애플리케이션 배포

1. 애플리케이션 배포환경 구성하기

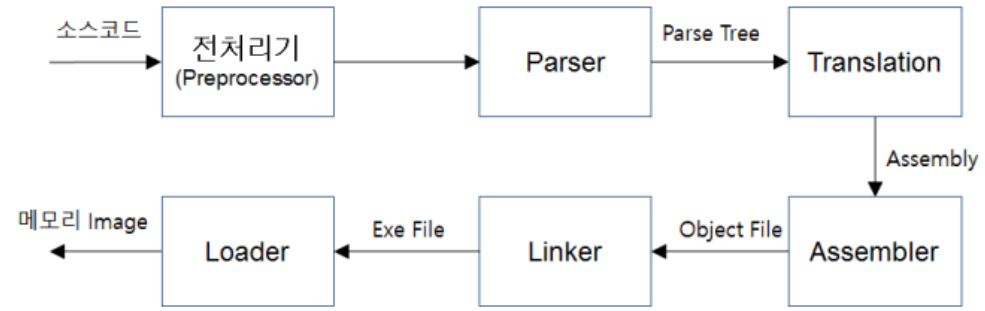
1 - 1 애플리케이션 배포환경 구성

① 소스코드 빌드 과정의 이해

빌드는 소스코드를 실행할 수 있는 상태로 변환하는 과정을 말하며, 프로그래밍 언어의 유형에 따라 빌드 과정이 상이하다

1. 컴파일 언어(C, C++ 등)

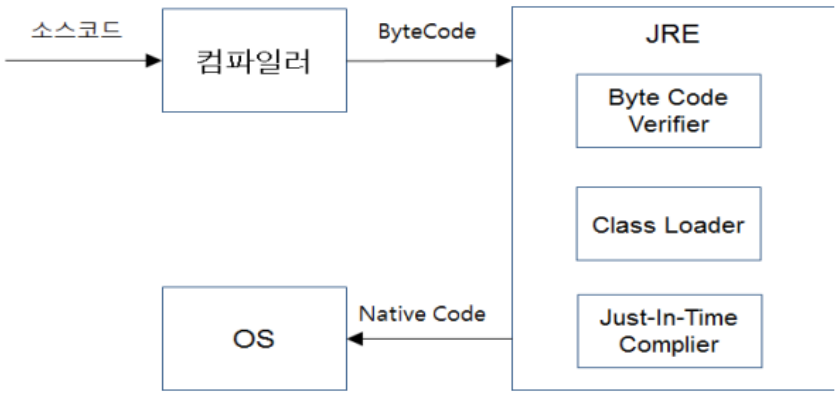
컴파일 언어는 기계어로 바로 변환되어 실행되기 때문에 가장 속도가 빠르고 보안에 유리하다. 하지만 소스 변경 시마다 컴파일 과정을 통해서 빌드 작업을 수행하기 때문에 빌드 과정이 오래 걸린다.
빌드 과정은 전처리기, 파싱, 번역, 어셈블리, 링킹 과정을 통해서 진행된다



[그림 1-1] 컴파일 언어의 빌드 과정

2. Byte Code 언어(Java, C# 등)

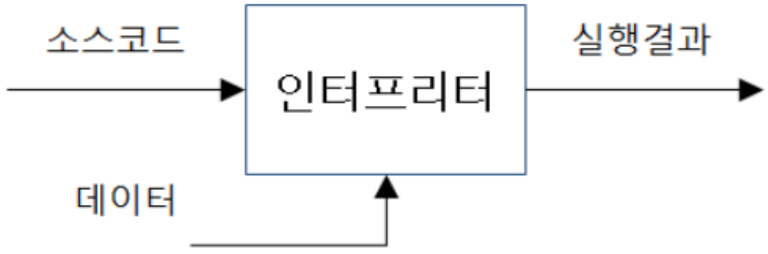
Byte Code 언어는 컴파일의 결과물이 실행파일이 아닌 'class'라는 바이트 코드 파일로 생성되고 가상 실행환경인 JRE(Java Runtime Environment), CLI(Common Language Infrastructure)에서 한 줄씩 실행하는 방식으로 빌드 된다. JRE, CLI 환 경에서 실행될 때 기계어로 변환되며, 컴파일 언어에 비해 빌드 과정이 빠르다



[그림 1-2] Byte Code 언어의 빌드 과정

3. **인터프리터 언어(Javascript, Python, Ruby 등)**

   인터프리터 언어는 컴파일 언어와 다르게 한 줄씩 번역되어 실행된다. 인터프리터 언어는 컴파일 하는 과정에서 메모리가 훨씬 적게 소모되고 빠른 시간에 컴파일을 진행할 수 있다



[그림 1-3] 인터프리터 언어의 빌드 과정

② 애플리케이션 배포(Release) 환경

애플리케이션 배포는 개발자 또는 사용자가 애플리케이션을 실행, 테스트할 수 있도록 컴파일된 프로그램, 실행에 필요한 리소스(이미지, 환경 설정 파일 등)를 서버상의 적합한 위치로 이동하는 작업을 말한다. 그러므로 애플리케이션 배포는 해당 애플리케이션의 동작환경의 영향을 받는다. 웹 애플리케이션의 경우 웹 서버, WAS(Web Application Server)를 동작 환경으로 사용한다.

1. **웹서버(Web Server)**

   사용자의 http 요청을 받아 웹 컨테이터에 요청을 전달하고 결괏값을 받아와 사용자에게 전송하는 역할을 수행한다. 애플리케이션 배포 시 이미지, 단순 HTML과 같은 정적인 리소스를 배포하며, 정적 리소스를 빠르고 안정적으로 처리한다.

2. **WAS(Web Application Server)**

   사용자의 요청을 받아 동적인 처리를 수행하는 프로그램 실행 부분을 배포한다. WAS 구성 및 운용을 위한 국제 표준 규격을 정하고 있으며, 제품마다 배포 방식과 설정이 일부 다르다. 웹 애플리케이션의 경우 UI 배포 영역(JSP, Servlet 등)과 Biz. 배포 영역(EJB, POJO 서비스 등)으로 구분되어 있다.

③ 애플리케이션 배포 단위

애플리케이션은 배포 시 단순히 컴파일된 실행 파일 또는 Byte Code를 복사하는 방식 이외에 다양한 단위로 묶음, 패키징을 통해 배포할 수 있다. Java의 경우 jar, war, ear등으로 패키징하여 배포할 수 있다.

1. **jar(Java Archive)**

   Java 라이브러리, 리소스, property 파일들을 포함한다. 프로그램에서 참조하는 라이브러리, 구현된 비즈니스 서비스를 배포할 때 jar 단위로 패키징하여 배포한다.

2. **war(Web Archive)**

   웹 컨테이너에 배포되는 배포 형식으로 Servlet, jar 파일과 WEB-INF 폴더에 있는 web.xml 파일로 구성된다. 웹 컨테이너상에 배포되어 독립적인 UI단 웹 애플리케이션 서비스를 제공할 수 있다.

3. **ear(Enterprise Archive)**

   jar와 war을 묶어서 하나의 완성된 웹 애플리케이션 서비스를 제공할 수 있다.

④ 형상관리(Configuration Management) 시스템

형상관리시스템은 서비스 제공 대상 형상항목을 식별하여 기준선(Baseline)을 설정하고, 형상항목 변경 과정에서 점검, 검증 등의 체계적인 통제를 통해 형상항목 간의 일관성과 추적성을 확보하기 위한 시스템이다. 형상관리시스템은 소프트웨어의 개발 및 운영/유지, 보수에 필요한 문서 관리, 변경 관리, 버전관리, 배포관리 및 작업 산출물에 대한 형상관리를 포함한다.

1. 형상관리의 범위

형상관리 범위는 신규 프로젝트 및 보완 개발, 업무시스템의 운영/유지·보수, 전산 설비 및 시스템 소프트웨어 등과 관련된 작업, 사용자(EUC: End User Computing) 6 파일 관리 등을 포함한다

2. 형상관리 시스템에서 사용하는 용어

(1) 형상관리(Configuration Management)

소프트웨어의 전체 생명 주기, 즉 계획부터 개발, 운영, 유지·보수, 폐기까지 발생하는 모든 활동을 지속적으로 관리하는 것을 의미한다. 이를 위해서는 형상항목들에 대해 식별 표시를 붙여 추적이 가능하게 만들고, 형상항목들에 대한 변경을 제어하고 관리하는 프로세스가 필요하다.

(2) 형상항목(Configuration Item)

형상관리 대상이 되는 항목을 의미하며, 유일한 식별자가 부여되어 개별적으로 관리되는 소스 파일, 문서, 기타 사항 등이 포함된다.

(3) 기준선(Baseline)

공식적으로 검토되고 협의되어 향후 기준이 되는 형상항목의 집합체를 의미한다.

(4) 마이그레이션(Migration)

개발 완료된 시스템이 운영 단계로 전환될 때 관련 소스 파일을 저장 공간(리포지터리)으로 이관시키는 작업을 의미한다.

(5) 리포지터리(Repository)

관리 대상을 형상관리 시스템으로 일괄 전송하여 압축, 암호화한 후에 저장, 관리하는 저장 공간을 의미한다. 일반적으로 업무 또는 디렉터리 단위로 구성된다.

(6) 워크플로(Workflow)

형상관리 활동을 수행하기 위해 미리 정해진 절차가 형상관리 시스템 안에 구현되어 있는 것.

(7) 반출(Check Out)

형상항목을 변경하기 위해 형상 리포지터리로부터 전송받는 것을 의미한다. 반출된 형상항목에 대해서는 잠금 상태가 유지된다.

(8) 반입(Check In)

반출된 형상항목을 변경 후 다시 형상 리포지터리로 전송하는 것을 의미한다. 반입 시 버전관리는 자동적으로 이루어진다.

2. 애플리케이션 소스 검증하기

2 - 1 애플리케이션 소스 검증

① 소스코드 검증도구

1. 검증도구의 구분

소스코드 검증도구는 구현된 SW를 실행하지 않고 테스트하는 정적 테스트 도구와 구현된 SW를 실행하여 동작을 보면서 테스트하는 동적 테스트 도구로 구분한다.

[그림 2-1] 소스코드 검증도구 구분

2. 소스코드 검증도구의 용도

(1) 정적 테스트 도구 사용 목적

정적 테스트 도구는 테스트하기 전에 코딩 오류, 성능 저하, 보안 취약점 등의 결 함을 조기에 발견할 수 있도록 지원한다. 이렇게 함으로써 개발의 생산성을 향상시키고, 운영환경에서 프로그램의 품질 향상을 제고하며, 정량적인 품질 관리시스템을 구축하게 한다

(2) 동적 테스트 도구 사용 목적

테스트 미수행 코드를 확인하고 분기(결정)문 등 특정 유형의 코드 구조가 충분히 테스트 되었는지를 확인하여 추가적인 테스트를 진행함으로써 애플리케이션의 안정성을 제고하고, 소스 품질 관리(통제) 활동을 할 수 있는 정량적인 품질 관리시스템을 구축할 수 있게 한다

② 소스코드 검증도구

코드 인스펙션은 정적 테스트의 가장 일반적인 유형으로, 사전에 정의된 코드 작성 규칙(Rule) 기반으로 소스코드를 점검하여 작성 규칙에 위반되는 소스코드를 추출하는 분석 도구로 애플리케이션 개발 시 대부분 사용되며, 빌드 도구와 연계하여 빌드·배포 수행 시 자동적으로 점검할 수 있다

1. 코드 인스펙션 Rule 유형

(1) 성능 개선

애플리케이션의 성능에 영향을 미칠 수 있는 코드를 점검하는 Rule이다. 메모리 누수, 미사용 변수, 메소드 여부 등을 확인하여 메모리를 낭비하는 코드를 식별한다.

(2) 코드 작성 규칙

개발언어에서 사전에 정의된 작성 규칙 또는 프로젝트 내에서 정의된 프로그램 명명규칙의 준수

여부를 점검하는 Rule. 작성 규칙을 미준수한 코드 내역을 추출하여 소스코드의 가독성을 향상시킴.

(3) 에러 발생 가능성

애플리케이션 동작 중 에러 발생 가능성이 있는 코드를 점검하는 Rule이다

2. 코드 작성 Rule 심각도 구분(예시)

(1) 필수, Blocker

애플리케이션의 성능에 영향을 미칠 수 있는 코드를 점검하는 Rule이다. 메모리 누수, 미사용 변수, 메소드 여부 등을 확인하여 메모리를 낭비하는 코드를 식별한다.

(2) 권고 상, Critical

개발언어에서 사전에 정의된 작성 규칙 또는 프로젝트 내에서 정의된 프로그램 명명규칙의 준수

여부를 점검하는 Rule. 작성 규칙을 미준수한 코드 내역을 추출하여 소스코드의 가독성을 향상시킴.

(3) 권고 중, Major

애플리케이션 동작 중 에러 발생 가능성이 있는 코드를 점검하는 Rule이다

(4) 권고 하, Minor

애플리케이션 동작 중 에러 발생 가능성이 있는 코드를 점검하는 Rule이다

(5) 정보, Info

애플리케이션 동작 중 에러 발생 가능성이 있는 코드를 점검하는 Rule이다

3. 정규표현식(Regular Expression)

정규 표현식은 특정한 규칙을 가진 문자열의 집합을 표현하는 범용적인 방식을 말한다. 코드 인스펙션 도구의 코드 작성 규칙은 일반적으로 정규식으로 표현되며, 정규식의 내용을 수정해서 점검 Rule의 내용을 수정할 수 있다.

③ 테스트 프레임워크(동적 분석 도구)의 구성

테스트 프레임워크는 테스트 케이스를 별도의 테스트 코드로 작성하고 동작시킬 수 있는 환경을 제공하는 도구로, 개발자의 반복적이고 시간이 많이 소요되는 테스트 작업을 자동화하여 테스트에 소요되는 시간과 노력을 절감할 수 있게 한다

1. 테스트 프레임워크의 구성

(1) 테스트 코드

테스트 코드 작성 및 자동화된 운영환경을 구성한다. 빌드 도구와 연계하여 빌드 수행 시 테스트 코드를 동작시켜 자동화된 테스트 환경을 제공한다.

(2) 테스트 저장소

테스트 수행을 위한 테스트 코드, 테스트 데이터, 관련 테스트 스크립트, 테스트 수행 결과를 저장, 관리한다

2. Junit 테스트 프레임워크

Java, 오픈소스 기반의 테스트 프레임워크로 Java 개발환경의 범용적인 표준으로 사용되며(Eclipse 개발 도구에 기본 내장 기능), 다음과 같이 테스트 가능한 Assert 함수를 제공한다.

〈표 2-2〉 Junit Assert 구문

제공 함수	설 명
assertArrayEquals(a,b)	배열 a와 b가 일치함을 확인
assertEquals(a,b)	객체 a와 b가 일치함을 확인
assertSame(a,b)	객체 a와 b가 같은 객체임을 확인
assertTrue(a)	a값이 참인지 확인
assertNotNull(a)	객체 a가 null이 아님을 확인

3. 애플리케이션 빌드하기

3 - 1 애플리케이션 빌드

① 지속적인 통합(CI: Continuous Integration) 환경

애플리케이션 개발 과정 중 지속적으로 개발된 프로그램을 통합, 빌드, 배포하여 애플리케이션의 개발 내역을 검증, 테스트할 수 있는 환경을 말한다. 통합 빌드 과정 중 테스트 도구, 소스코드 품질 측정도구 등과 연계할 수 있으며, 자동화된 스케줄 관리를 통해서 지속적이고 반복적인 프로그램 빌드, 테스트를 진행할 수 있다.

1. 빌드 도구

애플리케이션의 배포 단위, 형식에 따라 소스코드를 컴파일, 패키징하며, 배포하는 스크립트를 제공하고 수행하는 도구이다(Ant, Maven 등)

2. 테스트 도구

개발된 소스코드를 테스트할 수 있는 테스트 코드를 작성, 동작 시킬 수 있는 도구로, 통합빌드 수행 시 연결할 수 있다(Junit, DBUnit, StrutsTestCase 등).

3. 소스코드 품질 측정도구(코드 인스펙션)

정해진 소스코드 작성 규칙에 따라 소스코드를 점검하고 규칙 위반 여부를 체크하는 도구로, 통합 빌드 수행 시 연결할 수 있다(PMD, FindBugs 등).

4. 테스트 커버리지 측정도구

소스코드 내 테스트 가능한 경로 중 테스트 도구를 통해서 테스트된 커버리지를 측정하는 도구이다 (Clover, JCoverage, ElcEmma 등).

5. 빌드 스케줄 관리도구

작성된 빌드 스크립트를 정해진 조건, 시간에 기동하고 진행 상태, 수행 결과를 관리하는 도구이다 (Anthill, CruiseControl, Hudson 등).

② 테스트 커버리지

1. 테스트 커버리지의 의미

테스트 커버리지는 전체 프로그램의 범위 대비 테스트 수행 시 해당 테스트 수행을 위해 동작된 프로그램의 범위 비율을 의미한다.

2. 테스트 커버리지 축적 유형

(1) 라인 커버리지(또는 구문 커버리지, Statement Coverage)

개발 소스의 각 라인이 수행되었는지를 확인하는 측정 지표이다.

(2) 분기 커버리지(Decision Coverage)

개발소스의 각 분기문이 수행되었는지를 확인하는 측정 지표이다. 만약 소스 내에 if문에 대한 true/false 조건이 있다면, 두 가지 경우가 모두 테스트되어야 100%로 측정된다.

(3) 조건 커버리지(Condition Coverage)

각 분기문 내에 존재하는 조건식이 모두 테스트되었는지를 확인하는 측정 지표이다.
조건식 간의 조합에 대해서는 체크하지 않는다.

③ 빌드 스케줄 관리도구

1. 빌드 스케줄 관리도구의 구성

빌드 스케줄 관리도구는 별도의 웹 애플리케이션으로 구성되어 웹 서버상에 배포되고, 관리자 화면을 통해서 빌드 스크립트, 형상관리 도구 등과 연계되며, 이메일을 통해서 관련 개발자, 관리자들에게 빌드 수행 결과를 제공한다.

2. 빌드 스케줄 관리도구의 기능

(1) 빌드 작업 스케줄링

빌드 작업의 작업 주기, 작업 시간을 설정한다

(2) 빌드 작업 상태 및 이력 관리

등록된 빌드 작업의 진행 상태를 대기 중, 진행 중, 완료, 오류 유형으로 구분하여 관리하고 수행 이력을 관리한다.

(3) 빌드 도구 연계 관리

빌드 작업 수행을 위해 기작성된 빌드 스크립트와 접근 가능한 형상관리 도구를 연계 설정한다.

(4) 빌드 수행 결과 리포팅

빌드 수행 결과 오류 사항, 코드 인스펙션 점검 결과, 테스트 수행 결과 등을 웹 화면을 통해서 보여 주고, 관련자들이 공유할 수 있도록 리포팅을 제공한다.

4. 애플리케이션 배포하기

4 - 1 애플리케이션 배포

① 운영환경의 특징

기업의 IT 인프라 운영환경은 안정적인 IT 서비스 운영 관리를 위해서 애플리케이션 배포 및 변경 작업에 대한 관리, 통제를 강화하고 있으며, 여러 가지 제약 사항을 가지고 있다.

1. 네트워크 관점

운영환경은 일반적으로 인터넷망과 분리되어 직접적인 연결을 허용하지 않고 있으며, 별도의 내부망을 구성하고 방화벽을 통해서 인가된 IP를 통해서만 접근을 허용한다. 기업의 운영환경, 서버에 접근하기 위해서는 사전에 방화벽 허용 신청을 통해 서 배포 서버의 IP가 접근 가능하도록 설정되어야 한다.

2. 계정 관리 부문

기업의 운영환경은 접근 계정을 별도 IP 관리시스템을 통해서 관리하고 있으며, 정부의 보안 관리 기준에 따라 정기적으로 패스워드(password)를 변경하는 형태로 관리된다. 그러므로 애플리케이션 배포 전에 운영 서버에 접근 가능한 계정, 프로토콜, 접근 디렉터리를 정해서 신청해야하며, 아이디, 패스워드가 소스코드 또는 빌드 스크립트상에 하드 코딩되지 않도록 주의해야 한다.

3. 보안 취약점 부문

해킹 위협의 증가로 기업의 운영환경은 다양한 보안 취약점 점검을 주기적으로 수행하고 보안 취약점을 보완한다. 배포 시 이러한 보안 취약점은 애플리케이션 관점에서도 점검해야 한다.