1장. 빌드 시스템의 개요

빌드 시스템의 목적

- 가장 기본적인 목적은 사람이 읽을 수 있게 작성된 소스코드를 실행 프로그램으로 변환하는 데 있다.
- 빌드 시스템은 웹 기반 애플리케이션 패키징과 문서 생성, 소스코드 자동 분석 등의 다양한 관련 활동을 지원
- 프로세스의 대한 빌드 시스템의 세부 사항은 운영체제와 프로그래밍 언어의 종류에 따라 다르지만, 이런 가변성에도 빌드 시스템의 기본 개념은 모든 빌드 시스템에 포괄적으로 적용 된다.

빌드 시스템이란?

- C와 C++ 같은 전통적인 컴파일형 언어로 작성된 소프트웨어 컴파일로, 이때 소프트웨어는 자바나 C# 같은 최근 소개된 언어를 포함하여 확장될 수 있다.
- Perl이나 Python같은 인터프리트형 언어 로 작성된 소프트웨어의 패키지와 테스트
- 웹 기반 애플리케이션의 컴파일과 패키지로, 애플리케이션은 다양한 유행의 설정 파일을 포함한다. 이뿐만 아니라 정적 HTML 페이지, 자바나 C#으로 작성된 소스코드, JSP나 ASP, PHP 등의 문법으로 작성된 하이브리드 파일 등도 포함할 수 있다.
- 단위 테스트 수행으로, 소프트웨어의 작은 부분을 나머지 전체 소스코드로 부터 분리해 독립적으로 검증한다.

- 정적 분석 도구 실행으로, 프로그램의 소스코드로부터 버그를 찾아내기 위해서 정적 분석을 진행한다. 여기서 빌드 시스템의 출력물은 실행 프로그램이 아닌 버그 리포트 문서이다.
- PDF나 HTML 문서의 생성으로, 이런 유형의 빌드 시스템은 다양한 서식으로 작성된 입력 파일들을 처리하지만, 해당 출력물은 언제나 사람이 읽을 수 있는 형태의 문서이다.

컴파일형 언어

- 소스파일은 오브젝트 파일로 컴파일되며, 컴파일된 오브젝트 파일은 코드 라이브러리나 실행 프로그램으로 링크된다. 이 결과로 생성된 파일은 타겟 머신에 설치할 수 있는 하나의 릴리스 패키지로 묶인다.

컴파일형 언어를 위한 전통적인 빌드 시스템의 개요 구성 요소 설명

- 버전 관리도구 → 프로그램의 소스코드를 보관하며, 여러 개발자가 코드베이스를 동시에 수정할 수 있게 한다. 또한 코드의 이전 버전 검색을 용이하게 한다. 버전 관리 도구의 대표적인 예로는 CSV나 SubVision, Git, ClearCase등이 존재한다.
- 소스트리와 오브젝트 트리 → 한 명의 개발자가 작업하는 소스 파일이나 컴파일된 오브젝트 파일의 트리이다. 개발자는 트리를 통해 다른 사람에게 영향을 주지않고 자신만의 개인적 수정을 하는 것이 가능하다.
- 컴파일 도구 → 입력 파일을 받아 출력 파일을 생성하는 도구이다. 컴파일 도구의 대표적 예로는 C나 자바 컴파일러를 들 수 있으며, 이 두 도구는 컴파일 기능 외에도 문서와 단위 테스트 생성 기능을 함께 포함하고 있다.
- 빌드 머신 → 컴파일 도구가 실행되는 컴퓨팅 장치이다.
- 릴리스 패키징과 타겟 머신 → 소프트웨어 패키지가 만들어지고 최종 사용자에게 릴리스되며, 타겟 머신에 설치되는 방법이다.

인터프리트형 언어

인터프리트형 언어는 컴파일형 언어와는 빌드 시스템 모델이 조금 다르다.

인터프리트형 소스코드는 오브젝트 코드로 컴파일되지 않기 때문에 오브젝트 트리가 필요 없다. 대신 소스 파일이 타겟 머신에 설치될 수 있게 소스 파일을 바로 릴리스 패키지에 보관한다. 이런 유형의 빌드 시스템에도 컴파일 도구가 필요할 때가 종종 있는데, 이럴 때 컴파일 도구는 단지 소스 파일을 변환해 릴리스 패키지에 저장하기 위해 사용된다. 기게어 코드로의 컴파일은 빌드 타임 뿐만 아니라 런타임에서도 수행되지 않는다.

웹 기반 애플리케이션

컴파일형 코드와 인터프리트형 코드와 설정 파일, 데이터 파일이 함께 묶여 웹 기반 애플리케이션을 구성한다.

웹 기반 애플리케이션은 다른 어떤 파일들은 소스트리에서 릴리스 패키지로 바로 복사되며(HTML 파일), 이와 달리 우선 오브젝트 코드로 컴파일된 후에 릴리스 패키지로 복사되기도 한다.(자바 소스 파일), 또한 웹 애플리케이션 서버와 최종 사용자의 웹 브라우저는 코드의 인터프리팅과 컴파일을 수행하는데, 이는 빌드 시스템의 범위를 넘어선다.

전형적인 웹 애플리케이션은 다음과 같은 다양한 파일 유형과 관련이 있다.

- 정적 HTML 파일 → 웹 브라우저에서 표시되는 마크업 데이터만을 포함하고 있으며, 이런 파일은 바로 릴리스 패키지로 복사된다.
- 자바 스크립트 파일 → 최종 사용자의 웹 브라우저에 의해 인터프리트될 코드를 포함하며, 정적 HTML 파일과 동일하게 바로 릴리스 패키지로 복사가 된다.
- JSP나 ASP, PHP페이지 → HTML 코드와 프로그램 코드를 함께 갖고 있으며, 빌드 시스템이 아닌 웹 애플리케이션 서버가 컴파일해 실행이 된다. 이런 유형의 파일도 역시 릴리스 패키지로 바로 복사돼 웹 서버상에 설치될 수 있게 준비 된다.
- 자바 소스파일 → 오브젝트 코드로 컴파일되고 패키지로 만들어져 웹 어플리케이션이 일부가 되며, 여기서 빌드 시스템은 자바 클래스 파일의 패키징에 앞서 자바 소스 파일을 변환하는 역할을 수행한다. 자바 클래스는 웹 어플리케이션 서버나 웹 브라우저에서 실행된다.

> 💡 물론 빌드 시스템이 앞에서 살펴본 파일과는 다른 형식의 입력 파일로 부터 HTML, Javascript, JSP,ASP,PHP와 같은 파일을 자동 생성하는 예도 존재한다. 이는 빌드 시스템이 최종적으로 출력물을 릴리스 패키지로 만들기 까지 다양한 추가 컴파일 단계들을 거쳐야 할 수도 있음을 의미한다.

단위 테스트

단위 테스트 환경을 위한 빌드 시스템은 지금 까지 설명한 빌드 시스템 모델을 확장한 것이다. 빌드 시스템은 타겟 머신에 설치될 릴리스 패키지를 생성하는 대신 더욱 작은 크기의 단위 테스트 스위트를 다수 생성한다. 타겟 머신에서 각 스위트가 실행되는데, 스위트는 소프트웨어가 예상한 대로 동작하는지에 따라 '합격'이나 '실패'로 테스트의 결과를 생성한다.

단위 테스트 빌드 시스템은 사실 일반적인 빌드 시스템의 변형일 뿐이다. 인터프리트형 언어와 웹 기반 애플리케이션의 경우도 이와 유사한 단위 테스트 빌드 시스템을 생성할 수 있다.

정적 분석

커버리티 프리벤트나 록워크 인사이트, 파인드버그와 같은 정적 분석 도구는 잠재적 버그를 밝혀내기 위해 프로그램 소스코드를 검사한다. 이런 분석은 소프트웨어가 올바르게 동작하는지 실행을 통해 파악하지 않고 정적으로 수행한다.

정적 분석 시스템의 입력물은 일반 빌드 시스템에서 사용하는 소스코드와 같다. 하지만 오브젝트 트리와 릴리스 패키지를 새성하지 않으며, 결함 리포트 문서나 이와 유사한 유형의 출력물을 만든다.

문서 생성

이 빌드 시스템의 출력물은 PDF 파일이나 HTML 페이지의 묶음, 그래픽 이미지를 비롯해 문서라 여겨지는 모든 유형일 수 있다. 문서 생성 과정에서 데이터 파일이 필요할 수 있으며, 이 빌드 시스템에서도 오브젝트 트리가 적용된다. 이 경우는 타겟 머신을 포함하지 않는 대신 프린터나 웹 브라우저, PDF 뷰어 등의 문서를 읽을 수단이 필요하다.

빌드 시스템의 구성 요소

1. 버전 관리 도구

버전 관리 도구는 빌드 시스템의 첫 번째 구성 요소이다. 개발자는 어떤 소프트웨어든 컴파일에 이르기까지 소스코드의 개인 복사본을 유지해야만 한다. 개발자는 맡은 작업에 따라 소스코드를 수정하며, 빌드 시스템은 이를 바탕으로 소프트웨어를 컴파일한다.

버전 관리 도구를 통해 다음과 같은 작업을 수행할 수 있다.

- 소스코드의 복사본을 가져와 개인저그올 진행할 수정 작업을 준비한다.
- 체크인이나 커밋을 관리해 개인적 수정 사항을 나머지 개발자가 사용할 수 있게 한다.
- 다양한 코드 스트림의 생성을 원할히 해 동일 제품의 여러 다른 버전 을 개발하고 유지보수하는 과정을 관리한다.
- 파일 접근을 관리해 허가된 개발자만이 소스 파일을 수정할 수 있게 한다.
- 개발자가 소스 파일의 이전 버전을 열람할 수 있게 한다. 새로운 버전이 이전 내용을 대체했더라도 열람이 가능하다.

2. 소스와 오브젝트 트리

프로그램의 소스 코드는 여러 디스크 파일로 나눠 저장되며, 여러 디렉터리에 정렬된 파일의 상태를 소스 트리라 한다. 소스 트리의 소스코드 구조는 빌드 시스템 설계에 지대한 영향을 미친다.

소스 트리의 구조는 종종 소프트웨어의 아키텍처를 반영하기도 한다. 빌드 기술 파일은 그들이 처리할 소스 파일과 동일한 디렉터리에 저장돼 있다. 이는 빌드 기술 파일을 저장하는 유일한 방법은 아니지만, 각 디렉터리안에서 파일을 관리할 빌드 시스템의 하위 요소를 편리하게 배치할 수 있게 한다.

소스 트리의 옆에는 오브젝트 트리가 함께 나타나 있다. 오브젝트 파일을 소스 파일과 같은 디렉터리에 저장하는 방법도 가능하지만 지저분한 접근이 되고 마는 경우가 많다. 이보다는 빌드 프로세스가 생성하는 오브젝트 파일이나 실행 프로그램을 저장할 별도의 트리 계층을 만들어야 한다.

프로그램을 여러 소스 파일로 분할해 디스크의 여러 디렉터리에 나눠 담기 위한 중요 고려 사항은 다음과 같다.

- 이해(Comprehension): 사람들은 프로그램을 쪼개 논리적 세부 항목으로 나눌 때 더 쉽게 이해할 수 있다는 개념을 발견했다. 이 개념은 사람들이 프로그램을 서로 다른 여러 클래스의 묶음으로 이해할 수 있게 하는 객체지향 프로그래밍의 기본 전제다. 각 클래스는 프로그래머가 항상 명심해야 할 외부 행동과 클래스의 복잡도를 외부 관점으로부터 숨기는 내부 구현을 함께 담고 있어야 한다. 그러므로 빌드 시스템에서는 소스코드를 여러 항목으로 나눠 각 항목이 프로그램 기능의 특정 영역을 캡슐화하는 방법이 최선이다.
- 소스코드 관리: 프로그램의 소스코드가 다양한 파일과 디렉터리로 퍼져 나갈때 소스코드 관리 도구를 사용하면 소스코드를 관리하는 작업은 더욱 쉬워진다. 반대로 하나의 디스크 파일에 프로그램 전체가 저장된다면 여러 개발자가 간섭하지 않고 각자의 코드 수정 사항을 끊임없이 반영하기란 어렵다.
- 성능(Performance) : 편집기나 컴파일러와 같은 개발 도구는 작은 단위로 작업을 수행할 때 더욱 효율적이다. 따라서 개발 도구가 수 메가바이트에 이르는 큰 크기의 소스 파일을 다룰 수 있더라도 이런 큰 파일을 처리하는 동작은 매우 비효율 적이다.

3. 컴파일 도구와 빌드 도구

개발자가 소스 트리로 작업할 때에는 사랑미 읽을 수 있는 소스 파일을 장치가 인식할 수 있는 실행 프로그램으로 변환하는 방법을 알고 있어야 한다. 컴파일 도구는 입력 파일을 읽어 출력 파일로 변환하는 일종의 프로그램이다.

다음은 컴파일 도구의 일반적인 예이다.

- C 컴파일러 : 사람이 작성한 어떤 프로그램의 C언어 소스 파일을 읽고, 이를 기계어 코드로 변환해 오브젝트 파일을 생성한다. 이 시나리오에서 컴파일 도구의 출력물과 입력물의 기능은 서로 동일해야 한다. 출력물은 타겟 머신이 이해할 수 있는 형태에 가까워졌을 뿐이다.
- 링커(Linker) : 여러 오브젝트 파일을 합쳐 하나의 실행 가능한 프로그램 이미지를 생성한다. 앞선 빌드 단계에서 컴파일러의 출력물이었던 오브젝트 파일이 여기서는 링커 도구의 입력물이 된다. 즉, 소스 파일과 오브젝트 파일이 아니라 입력 파일과 출력 파일의 측면에서 이해해야 한다.
- UML 기반의 코드 생성 : 입력물로 UML모델을 읽어 자바나 C++, C# 같은 범용 프로그래밍 언어로 작성된 동일한 프로그램을 생성한다.
- 문서 생성기 : 마크업 언어로 작성된 파일을 읽고 출력물로 PDF파일을 생성한다.
- 새로운 디렉터리를 만들기 위한 커맨드라인 도구: 파일 시스템에 새로운 디렉터리를 만든다. 이 시나리오에서 유일한 입력물은 새로운 디렉터리의 이름이다.

컴파일러와 컴파일 도구 간의 차이점

컴파일러는 일반적으로 상위 수준의 프로그래밍 언어로 작성된 소스코드를 오브젝트 코드로 변환한다. 하지만 컴파일 도구는 입력 데이터를 출력 데이터로 변환하는 모든 도구를 포함한다.

반면 빌드 도구는 컴파일 도구보다 한 단계 상위 수준의 기능을 포함한 프로그램이다. 이는 빌드 도구가 빌드 프로세스 전반을 조직하기 위해 소스 파일과 오브젝트 파일 간의 관계에 관한 충분한 지식이 있어야 함을 의미한다. 빌드 도구는 최종 빌드 출력물을 생산하는 데 필요한 컴파일 도구를 호출한다.

4. 빌드 머신

처음에는 잘 드러나지 않을 수도 있지만, 컴파일과 빌드 도구가 실행하는 머신은 빌드 시스템 관리에서 중요한 역할을 수행한다. 모든 도구는 빌드 머신에서 실행할 수 있어야 하고, 이는 시간이 흘러 하드웨어나 운영체제가 변경되더라도 마찬가지다.

또한 소프트웨어가 동일한 유형의 머신에서 컴파일 돼 실행되는지, 혹은 소프트웨어가 완전히 다른 환경에서 실행되는지를 고려해야 한다. 환경에는 두 가지, 네이티브 컴파일 환경, 크로스 컴파일 환경이 존재한다.

- 네이티브 컴파일 환경은 소프트웨어는 빌드 머신과 동일한 타겟 머신에서 실행된다.
- 크로스 컴파일 환경은 서로 다른 두 머신이 필요하며, 타겟 머신과 빌드 머신은 운영체제나 CPU가 다르다.

5. 릴리스 패키징과 타겟 머신

빌드 도구가 수행하는 작업 대부분은 오브젝트 파일과 실행 프로그램을 생성하는 데 초점을 맞추고 있지만, 최종 패키징 단계에 이르면 결국은 사용자 머신에 실제로 설치할 수 있는 무언가를 만들어야 한다. 초보자에게 여러 실행 프로그램과 데이터 파일을 건네주고는 알아서 설치하고 설정하기를 바라는 생각은 비현실 적이다. 대신 다운로드할 수 있는 하나의 파일이나, 컴퓨터의 CD-ROM 드라이브에서 사용할 수 있는 하나의 CD, 혹은 DVD의 형태로 제공해야 한다. 특히 일반 가정 소비자의 시장을 대상으로 작성된 소프트웨어의 설치 과정은 아이콘을 더블클릭하고 간단한 일부 질문에 대답하는 정도 보다 복잡해서 안 된다.

그러므로 빌드 프로세스의 최종 단계에서는 소스 트리와 오브젝트 트리로부터 관련된 파일을 추출해 릴리스 패키지에 저장한다. 릴리스 패키지는 가급적 하나의 디스크 파일이어야 하고 압축돼야 한다. 이를 통해 다운로드에 소요되는 시간과 파일을 담는 데 필요한 DVD의 수를 줄일 수 있다. 이에 더해 소프트웨어의 설치를 방해하지 않게 불필요한 디버그 정보는 제거되어야 한다.

소프트웨어를 패키징하고 설치하기 위한 다음과 같은 세 가지의 일반적인 방법이 존재한다.

- 아카이브 파일 : 이는 파일을 압축해 하나의 디스크 파일로 묶는 가장 직관적이고 쉬운 방법이다. 최종 사용자는 아카이브 파일을 만드는 동작의 반대 작업을 통해 소프트웨어를 설치해야 한다.
- 패키지 관리 도구 : 완전한 소프트웨어 패키지를 인터넷에서 다운로드해 이를 운영체제의 추가 요소로 설치하는 유닉스 계열 환경에서 일반적으로 사용된다. 설치는 한 단계로 구성되는 프로세스이며, 설치에 필요한 그 외 모든 선행 패키지도 함께 설치된다. 일반적 예로는 .rpm, .deb를 포함한 패키지 파일이 존재한다.
- 커스터마이즈된 GUI 설치 도구 : 마이크로소프트 윈도우 운영체제에서 소프트웨어를 설치해 본 경험이 있는 이라면 누구나 익숙한 방법이다. 설치 프로세스는 단순히 아이콘의 더블클릭으로 시작하며, 최종 사용자는 소프트웨어의 설치를 위해 작성된 별도의 GUI 설치 도구와 상호 작용한다.

마지막으로 고려 사항으로 소프트웨어 일부만 설치돼 나머지 부분은 런타임에 접근 할 수도 있음을 간략히 살펴보자. 최종 사용자의 컴퓨터에는 소프트웨어의 일정부분만이 설치되며, 프로그램을 실행할 때 나머지 코드나 데이터에 접근한다. 대표적인 예로는 그래픽 이미지와 영상과 소리 파일을 필요할 때마다 DVD로부터 읽어올 뿐 절대 타겟 머신의 하드디스크에는 저장하지 않는 비디오 게임을 들 수 있다. 또한 구글 어스 같은 도구도 클라이언트 프로그램이 설치 되지만, 나머지 데이터는 필요에 따라 인터넷에서 다운로드하는 방식으로 동작한다.

릴리스 패키지의 생성은 가장 소프트웨어 빌드 프로세스의 가장 마지막 단계이다.

6. 빌드 프로세스와 빌드 기술

사람이 이 프로세스를 다이어그램 형태로 배치하는 일은 쉽더라도 빌드 도구에게는 텍스트 기반 형태로 작성된 빌드 기술이 필요하다. 예로 Make를 사용할 때에는 파일 간 종속적 정보가 규칙의 형태로 설정되며, 이는 Makefile이라는 이름의 파일에 저장된다.

Stock 프로그램은 작은 수의 소스 파일로만 구성되기 때문에 빌드 기술이 간단하며, 하나의 파일에 보기 좋게 담길 수 잇다. 이보다 큰 프로그램에서는 수백의 작은 파일들이 빌드 기술로 구성되고는 하는데, 이 파일은 서로 함께 동작하며 전체 프로그램에 대한 빌드 처리법을 찾아낸다.

7. 빌드 관리 도구

빌드 관리 도구는 버전 관리 도구와 통신해 빌드 트리를 체크아웃하고, 소프트웨어를 컴파일하기 위해 빌드 시스템을 호출하며, 개발자에게 빌드의 종료를 알린다. 관점에 따라 빌드 관리 도구를 빌드 시스템에 속하는 일부분으로 볼 수도 있지만 이 책에서는 분리해 살펴본다.

훌륭한 빌드 관리 도구는 다음과 같은 기능을 제공한다.

- 미리 정해진 일정이나 단순히 새로운 코드가 커밋될 때 소스 트리를 체크아웃하고 빌드한다.
- 여러 빌드 머신으로 구성된 풀을 여러 빌드 작업이 함께 공유할 수 있게 큐 메커니즘을 제공한다. 충분한 수의 머신이 사용 가능한 상태가 되면 다음 작업이 시작된다.
- 다양한 그룹의 사용자에게 이메일로 알람 메세지를 발송한다.
- 언제 빌드가 이뤄졌는지와 빌드의 성공 여부를 보여주는 그래픽 사용자 인터페이스를 제공한다.
- 각 빌드가 성공한 후 버전 번호를 증가시키며 버전 번호를 관리한다.
- 테스터가 사용할 수 있게 최종 소프트웨어 패키지를 아카이브 디렉터리에 보관한다.
- 모든 성공한 빌드를 대상으로 온전성 검사를 시작한다.
- 누가 최근에 나쁜 코드를 체크인했는지의 guilty list를 통해 어떤 개발자들이 이 리스트에 있는지를 알 수 있다.

빌드 도구 관리는 어느 정도 이상의 사람이 모인 모든 소프트웨어 프로젝트에서 중요한 역할을 수행한다. 상용으로 개발된 도구는 물론이고 오픈소스의 지원을 받거나 오픈소스로 개발된 도구와 같은 다양한 도구를 선택할 수 있다. 잘 알려진 도구로는 Build Forge, ElectricCommander, CruiseControl, Hudson등이 있다.

8. 빌드 시스템 품질

- 편의성 : 도구와 빌드 기술 파일은 사용하기 편리해야 하고 이를 사용하는 개발자에게 너무 많은 짐을 지워서는 안된다. 개발자는 빌드 도구의 복잡한 사항을 처리하기 보다는

소스코드의 작성에 집중할 수 있어야 한다.

- 정확성 : 빌드 도구는 언제나 올바른 컴파일러 옵션을 사용해 정확한 파일을 컴파일/링크해야 한다. 또한 컴파일 순서가 결과에 영향을 미치는 경우 빌드 도구는 최종 실행 프로그램이 소스 파일의 내용을 항상 잘 반영하게 올바른 순서로 파일을 컴파일해야 한다.
- 성능 : 이상적인 환경에서의 빌드 프로세스는 어떤 두드러진 지연도 없이 완료될 수 있겠지만, 현실적으로 보면 빌드는 실행되는 컴퓨팅 장치가 지원하는 한 빠르게 수행돼야 한다.
- 확장성 : 빌드 도구는 큰 프로그램을 빌드 할때에도 편리하고, 올바른 릴리스 이미지를 제공하며, 잘 동작해야 한다.