다.

- "한 번 작성하고, 모든 곳에서 실행하기"라는 자바의 가치 제안을 강화합니다. 이것은 JDBC 드라이버가 네이티브 메소드를 포함하고 있지 않으며, ODBC 또는 데이터베이스 공급자가 제공하는 특정 데이터베이스 네트워크 소프트웨어와 같은 대안의 데이터 액세스 표준을 따르지 않아도 되는 일반적인 형식입니다.
- 3-티어 아키텍처가 아니어도 JDBC 호환 애플릿의 구현을 수월하게 하여, "관리 불필요"라는 자바 가치 제안을 강화합니다.
- 성능과 기능이 JDBC 드라이버의 독자적인 특성이 되므로, ODBC 또는 특정 데이터베이스 네트워크 소프트웨어의 능력에 의존하지 않습니다.
- 데이터베이스 독립적인 자바 응용프로그램/애플릿/빈의 개발을 가능하게 함으로써 JDBC 가치 제안을 강화합니다.
- 이것은 인터넷/인트라넷/엑스트라넷 내에서 JDBC를 구현하기 위한 매우 경제적인 방법입니다.

단점

단점은 다음과 같습니다.

- 성능과 기능(보안, 설정 관리 등)이 JDBC 드라이버의 데이터베이스 독립적인 네트워크 관련 기반 능력에 의존합니다. 만약 이것이 느리거나 기능이 떨어지면, 기반 데이터베이스와 상관없이, 실제 JDBC를 사용할 때 역시 느리고 기능이 떨어지는 경험을 하게 될 것입니다.
- ODBC 기반의 "전선을 통한" 프로토콜 또는 전적으로 ODBC에 기반한 JDBC 서버 구성요소는 결국 JDBC 드라이버의 영역을 그 기반이 되는 ODBC 드라이버의 영역으로 제한합니다. 궁극적으로 이것은 "한 번 작성하고, 모든 곳에서 실행하기"라는 자바의 가치 제안에 합당하지 않습니다.

JDBC 드라이버 유형 4

자체 프로토콜의 100% 자바(native protocol all-Java) 드라이버는 JDBC 호출을 DBMS에서 직접 사용되는 네트워크 프로토콜로 변환합니다. 이것은 클라이언트 기계에서 DBMS 서버로의 직접 호출을 가능하게 하며, 인트라넷 액세스를 위한 실질적인 해결책이 됩니다. 이러한 프로토콜의 다수가 공급자의 독점적인 소유물이므로, 데이터베이스 공급자 자체가 이러한 종류의 드라이버의 주요 소스가 될 것입니다. 현재 몇몇 데이터베이스 공급자가 이러한 프로토콜을 개발하고 있습니다.

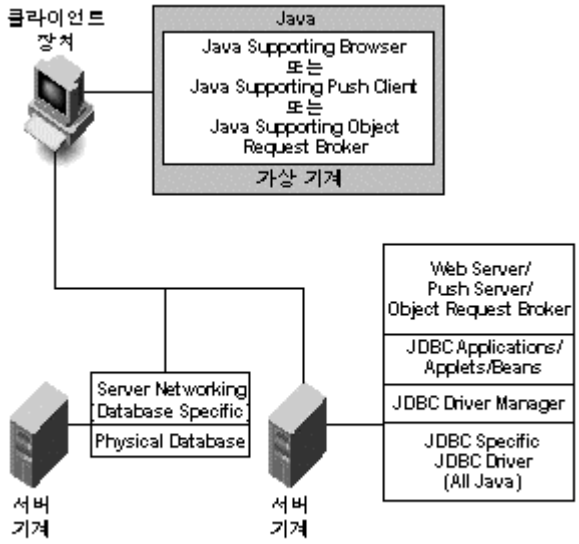


그림 5: JDBC 드라이버 유형 4

장점

장점은 다음과 같습니다.

- 만약 기업 차원에서 단일 데이터베이스 엔진을 사용하며, 데이터베이스 공급자가 모든 데이터베이스 관련 인터페이스를 제공한다는 견해에 동의한다면, 이 JDBC 드라이버 형식은 기존의 하부구조(자체 드라이버 기반의)에 간단히 적용될 것입니다.
- 이것은 "관리가 필요 없는" JDBC 호환 애플릿의 구현을 지원합니다.

단점

단점은 다음과 같습니다.

- 이것은 JDBC 호환 응용프로그램/애플릿/빈의 액션을 서로 다른 데이터베이스 엔진들에서, 특히 데이터베이스 핵심 기능과 반대되는 JDBC API 구현 영역에서(DBMS 엔진이 스스로를 차별화할 수 있는 올바른 위치) 거의 예견할 수 없다는 점에서 JDBC의 데이터베이스 독립성을 훼손합니다.
- JDBC 드라이버 성능과 기능은 해당 DBMS 공급자에 의해 자바에서 구현되는 특정한 기반 데이터베이스 프로토콜의 성능에 전적으로 의존합니다.
- 이것은 결국 유형(제품에 대한 실제 및 숨겨진 비용)과 무형(관리, 교육, 구현)의 측면에서 모두 JDBC로 가기 위한 매우 값비싼 접근법입니다. 프론트 엔드 JDBC 드라이버가 처음에는 매우 경제적인 것처럼 보일지 모르지만, 결국 백 엔드 DBMS 라이센스 비용을 감당하기가 힘들어 질 것입니다. ODBC와 마찬가지로 JDBC도 데이터베이스 엔진 상품화가 곧 이루어질 것이므로, 시장에서 이러한 상황을 모르고 있는 한 대부분의 DBMS 공급자는 이에 저항할 것입니다.
- 이것은 곧 쓸모 없게 될 "원 스톱 샵(one-stop shop)" 개념을 기술 공급자와 소비자 사이에서 영속시키려고 합니다.

JDBC 활용 기술

JDBC 어법에 따르면, 데이터 소스는 리소스를 말합니다. JDBC 는 인터넷 데이터 액세스 메커니즘이기 때문에 이것이 다른 인터넷 프로토콜 및 기술에서 사용되는 기존의 리소스 바인딩 메커니즘을 모방하는 것은 어쩔 수 없는 일입니다. JDBC 드라이버는 JDBC URL 을 통해 데이터 소스를 검색 및 연결하며, 다음의 형식을 취합니다.

<Protocol>:<Sub-protocol>:[<sub-name>]:

[//<host name or address>[:port number]]

[/Driver Attribute 1 ./Driver Attribute n]

위의 URL 구조는 드라이버에 의존적이므로, 사각형 괄호 안의 내용을 분석하고 해석하는 것은 드라이버가 할 일입니다. 그러므로 다른 JDBC 드라이버 공급자가 제공하는 JDBC 드라이버는 또 다른 URL 형식을 요구하게 됩니다.

JDBC URL 예

1. Sun 사의 JDBC-ODBC 연결

아래의 URL 은 이 JDBC 드라이버가 로드될 컴퓨터(일반적인 PC)에 이미 존재하는 "Accounts"라는 ODBC Data Source Name(DSN)으로 연결됩니다.

jdbc:odbc:Accounts

2. OpenLink Software 사의 JDBC 드라이버

아래의 URL 은 이 JDBC 드라이버가 로드될 서버 컴퓨터(호스트 이름은 Jupiter)에 이미 존재하는 "Accounts"라는 논리적 Data Source Name (DSN)에 연결합니다.

jdbc:openlink://jupiter:5000/DSN=Accounts

오늘날 JDBC의 활용

현재 JDBC는 자바 응용프로그램과 애플릿 개발자들이인터넷과 인트라넷 기반의 데이터베이스 독립적인 응용프로그램/애플릿을 만들기 위해 사용합니다. 애플릿 클래스는 HTTP를 통해 JVM을 호출하는 반면, 응용프로그램 클래스는 푸쉬 기술 제품을 통하여 JVM을 호출하는 방향으로 나아갑니다.

JDBC 의 활용

자바/CORBA 는 불가항력적으로 거대한 힘을 발휘하고 있으며, OMG 의 IIOP 프로토콜을 사용하여 원격 자바 클래스를 Object Request Broker(ORB) 방식으로 결합하는 시대가 곧 도래할 것입니다. ORB-IIOP 기반의 자바 클래스 결합 방식이 도입되면 푸쉬 기술보다 데이터 전송량이 훨씬 적어지고, 자바와 JDBC 각각의 가치 제안을 충족시키면서도 애플릿이 HTTP 에 의존하지 않게 될 것입니다.

Javasoft 는 또한 Remote Method Invocation(RMI)이라는 자바 전용 기반의 원격 결합 및 메서드 실행 메커니즘을 추가로 제공하였습니다. RMI 는 개념상으로는 ORB-IIOP 접근 방식과 매우 유사하지만, RMI 가 자바 전용 솔루션인 데 비해, ORB-IIOP 표준은 광범위한 객체 구현 방식을 통합합니다. 그러나 Javasoft 가 CORBA 구현을 선호하기 때문에 RMI 는 중단될 것입니다.

JDBC 쟁점(가치 제안 위반 요인)

JDBC 가치 제안은 현재 큰 취약점을 가지고 있고 이미 훼손되었으며, 그 타당성과 사용 가능성은 앞으로도 늘 쟁점이 될 것입니다. 이러한 취약성이나 가치 제안 위반 요인은 다양한 양태를 취하고 있으며, 다음에 소개하는 내용들로 요약할 수 있습니다.

애플릿의 HTTP 의존성

JDBC 애플릿은 현재 매우 유행하고 있습니다. 이는 한 번 개발하여 모든 곳에서 실행할 수 있으며, 데이터베이스 독립적이어서 "관리가 필요 없는" 자바 솔루션을 모든 사람이 원하고 있기 때문입니다. 그러나 이 경우에 "모든 곳"이라는 말은 실제로 무엇을 의미할까요? 현재 이것은 JDBC 호환 애플릿을 담고 있는 페이지를 보기 위해 사용되는 웹 브라우저의 일부인 JVM 과 함께 HTTPD(웹 서버) 프로세스가 실행되는 "모든 곳"을 의미합니다.

JDBC 는 JDK 1.02 의 일부가 아니라, 현재 1.02 JVM 과 함께 제공되는 자바 지원 웹 브라우저(특히 Netscape 와 Microsoft Explorer)의 거의 대부분입니다. 일반적으로 브라우저 공급자가 자바를 지원한다는 것은 JDK 1.02 의 핵심적인 클래스가 인터넷 장치(일반적으로 PC)에 미리 설치되어 있다는 것을 의미합니다. 또한 이것은, 전달 메커니즘(HTTP 또는 IIOP)에 상관 없이, 브라우저가 원격 서버로부터 핵심적인 자바 클래스를 로드하지 않을 것임을 의미합니다. 그러므로 지역 JVM 구현의 일부가 아닌 핵심적인 자바 클래스를 사용하여 개발된 애플릿을 로드하려는 시도는 거부되며, 이러한 거부는 "애플릿 보안 모델(applet security

model)" 위반으로 이해됩니다.

JDBC 는 JDK 1.1 의 일부이지만, 브라우저 공급자는 현재 각 브라우저에 대해 JDK 1.1 기반의 JVM 구현을 위해 작업하고 있습니다.
VM 1.1 브라우저가 개발될 때까지는 JDBC 호환 자바 애플릿을 개발하거나 사용하는 것이 사실상 불가능합니다.

푸쉬 기술

푸쉬 기술은 앞에서 언급하였듯이, 브라우저와 HTTP 의 의존성을 피할 수 있는 방법을 제공합니다. 그러나 푸쉬 기술 방식은 스스로 가치 제안 위반 요인을 내포하고 있습니다. 예를 들어, 이러한 제품은 서버 측에서 자바로 작성되지 않았으므로, 한 번 작성하여 "모든 곳에서" 실행한다는 제안은 실제로 "일부에서"의 의미에 더 가깝게 되었습니다.

이러한 제품들은 객체 참조가 아닌, 전체 객체 이전이라는 개념을 따르고 있습니다. 이는 인터넷 장치 상의 실재 자산인 물리적 디스크와 같은 문제를 인정하거나 이해하지 않기 때문입니다(이러한 제품은 PC 에 디스크 공간과 메모리가 많다고 추정합니다). 확실히 이러한 제품은 JDBC 호환 자바 응용프로그램의 초기화 시간을 단축시키지만, 실제로 가치 제안을 지키지는 못합니다.

Object Request Brokers (ORBS)

우리 사회의 중개인처럼, ORBs 도 사용자를 서비스 제공자와 연결시켜 줍니다. 그러나 실생활의 공급자와 마찬가지로(리모콘), ORBs 도 서비스 제공자(TV 세트) 또는 실제 서비스(시청하는 프로그램) 전체를 사용자에게 가져다 주지는 않습니다. 대신, 사용자는 제공된 서비스를 원격에서 제어하고 소비합니다.

자바와 ORBs 는 놀라운 속도로 결합되고 있지만, ORB 의 상호 운영(cross-ORB interoperability)에 대한 문제는 여전히 남아 있습니다. IIOP 표준은 이러한 문제를 다루기 위해 올바른 방향으로 한 걸음 나아간 것입니다. 그러나 이 표준도 여전히 자바 객체와 이를 중개하는 ORBs 객체가 독립적임을 보장하지 못합니다. 그러므로 오늘날의 자바 응용프로그램 개발자는 실제로 도처에 있는 모든 가능성에 대비하기 위하여 수많은 ORB 인터페이스를 만들어야 한다는 결론이 나오게 됩니다. 이것은 확실히 자바와 JDBC 가치 제안을 만족시키지 못합니다.

자바로 작성된 ORBs 는 거의 없습니다. JVM 이 상주하는 호스트 운영 환경에서 자바 객체가 ORB 구현을

못할지도 모르므로, 이것은 또 다른 가치 제안 위반 요인을 잠재적으로 내포합니다.

성능

만약 JDBC 드라이버가 여러 플랫폼과 데이터베이스 엔진에서 일관되게 합리적인 성능을 제공하지 못한다면, JDBC 가치 제안은 깨질 것입니다.

ODBC 연결

최근에 마이크로소프트의 Open Database Connectivity(ODBC) 표준과 JDBC 는 잘못된 이유로 인하여 어쩔 수 없이 연결되었습니다. 물론, JDBC 드라이버가 오늘날 널리 만연되어 있는 ODBC 드라이버를 활용할 수 있다는 사실은 중요하지만, JDBC 를 다룰 때 ODBC 가 반드시 개입되어야 하는 것은 아닙니다. 불행하게도 현재 JDBC 기술을 채택하려는 대부분의 사람들은 이러한 잘못된 개념이 미치게 될 영향을 알거나 이해하지 못하고 있습니다.

ODBC 는 특정 운영 환경, 즉 윈도우를 염두에 두고 개발된 것이며 그게 전부입니다. 비록 다른 플랫폼에서도 구현이 가능하지만, 마이크로소프트 운영 체제 이외에서 사용하는 경우는 매우 드뭅니다. ODBC 와 JDBC 의 어쩔 수 없는 연결은 확실히 JDBC 가치 제안을 만족시키지 못합니다.

레코드 스크롤과 데이터 민감성

JDBC 는, 운영 체제의 대응물인 ODBC 와는 달리, 현재 레코드 스크롤 능력이 부족합니다. JDBC Record Set Object 는 "Next" 메서드만을 가지고 있습니다. 이것은 JDBC 정의 자체의 큰 결함으로서, 데이터에 민감한 기업용 솔루션의 개발을 위하여 실제로 JDBC 를 사용할 수 없게 만듭니다. 현재 대부분의 JDBC 개발자와 드라이버 공급자들은 기초적인 JDBC 레벨 위에서 추출을 통해(by abstracting) 스크롤 능력을 구현합니다. 한편 커서 이름 등을 찾기 위해 명세에서 제공되는 삽입된 SQL 훅(embedded SQL hooks)을 사용하려는 시도도 이루어지고 있습니다. 불행하게도 이러한 접근법은 시대에 뒤떨어졌으며, 실질적인 해결책과 너무 동떨어져 있습니다. 만약 데이터에 민감한 뛰어난 JDBC 호환 응용프로그램/애플릿/빈을 만들 수 없다면, JDBC 의 사용은 훨씬 줄어들게 됩니다.

데이터 액세스 요소가 본질적으로 정적이라면, JDBC 개발자 및 사용자가 핵심적인 JDBC 가치 제안에서만 전달되는 JDBC 솔루션을 기꺼이 만들거나 사용하게 되리라고 기대할 수 없을 것입니다. 동적 데이터에 대한 액세스와 조작을 기반으로 한 정보 혁명 및 전세계 시장 연결의 힘을 얻어 세계는 매일매일 더 작고 보다

동적인 장소가 되어가고 있습니다.

프록시와 안전성

인터넷, 인트라넷 및 엑스트라넷은 오늘날의 TCP/IP 가 지배하는 분산 컴퓨팅 환경을 정의하는 세 가지 형식입니다. 인터넷은 전반적인 하부 구조이고, 인트라넷은 회사의 방화벽 뒤에 존재하는 인터넷의 부분이며, 엑스트라넷은 보다 넓은 인터넷으로 드러나게 되는 인트라넷의 일부입니다.

이러한 분산 컴퓨팅 형식 내에서 JDBC 드라이버를 사용하려면, 보안을 희생하지 않고서 회사 방화벽 뒤의 데이터베이스로 액세스하는 능력을 JDBC 드라이버가 제공해야 합니다. 프록시 기능을 처리하는 JDBC 드라이버는 이러한 요구를 충족시킵니다.

만일 인터넷, 인트라넷 및 엑스트라넷 내에서 JDBC 드라이버를 사용할 수 없다면, JDBC 가치 제안이 깨질 것입니다.

사용자와 조직의 다양성

어느 기술에서든지 가장 큰 약점 가운데 하나는 소비자들의 다양성을 예견하지 못하는 것입니다. 예를 들어, 한 회사 내에서 부서에 상관없이 컴퓨터 사용 기술 수준이 다양하다는 점은 비밀이 아닙니다. 또한, 비록 모두가 모든 컴퓨터 기술 전문가라고 하더라도, 회사 조직은 전문적이긴 하나 혼란한 응용프로그램에서가 아니라 회사의 업무 체계에 대한 응집된 개발, 습득 및 적용을 통해 각각의 전문성을 체계화한 응용프로그램에 안주할 것이라는 점도 사실입니다.

JDBC 는 조직 내에서 사용자의 다양성을 인정하는 응용프로그램의 개발을 가능하게 해야 합니다. 그렇지 않다면 JDBC 응용프로그램/애플릿/빈은 JDBC 기술 사용자에 대한 고정관념 때문에 융통성을 갖지 못하게 될 것입니다. JDBC 는 물리적으로가 아니라 논리적으로 스스로를 사용자 그룹과 연결시켜야 합니다. 그렇지 못하면 가치 제안이 깨지고 말 것입니다.

OpenLink 일반 고성능 JDBC 드라이버

OpenLink 일반 고성능 JDBC 드라이버는 인터넷 상에 또는 인트라넷이나 엑스트라넷 내에 있는 JDBC 호환 응용프로그램/애플릿/빈으로부터 데이터 소스로 쉽게 액세스할 수 있게 해줍니다. 이러한 드라이버는 "OpenLink data access drivers suite"로 알려진 상업용 제품의 필수 부분입니다.

OpenLinkJDBC 가치 제안

OpenLink JDBC 가치 제안은 "데이터베이스에 상관없이 한 번 작성하여, 사용 가능성이나, 보안성, 성능 또는 자바 가치 제안의 어떤 부분에 대해서도 훼손 없이 모든 곳에서 실행한다"라는 것입니다.

구현 및 아키텍처

OpenLink 소프트웨어는 유형 1, 유형 2, 유형 3의 JDBC 드라이버를 제공합니다. 유형 1과 2 드라이버들은 OpenLink Data Access Drivers Suite(Lite 버전)의 필수 부분인 반면, 유형 3 드라이버들은 OpenLink Data Access Drivers Suite(워크그룹과 엔터프라이즈 버전)의 필수 부분입니다.

OpenLink JDBC Lite 유형 1 JDBC 드라이버

OpenLink JDBC Lite 유형 1 JDBC 드라이버는 JDBC-ODBC 연결 드라이버로서, ODBC 드라이버 관리자 및 JDBC 방식의 응용프로그램/애플릿/빈의 호스트 기계에 설치된 ODBC 드라이버의 존재, 기능 및 능력에 대해 의존적입니다.

JDBC 드라이버 클래스는 네이티브 메소드(native methods)를 통해 구현되는데, 이는 JDBC 드라이버 내에서 구현되는 메소드가 동적인 혹은 공유된 개별 라이브러리에서 C 함수 또는 C++ 객체로의 매핑을 통해 구현됨을 의미합니다(호스트 운영 체제에 의존적임). 이러한 라이브러리는 기본적으로 ODBC 방식의 서비스 수요자(클라이언트)입니다.

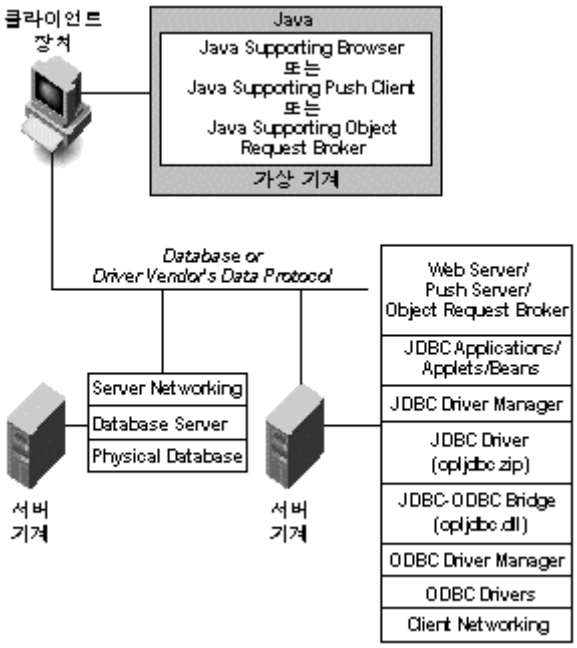


그림 6: OpenLink JDBC Lite(유형 1 JDBC 드라이버)

OpenLink JDBC Lite 유형 2 JDBC 드라이버

OpenLink JDBC Lite 유형 2 JDBC 드라이버는 JDBC 자체의 데이터베이스 인터페이스 연결 드라이버로서, 관련 데이터베이스 공급자의 클라이언트 인터페이스 및 JDBC 방식의 응용프로그램/애플릿/빈의 호스트 기계에 있는 데이터베이스 전용 네트워크 미들웨어의 존재, 기능 및 능력에 대해 의존적입니다.

JDBC 드라이버 클래스는 네이티브 메소드(native methods)를 통해 구현되는데, 이는 JDBC 드라이버 내의 네이티브 메소드가 동적인 혹은 공유된 개별 라이브러리에서 C 함수 또는 C++ 객체로의 매핑을 통해 구현됨을 의미합니다(호스트 운영 체제에 의존적임). 이러한 라이브러리는 관련된 데이터베이스 서버의 자체 호출-레벨(call-level) 인터페이스를 사용하여 구축됩니다.

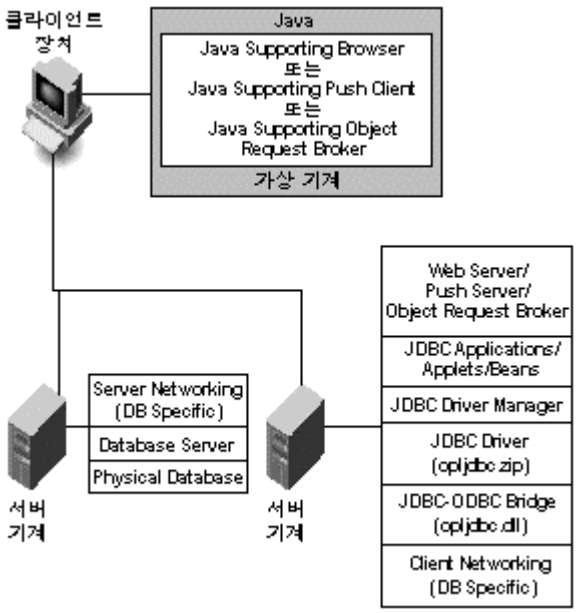


그림 7: OpenLink JDBC Lite(JDBC 자체 연결)

OpenLink JDBC-ODBC 유형 3 JDBC 드라이버(A 형식)

OpenLink JDBC-ODBC 유형 3 JDBC(A 형식) 드라이버는 네이티브 메소드의 필요성을 부정하며 전적으로 자바로 작성된 JDBC 드라이버 메소드로 구현하는 일반적인 드라이버입니다. 이는 또한 데이터베이스 독립적인 네트워크 층을 통합하는 씬 클라이언트(thin client) 층 구현을 가능하게 해주는 멀티-티어(multi-tiered) 구현 방식입니다. 이 드라이버 형식은 낮은 레벨의 데이터 액세스 서비스를 위해 ODBC 드라이버를 사용합니다.

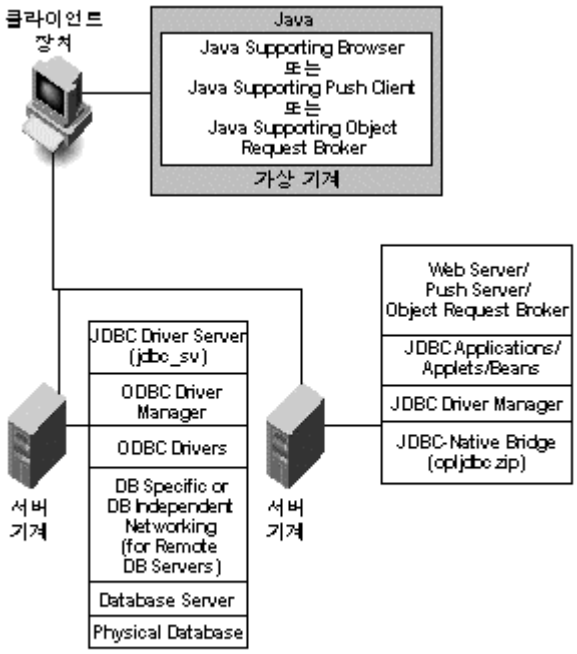


그림 8: OpenLink Generic JDBC 드라이버 (멀티-티어 아키텍처의 ODBC 서버)

OpenLink JDBC Drivers 유형 3(B 형식 드라이버)2

OpenLink JDBC Drivers 유형 3(B 형식) 드라이버는 네이티브 메소드의 필요성을 부정하며 전적으로 자바로 작성된 JDBC 드라이버 메소드로 구현하는 일반적인 드라이버입니다. 이는 또한 씬 클라이언트(thin client) 층 구현을 가능하게 해주는 멀티-티어(multi-tiered) 구현 방식이며, 데이터베이스 독립적인 네트워크 층을 통합합니다. 이 드라이버 형식은 낮은 레벨의 데이터 액세스 서비스를 위해 자체의 데이터베이스 드라이버(OpenLink 데이터베이스 에이전트)를 사용합니다.

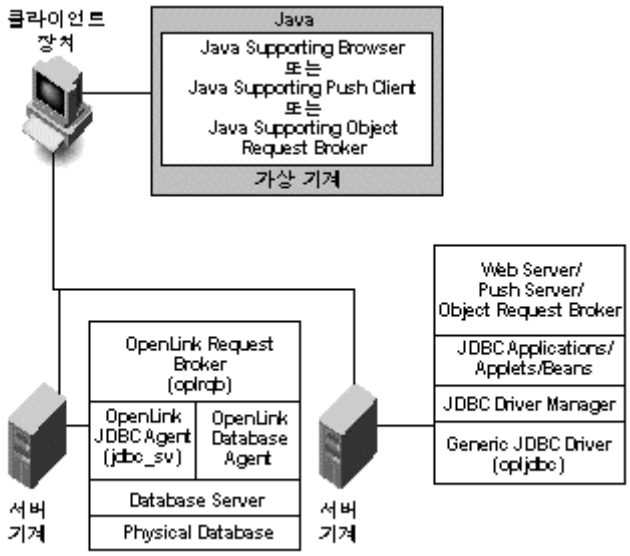


그림 9: OpenLink 일반 JDBC 드라이버 (멀티-티어 아키텍처의 자체 서버)

맺음말

Informix 사용자 관련 사항

자바와 JDBC 는 관리가 불필요하며 운영 체제에 의존하지 않는 차세대 Informix 응용프로그램의 구현을 위한 초석입니다. 구식의 데스크탑과 "팻 클라이언트(fat client)"의 클라이언트-서버 응용프로그램 모델과 연관된 관리상의 오버헤드와 데스크탑의 복잡성은 현재 컴퓨터 모델에 있어서 2-티어에서 N-티어 분산 컴퓨팅 모델로의 신속한 이전에 힘입어 곧 과거의 유물이 될 것입니다.

Informix 개발자 관련 사항

수직적 시장(vertical market)의 응용프로그램 개발은 여전히 Informix 의 중추가 되지만, 자바, JDBC 및 N-티어 분산 컴퓨팅의 출현은 제품 유지, 지원 및 구현상의 오버헤드가 훨씬 감소되는 폭 넓은 시장을 형성할 것입니다. JDBC 를 기반으로 한 자바 응용프로그램 프레임웍의 급속한 출현으로 자바는 빠르게 복합 응용프로그램 서버가 되고 있으며, 제품 개발 기간을 극적으로 단축시켜주는 재사용이 가능한 템플릿과 구성요소가 제공되고 있습니다. 자바와 JDBC 는 Informix 응용프로그램 개발자에게 훨씬 폭 넓은 시장을 제공합니다.

업계 관련 사항

자바는 정당한 사유로 세상의 이목을 사로 잡았습니다. JDBC 와 자바 빈은 기술적으로나 상업적으로 확실히 자바를 지원합니다. CORBA 를 지원하는 ORBs 와 IIOP 프로토콜 시대의 도래는 자바와 함께 IT 의 비약적인 도약을 위한 무대를 마련합니다.

구성요소 기반의 분산 컴퓨팅 시대가 다가옵니다. 그 잠재력과 파급 효과를 제대로 수용하거나 이해하지 못하면 IT 개발자와 사용자 모두 손해를 보게 될 것입니다.

IT 의 미래를 예측할 수 있다면 이윤의 증대, 직원의 감소, 군살 없는 기업, 보다 자유로운 환경에서 나오는 혁신적인 소프트웨어 기획, 구성요소화를 통한 소프트웨어 품질의 향상 등과 같은 기업의 중요한 성공 요소에 긍정적인 역할을 할 것이며, 궁극적으로 IT 투자에 대한 더 많은 보상을 얻게 될 것입니다.

참조

OpenLink Software 사는 고성능의 안전한 Universal Data Access 소프트웨어 개발 및 구현을 추구하는 앞서가는 개발업체입니다. 이 회사는 Java Database Connectivity(JDBC), Open Database Connectivity(ODBC) 및 Universal Database Connectivity(UDBC)와 같은, 업계의 중요한 데이터 액세스 표준에 기술력을 집중시키고 있습니다. OpenLink 의 고성능 Universal Data Access 소프트웨어는 인터넷/인트라넷/엑스트라넷 솔루션의 개발자와 사용자에게 여러 운영 환경을 지원하는 웹 사이트, 자바 응용프로그램/애플릿/ServerLets/Bean 구성요소와 ActiveX 기반 구성요소에서 사용할 데이터베이스의 개발과 구현을 위한 일관된 데이터베이스 인터페이스를 제공합니다.

OpenLink Software사에 대한 추가 정보는 http://www.openlinksw.com 웹 사이트를 참고하십시오.

또한, OpenLink Software사는 nntp://news.openlinksw.com의 뉴스 서버에서 ODBC, JDBC, UDBC 및 OLE-DB를 다루는 토론 포럼을 지원하는 후원업체입니다.

ONSTAT 유틸리티 살펴보기

소개

위한 이 유틸리티의 여러 가지 사용 방법을 소개할 것입니다.

ONSTAT

ONSTAT는 가장 널리 사용되는 명령줄 유틸리티입니다. ONSTAT는 OnLine의 공유 메모리 구조를 읽고 서버 상태에 관한 유용한 정보들을 제공합니다. 이 유틸리티는 공유 메모리 구조에 로크를 두지 않고 오버헤드를 거의 사용하지 않기 때문에 언제든지 사용 가능합니다. 명령 사용 결과로 나타나는 정보는 명령이 보내진 시점의 것이라는 점과 명령을 보낸 후 데이터가 변경될 수 있다는 점을 염두에 두어야 합니다.

ONSTAT는 다른 어떤 Informix 유틸리티 보다도 더 많은 옵션을 제공합니다. 그 옵션들 중 다수는 제대로 문서화되어 있지 않으며 일반 데이터베이스 관리자(DBA)들이 쉽게 이해할 수 없는 디버깅 매개변수들입니다. ONSTAT는 또한 서버 관리에 필요한 여러 유용한 옵션들도 제공합니다. 이 글에서는 이와 같은 옵션들의 사용법에 초점을 맞출 것입니다. 구문과 옵션의 전체 목록은 표 1 에 있습니다. Informix Dynamic Server에서, ONSTAT 명령은 새로운 감시 및 디버깅 옵션들로 크게 향상되었습니다. 이 옵션들은 -g로 시작되며 표 2 에 정리되어 있습니다.

onstat [abcdfghklmpstuxzBCDFRX] [-I] [-r seconds] [-o file] [infile]	
-t	테이블 영역 출력
-u	사용자 스레드 정보 출력
-x	트랜잭션 정보 출력
-z	공유 메모리 통계치를 0으로 돌리기
-B	모든 버퍼 정보 출력
-C	b 트리 제거 요구 출력
-D	DB 영역과 자세한 청크 상태 출력
-F	페이지 플러셔 정보 출력
-R	LRU 큐 정보 출력
-X	버퍼에 대한 공유자 및 대기자 전체 목록 출력
-r	매 n 초마다 onstat 반복 수행 옵션 (기본값: 5)

-s	래치 정보 출력
-t	테이블 영역 출력
-u	사용자 스레드 정보 출력
-x	트랜잭션 정보 출력
-z	공유 메모리 통계치를 0으로 돌리기
-B	모든 버퍼 정보 출력
-C	b 트리 제거 요구 출력
-D	DB 영역과 자세한 청크 상태 출력
-F	페이지 플러셔 정보 출력
-R	LRU 큐 정보 출력
-X	버퍼에 대한 공유자 및 대기자 전체 목록 출력
-r	매 n 초마다 onstat 반복 수행 옵션 (기본값: 5)
-o	특정 파일로 공유 메모리 내용 출력 (기본값: onstat.out)
infile	공유 메모리 정보를 얻기 위해 infile 사용
-	OnLine 모드 표시

표 1: ONSTAT 구문과 옵션

onstat -g [아래 목록의 옵션]	
all	모든 MT 정보 출력
ath	모든 스레드 정보 출력
wai	대기 스레드 정보 출력
act	활동 스레드 정보 출력
rea	준비 스레드 정보 출력

sle	모든 휴면 스레드 정보 출력
spi	긴 스핀이 있는 스핀 로크 정보 출력
sch	VP 스케줄러 통계 정보 출력
lmx	모든 로크된 뮤텍스 정보 출력
wmx	대기자가 있는 모든 뮤텍스 정보 출력
con	대기자가 있는 조건 정보 출력
stk	<tid> 특정한 스레드의 스택 덤프
glo	MT 전역 정보 출력
mem	<pool name|session id> 풀(pool) 통계 출력
seg	메모리 세그먼트 통계 정보 출력
rbm	상주 세그먼트에 대한 블록 맵 정보 출력
nbm	비상주 세그먼트에 대한 블록 맵 정보 출력
afr	<pool name|session id> 할당된 풀 분할 정보 출력
ffr	<pool name|session id> 유휴(free) 풀 분할 정보 출력
ufr	<pool name|session id> 풀 사용 고장(breakdown) 정보 출력
iov	VP에 의한 디스크 입출력 통계 출력
iof	청크/파일에 의한 디스크 입출력 통계 출력
ioq	큐에 의한 디스크 입출력 통계 출력
iog	AIO 전역 정보 출력
iob	IO VP 클래스에 의한 큰 버퍼 사용 내역 출력
ppf	[<partition number> | 0] 파티션 프로파일 출력
tpf	[<tid> | 0]스레드 프로파일 출력
ntu	네트 사용자 스레드 프로파일 정보 출력

ntt	네트 사용자 스레드 액세스 시간 출력
ntm	네트 메시지 정보 출력
ntd	네트 급송 정보 출력
nss	<session id>네트 공유 메모리 상태 출력
nsc	<client id>네트 공유 메모리 상태 출력
nsd	네트 공유 메모리 데이터 출력
sts	최대 스톡(stock) 크기와 현재 스톡 크기 출력
dic	사전 캐쉬 정보 출력
qst	큐 통계 출력
wst	스레드 대기 통계 출력
ses	<session id>세션 정보 출력
sql	<session id>sql 정보 출력
dri	데이터 복제 정보 출력
pos	/InformixDIR/etc/.infos.DBSERVERNAME 파일 출력
mgm	mgm 리소스 관리자 정보 출력
ddr	DDR 로그 사후 처리 정보 출력

표 2: onstat -g 구문에 대한 옵션

ONSTAT 명령들은 너무 방대해서 이 글에서 모두 다룰 수는 없습니다. 대신 여기에서는 서버를 감시(monitor)하는 데 있어 DBA에게 가장 유용한 핵심적인 ONSTAT 명령들에 초점을 맞출 것입니다.

서버의 현재 상태: onstat -

onstat - 명령은 서버의 현재 상태를 나타내는 간단한 메시지를 나타냅니다. 그림 1 은 예제 출력입니다:

```
lester@merlin >onstat -

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 11:54:44 -- 10656 Kbytes
```

그림 1: onstat - 명령은 서버의 현재 상태를 나타냅니다.

onstat - 명령은 Infomix Dynamic Server의 버전, 서버의 모드, 열려서 실행된 기간 그리고 사용 중인 메모리 양을 나타냅니다. 서버가 다운된 경우 그림 2 와 같이 "shared memory not initialized"라는 오류 메시지가 나타납니다.

```
lester@merlin >onstat -

shared memory not initialized for InformixSERVER 'merlindb713'
```

그림 2: 서버가 다운되었을 때 현재 상태입니다.

데이터베이스 서버 프로파일: onstat -p

-p 옵션은 기본적인 입출력과 시스템의 성능 프로파일을 나타냅니다. 그림 3 은 예제 출력입니다. 이 통계들은 서 버가 마지막으로 재부팅되거나 onstat -z 옵션으로 통계들이 마지막으로 재설정된 시점의

것들입니다.

```
lester@merlin >onstat -p
```

ovlock	ovuserthread	ovbuff	usercpu	syscpu	numckpts	flushes
0	0	168	6625.92	722.70	35	4320

	됩니다. ONCONFIG 파일의 BUFFERS 매개변수가 이 값에 영향을 줄 것입니다. BUFFERS 매개변수 값을 올릴 때에는 주의해야 합니다. BUFFERS를 너무 크게 하면 다른 프로세스들을 위한 메모리를 가져와서 사용하므로 시스템 전체를 느리게 만들 수 있습니다. BUFFERS 값을 올릴 때에는, 사용자 운영 체제의 스와핑과 페이징을 감시하십시오.
ovlock	이 값은 0이어야 합니다. 다른 숫자는 시스템이 마지막으로 재설정된 이후로 로크가 모두 소모되었음을 나타냅니다. ONCONFIG 파일에서 LOCKS 매개변수 값을 올리십시오.
ovuserthread	이 값은 0이어야 하고, 사용자가 연결을 시도하거나 현재 사용자가 ONCONFIG 파일에서 설정한 사용자 스레드의 최대값을 초과할 때마다 증가됩니다. 사용자 스레드의 최대값은 ONCONFIG 파일에서 NETTYPE 매개변수의 세 번째 값입니다.
ovbuff	이 값 역시 0이어야 합니다. 서버가 ONCONFIG 파일에 있는 BUFFER 매개변수에서 설정한 것보다 많은 버퍼를 얻으려고 할 때마다 증가됩니다.
bufwaits	이 값은 0이어야 하고, 사용자 스레드가 BUFFER를 위해 대기한 회수를 나타냅니다.
lokwaits	이 값은 0이어야 하고, 사용자 스레드가 LOCK을 위해 대기한 회수를 나타냅니다.
deadlks	이 값은 0이어야 합니다. 데드록이 감지되고 중단된 회수를 나타냅니다.
dltouts	이 값은 0이어야 하고, 분산 데드록이 감지된 회수를 나타냅니다.

표 3: onstat -p 옵션의 핵심 요소입니다.

메시지 로그 파일 표시: onstat -m

onstat -m 옵션은 서버 메시지 로그에 있는 마지막 내용을 나타냅니다. 그림 4 를 참조하십시오. 이 메시지 파일 은 서버에 관한 모든 메시지를 포함하고 있으며, 감시를 위한 핵심적인 시스템 구성 요소입니다. 그림 4 는 예제 를 나타냅니다. 이 옵션을 사용하여 로그 파일에 있는 마지막 20 개의 메시지를 볼 수 있습니다.

```
lester@merlin >onstat -m

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:41:12 --

10656 Kbytes

Message Log File: /u3 / Informix7 / online1.log
21:05:15 Checkpoint Completed: duration was 8 seconds.
21:05:43 Checkpoint Completed: duration was 6 seconds.
21:10:58 Checkpoint Completed: duration was 7 seconds.
21:16:06 Checkpoint Completed: duration was 7 seconds.
21:21:13 Checkpoint Completed: duration was 7 seconds.
21:26:20 Checkpoint Completed: duration was 7 seconds.
21:31:28 Checkpoint Completed: duration was 7 seconds.
21:36:36 Checkpoint Completed: duration was 8 seconds.
21:41:43 Checkpoint Completed: duration was 7 seconds.
21:46:51 Checkpoint Completed: duration was 8 seconds.
21:52:00 Checkpoint Completed: duration was 9 seconds.
21:57:09 Checkpoint Completed: duration was 8 seconds.
22:00:42 Logical Log 20 Complete.
22:00:43 Process exited with return code 1: /bin/sh /bin/sh -c
/u3/Informix7/lo
g_full.sh 2 23 "Logical Log 20 Complete." "Logical Log 20 Complete."
22:02:17 Checkpoint Completed: duration was 8 seconds.
```

그림 4: 메시지 로그 파일을 나타내기 위해 onstat -m을 사용합니다.

주의

화면 윈도우에 항상 DBA의 로그 파일이 나타나도록 하는 것이 좋습니다. 이렇게 하려면 -f 옵션과 함께 UNIX tail 명령을 사용하십시오. 이 명령은 파일에 줄이 추가될 때마다 마지막 줄을 계속해서 읽습니다. 시스템에서 이 로 그를 계속 감시하기 위해 다음과 같은 명령을 실행할 수 있습니다

(메시지 로그 파일이 $InformixDIR/online.log라고 가정합니다).

```
tail -f $InformixDIR/online.log
```

사용자 상태: onstat -u

사용자 활동을 감시하기 위해 사용하는 ONSTAT 옵션은 -u입니다. 그림 5는 이 명령을 사용한 예제 출력입니다. 핵심 필드는 서버가 사용자를 내부적으로 추적하는 데 사용하는 sessid인데, 이것은 사용자의 세션 ID를 나타 냅니다. 이 숫자는 특정 사용자의 세션을 제거하는 데 사용됩니다. 보다 자세한 내용은 onmode -z 명령을 참조 하십시오.

```
lester@merlin >onstat -u



5 active, 128 total, 17 maximum concurrent
```

그림 5: onstat -u 옵션은 사용자 상태를 나타냅니다. .

"flags" 칼럼은 사용자 활동을 나타냅니다.

다음은 플래그 필드 내에서 각각의 위치에 따른 중요한 플래그들입니다:

위치 1의 플래그	설명
B	버퍼(Buffer)에서 대기 중
C	검사점(Checkpoint)에서 대기 중
G	논리 로그 버퍼 쓰기(Logical log buffer write)에서 대기 중
L	로크(Lock)에서 대기 중
S	뮤텍스(Mutex)에서 대기 중
T	트랜잭션(Transaction)에서 대기 중
Y	조건(Condition)에서 대기 중
X	트랜잭션 롤백(Transaction rollback)에서 대기 중
위치 2의 플래그	설명

```
        P--
        -

5 active, 128 total, 17 maximum concurrent
```

그림 5: onstat -u 옵션은 사용자 상태를 나타냅니다. .

"flags" 칼럼은 사용자 활동을 나타냅니다.

다음은 플래그 필드 내에서 각각의 위치에 따른 중요한 플래그들입니다:

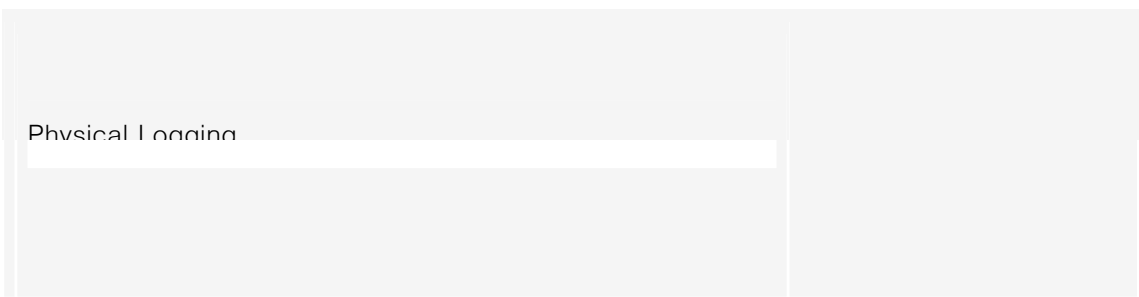
위치 1의 플래그	설명
B	버퍼(Buffer)에서 대기 중
C	검사점(Checkpoint)에서 대기 중
G	논리 로그 버퍼 쓰기(Logical log buffer write)에서 대기 중
L	로크(Lock)에서 대기 중
S	뮤텍스(Mutex)에서 대기 중
T	트랜잭션(Transaction)에서 대기 중
Y	조건(Condition)에서 대기 중
X	트랜잭션 롤백(Transaction rollback)에서 대기 중
위치 2의 플래그	**설명**
*	입출력 오류 동안의 활성 트랜잭션
위치 3의 플래그	**설명**
A	DB 영역 백업 스레드(Dbspace backup thread)
B	작업 시작(Begin work)
P	완료 작업을 위한 준비(Prepared for commit work)
X	완료 작업을 위한 TP/XA 준비
C	작업 완료하기(Committing work)

R	작업 롤백(Rolling back work)
H	경험적으로 작업 롤백(Heuristically rolling back work)
위치 4의 플래그	**설명**
P	세션을 위한 기본 스레드(Primary thread)
위치 5의 플래그	**설명**
R	호출 읽기(Reading call)
X	트랜잭션 완료 중(Transaction is committing)
위치 6의 플래그	**설명**
없음	
위치 7의 플래그	**설명**
B	B+ Tree 제거 스레드(B+ Tree cleaner thread)
C	종료 사용자 제거(Cleanup of terminated user)
D	데몬 스레드(Daemon thread)
F	페이지 플러셔 스레드(Page flusher thread)
M	ON-Monitor 사용자 스레드

표 4: 플래그와 플래그 필드에서의 위치

논리 로그 상태 onstat -l

ONSTAT의 -l 옵션은 논리 로그의 현재 상태를 나타냅니다. 그림 6 은 사용 예제입니다. 이 옵션의 출력에서 한 가지 문제점은 어떤 로그가 재사용할 준비가 되어 있는지 알 수 없다는 것입니다. Informix OnLine 버전 5.x 에서는 로그가 백업되자마자 그리고 오픈된 트랜잭션이 없으면 플래그 칼럼에 "F" 와 함께 유휴 상태(free)임 이 표시되었습니다. Infomix Dynamic Server의 버전 7.x에서 로그는 재사용되기 전까지 유휴 상태(free)로 표 시되지 않습니다. ONSTAT를 사용하여 재사용 가능한 로그를 표시하는 한 가지 방법은 -x와 함께 onstat -l을 사용하여 모든 활동 세션을 나타내는 것입니다.



	bufused	bufsize	numpages	numwrits	pages/io
Buffer	0	16	236	60	3.93
P-1	phybegin	physize	phypos	phyused	%used
	1003f	1000	967	0	0.00

Logical Logging

Buffer	bufused	bufsize	numrecs	numpages	numwrits	recs/pages	pages/io
L-3	0	16	90303	1522	275	59.3	5.5

address	number	flags	uniqid	begin	size	used	%used
a1ee3e4	1	U-B----	13	100427	500	500	100.00
a1ee400	2	U-B----	14	10061b	500	500	100.00
a1ee41c	3	U-B----	15	10080f	500	500	100.00
a1ee438	4	U-B----	16	100a03	500	500	100.00
a1ee454	5	U-B----	17	100bf7	500	432	84.40
a1ee470	6	U-B----	18	100deb	500	500	100.00
a1ee48c	7	U-B----	19	100fdf	500	500	100.00
a1ee4a8	8	U-B----	20	1011d3	500	500	100.00
a1ee4c4	9	U---C-L	21	1013c7	500	23	4.60
a1ee4e0	10	U-B----	10	1015bb	500	500	100.00
a1ee4fc	11	U-B----	11	1017af	500	500	100.00
a1ee518	12	U-B----	12	1019a3	500	500	100.00

그림 6: onstat -l 명령은 논리 로그 상태를 표시합니다.

플래그(flags) 칼럼은 각각의 로그에 대한 상태 정보를 다음과 같이 나타냅니다:

플래그	설명
A	새로 추가됨(Newly added): 사용 전에 보관(archive)을 실행해야 함
B	테이프나 "/dev/null"로 백업(Backup)됨
C	현재(Current) 사용 중인 논리 로그 파일
F	유휴 상태(Free)이며 사용 가능. 단 한 번도 사용되지 않은 로그를 제외하고는 로그는 유휴 상태로 표시되지 않으므로, 이 플래그는 Informix Dynamic Server를 처음으로 초기화한 때를 제외하고는 거의 보기 힘들 것입니다.
L	마지막 검사점(Last checkpoint)이 이 논리 로그에 있음
U	사용된 논리 로그(Used logical log): 백업되어 있거나 활성 트랜잭션을 포함하고 있지 않으면 유휴 상태일 것입니다.

표 5: onstat -l에 대한 플래그

트랜잭션 표시: onstat -x

onstat -x 옵션은 현재의 모든 트랜잭션을 나타냅니다. 가장 유용한 칼럼은 "log begin"입니다. 이 옵션은 트랜잭션이 시작되는 논리 로그에 관한 정보를 제공하며, onstat -l 명령과 함께 사용하면 어떤 로그가 유휴 상태이며 재사용할 수 있는지를 알 수 있습니다. 어떤 논리 로그가 가장 이른(earliest) 활성 트랙잭션을 가지고 있는지 알아보기 위해 "log begin" 칼럼에서 가장 이른 논리 로그 숫자를 찾아볼 수 있습니다. 가장 이른 트랜잭션의 로그 이전에 백업된 논리 로그는 서버에서 자동으로 재사용될 것입니다.

```
lester@merlin >onstat -x
a2f413c  A---      a2d0458  0       0       COMMIT     0
a2f4260  A---      a2d0898  0       0       COMMIT     0
a2f4384  A---      a2d1118  0       0       NOTRANS    0
```

```
                 -
a2f413c  A---    a2d0458   0     0     COMMIT     0
         -
a2f4260  A---    a2d0898   0     0     COMMIT     0
         -
a2f4384  A---    a2d1118   0     0     NOTRANS    0
         -
a2f44a8  A---    a2d1558   0     0     COMMIT     0
         -
a2f45cc  A-B-    a2d1118   2     21    NOTRANS    0
         -

 6 active, 128 total, 7 maximum concurrent
```

그림 7: onstat -x 옵션은 트랜잭션의 상태를 나타냅니다.

로크 표시: onstat -k

onstat -k 옵션은 활성 상태의 모든 로크를 나타냅니다. 그러나, 이 표시가 길어질 수 있다는 점에 주의해야 합니 다. ONCONFIG 파일에서 LOCKS의 값으로 큰 수가 정의되어 있고 사용자가 많을 경우에는 이 명령으로 수 천 줄이 나타날 수도 있습니다. 그림 8 은 사용 예제입니다.

```
lester@merlin >onstat -k
```

그림 8: onstat-k 옵션은 모든 로크를 나타냅니다.

```
1 active, 20000 total, 16384 hash buckets
```

그림 8: onstat-k 옵션은 모든 로크를 나타냅니다.

로크의 소유는?

"owner" 칼럼은 로크를 소유한 사용자의 공유 메모리에 있는 주소를 나열합니다. onstat -u 옵션과 함께 이 칼럼 을 사용하면 모든 사용자를 볼 수 있으며, 이 출력을 "address" 칼럼과 비교하여 소유자들의 username 을 확인 할 수 있습니다.

로크된 테이블은?

"tblsnum" 칼럼은 로크되어 있는 테이블을 표시합니다. 테이블의 partnum을 16 진수로 변환하려면 이 출력을 다음에 나오는 SQL 문과 비교하십시오. 이 문의 출력을 보면 어떤 테이블이 잠겼는지 알 수 있을 것입니다.

```
select tabname, hex(partnum) tblsnum from systables

   where tabid > 99;
```

```
EOF

database selected

tabname     tblsnum
genjournal  0x0010009E
gjsum       0x0010009F

2. 로크된 테이블 찾기
onstat -k

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 01:47:38 --
10656 Kbytes

Locks
address  wtlist  owner    1klist   type    tblsnum  rowid  key#/bsiz
a103e44  0       a2d1118  a103de4  HDR+X   10009f   0          0

3. 1 단계와 2 단계를 비교해보십시오. gjsum 테이블에
로크되어 있음을 알 수 있습니다.
```

그림 9: 어떤 테이블에 로크되어 있는지를 알아보기 위한 출력입니다.

tblsnum 100002는 데이터베이스 로크를 표시합니다. 데이터베이스를 사용하는 각각의 사용자들은 데이터베이스에 공유된 로크를 두게 될 것입니다.

로크 형태

다음은 사용 가능한 로크 형태와 로크를 식별하기 위한 정보를 나열한 것입니다:

로크 형태	설명

Database	tablespace 100002에 로크되어 있는 경우, Database 레벨의 로크입니다.
Table	rowid가 0 인 경우, Table 레벨의 로크입니다.
Page	rowid가 00 으로 끝나는 경우, Page 레벨의 로크입니다.
Row	실제 rowid(00 으로 끝나지 않는 경우)에 로크가 걸려 있는 경우, Row 레벨의 로크입니다.
Byte	key#/bsize에 실제 size 값이 들어 있는 경우, Byte 레벨의 로크입니다.
Key	16 진수 rowid를 가지는 경우(f로 시작하는 rowid), Key 레벨의 로크입니다.

표 6: onstat -k에 대한 로크

로크 형태 플래그

다음은 onstat -k의 "flags" 칼럼에 있는 로크 플래그의 목록입니다.

로크 플래그	설명
HDR	Header
B	Bytes lock
S	Shared lock
X	Exclusive lock
I	Intent lock
U	Update lock
IX	Intent-exclusive lock
IS	Intent-shared lock
SIX	Shared, intent-exclusive lock

표 7: onstat -k에 대한 로크 플래그

DB 영역과 청크 상태: onstat -d

onstat -d 명령은 DB 영역과 디스크 청크의 레이아웃 그리고 각각의 청크와 DB 영역의 상태에 관한 중요한 정보 를 제공합니다. 이 명령의 출력을 보관해두는 것이 좋습니다. 복원(restore) 작업 시 이 내용이 필요할 수 있습니 다. 출력에는 시스템을 재구축하는 데 필요한 각각의 DB 영역과 청크가 표시되어 있습니다. 그림 10 은 예제 출 력입니다.

```
lester@merlin >onstat -d

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:54:44 -- 10656 Kbytes

Dbspaces

address  number  flags  fchunk  nchunks  flags  owner     name

a2ce100  1       1      1       1        N      Informix  rootdbs

a2ce508  2       1      2       1        N      Informix  dbspace1

a2ce578  3       1      3       1        N      Informix  dbspace2

a2ce5e8  4       1      4       1        N      Informix  dbspace3

  4 active, 2047 maximum

Chunks

address  chk/dbs  offset  size    free   bpages  flags  pathname

a2ce170  1    1   0       250000  62047          PO-    /u3/dev/rootdbs1

a2ce280  2    2   0       10000   9587           PO-    /u3/dev/dbspace1

a2ce358  3    3   0       10000   9947           PO-    /u3/dev/dbspace2

a2ce430  4    4   0       10000   9587           PO-    /u3/dev/dbspace3

  4 active, 2047 maximum
```

그림 10: onstat -d 출력에는 DB 영역과 청크 상태가 나타납니다.

onstat -d 명령의 출력에는 각 청크에 있는 여유 공간과 각 청크의 상태가 표시되어 있습니다.

DB 영역에 대한 "flags"는 다음과 같습니다:

위치 1	
	P - 물리적 복구 중(Physical recovery underway)
	L - 논리적 복구 중(Logical recovery underway)
	R - 복구 중(Recovery underway)
위치 3	
	B - Blob 영역(Blobspace)

청크에 대한 "flags"는 다음과 같습니다.

	B - Blob 영역(Blobspace)
	- - DB 영역(Dbspace)
	T - 임시 DB 영역(Temporary Dbspace)

표 8: onstat -d에 대한 플래그

DB 영역과 청크 입출력: onstat -D

onstat -D 옵션은 청크 입출력을 나타냅니다. 이 옵션은 성능 조정(performance tuning) 시 매우 유용합니다. 모 든 청크에 읽기와 쓰기를 균등하게 배포하는 것이 목적입니다. 그림 11 은 한 청크만이 모든 입출력에 사용되고 다른 청크들은 비활성 상태임을 나타내는 예제입니다. 여기에서는 입출력이 청크들 사이에서 고르게 이루어지 지 않고 있는데, 이는 디스크가 효율적으로 사용되지 않음을 나타냅니다.

```
lester@merlin >onstat -D
```

그림 11: onstat -D 옵션은 DB 영역,청크의 입출력 사용을 나타냅니다.

```
a2ce280  2    2    0      3          0          /u3/dev/dbspace1
a2ce358  3    3    0      2          0          /u3/dev/dbspace2
a2ce430  4    4    0      2          0          /u3/dev/dbspace3
  4 active, 2047 maximum
```

그림 11: onstat -D 옵션은 DB 영역,청크의 입출력 사용을 나타냅니다.

페이지 쓰기 상태: onstat -F

서버가 공유 메모리에서 디스크로 페이지 쓰기를 하는 데에는 세 가지 방법이 있습니다. 첫 번째 방법이 포그라 운드 쓰기(foreground writes)인데, 이는 서버에 버퍼가 필요하며 여유 버퍼를 만들기 위해 디스크에 대한 버퍼 플러시 처리 과정을 중단해야 할 경우 발생합니다. 이는 가장 바람직하지 못한 쓰기 형식입니다. 두 번째 방법인 백그라운드 쓰기(LRU Writes)는 일정 퍼센트의 버퍼가 더티(dirty) 상태일 때 발생합니다. 이것은 ONCONFIG 파일의 LRU 매개변수에 의해 조절됩니다. 이 쓰기 방법은 사용자 처리를 중단하지 않기 때문에 대화형 시스템 에 가장 적합합니다. 마지막 방법인 청크 쓰기(Chuck Writes)는 검사점(checkpoint)에서 발생하는데, 이 때에는 모든 더티 버퍼 페이지가 디스크에 쓰여집니다. 따라서 더티 페이지가 많을수록 완료하는 데 시간이 더 오래 걸 립니다. 검사점 쓰기는 정렬되고(sorted) 최적화되지만(optimized), 검사점이 길수록 사용자의 활동을 더 오래 방해할 것입니다. 검사점 쓰기는 배치(batch) 시스템에 가장 적합합니다. 이 활동을 감시하기 위한 ONSTAT 옵 션은 -F입니다. 이것의 목표는 포그라운드 쓰기(Fg Writes)가 0 이 되도록 하는 것입니다. 그림 12 는 예제를 나타 냅니다.

```
lester@merlin >onstat -F
```

그림 12: onstat -D 옵션은 페이지 쓰기 상태를 나타냅니다.

```
states: Exit Idle Chunk Lru
```

그림 12: onstat -D 옵션은 페이지 쓰기 상태를 나타냅니다.

새로운 모니터링과 디버깅 명령들 (Informix Dynamic Server: 버전 7.x): onstat -g

-g 명령들은 Infomix Dynamic Server의 7.x 버전을 시작할 때 사용할 수 있는 새로운 명령들입니다. -g 명령들의 목록은 표 2에 있습니다. 여기에서는 이 -g 명령들 중 일부에 대해 설명할 것입니다.

모든 스레드 나열: onstat -g ath

onstat -g ath 옵션은 모든 활성 스레드를 나열합니다. 그림 13 은 예제를 나타냅니다.

```
Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:56:09 -- 10656 Kbytes

8       a34ab48  0        2  running           1cpu  sm_poll
9       a34b770  0        2  running           8tli  tlitcppoll
10      a34bce0  0        2  sleeping (Forever)  1cpu  sm_listen
11      a3c4a28  0        2  sleeping (secs: 2)  1cpu  sm_discon
12      a3c4e58  0        3  sleeping (Forever)  1cpu  tlitcplst
13      a3d0680  a2d0458  2  sleeping (Forever)  1cpu  flush_sub(0)
14      a3d0e40  a2d0898  2  sleeping (secs: 8)  1cpu  btclean
30      a35ea58  a2d1558  4  sleeping            1cpu  onmode_mon
```

				(secs: 1)		
8	a34ab48	0	2	running	1cpu	sm_poll
9	a34b770	0	2	running	8tli	tlitcppoll
10	a34bce0	0	2	sleeping (Forever)	1cpu	sm_listen
11	a3c4a28	0	2	sleeping (secs: 2)	1cpu	sm_discon
12	a3c4e58	0	3	sleeping (Forever)	1cpu	tlitcplst
13	a3d0680	a2d0458	2	sleeping (Forever)	1cpu	flush_sub(0)
14	a3d0e40	a2d0898	2	sleeping (secs: 8)	1cpu	btclean
30	a35ea58	a2d1558	4	sleeping (secs: 1)	1cpu	onmode_mon
283	a39ef38	a2d2218	2	cond wait (sm_read)	1cpu	sqlexec

그림 13: onstat -g ath 명령은 모든 활성 스레드를 나열합니다.

모든 스레드 나열: onstat -g ath

onstat -g ath 옵션은 모든 활성 스레드를 나열합니다. 그림 13 은 예제를 나타냅니다.

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:56:09 -- 10656 Kbytes

2	a336b70	0	2	sleeping	3lio	lio vp 0

tid	tcb	rstcb	prty	status	vp-class	name
2	a336b70	0	2	sleeping (Forever)	3lio	lio vp 0
3	a336dd0	0	2	sleeping (Forever)	4pio	pio vp 0
4	a337088	0	2	sleeping (Forever)	5aio	aio vp 0
5	a337340	0	2	sleeping (Forever)	6msc	msc vp 0
6	a337af8	0	2	sleeping (Forever)	7aio	aio vp 1
7	a337e00	a2d0018	4	sleeping (secs: 1)	1cpu	main_loop()
8	a34ab48	0	2	running	1cpu	sm_poll
9	a34b770	0	2	running	8tli	tlitcppoll
10	a34bce0	0	2	sleeping (Forever)	1cpu	sm_listen
11	a3c4a28	0	2	sleeping (secs: 2)	1cpu	sm_discon
12	a3c4e58	0	3	sleeping (Forever)	1cpu	tlitcplst
13	a3d0680	a2d0458	2	sleeping (Forever)	1cpu	flush_sub(0)

14	a3d0e40	a2d0898	2	sleeping (secs: 8)	1cpu	btclean
30	a35ea58	a2d1558	4	sleeping (secs: 1)	1cpu	onmode_mon
283	a39ef38	a2d2218	2	cond wait (sm_read)	1cpu	sqlexec

그림 13: onstat -g ath 명령은 모든 활성 스레드를 나열합니다

가상 프로세서 상태 나열 onstat -g sch

onstat -g sch 옵션은 어떤 oninit UNIX 프로세서가 어떤 서버 가상 프로세서(VP)와 대응하는지를 표시하기 위 해 사용합니다. UNIX에서 ps -ef 명령을 수행할 때, 여러 oninit 프로세서가 실행되고 있는 것을 볼 수 있습니다. 각각은 데이터베이스 서버에 대한 특수한 작업을 수행합니다. ps -ef 명령으로 얻은 UNIX "pid" 칼럼을 사용하여 onstat -g sch에서 나온 "pid" 칼럼에 프로세스를 상호 연결시킬 수 있습니다. 그림 14 는 이 명령에서 나온 예제 출력입니다.

```
lester@merlin >onstat -g sch
```

그림 14: onstat -g sch 명령은 가상 프로세서의 상태를 나타냅니다.

5	234	aio	144794	0	0	
6	235	msc	756	0	0	
7	236	aio	64028	0	0	
8	237	tli	3	0	0	

그림 14: onstat -g sch 명령은 가상 프로세서의 상태를 나타냅니다.

SQL 문 유형 나열: onstat -g sql

이것은 새로운 옵션들 중에서 가장 흥미로운 옵션입니다. 이것을 사용하면 사용자가 실행하고 있는 실제 SQL 문을 찾아가서 볼 수 있습니다. 그림 15 의 예제는 실행 중인 모든 SQL 문의 요약을 나타냅니다. session id를 사용하면 세부 사항과 실행되고 있는 실제 SQL 문을 볼 수 있습니다. 그림 16 은 세부 사항에 대한 예제입니다.

lester@merlin >onstat -g sql

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:52:02 --10656 Kbytes

Sess ID	SQLStmt type	Current Database	Iso Lvl	Lock Mode	SQL ERR	ISAM ERR	F.E. Vers
264	INSERT	ffsdw	NL	Not Wait	- 264	0	7.23

그림 15: onstat -g sql명령은 모든 sql 문을 나열합니다.

특정 사용자가 사용 중인 SQL 문 나열: onstat -g sql sid

lester@merlin >onstat -g sql 264

Sess	SQL Stmt	Current	Iso	Lock	SQL	ISAM	F.E.

VP Scheduler Statistics:

Sess ID	SQL Stmt type	Current Database	Iso Lvl	Lock Mode	SQL ERR	ISAM ERR	F.E. Vers
264	INSERT	ffsdw	NL	Not Wait	-264	0	7.23

Current SQL statement:

```
  insert into gjsum select exp_org, exp_prog, bud_obj_code, job_num,
    sum (exp_amount) from genjournal group by 1, 2, 3, 4
```

Last parsed SQL statement:

```
  insert into gjsum select exp_org, exp_prog, bud_obj_code, job_num,
    sum (exp_amount) from genjournal group by 1, 2, 3, 4
```

그림 16: onstat -g sid 명령은 특정 사용자가 사용 중인 SQL 문을 나타냅니다.

이 옵션은 SQL 코드로의 액세스가 불가능하거나 데이터베이스 테이블과 색인을 최적화해야 할 때와 같은 경우 에 매우 유용합니다. 이 명령을 반복적으로 수행함으로써 처리되고 있는 SQL 문들을 볼 수 있습니다. 그러면 SQL 을 모아서 검토해봄으로써 시스템의 전체 성능을 개선하기 위해 색인을 어디에 추가해야 할지를 결정할 수 있습니다.

이 옵션은 또한 프로그램 트랜잭션 디버깅을 위해 사용할 수 있습니다. 예를 들면 프로그래머는 이 옵션을 사용 하여 프로그램이 운영되는 동안 onstat -g sql *sid*를 실행함으로써 프로그램을 디버그할 수 있습니다. 이렇게 하 면 프로그래머가 프로그램에서 놓친 오류 조건과 SQL 오류들을 찾을 수 있습니다.

사용자 세션 나열: onstat -g ses

onstat -g ses 옵션은 각 세션에서 사용되는 메모리의 양을 비롯하여 사용자 세션에 관한 추가 정보를 제공합니 다. 그림 17 은 사용 예제입니다. 이 옵션은 세션에 관한 자세한 정보를 나타내는 데도 사용됩니다.

```
lester@merlin >onstat -g ses
```

Informix OnLine Version 7.23.UC1 -- On-Line -- Up 7 days 12:57:48 --10656 Kbytes

session

id	user	tty	pid	hostname	#RAM threads	total memory	used memory
265	Informix	-	0	-	0	8192	4680
264	lester	4	4249	merlin	1	106496	97840
10	Informix	-	0	-	0	8192	4680
7	Informix	-	0	-	0	16384	13144
6	Informix	-	0	-	0	8192	4680
4	Informix	-	0	-	0	16384	13144
3	Informix	-	0	-	0	8192	4680
2	Informix	-	0	-	0	8192	4680

그림 17: onstat -g ses 명령은 사용자 세션의 목록을 나타냅니다.

ONSTAT 명령 반복: -r

계속적으로 ONSTAT 명령을 반복하려면 -r [# of seconds] 옵션을 사용합니다. 이 옵션은 상황을 모니터할 때 매 우 유용합니다. 다음의 예제는 매 10 초마다 논리 로그의 상태를 표시해줍니다.

```
onstat -l -r 10
```

ONSTAT 공유 메모리 통계 삭제: onstat -z

서버 통계는 서버를 재시동할 때마다 재설정됩니다. 서버가 실행 중일 때(종료하지 않고) 통계를 재설정하려면 다음 명령을 사용하십시오.

```
onstat -z
```

위의 명령은 ONSTAT와 SMI 테이블에 대한 모든 통계를 삭제할 것입니다.

맺음말

ONSTAT 유틸리티는 DBA에게 데이터베이스 서버를 관리하고 감시하기 위한 강력한 툴킷을 제공합니다. 이 글 이 ONSTAT 유틸리티의 사용의 발판이 되기를 바랍니다.