애플리케이션 테스트 및 성능 개선

Junit & SpringTest-MVC & Selenium & Jmeter & 정적분석도구

- ❑ **웹기반어플리케이션 → 애플리케이션 테스트 수행**
- ❑ **애플리케이션 통합 → 애플리케이션 테스트 관리**

– 참고 사이트

- https://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev
- https://pmd.github.io/
- http://emma.sourceforge.net/
- http://www.eclemma.org/
- http://www.eclemma.org/jacoco/index.html
- http://metrics2.sourceforge.net/
- http://www.jidum.com/jidums/view.do?jidumKindCd=Te
- http://pseg.or.kr/pseg/
- https://junit.org/junit5/docs/current/user-guide/
- https://code.google.com/archive/p/hamcrest/wikis/Tutorial.wiki

대덕인재개발원 : 최희연

❑ 애플리케이션 테스트 개요

– 개발된 애플리케이션이나 시스템이 사용자가 요구하는 기능의 동작과 성능, 사용성, 안정성 등을 만족하는지 확인하기 위하여 소프트웨어의 결함을 찾아내는 활동.

1. 애플리케이션 테스트 원리
   ① 테스팅은 결함이 존재함을 밝히는 활동이다.
   ② 완벽한 테스팅은 불가능하다.
   ③ 테스팅은 개발 초기에 시작해야 한다.
   ④ 결함의 대부분은 소수의 특정 모듈에 집중되어있다(결함집중).
   ⑤ 동일한 테스트 케이스로 반복 실행하면 결함을 발견할 수 없으므로 주기적으로 테스트 케이스를 개선해야 한다
   (살충제 패러독스).
   ⑥ 테스팅은 Context(정황)에 의존한다.
   ⑦ 사용자의 요구사항을 만족하지 못하는 오류를 발견하고 그 오류를 제거하더라도 해당 애플리케이션의 품질이 높다고 말할 수는 없다(오류-부재의 궤변).
2. 테스트 프로세스와 산출물

   ① 테스트 계획서 : 테스트 목적과 범위 정의, 대상 시스템 구조 파악, 테스트 수행 절차, 테스트 일정, 조직의 역할 및 책임 정의 종료 조건 정의 등 테스트 수행을 계획한 문서.
   ② 테스트 케이스 : 테스트를 위한 설계 산출물로, 응용 소프트웨어가 사용자의 요구사항을 준수하는지 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과로 구성된 테스트 항목의 명세서.
   ③ 테스트 시나리오 : 테스트 수행을 위한 여러 개의 테스트 케이스의 집합으로 테스트 케이스의 동작 순서를 기술한 문서이며, 테스트를 위한 절차를 명세한 문서.
   ④ 테스트 결과서 : 테스트 결과를 정리한 문서로 테스트 프로세스를 리뷰하고, 테스트 결과를 평가하고 리포팅하는 문서.
3. 테스트 유형
   1) 프로그램 실행 여부
      ① 정적 테스트 : 프로그램 실행 없이 소스 코드의 구조를 분석하고 논리적으로 검증하는 테스트로 인스펙션, 코드 검사, 워크스루 등이 있다.
      ② 동적 테스트 : 프로그램의 실행을 요구하는 테스트로 화이트박스 테스트와 블랙박스 테스트가 있다.
         a. 화이트박스 테스트 : 프로그램 내부 로직을 보면서 테스트를 수행한다.
         b. 블랙박스 테스트 : 프로그램 외부 사용자 요구사항 명세를 보면서 테스트하며, 주로 구현된 기능을 테스트 한다.
   2) 테스트 목적
      ① 회복(Recovery) 테스트 : 시스템에 고의적 실패를 유도하고, 정상적으로 복귀하는지 테스트
      ② 안전(Security) 테스트 : 소프트웨어의 보안적인 결함을 미리 점검하는 테스트

③ 강도(Stress) 테스트 : 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 검증하는 테스트

④ 성능(Performance) 테스트 : 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간 내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트

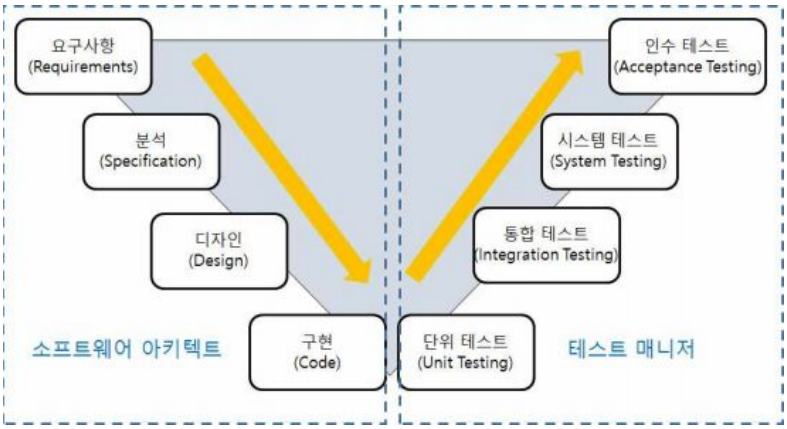
⑤ 구조(Structure) 테스트 : 시스템의 내부 논리 경로, 소스코드의 복잡도를 평가하는 테스트

⑥ 회귀(Regression) 테스트 : 변경 또는 수정된 코드에 대하여 새로운 결함 발견 여부를 평가하는 테스트

⑦ 병행(Parallel) 테스트 : 변경 시스템과 기존 시스템에 동일한 데이터를 입력 후 결과를 비교하는 테스트

❑ 애플리케이션 테스트 수행

1. 프로젝트 수행 단계에 따른 테스트의 접근 방법



1) 단위 테스트 : 테스트 가능한 단위(컴포넌트, 모듈)로 분리된 소프트웨어 내에서 결함을 찾고 기능을 검증하는 테스트 활동

① 구조적 테스트, 기능성 테스트, 리소스 테스트, 강도 테스트 등 특정 비기능성 테스트 등이 포함되어 수행되며, 컴포넌트 명세, 소프트웨어 상세설계, 데이터 모델 명세 등을 이용하여 테스트

② 단위 테스트 방법

a. 구조 기반 테스트 : 소프트웨어 내부 논리 흐름에 따라 테스트 케이스를 작성하여 프로그램 내부 구조 및 복잡도를 검증하는 테스트 (화이트박스 테스트)

b. 명세 기반 테스트 : 목적 및 실행 코드 기반의 실행을 통해 주어진 명세를 빠짐없이 테스트 케이스로 구현하고 있는 지 확인하는 테스트(블랙박스 테스트)

2) 통합 테스트 : 모듈 사이의 인터페이스, 통합된 컴포넌트 간의 상호 연동이 정상적으로 작동하는지 여부, 즉 운영체제, 파일시스템, 하드웨어 또는 시스템간 인터페이스 와 같은 각각 다른 부분과 상호연동이 정상적으로 작동하는지 여부를 테스트

① 특징 : 빅뱅 방식보다는 순차적 형태와 아키텍처에 대한 이해를 바탕으로 진행됨.

② 테스트 방법.

구분	내용
하향식 테스트	- 주요 제어 모듈을 테스트 드라이버로 활용하고, 주요 제어 모듈에 직접 종속되는 모듈은 스텁으로 활용하여, 상위 모듈로부터 시작된 테스트가 하위 모듈을 점진적으로 포함하며 진행
상향식 테스트	- 최종단의 구현 모듈의 응집도를 기초로 클러스터링 진행 및 단위 클러스터의 테스트 드라이버를 생성 및 테스트 수행 -하위 클러스터로부터 시작된 테스트가 상위의 클러스터를 점진적으로 포함하며 진행
Big Bang	- 모든 모듈을 동시에 통합하여 테스트하며 단시간에 테스트가 종료 가능. - 식별된 결함의 격리가 어려움
혼합식 통합 테스트	- 통합을 시도할 모듈을 선정하고, 해당 모듈을 기준으로 하향식 또는 상향식 테스트 수행

3) 시스템 테스트 : 통합된 단위 시스템의 기능이 정상적으로 수행되는지, 성능 및 장애를 테스트

① 특징 : 개발 프로젝트 차원에서 정의된 전체 시스템의 동작과 관련이 있으며, 환경(Context) 제한적 장애 관련 리스크를 최소화하기 위해 실제 환경과 유사하게 시스템 성능, 관련된 고객의 기능/비기능적 요구사항 등이 완벽하게 수행되는 지를 테스트하며, 요구사항 명세서, 비즈니스 절차, 유스케이스, 리스크 분석 결과 등을 이용함.

② 시스템 테스트 방법

a. 기능적 요구사항 : 요구사항 명세서, 비즈니스 절차, 유스케이스 등 명세서 기반의 블랙박스

b. 비기능적 요구사항 : 성능 테스트, 회복 테스트, 보안 테스트, 내부 시스템의 메뉴 구조, 웹 페이지의 네비게이션 등의 구조적 요소에 대한 화이트박스 테스트

4) 인수 테스트 : 최종 사용자와 업무에 따른 이해관계자 등이 테스트를 수행함으로써 개발된 제품에 대해 운영 여부를 결정하는 테스트.

테스트 종류	테스트 내용
사용자 인수 테스트	비즈니스 사용자가 시스템 사용의 적절성 여부를 확인
운영상의 인수 테스트	시스템 관리자가 시스템 인수 시 수행하는 테스트 활동으로 백업/복원 시스템, 재난 복구, 사용자 관리, 정기 점검 등을 확인
계약 인수 테스트	계약상의 인수/검수 조건을 준수하는지 여부를 확인
규정 인수 테스트	정부 지침, 법규, 규정 등 규정에 맞게 개발하였는지 확인
알파 테스트	조직 내 잠재 고객에 의해 테스트가 수행되나, 개발 현장에서 통제된 상태에서 수행됨.
베타 테스트	최종사용자에 의해 실제 서비스 환경에서 수행되는 테스트

2. 단위 테스트 이해

1) 테스트 목적

① 소스 코드의 변형 및 관리를 통해 테스트 대상 모듈의 정상적인 동작 여부 확인

② 요구사항 대상의 기능/비 기능 충족 여부 체크

③ 논리적 설계 구현의 검증과 개발 목적의 적합 여부 검증

④ 구현 모듈의 불확실성에 입각한 모듈의 점검 및 이슈 제가

⑤ 단위 테스트 커버리지를 통해 단위 테스트 완성도 측정

⑥ 애플리케이션 품질에 대한 신뢰성 부여

2) 테스트 동적 설계 기법

명세 기반 기법	구조 기반 기법	경험 기반 기법
- 등가분할 - 분류 트리 기법 - 경계값 분석 - 상태 전이 테스트 - 결정 테이블 테스트 - 원인-결과 분석 - 조합 테스트 기법 - 시나리오 테스트 - 오류 추정	- 구문 테스트 - 결정 테스트 - 조건 테스트 - 데이터 흐름 테스트	- 구문 테스트 - 결정 테스트 - 조건 테스트 - 데이터 흐름 테스트

3) 테스트 커버리지 : 테스트 대상의 전체 범위에서의 테스트 수행 정도를 의미하는 실행 기반 테스트의 화이트박스 테스트이며, 구조 기반 테스트 기법이 적용된다. 테스트 수행을 통해 테스트 볼륨의 크기와 테스트의 정확성을 판단하는 척도이며, 일반적으로 80% 이상을 성공적 커버리지 비율로 판단한다.

① Statement Coverage : 소스 코드의 모든 코드 라인 대상의 테스트 수행 정도(Line/구문 커버리지)
ex) if(){}else if(){}else 중 어느 구문이든 한번 수행되면 커버리지 카운트

② Decision Coverage : 소스 코드 내 전체 조건 분기문 대상의 테스트 수행 정도(Branch/결정 커버리지)
ex) if(전체 조건문){}else{} 에서 전체 조건문 내 세부 조건들의 참/거짓 여부에 관계없이 if 구문 전체의 true 또는 false 결과에 대해 각각 테스트 수행 시 커버리지 카운트

③ Condition Coverage : 소스 코드 내 전체 조건 분기문 내 세부 조건들이 true 또는 false 조건에 대해 각각 테스트 수행 정도(Term/조건 커버리지)
ex) if(세부조건 and|or 세부조건){}else{} 에서 각각의 개별 세부 조건문들을 대상으로 테스트 수행 정도, if 구문 전체의 true false 결과와 무관함.

④ MC/DC Coverage : Decision Coverage 와 Condition Coverage가 병합된 커버리지 (Modified/Decision 커버리지)

3. 테스트 도구

1) 테스트 하네스(Harness)
: 애플리케이션 컴포넌트 및 모듈을 테스트 하는 환경의 일부분으로, 테스트하기 위한 코드와 데이터를 말하며, 단위 또는 모듈 테스트에 사용하기 위해 코드 개발자가 작성한다.

① 테스트 드라이버 : 테스트 대상 하위 모듈을 호출하고, 파라미터를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 등 상향식 테스트에 필요.

② 테스트 더블 : 테스트 대상 객체의 역할을 대신하는 객체

a. 테스트 스텁 : 제어 모듈이 호출하는 타 모듈의 기능을 단순 수행하는 도구로 하향식 테스트에 필요

b. Mock 객체 : 사용자의 행위를 사전에 조건부 입력해두고, 그 상황에 예정된 행위를 수행하는 객체

③ 테스트 수트 : 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합.

④ 테스트 케이스 : 명세 기반 테스트의 설계 산출물로, 특정한 프로그램의 일부분 또는 경로에 따라 수행하거나, 특정한 요구사항을 준수하는지 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과로 구성된 테스트 항목의 명세서를 말한다. 정확성, 재사용성, 간결성 보장을 위해 다음 절차에 따라 작성한다.

a. 테스트 계획 검토 및 자료 확보

b. 위험 평가 및 우선순위 결정

c. 테스트 요구사항 정의

d. 테스트 구조 설계 및 테스트 방법 결정

e. 테스트 케이스 정의

f. 테스트 케이스 타당성 확인 및 유지보수

⑤ 테스트 스크립트 : 자동화된 테스트 실행 절차에 대한 명세

2) 테스트 오라클 : 테스트 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참 값을 입력하여 비교하는 기법 및 활동을 말한다.

① 테스트 오라클 유형

a. 참(true) 오라클 : 모든 입력 값에 대하여 기대하는 결과를 생성함으로써 발생된 오류를 모두 검출할 수 있는 오라클

b. 샘플링 오라클 : 특정한 몇 개의 입력 값에 대해서만 기대하는 결과를 제공해 주는 오라클

c. 휴리스틱 오라클 : 샘플링 오라클을 개선한 오라클로, 특정 입력 값에 대해 올바른 결과를 제공하고, 나머지 값들에 대해서는 휴리스틱(추정)으로 처리하는 오라클

d. 일관성 검사 오라클 : 애플리케이션 변경이 있을 때, 수행 전과 후의 결과 값이 동일한지 확인하는 오라클

② 오라클 적용 방안 : 참 오라클은 주로 항공기, 임베디드, 발전소 등 미션 크리티컬한 업무에 적용하고, 샘플링/추정 오라클은 일반, 업무용, 게임, 오락 등의 업무에 적용한다.

4. 공개 SW 기반의 테스트 도구(참고 : 소프트웨어 공학포털)

1) 정적 분석 도구

분류	제품명	세부정보	
결함 예방 식별 관리	PMD(CPD)	개요	자바 코드 대상 결함 패턴 식별 도구
		지원환경	Windows, Linux
		IDE	Eclipse, NetBeans, IntelliJ 등
		사이트	https://pmd.github.jo
	FindBugs	개요	자바 바이트 코드 대상 결함 패턴 식별 도구
		지원환경	Cross-Platform
		IDE	Eclipse, NetBeans, IntelliJ 등
		사이트	http://findbugs.sourceforge.net/
	Sonar Qube	개요	지속적인 소스 코드 품질 검사 수행 통합 플랫폼
		지원환경	Cross-Platform
		IDE	Eclipse, IntelliJ
		사이트	http://www.sonarqube.org/
코딩표준	CheckStyle	개요	자바 코드 대상 코딩룰 준수 여부 식별
		지원환경	Cross-Platform
		IDE	Ant, Eclipse, IntelliJ, NetBeans 등
		사이트	http://checkstyle.sourceforge.net/
코드 복잡도	Eclipse Metrics	개요	소스 코드의 복잡도 분석 및 통계 정보 제공
		지원환경	Cross-Platform
		IDE	Eclipse
		사이트	https://sourceforge.net/projects/metrics/

2) 동적 분석 도구

제품명	세부정보	
VisualVM	개요	JVM 내 메모리, 스레드 현황 및 CPU 사용률, GC, 로그 식별
	지원환경	Cross-Platform
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	https://visualvm.java.net/
JVM Monitor	개요	모듈별 JVM 내 메모리, 스레드 현황 및 CPU 사용률
	지원환경	Cross-Platform
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://jvmmonitor.org/

3) 테스트 커버리지 도구

제품명	세부정보	
Emma	개요	Jacoco 베이스의 코드 커버리지(Branch coverage 미제공)
	지원환경	Cross-Platform
	IDE	Eclipse, NetBeans, StandAlone 등
	사이트	http://emma.sourceforge.net/, http://www.eclemma.org
JaCoCo	개요	코드 커버리지 및 리포팅
	지원환경	Windows, Linux
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://cobertura.sourceforge.net

4) 테스트 프레임워크

제품명	세부정보	
Xunit	개요	Junit 을 활용한 자바 단위 테스트 프레임워크
	지원환경	Platform 별 지원 상이
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://junit.org/junit4/
Hamcrest	개요	다양한 테스트 프레임웤을 지원하는 Matcher
	지원환경	Platform 별 지원 상이
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://hamcrest.org/
EsayMock	개요	모듈간 종속성 해소를 위해 테스트 더블을 이용한 테스트 테스트 더블 : 단위 테스트 수행시 의존성을 가진 활용 불가능한 모듈을 대신할 가상의 모듈, Mock, Fake, Stub
	지원환경	Platform 별 지원 상이
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	Http://easymock.org/
Mockito	개요	모듈 간 종속성 해소를 위해 테스트 더블을 이용한 테스트
	지원환경	Platform 별 지원 상이
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://mockito.org/
SpringMVC-Test	개요	Spring webMVC 모듈과 연동한 테스트 지원
	지원환경	Cross-Platform
	IDE	Eclipse, NetBeans, IntelliJ 등
	사이트	http://spring.io/
FitNesses	개요	웹 기반 테스트 케이스 설계/실행/결과 등을 지원
	지원환경	Cross-Platform
	IDE	Eclipse
	사이트	http://fitness.org/

5) 성능 테스트 도구

제품명	세부정보	
JMeter	개요	Http, Ftp 등의 다양한 프로토콜 지원 서버 부하 테스트
	지원환경	Cross-Platform
	IDE	Eclipse, Ant, IntelliJ 등
	사이트	Http://jmeter.apache.org/
LoadUI	개요	Http, JDBC 등의 다양한 프로토콜 지원 서버 모니터링, 사용자 편의성이 강화된 테스트 도구
	지원환경	Cross-Platform
	IDE	-
	사이트	http://www.loadui.org/

6) 시스템 모니터링 도구

제품명	세부정보	
Nagios	개요	웹 기반 서버, 서비스, 애플리케이션의 모니터링 도구
	지원환경	Cross-Platform
	사이트	http://www.nagios.org/
Zabbix	개요	웹 기반 서버, 서비스, 애플리케이션의 모니터링 도구
	지원환경	Cross-Platform
	사이트	http://www.zabbix.com/

7) 테스트 케이스(수트) 관리 도구

제품명	세부정보	
TestLink	개요	요구사항 기반의 테스트 케이스와 테스트 수트의 작성 관리 및 테스트 수행 계획과 테스트 결과 관리 및 리포팅 지원
	지원환경	Cross-Platform
	사이트	http://testlink.org/
Impasse	개요	Testlink를 레드마인 에서 사용할 수 있도록 개발된 도구
	지원환경	Cross-Platform
	사이트	http://www.redmine.org/plugins/impasse/

8) 지속적 통합 테스트 관리 도구

제품명	세부정보	
Hudson	개요	다양한 플러그인을 활용하여 개발 산출물 통합 및 테스트 자동화에 활용
	지원환경	Cross-Platform
	사이트	http://hudson-ci.org/
Jenkins	개요	다양한 플러그인을 활용하여 개발 산출물 통합 및 테스트 자동화에 활용
	지원환경	Cross-Platform
	사이트	http://jenkins.io/

❑ 테스트 프레임워크 활용

1. Junit & Hamcrest & Spring MVC-Test 를 이용한 단위 테스트 및 Mock 객체 활용과 Builder 패턴의 활용

1) Junit 과 Hamcrest 를 이용한 테스트 케이스 작성

```
<dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
</dependency>
<dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-all</artifactId>
        <version>1.3</version>
        <scope>test</scope>
</dependency>
```

```
@BeforeClass
public static void setUpBeforeClass() throws Exception {
        // 테스트 클래스의 테스트 수행을 위한 초기화 로직 구성 및 최초 1회 Call Back.
}
@AfterClass
public static void tearDownAfterClass() throws Exception{
        // 테스트 클래스의 테스트 종료를 위한 자원 반납 로직 구성 및  최초 1회 Call Back.
}
@Before
public void setUp() throws Exception {
        // 개별 테스트 메서드의 테스트 수행 전 반복 Call Back.
}
@After
public void tearDown() throws Exception{
        // 개별 테스트 메서드의 테스트 수행 후 반복 Call Back.
}
@Test(
        timeout=1000, // 수행 시간 제한(ms)
        expected=Exception.class // 발생 예측 가능한 예외 타입 (예외 발생시 테스트 성공)
)
public void testPlus() {
        // 테스트 수행 코드 1. Asserts
        assertEquals(expected, actual);  // 기대값과 실제 값이 같은지 검증
        assertNotEquals(unexpected, actual); //  기대값과 실제 값이 같지 않은지 검증
        assertSame(expected, actual); // 동일 객체 인지 검증
        assertNotSame(unexpected, actual); // 서로 다른 객체인지 검증
        assertNull(expected); // Null 검증
        assertNotNull(unexpected); // Not  Null 검증
        assertArrayEquals(expected, actual); // 요소와 순서까지 고려한 배열 비교
        // Assert 메소드 종류 부족을  Matcher
        assertThat(value, matcher); //  Matcher 를 이용한 검증(Hamcrest Matcher 활용)
}
```

**** java7 부터의 static import 구문을 사용하기 위한 이클립스 설정 변경**

- Window -> preferences -> Java -> Editor -> Content Assist -> Favorites -> New Type
  org.hamcrest.Matchers, org.junit.Assert 추가
- Window -> preferences -> Java -> Code Style -> Organize Import
  Number of static imports need for = "1"

- Hamcrest Matcher 의 종류

분류	Matcher	설명
Core	anything	테스트 객체와 관계없이 true
	describedAs	테스트 실패시 설명 추가
	is	가독성을 높이기 위해 추가적으로 사용 가능.
Logical	allOf	모든 Macher 가 매치되는지 판단(short-circuit)
	anyOf	Matcher 중 어느 하나가 매치되는지 여부(short-circuit)
	not	Matcher 에 매칭되지 않는 경우를 확인
Objects	equalTo	테스트 대상 객체들의 상태 비교
	hasToString	toString 메서드 테스트
	instanceOf , isCompatibleType	타입 검증
	notNullValue , nullValue	null 값 검증
	sameInstance	동일 객체인지 식별성 검증
Numbers	closeTo	실수에 대해 주어진 값에 가까운지 검증
	greaterThan , greaterThanOrEqualTo, lessThan, lessThanOrEqualTo	값의 크기 검증
Beans	hasProperty	자바빈의 프로퍼티 검증
Collections	array	배열 엘리먼트들에 대한 순차적 검증
	hasEntry, hasKey , hasValue	맵에 대해 key/value/entry 검증
	hasItem, hasItems	컬렉션 객체에 대해 요소를 포함하고 있는지 검증
	hasItemInArray	배열에 요소가 포함되었는지 검증
Text	equalToIgnoringCase	대소문자 구별없이 문자열 비교
	equalToIgnoringWhiteSpace	공백제거 후 문자열 비교
	containsString, endsWith, startsWith	문자열 매칭 여부 검증

```
<dependency>
        <groupId>org.hamcrest</groupId>
        <artifactId>hamcrest-all</artifactId>
        <version>1.3</version>
        <scope>test</scope>
 </dependency>
```

```
// 테스트 객체의 타입 검증
assertThat(testObj1, instanceOf(List.class));
// 객체의 상태 검증
assertThat(testObj1, equalTo(testObj2));
// 동일 객체 여부 검증
assertThat(testObj1, sameInstance(testObj2));
// 테스트 객체의 부모 타입 검증
assertThat(String.class, typeCompatibleWith(CharSequence.class));
// 객체 검증 실패시 메시지 추가
assertThat(testObj1, describedAs("두객체는 다른 객체임", sameInstance(testObj2)) );
```

```
// toString 반환값 검증
assertThat(testObj1, hasToString(not(isEmptyOrNullString())));
// 객체의 프로퍼티 존재 검증
assertThat(testObj1, hasProperty("anyProperty"));
// 배열의 요소 검증(순서대로 모든 matcher 들을 만족하는지 검증)
assertThat(testArray, array(equalTo("element1"), containsString("element2")));
// 배열이 임의의 요소를 가지고 있는지 검증
assertThat(testArray, hasItemInArray("element") );
//  컬렉션이 임의의 요소를 가지고 있는지 검증
assertThat(testList, hasItem("listElement"));
// 대소문자 구분없이 문자열 비교
assertThat("text", equalToIgnoringCase("tExT"));
// 앞뒤의 공백을 제거하고 문자열 비교
assertThat("text", equalToIgnoringWhiteSpace(" text "));
// 숫자 크기 검증
assertThat(testNumber, greaterThan(3));
//10<=testNumber<=20,  allOf : 모든 Matcher 들을 만족하는지  검증
assertThat(testNumber, allOf(greaterThanOrEqualTo(10), lessThanOrEqualTo(20)));
//testNumber > 20 or testNumber < 0 ,  anyOf : Matcher 중 어느 하나이상 만족하는지 검증
assertThat(testNumber, anyOf(greaterThan(20), lessThan(0)));
//10<=testNumber<=20 , 해당 Matcher 와 반대되는 경우인지 검증
assertThat(testNumber, not(anyOf(greaterThan(20), lessThan(0))));
```

2) Junit 과 Spring MVC-Test 기반의 Mock 객체를 활용한 테스트 케이스 작성

spring-webMVC 컨텍스트에서 동작하는 컨트롤러에 대한 테스트 케이스 작성 상황을 가정함.

```
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.log;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*;
import static org.hamcrest.Matchers.*;
// junit 에 내장된 Runner 대신 테스트용 컨텍스트를  생성하고 관리하기 위한 별도의 Runner 지정.
@RunWith(SpringJUnit4ClassRunner.class)
//  테스트를 위해 ApplicationContext 를 로딩하기 위한 선언
@WebAppConfiguration // 웹 컨텍스트 형성을 위한 어노테이션
// Bean Configuration 파일을 통해 ApplicationContext 계층 구조를 형성하기 위한 설정
@ContextHierarchy({
        @ContextConfiguration("file:webapp/WEB-INF/spring/*-context.xml"),
        @ContextConfiguration("file:webapp/WEB-INF/spring/appServlet/servlet-context.xml“)
})
// 테스트 케이스 실행시 트랜잭션 관리를 위한 기본 정책을 설정하는 어노테이션
@Transactional
public class BoardRetrieveControllerTest{
        @Inject
        // Child WebApplicationContext 를 주입
        public WebApplicationContext context;
        //Spring MVC test 이용을 위한 entry point  mock 객체로 mock Request 등을 사용할 수 있다.
        public MockMvc mockMvc;
        // 테스트 대상 컨트롤러 주입
        @Inject
        public  BoardRetrieveController controller;
        @Before
        public void setUp(){
                // WebApplicationContext 객체를 전체적으로 초기화 ,Dispatcher Servlet 동작
                mockMvc = MockMvcBuilders.webAppContextSetup(context).build();
                // 일부 컨트롤러들을 대상으로 한 최소한의 초기화
                mockMvc = MockMvcBuilders.webAppContextSetup(context).build();
        }
```

```
	@Test
	public void testList() throws Exception {
		mockMvc.perform( get("/board/boardList.do")
				.param("page", "1")
				.param("searchType", "title")
				.param("searchWord", "은대"))
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("pagingVO"))
				.andExpect(model().attribute("pagingVO", hasProperty("dataList",
							hasSize(greaterThan(5)))))
				.andExpect(view().name("board/boardList"))
				.andDo(log());

		mockMvc.perform( get("/board/boardList.do").param("page", "2") )
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("pagingVO"))
				.andExpect(view().name("board/boardList"))
				.andDo(log());
		mockMvc.perform( get("/board/boardList.do")
				.param("page", "1")
				.param("searchType", "writer")
				.param("searchWord", "은대"))
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("pagingVO"))
				.andExpect(model().attribute("pagingVO", hasProperty("dataList",
					hasSize(greaterThan(5)))))
				.andExpect(view().name("board/boardList"))
				.andDo(log());
		mockMvc.perform( get("/board/boardList.do")
				.param("page", "1")
				.param("searchType", "content")
				.param("searchWord", "은대"))
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("pagingVO"))
				.andExpect(model().attribute("pagingVO", hasProperty("dataList", empty())))
				.andExpect(view().name("board/boardList"))
				.andDo(log());
		mockMvc.perform( get("/board/boardList.do")
				.param("page", "1")
				.param("searchWord", "은대"))
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("pagingVO"))
				.andExpect(model().attribute("pagingVO", hasProperty("dataList",
					hasSize(greaterThan(5)))))
				.andExpect(view().name("board/boardList"))
				.andDo(log());
	}
	@Test
	public void testBoardView() throws Exception{
		mockMvc.perform(get("/board/boardView/96"))
				.andExpect(status().isOk())
				.andExpect(model().attributeExists("board"))
				.andExpect(view().name("board/boardView"))
				.andDo(log());
	}
}
```

3) Json 객체 탐색을 위한 의존성 (json-path)추가.

참고 : https://github.com/json-path/JsonPath/blob/master/README.md | http://jsonpath.herokuapp.com/

```
<dependency>
	<groupId>com.jayway.jsonpath</groupId>
	<artifactId>json-path</artifactId>
	<version>2.4.0</version>
	<scope>test</scope>
</dependency>
```

```
	@Test
	public void testAjaxList(){
		mockMvc.perform(get("/board/boardList.do")
			.accept(MediaType.APPLICATION_JSON_UTF8)
			)
			.andExpect(status().isOk())
			.andExpect(jsonPath("$.dataList").exists())
			.andExpect(jsonPath("$.dataList.length()", greaterThan(3)))
			.andExpect(jsonPath("$.dataList[0].bo_no", greaterThan(100)))
			.andExpect(jsonPath("$.dataList[?(@.bo_hit>0)]").exists())
			.andDo(log())
			.andReturn();
	}
```

4) Json-path 의 객체 탐색을 위한 질의 방법 : jsonPath(query, Matcher) , jsonPath(query).matcher()

① 연산자

`$`	The root element to query. This starts all path expressions.
`@`	The current node being processed by a filter predicate.
`*`	Wildcard. Available anywhere a name or numeric are required.
`..`	Deep scan. Available anywhere a name is required.
`.<name>`	Dot-notated child
`['<name>' (, '<name>')]`	Bracket-notated child or children
`[<number> (, <number>)]`	Array index or indexes
`[start:end]`	Array slice operator
`[?(<expression>)]`	Filter expression. Expression must evaluate to a boolean value.

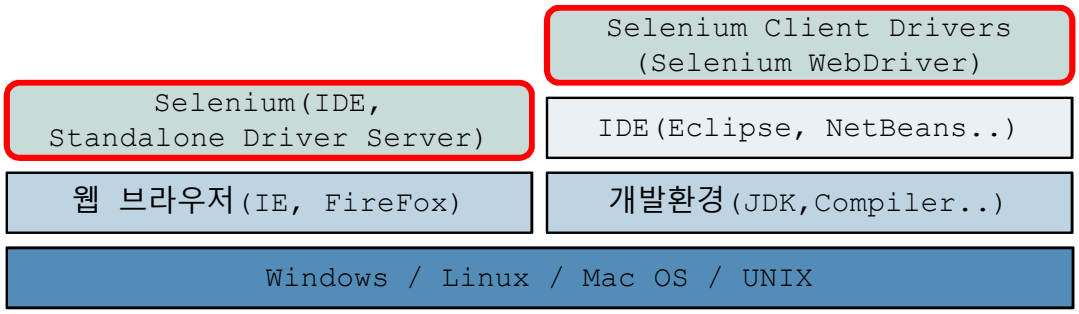
② 함수

min()	Provides the min value of an array of numbers	Double
max()	Provides the max value of an array of numbers	Double
avg()	Provides the average value of an array of numbers	Double
stddev()	Provides the standard deviation value of an array of numbers	Double
length()	Provides the length of an array	Integer

③ 필터연산자

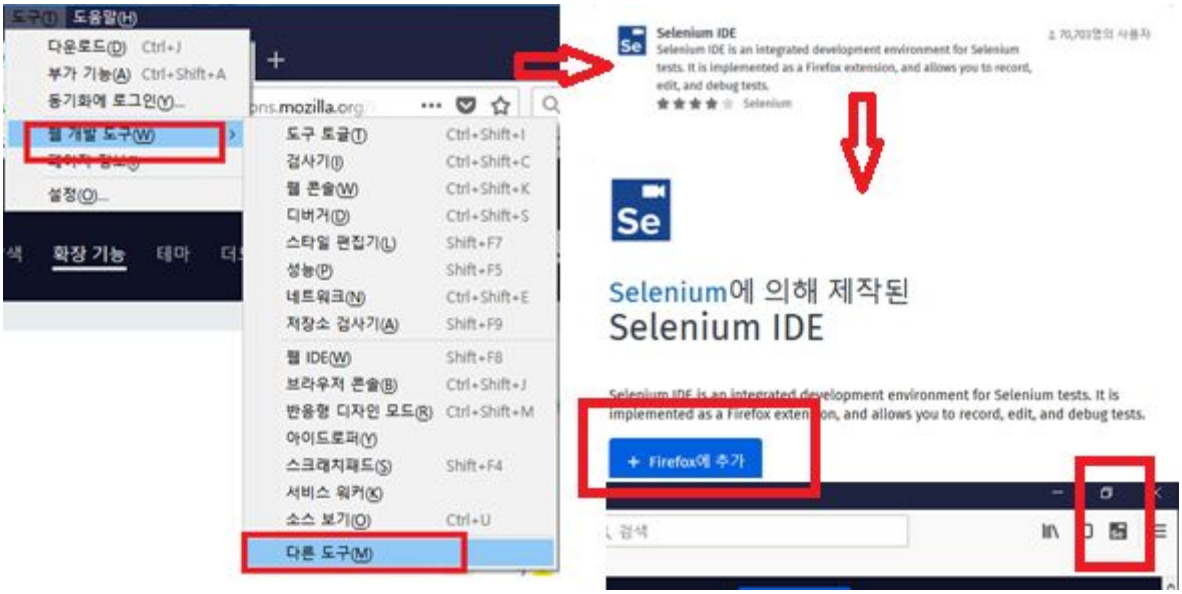
==	left is equal to right (note that 1 is not equal to '1')	=~	left matches regular expression [?(@.name =~ /foo.*?/i)]
!=	left is not equal to right	in	left exists in right [?(@.size in ['S', 'M'])]
<	left is less than right	nin	left does not exists in right
<=	left is less or equal to right	subsetof	left is a subset of right [?(@.sizes subsetof ['S', 'M', 'L'])]
>	left is greater than right	size	size of left (array or string) should match right
>=	left is greater than or equal to right	empty	left (array or string) should be empty

2. Selenium 을 이용한 UI 테스트 (https://www.seleniumhq.org/)
: 웹애플리케이션을 위한 고이식성 테스트 프레이워크로 각 브라우저별 플러그인이 지원됨.



1) Selenium 도구 설치

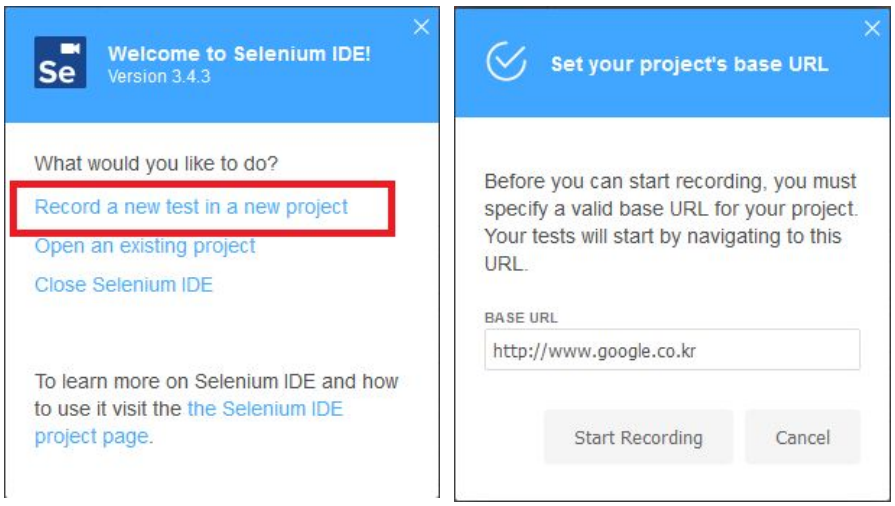
① SeleniumIDE



② Selenium WebDriver 다운로드(https://www.seleniumhq.org/download/)
: Maven 에 의존성 추가 (selenium-java)

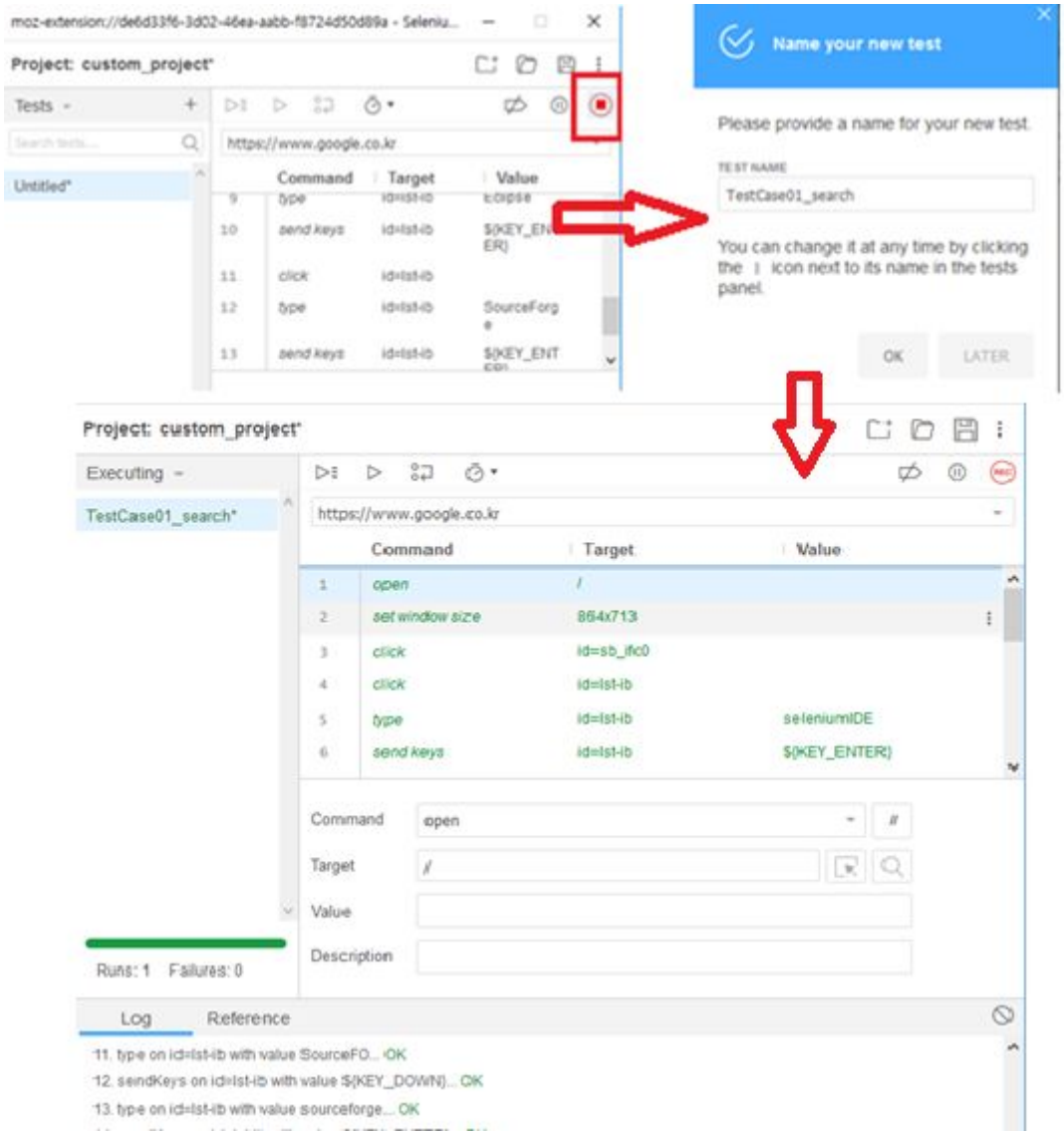
Language	Client Version	Release Date			
Java	3.14.0	2018-08-02	Download	Change log	Javadoc
C#	3.14.0	2018-08-02	Download	Change log	API docs
Ruby	3.14.0	2018-08-03	Download	Change log	API docs
Python	3.14.0	2018-08-02	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

2) SeleniumIDE 사용

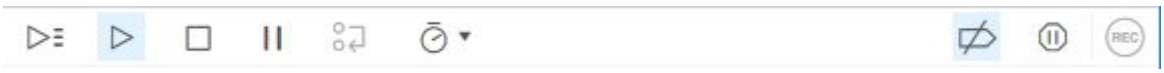


② 테스트 대상 모듈에 대해 테스트 수행(기능 동작)

③ 테스트케이스 생성



④ 테스트 케이스나 테스트 스윗 실행



- Run All tests in suite : 테스트 스윗 단위로 테스트 수행
- Run Current test : 현재 선택된 테스트 케이스 대상으로 테스트 수행
- Stop / Pause : 정지 / 일시정지
- Step Over Current Command : breakpoint 단위로 진행
- Test Execution speed : 테스트 케이스 재생 속도 조절
- Disable breakPoints : 모든 breakpoint 비활성화
- Pause on exceptions : 예외 발생시 일시 중지 설정
- Record : 녹화

3) Selenium WebDriver 사용

① Maven에 의존성 추가(pom.xml)

```
<dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.7.1</version>
        <scope>test</scope>
</dependency>
```

② 각 브라우저별 드라이버 플러그인 다운로드 및 path 추가
(https://www.seleniumhq.org/download/)

Third Party Browser Drivers NOT DEVELOPED by seleniumhq

Browser				
Mozilla GeckoDriver	latest	change log	issue tracker	Implementation Status
Google Chrome Driver	latest	change log	issue tracker	selenium wiki page

③ 브라우저별 WebDriver 를 이용한 테스트 케이스 작성

```
@Test
public void googleSearch () throws IOException{
        // 테스트 브라우저별 WebDriver 생성
        WebDriver driver =  new FirefoxDriver();
        // get 요청 주소 설정
        driver.get("http://www.google.co.kr");
        // 주어진 조건에 맞는 첫번째 엘리먼트 검색
        WebElement element = driver.findElement(By.name("q"));
        // 해당 엘리먼트에 value 타이핑
        element.sendKeys("selenium");
        // 해당 엘리먼트가 form 이거나 form 의 입력 태그일때 전송.
        element.submit();
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
                @Override
                public Boolean apply(WebDriver d) {
                        return d.getTitle().toLowerCase().startsWith("selenium");
                }
        });
        driver.findElements(By.cssSelector(".r")).get(0).findElement(By.tagName("a")).click();
}
```

4) Selenium standalone server 사용

: 테스트 대상 디바이스가 테스트 수행 디바이스와 다른 경우 별도로 운영되는 서버를 사용할 수 있음.

① Standalone server 다운로드 (https://www.seleniumhq.org/download/)

Selenium Standalone Server

The Selenium Server is needed in order to run Remote Selenium V
capable of running Selenium RC directly, rather it does it through
WebDriverBackedSelenium interface.

Download version 3.14.0

② 테스트 대상 디바이스에 드라이버 플러그인 설치 및 PATH 추가
(https://www.seleniumhq.org/download/)

```
pi@raspberry:~/selenium-server/driver-plugins $ ls
geckodriver
pi@raspberry:~/selenium-server/driver-plugins $ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games
:/usr/games:/home/pi/selenium-server/browsers/firefox:/home/pi/selenium-serve
r/driver-plugins
pi@raspberry:~/selenium-server/driver-plugins $ sudo apt-get install chromium
-driver
```

③ 테스트 대상 디바이스에 서버 구동(ex. 192.168.0.82:4444)

```
pi@raspberry:~/selenium-server $ java -jar selenium-server-standalone-3.14.0.jar

17:50:21.809 INFO [GridLauncherV3.launch] - Selenium build info: version: '3.14.
0', revision: 'aaccccе0'
17:50:21.817 INFO [GridLauncherV3$1.launch] - Launching a standalone Selenium Se
rver on port 4444
2018-10-18 17:50:22.540:INFO::main: Logging initialized @3217ms to org.seleniumH
q.jetty9.util.log.StdErrLog
17:50:23.588 INFO [SeleniumServer.boot] - Selenium Server is up and running on p
ort 4444
```

④ RemoteWebDriver 를 이용한 테스트 케이스 작성

```
@Test
public void googleSearch() throws IOException{
        // 테스트 브라우저별 WebDriver 생성
        DesiredCapabilities dc = DesiredCapabilities.firefox();
        WebDriver driver = new RemoteWebDriver(new URL("http://192.168.0.82:4444/wd/hub"), dc);
        // get 요청 주소 설정
        driver.get("http://www.google.co.kr");
        //…….
}
```

5) 페이지 로딩 및 Element 검색 API (WebElements)

① 페이지 로딩

```
// 정적 페이지를 대기 설정 없이 로딩할 때 사용
driver.get("http://www.google.co.kr"); // driver.navigate.to("http://www.google.co.kr");
// 동적 페이지가 로딩시 대기 설정
// implicit wait 설정 , 일정 시간 대기
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://www.google.co.kr");
// explicit wait 설정 , 특정 엘리먼트가 나타날때까지 대기
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = (new WebDriverWait(driver, 10))
                .until(ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));
```

② Element 검색

```
WebElement element = driver.findElement(By.id("cheese"));
WebElement element = driver.findElement(By.className("cheese");
WebElement element = driver.findElement(By.tagName("iframe"));
WebElement element = driver.findElement(By.Name("cheese"));
WebElement element = driver.findElement(By.linkText("cheese"));
WebElement element = driver.findElement(By.partialLinkText("cheese"));
WebElement element = driver.findElement(By.cssSElector("#cheeses span.cheese"));
```

③ JavaScript 실행

```
WebElement element = (WebElement)
                ((JavascriptExecutor)driver).executeScript("return $('.cheese')[0]");
List<WebElement> labels = driver.findElements(By.tagName("label"));
List<WebElement> inputs = (List<WebElement>)
                ((JavascriptExecutor)driver).executeScript( "var labels = arguments[0], inputs = []; for
(var i=0; i < labels.length; i++){" + "inputs.push(document.getElementById(labels[i].getAttribute('for'))); }
return inputs;", labels);
```

④ Text 가져오기

```
WebElement element = driver.findElement(By.id("elementID"));
element.getText();
```

⑤ Form 입력값 채우기

```
WebElement select = driver.findElement(By.tagName("select"));
List<WebElement> allOptions = select.findElements(By.tagName("option"));
for (WebElement option : allOptions) {
        System.out.println(String.format("Value is: %s", option.getAttribute("value")));
        option.click();
}
```

```
Select select = new Select(driver.findElement(By.tagName("select")));
select.deselectAll();
select.selectByVisibleText("Edam");
```

```
driver.findElement(By.id("submit")).click();
element.submit();
```

⑥ Window / Frame 간 이동

```
driver.switchTo().window("windowName");
driver.switchTo().frame("frameName");
for (String handle : driver.getWindowHandles()) {
                    driver.switchTo().window(handle);
}
```

⑦ Popup 창

```
Alert alert = driver.switchTo().alert();
```

⑧ Histroy / Location 객체 이용

```
driver.navigate().to("http://www.google.co.kr"); // window.location.href = "http://www.google.co.kr"
driver.navigate().forward(); // window.history.forward()
driver.navigate().back(); // window.history.back()
```

⑨ Cookie 설정

```
driver.get("http://www.google.co.kr");
// 도메인 전체를 대상으로 한 쿠키 생성 (name, value, path, expireDate) 등을 설정 가능함.
Cookie cookie = new Cookie("key", "value");
driver.manage().addCookie(cookie);
// 현재 도메인을 대상으로 유효한 모든 쿠키 조회
Set<Cookie> allCookies = driver.manage().getCookies();
for (Cookie loadedCookie : allCookies) {
    System.out.println(String.format("%s -> %s", loadedCookie.getName(), loadedCookie.getValue()));
}
// 이름으로  쿠키 삭제
driver.manage().deleteCookieNamed("CookieName");
// 쿠키 객체로 삭제
driver.manage().deleteCookie(loadedCookie);
// 모든 쿠키 삭제
driver.manage().deleteAllCookies();
```

⑩ User Agent 변경

```
// about:config 를 통해 설정 가능한 preference 확인
FirefoOptions options= new FirefoxOptions();
options.addPreference("general.useragent.override", "some UA string");
options.addPreference("network.http.accept.default", "text/html");
WebDriver driver = new FirefoxDriver(options);
// chrome : https://src.chromium.org/viewvc/chrome/trunk/src/chrome/common/chrome_switches.cc 참고
ChromeOptions options = new ChromeOptions();
options.addArguments("user-agent=some UA string");
driver = new ChromeDriver(options);
```

6) REST response 확인 : driver.getPageSource()

① Xml

```
// Firefox : 프롤로그를 제외한 xml 형태 그대로 로딩
// Chrome : <body><div id="webkit-xml-viewer-source-xml">XML Text</div>~~~~</body>
// Edge / MSIE : parsing 규칙 없음.
```

② Json

```
// Firefox :<body><div id="content"><div id="json">JSON String</div></div></body> 구조의 DOM  형성
// Chrome / Edge / MSIE : <body><pre>JSON String</pre></body> 구조의 DOM 형성
```

❑ 테스트 및 분석 도구 활용

1. Emma 를 활용한 테스트 커버리지 측정

1) 테스트 커버리지의 정의 : Test Coverage는 개발자가 작성한 테스트 코드가 대상 소스 코드에 대해 테스트하는 코드를 작성했는지 그 커버하는 정도를 백분률과 코드 라인을 통해 알려주는 것을 말하며, Test Coverage를 분석하고 그 결과를 리포팅하는 것에 대해 설명한다.

* instrument : 테스트 대상 코드를 실행하거나 단위 테스트를 수행하는 동안 대상 코드에 테스트 코드 정보를 삽입하는 작업.

2) 테스트 커버리지 툴 EMMA

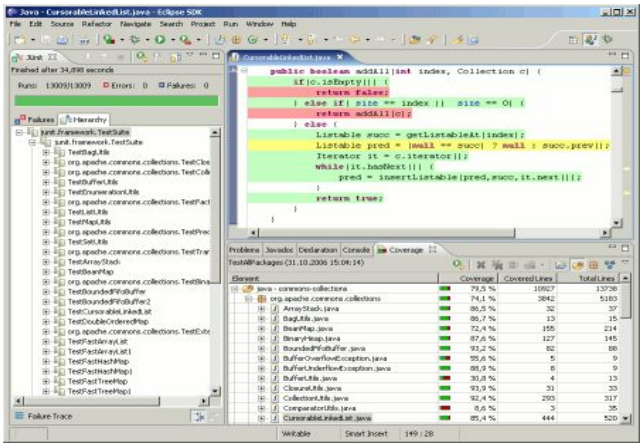
① 서비스 : http://emma.sourceforge.net/

② 특징

a. Coverage instrument 를 클래스 로드 되기 전이나 로드 중에 할 수 있다.
b. 지원 커버리지 유형 : class, method, line, basic block
c. 리포트 종류 : text, html, xml 등으로 소스와의 링크를 제공한다.
d. 개별 class 파일이나 전체 .jar 파일을 instrument 할수 있다.
e. 매우 빠르고 메모리 누수가 적다.
f. 이클립스에서 EclEmma 라는 플러그인을 통해 사용할 수 있다.

3) Egovframework 에서 테스트 커버리지 실행

① 툴바 이용 : EclEmma를 설치하면 툴바에 다음과 같은 아이콘이 생기고, **Coverage Configuration**



② 커버리지 측정 프로젝트의 컨텍스트 메뉴 -> Coverage As -> Junit Test

③ Coverage View 확인 : JUnit Test를 수행하면서 Test Coverage instrument 작업이 끝나면 Coverage View에 결과가 나타나며, 해당 항목을 더블클릭하면 해당 소스가 에디터 View를 통해 열린다.

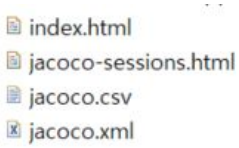
4) 메이븐을 이용한 coverage reporting(JaCoCo(JavaCodeCoverage) 플러그인을 이용)

① pom.xml 에 플러그인 설정

```
<plugin>
        <groupId>org.jacoco</groupId>
        <artifactId>jacoco-maven-plugin</artifactId>
        <version>0.8.0</version>
        <executions>
                <execution>
                        <id>jacoco-initialize</id>
                        <goals>
                        <goal>prepare-agent</goal>
                        </goals>
                </execution>
                <execution>
                        <id>jacoco-report</id>
                        <phase>test</phase>
                        <goals>
                        <goal>report</goal>
                        </goals>
                </execution>
        </executions>
</plugin>
```

② 메이븐 플러그인 실행 : ex) mvn test

③ target/site/jacoco 폴더에서 리포트 확인



2. PMD, FindBugs, CodeMetrics 를 활용한 정적 분석

1) PMD : 애플리케이션 배포 능력단위 부교재 참고

① 정의 : 전체 모듈의 소스 코드를 일괄 스캐닝하고 분석하여 부적절한 소스코드 부분을 탐지하기 위해 미리 정의된 78개의 탐지 가능한 오용된 코드, 솔루션, 레퍼런스의 패턴들과 사용자에 의해 수정 및 등록된 패턴을 활용하여 검색 및 탐지, 리포팅 서비스를 제공하는 정적 분석 도구

② 서비스 : https://pmd.github.io/

③ 설치 : 이클립스 -> Help -> Install New Software -> [Add] -> name : pmd plugin, location : https://dl.bintray.com/pmd/pmd-eclipse-plugin/updates/ -> 인스톨 -> 이클립스 재실행

④ 설정

```
Window -> Preferences -> PMD ->
        1) Priority levels : Blocker   - 심각한 버그 발생 가능 코드로 룰 적용 필수.
                                 Critical  - 버그 발생 가능 코드로 룰 적용.
                                 Urgent    - 스파케티 코드, 표준코드 및 보안과 성능관련 이슈 발생 가능.
                                 Important - 미사용 코드 선언.
                                 Warning   - 패키지, 클래스, 필드 네이밍 표준 규칙 미준수
        2) logging options : pmd 로그 파일 위치 변경.
        3) Reports
        4) Rule Configuration : 출력 룰셋 전체 선택 및 삭제 -> [Import rule set..] 선택 -> [Browse] 선택 ->
            가) 전자정부표준프레임워크 룰셋 환경 설정 및 내용 참고
http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev2:imp:inspection#전자정부_표준프레임워크_표준_inspection_룰셋
 [표준 Inspection 룰셋 한글/영문판의 압축파일 : 개발환경 3.5 이상 버전 사용]
```

⑤ 활용 :
a) 대상 프로젝트 선택 -> 오른쪽 마우스 클릭 -> PMD -> Check Code
-> Violations Overview View 활용
Violations Outline View
b) 대상 프로젝트 선택 -> 오른쪽 마우스 클릭 -> PMD -> Clear Violations 선택으로 삭제
c) 대상 프로젝트 선택 -> 오른쪽 마우스 클릭 -> PMD -> Generate Reports 선택으로 보고서 작성

⑥ 오류대응 :
http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:dev2:imp:inspection:usetool#전자정부_표준_inspection 참고

2) FindBugs

① 정의 : 전체 모듈의 소스 코드를 일괄 스캐닝하고 분석하여 잠재적인 오류 발생 가능 및 부적절한 소스코드, 보안 취약점을 Scariest, Scary, Troubling, Concern의 4등급으로 구분 탐지하고, 9개의 카테고리 별 전체 400여개의 패턴 규칙과 사용자 정의 패턴을 활용하여 검색 및 탐지, 리포팅 서비스를 제공하는 정적 분석 도구

② 서비스 : http://findbugs.sourceforge.net

③ 설치 : 이클립스 -> Help -> Eclipse Marketplace -> findbugs 검색 -> FindBugs Eclipse Plugin 3.0.1 인스톨 -> 이클립스 재실행.

④ 설정

```
Window -> Preferences -> Java -> FindBugs -> 설정
        1) analysis effort : Minimal 선택
        2) Report Configuration 탭
            - Malicious code vulnerability : JCF 보안 취약점 체크
            - Dodgy code : 형변환 및 캐스팅 부정확 및 오류 체크
            - Bad Practice : 명명규칙 위반 및 널 처리 등 체크
            - Correctness : 상수 선언 미스 및 무의미 메서드 콜 등 체크
            - Internationalization : 인코딩 및 국제화 미스 체크
            - Performance : 미사용 필드, 비효율적 객체 인스턴스와 보일러판 코드 체크
            - Security : 보안 취약 코드 체크
            - Multithreaded correctness : 멀티 스레드 환경에서 불완전 객체 활용 등 체크
            - Experimental : 스트림 자원 미회수 또는 리소스 해제 미스 등 체크
        3) Detector configuration 탭의 Pattern명과 Category 및 하기의 오류탐지 유형 참고.
```

⑤ 활용 : 버그 리포팅 대상 프로젝트 선택 -> 오른쪽 마우스 클릭 -> Find Bugs ->
Find Bugs perspective -> Bug Explorer View와 Bug Info View 활용

⑥ Export : 버그 리포팅 대상 프로젝트 선택 -> 오른쪽 마우스 클릭 -> Find Bugs ->
-> Save XML

⑦ 오류탐지유형 : http://findbugs.sourceforge.net/bugDescriptions.html

3) CodeMetrics

1. 설치
   1.1 이클립스 -> Help -> Install New Software -> [Add]
   Name : Code Metrics
   Location : http://metrics2.sourceforge.net/update/
   1.2 이클립스 -> Window -> Preferences -> Metrics Preferences
   -> XML Export -> Metrics flat 체크 확인
   1.3 이클립스 -> Package Explorer View -> 대상 프로젝트 선택 -> Properties -> Metrics
   -> Enable Metrics 체크
   1.4 이클립스 -> Window -> Show View -> Metrics -> Metrics View or Dependency Graph View 선택
   1.5 해당 프로젝트 properties -> Metrics -> Enable Metrics
2. 특징
   2.1 소스코드를 정적분석을 통해 다양한 기준으로 수치화하여 이를 표현
   2.2 메트릭을 통해 나타나는 수치들은 가시적인 수치로, 각 항목에 대해 수치에 대한 의미 이해 필요
   2.3수치를 통해서 소스코드의 상태와 리팩토링할 영역을 좁힐수 있음
3. 분석항목

분석 항목	설명
Number of Classes	전체 클래스 개수
Number of Children	상속 또는 구현된 서브 클래스 갯수(직접 코딩된 부모 클래스 또는 인터페이스 포함)
Number of Interfaces	전체 인터페이스 갯수
Depth of Inheritance Tree (DIT)	상속 또는 구현 구조 내 서브 클래스의 부모 존재 수
Number of Overridden Methods (NORM)	상속 또는 구현시 부모 생성자 호출에 따른 부모 상속 구조 상 연쇄 호출 수
Number of Methods (NOM)	전체 메서드 갯수(static 메서드 제외)
Number of Static Methods (NOM)	전체 static 메서드 갯수
Number of Attributes	전체 필드 갯수(static 필드 제외)
Number of Static Attributes	전체 static 필드 갯수
Total Lines of Code TLOC(Total Lines Of Code)	공백 라인과 주석을 제외한 코드 라인 수
MLOC(Method Lines Of Code)	선언된 전체 메서드 대상 각 메서드 내 공백 라인과 주석을 제외한 코드 라인 수
Specialization Index : 특화지수	NORM * DIT / NOM 연산 결과
McCabe Cyclomatic Complexity	전체 메서드 대상 각 메서드 내 코드 중 분기와 반복 및 메서드 호출 등의 지수
Weighted Methods per Class (WMC)	전체 클래스 대상 각 클래스별 McCabe Cyclomatic Complexity 지수
Lack of Cohesion of Methods (LCOM*)	전체 클래스 대상 각 클래스별 응집도 지수로 1에 가까울수록 응집도가 부족.
Afferent Coupling (Ca)	전체 패키지 내 각 패키지를 의존하는 외부 패키지의 클래스 갯수(JSP 파일 제외)(결합도)
Efferent Coupling (Ce)	전체 패키지 내 각 패키지별 다른 패키지를 의존하는 해당 패키지 내 클래스 갯수(JSP 파일 제외)(결합도)
Instability (I) : 결합도지수	Ce / (Ca + Ce) 연산 결과
Abstractness (A) : 추상화지수	전체 패키지 대상 각 패키지별 추상클래스 및 인터페이스 갯수
Normalized Distance(Dn) : 균형지수	A + I - 1 연산결과 추상화 정도와 불안정성 사이의 균형을 의미하며, 0에 가까울수록 설계 품질이 높음.

❑ Jmeter를 활용한 성능측정

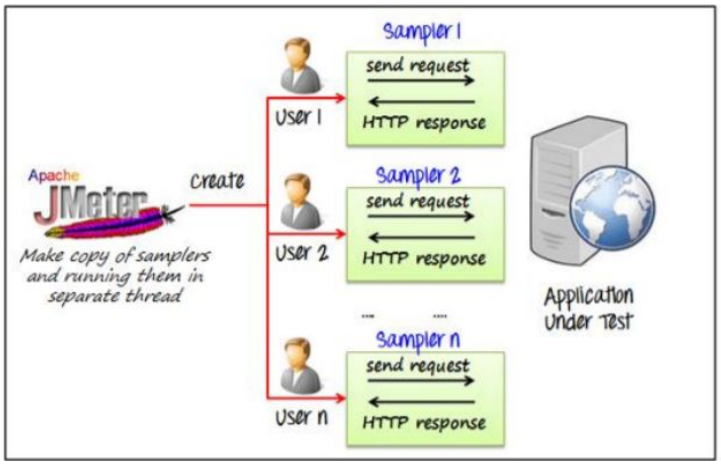
성능 측정 및 개선이 필요한 이유

(https://developers.google.com/web/fundamentals/performance/why-performance-matters/?hl=ko)

1. 개요 (출처 : oss.kr – 오픈소스소프트웨어 포탈)

소개	• Apache Jmeter 는 오픈 소스 소프트웨어로 부하 테스트 기능 동작과 퍼포먼스, 성능을 측정하기 위해 설계된 자바 어플리케이션		
주요기능	• 다양한 앱/서버/프로토콜 - Web, SOAP, REST, FTP, JDBC, LDAP, JMS, 메일, TCP, Java 오브젝트 등 테스트 • 다양한 응답 포멧 지원 - HTML, JSON, XML 혹은 텍스트 응답 포멧으로부터 데이터 추출 • 플러그인 확장 기능 - 플러그인을 통해 테스팅 능력, 데이터 분석, 가시화, 지속적인 통합 라이브러리 확장		
대분류	• 시스템 SW	소분류	• SW공학도구
라이선스 형태	• Apache License V2.0	사전설치 솔루션	• Java
실행 하드웨어	• Cross-platform	버전	• apache-jmeter-5.0(2018년10기준)
특징	• Multithreading 기능을 이용하여 동시에 많은 Thread를 발생 시킬 수도 있으며 혹은 독립된 Thread를 연속적으로 발생시켜 테스팅 • HTTP나 FTP서버 뿐만 아니라 임의의 데이터베이스 쿼리도 성능 테스트 • 높은 확장성과 GUI 환경으로 빠른 작업을 정확하게 할 수 있음		
보안취약점	• 취약점 ID : CVE-2018-1297 • 심각도 : 9.8 CRITICAL(V3) • 취약점 설명 : 분산 테스트 (RMI 기반)를 사용할 때, jmeter는 보안되지 않은 RMI 연결, 이로 인해 공격자는 JMeterEngine에 대한 액세스 권한을 얻고 무단 코드 사용 • 대응방안 :4.0 이상으로 업데이트 • 참고 경로 : http://mail-archives.apache.org/mod_mbox/www-announce/201802.mbox/%3CCAH9fUpaNzk5am8oFe07RQ-kynCsQv54yB-uYs9bEnz7tbX-O7g%40mail.gmail.com%3E		
개발회사/커뮤니티	• Apache Software Foundation		
공식 홈페이지	• https://jmeter.apache.org/		

1) 주요 기능



① JSP, Servlets 및 AJAX와 같은 동적 리소스 뿐만 아니라 JavaScript 및 HTML 과 같은 정적 리소스의 성능을 테스트하는데 사용

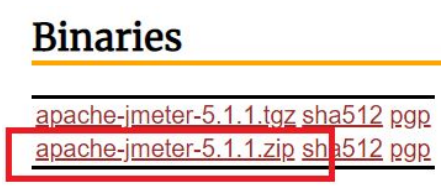
② 웹 사이트에서 처리 할 수 있는 동시 사용자의 최대 수를 테스트 할 수 있으며, 다양한 플러그인을 통해 성능 보고서와 그래픽 분석 제공

2) 시스템 요구 사항

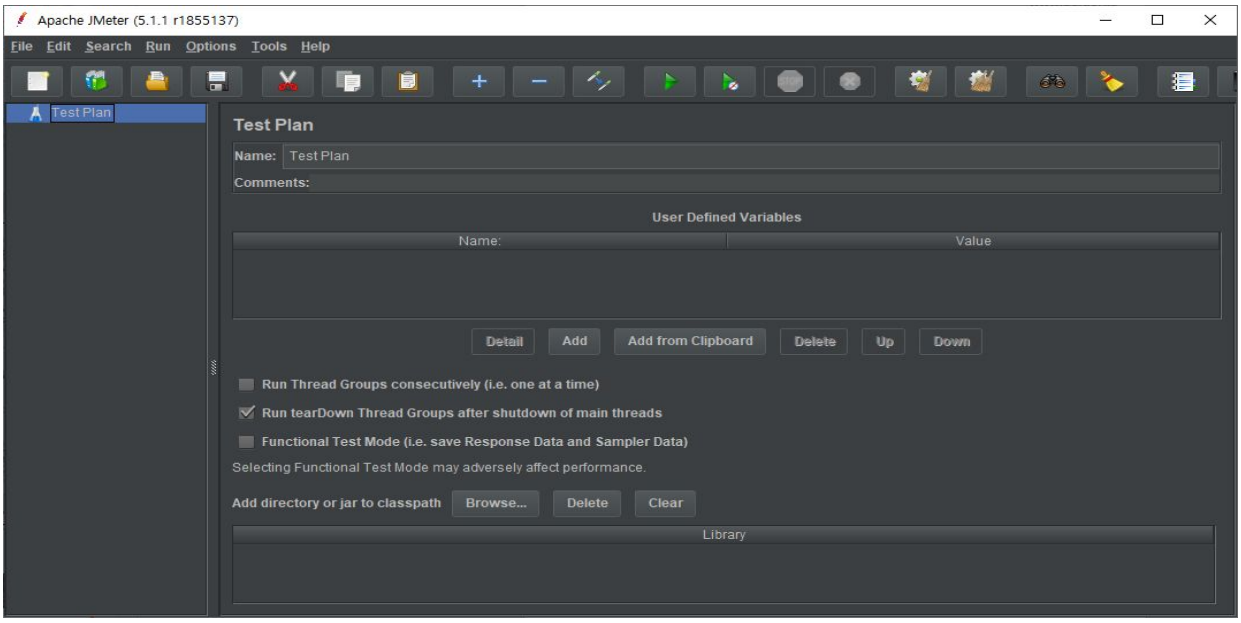
구 분	요구사항
JAVA (Java Virtual Machine Installed)	Jmeter는 Java 기반 응용 프로그램이며, Java Runtime을 실행 해야하며, Jmeter 4.0 부터 Java 8 이상 지원
CPU	코어가 4개 이상인 멀티코어 CPU를 제안하며, Jmeter는 다중 스레드를 많이 사용하므로 많은 CPU 차지
메모리	16GB RAM을 제안하며, 충분한 동시 사용자를 시뮬레이트 하고도 운영체제에 충분한 메모리를 확보 권장
디스크	Jmeter는 디스크에 별로 의존하지는 않지만 SSD 권장
네트워크	1Gbps LAN 권장하며, Jmeter는 네트워크 대역폭에서 많은 동시 사용자 시뮬레이션 함
etc	다른 시스템 하드웨어 사양은 별로 중요하지 않음

2. Jmeter 설치 및 plugin manager 설치 (https://jmeter.apache.org/download_jmeter.cgi)

1) 다운로드

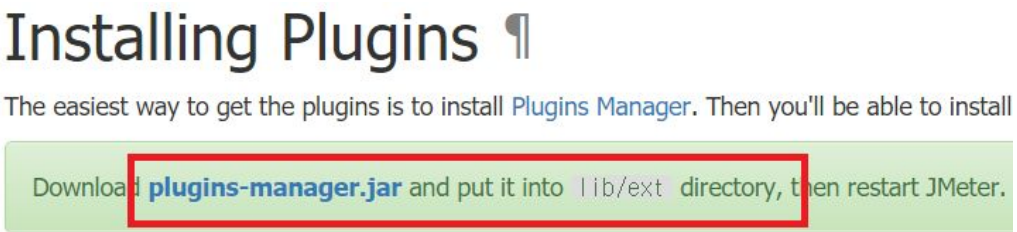


2) 설치 확인 및 실행

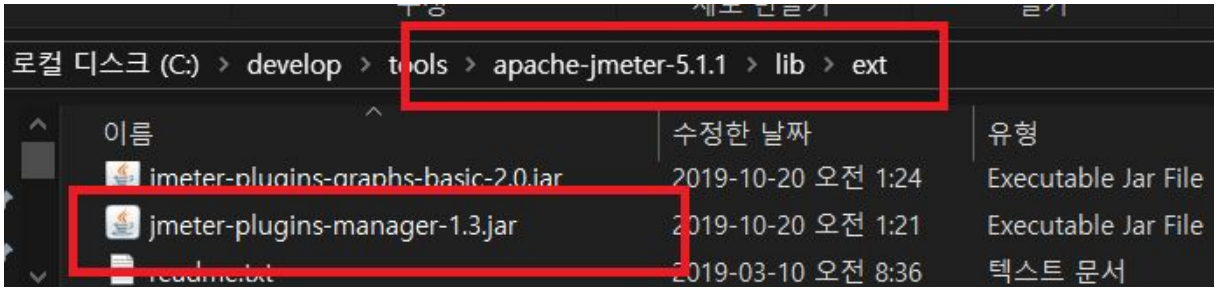


3) 플러그인 매이저 설치(https://jmeter-plugins.org/install/Install/)

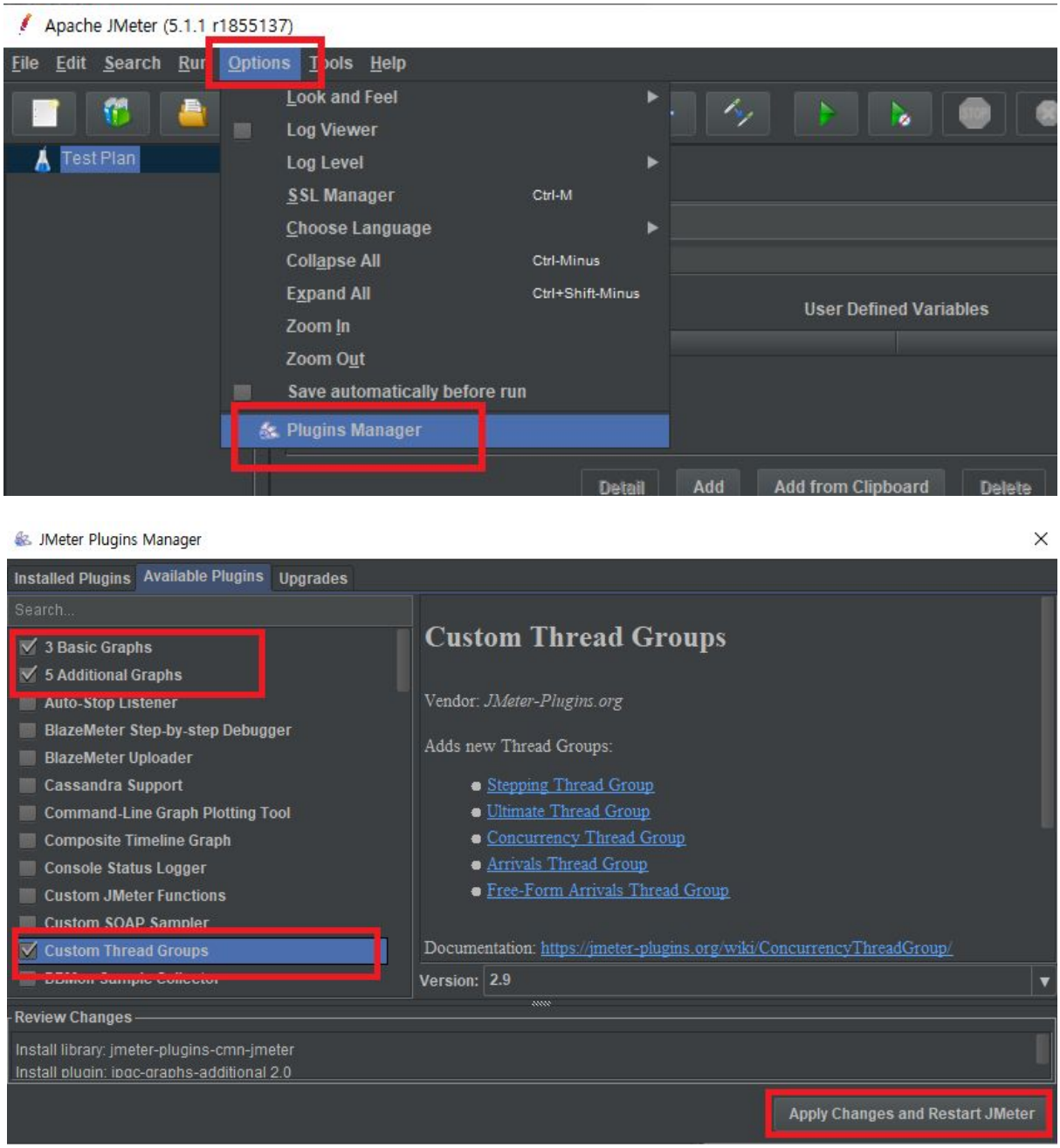
① 다운로드



② Jmeter설치경로₩lib₩ext 에 해당 파일 복사



③ Jmeter 실행 후 플러그인 추가



3. Jmeter 를 이용한 성능 테스트

1) 기본 용어

① ThreadGroup : 테스트 플랜이 수행할 하나의 작업 그룹, 모든 테스트 플랜의 시작이 됨.

② Sampler : 사용자의 행동을 가정한 하나의 작업

ex) Http Request

③ Logic Controller : Sampler 와 비슷한 개념이나, sampler 에서 단순 작업을 정의한다면, controller 에서는 작업에 추가적인 조건을 커스터마이징 할 수 있음.

ex) Once Only Controller, Interleave Logical Controller

④ Listener : 처리 결과를 보여주기 위한 리포트 종류

ex) View Result Tree, Summary Report , Transaction per Seconds, Response Latencies Over Time

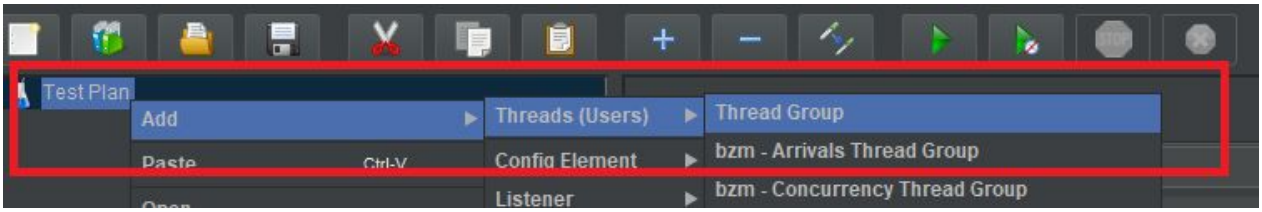
⑤ Config element : Sampler 에 추가적인 속성을 부여하기 위한 설정 종류

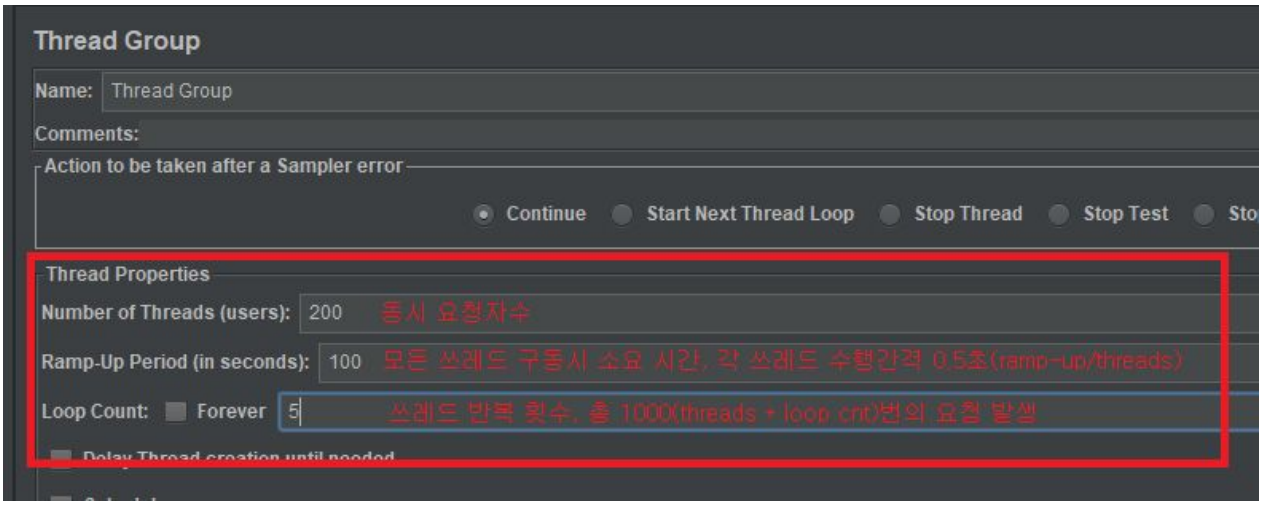
ex) Http Cookie Manager, Http Cache Manager, Http Header Manager, Http Request Default

2) 테스트

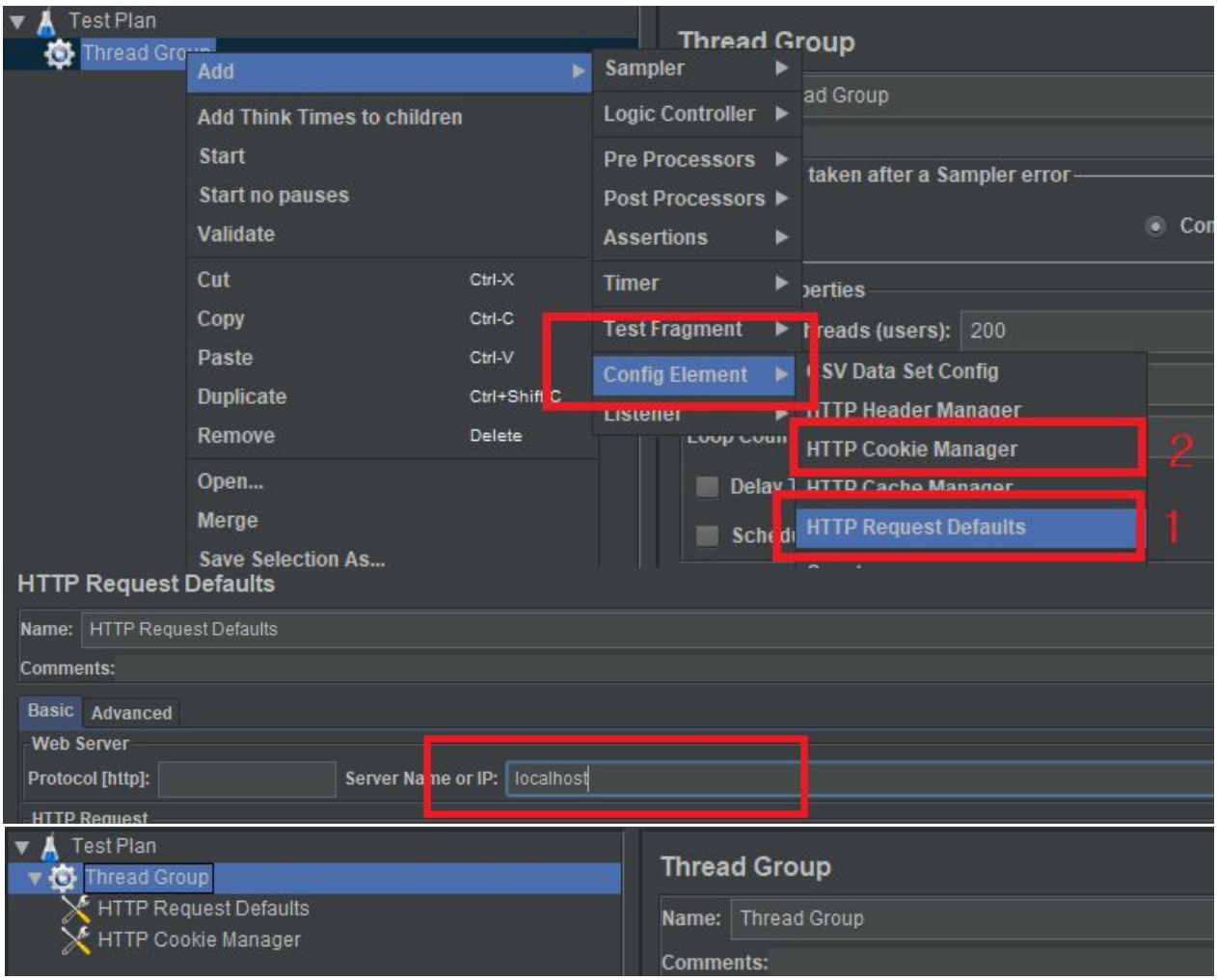
사용자 리스트 조회 성능 부하 테스트(선행 조건 : 로그인)

① TheadGroup 추가

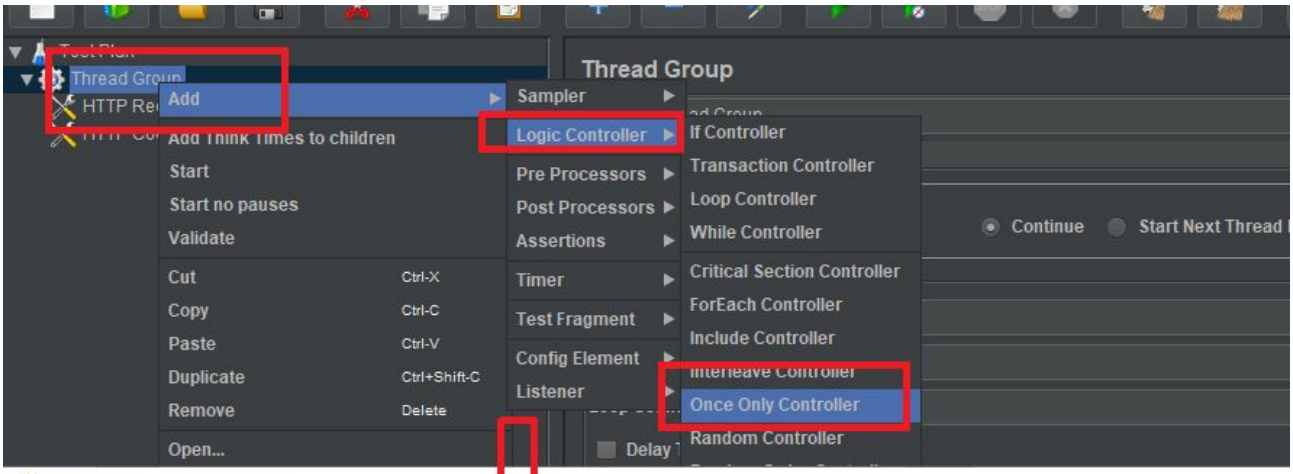


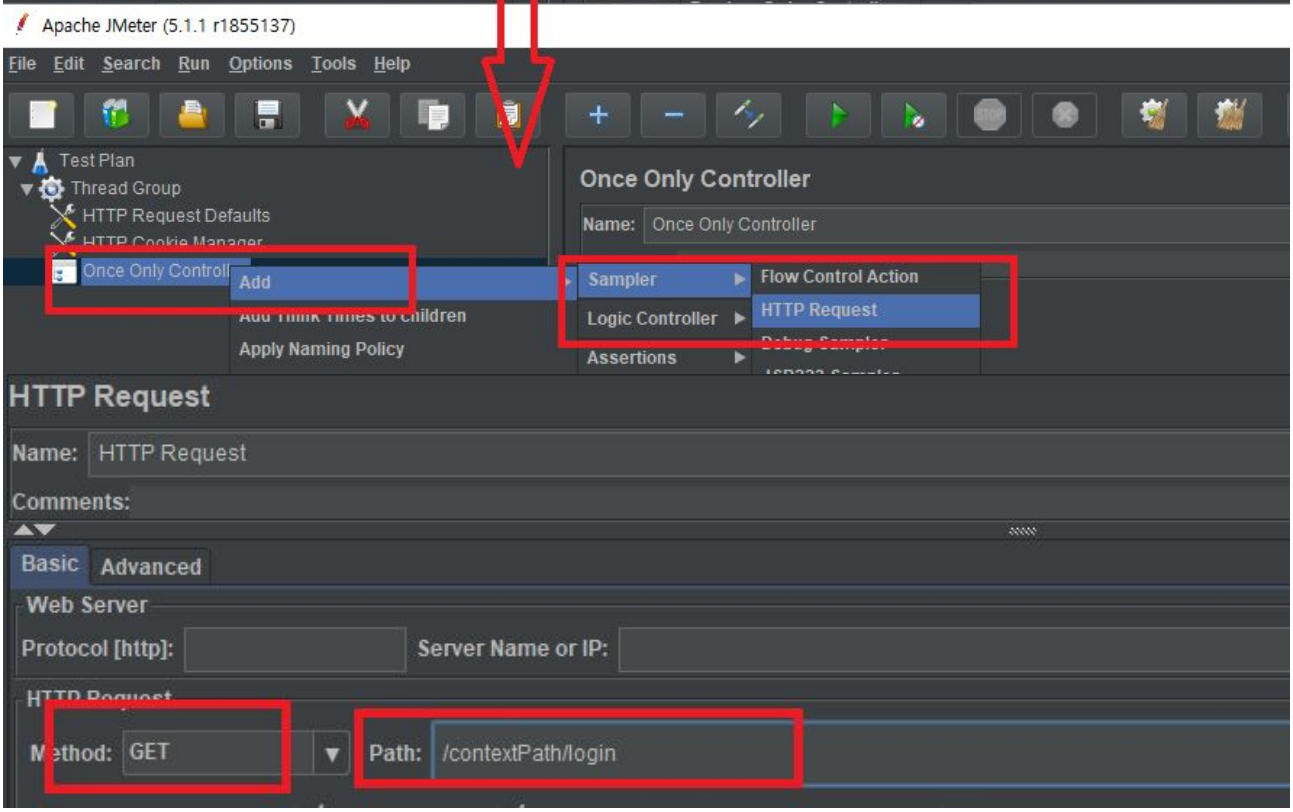


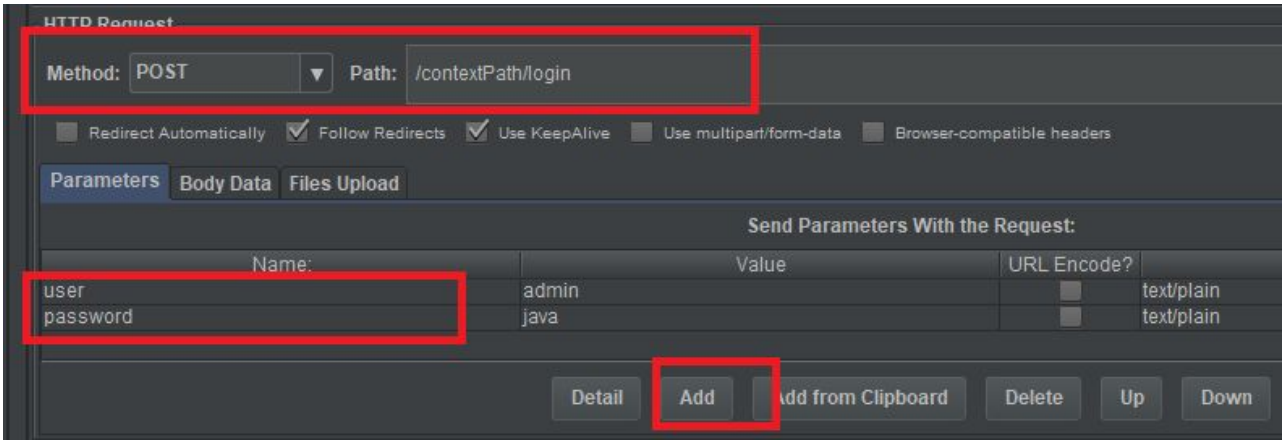
② Config Element 추가



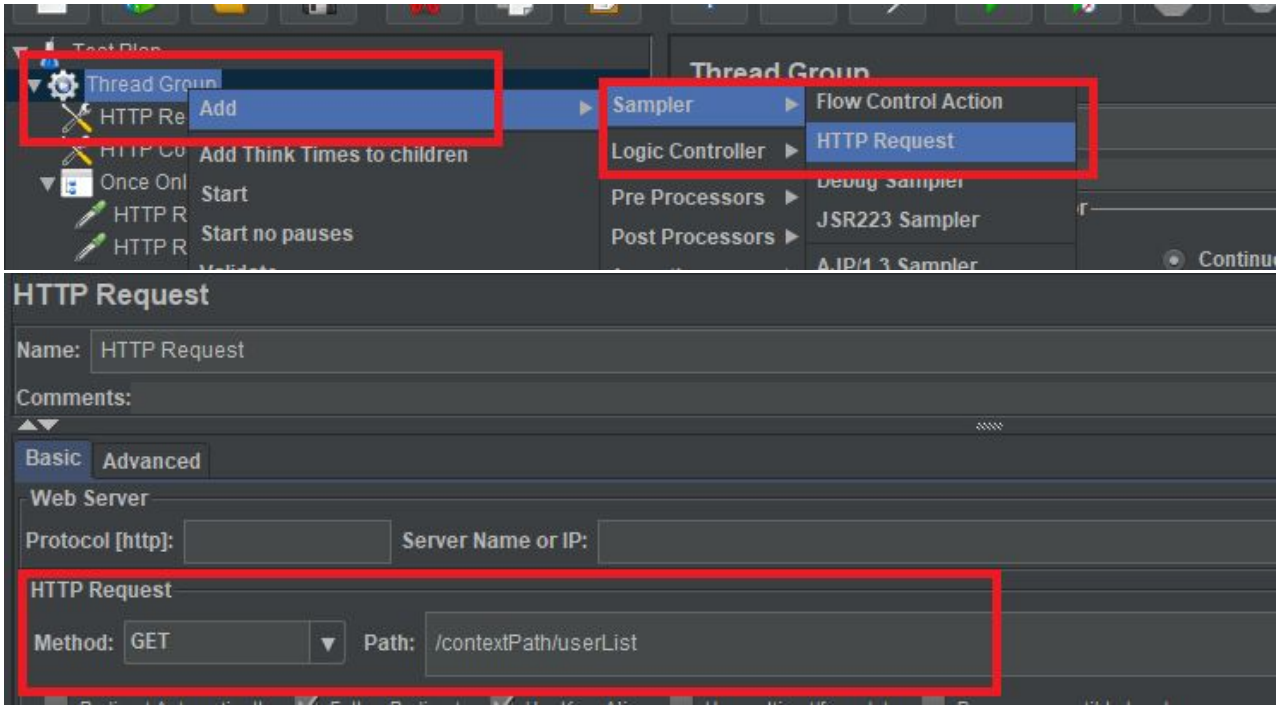
③ Logic Controller 추가 (로그인 선행을 위한 컨트롤러)



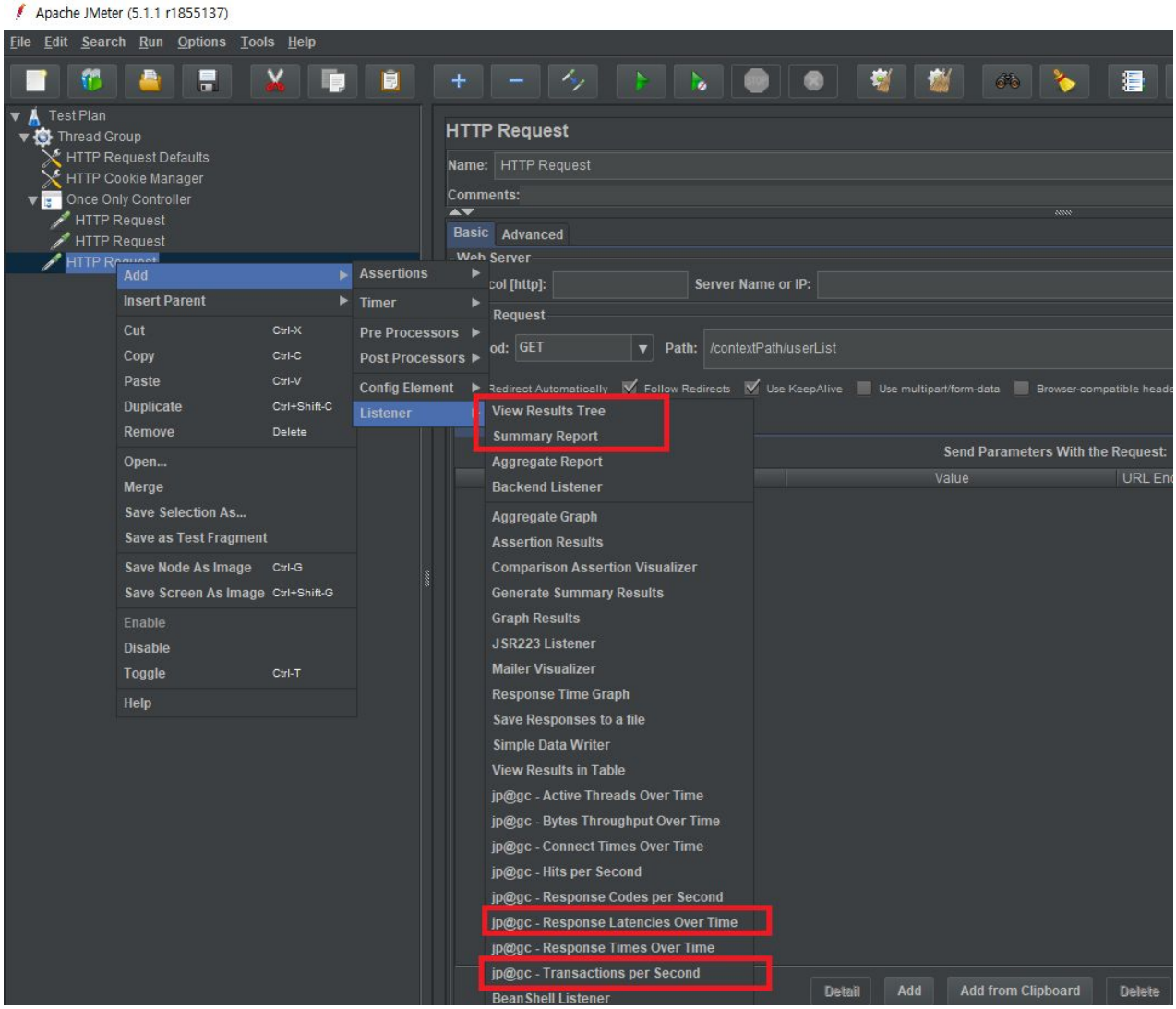




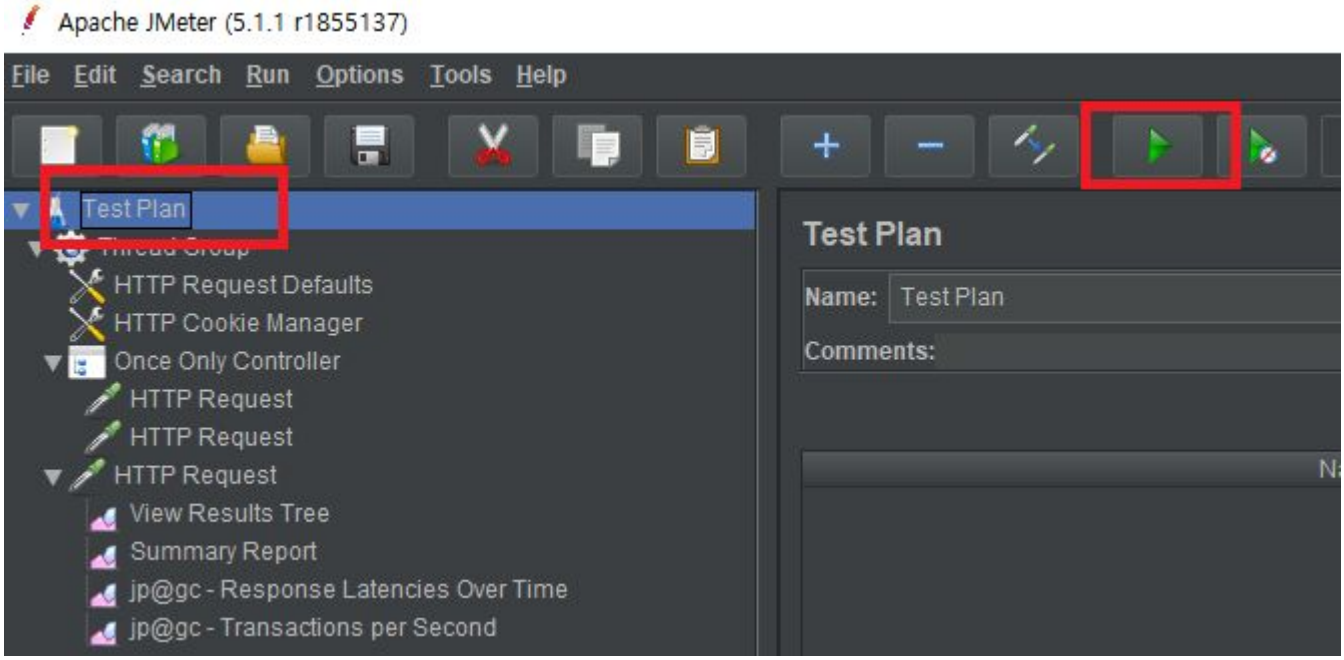
④ Sampler -> Http Request 추가 (성능 측정 대상 테스트)



⑤ Listener 추가 (측정 결과 확인을 위한 리포팅)
: View Result Tree, Summary Report, Response Latencies Over Time, Transaction Per Seconds

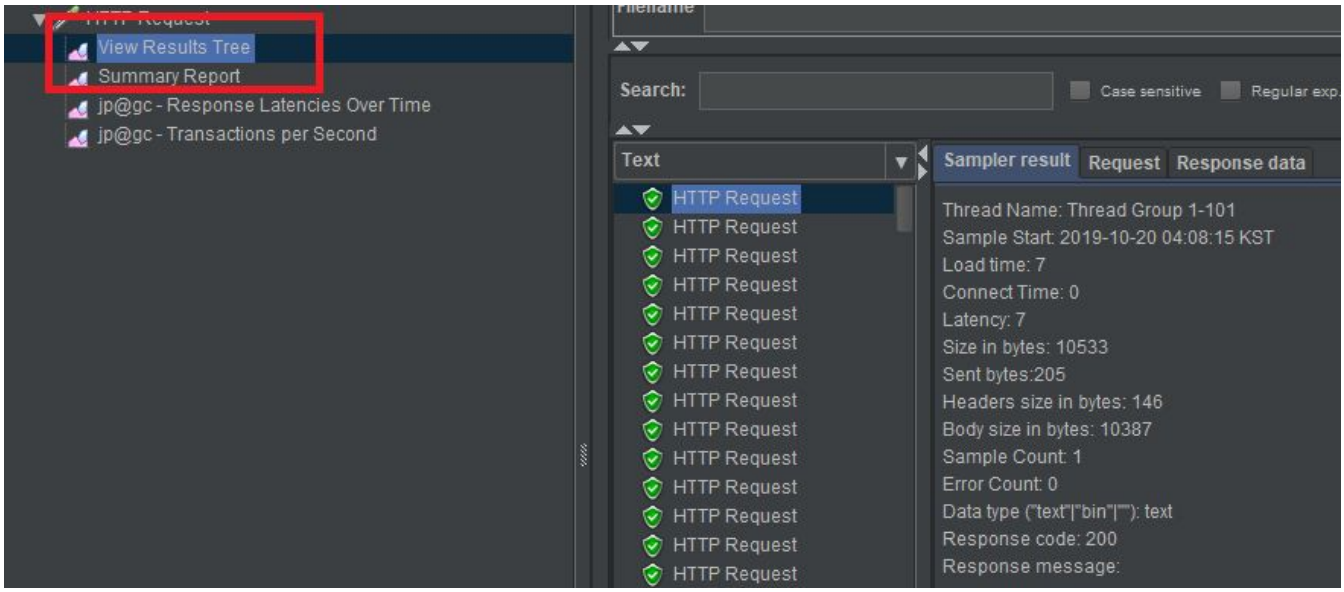


⑤ 측정 시작

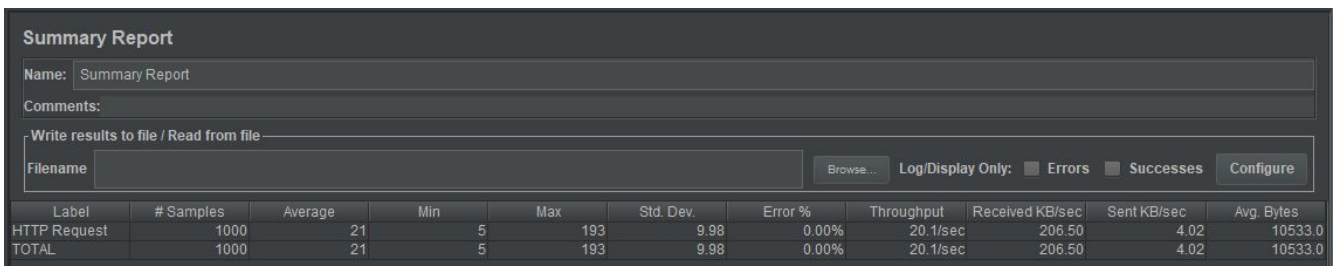


⑥ 결과 확인

- View Result Tree : 개별 요청에 대한 측정 결과

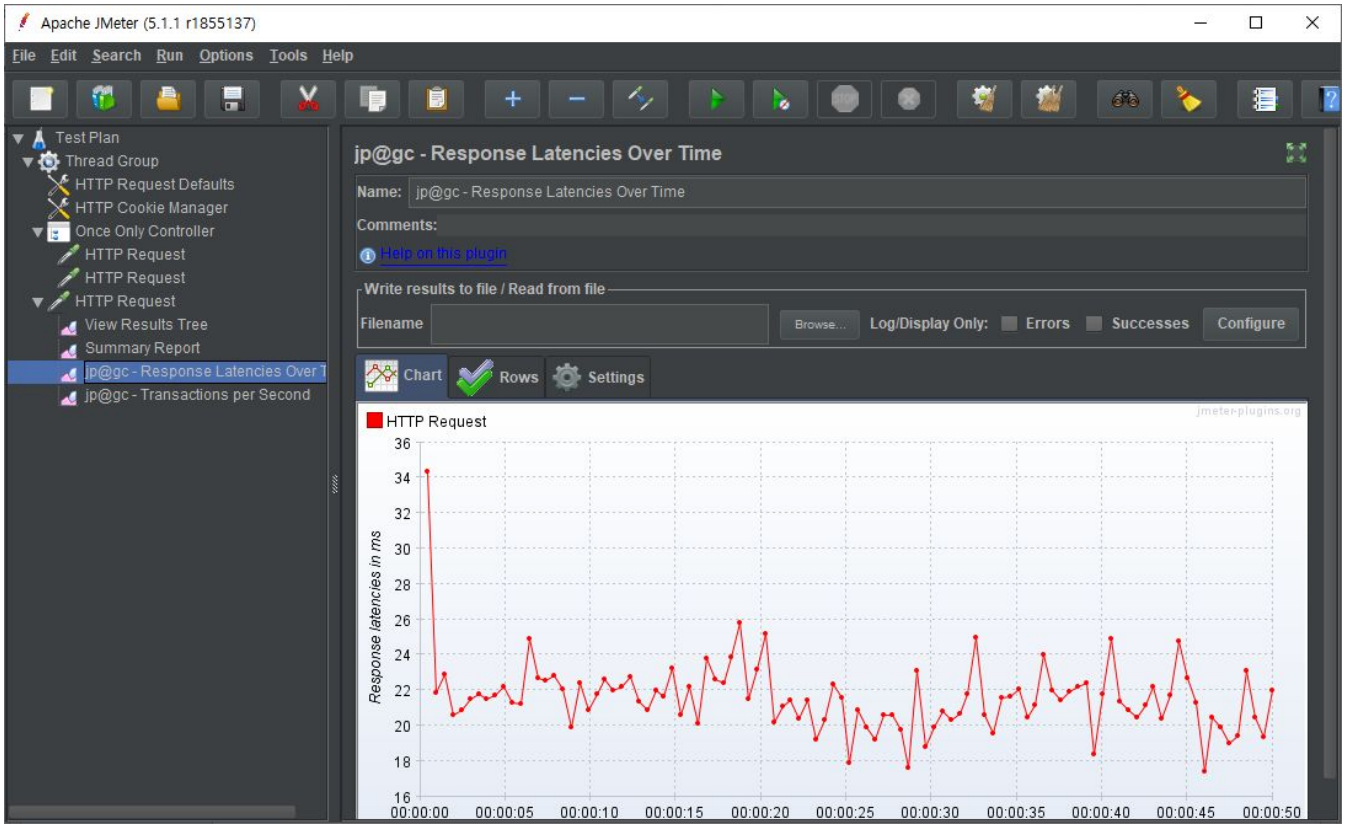


- Summary Report : 전체 측정 결과의 요약 보고서
  - Samples : 전체 요청 수
  - Average : 평균 응답 시간(ms)
  - Min : 최소 응답 시간(ms)
  - Max : 최대 응답 시간(ms)
  - Std. Dev. : 표준 편차, 값이 작을 수록 성능이 안정적임.
  - Error : 에러 발생율
  - Troughput : 평균 처리율(TPS)
  - Received : 초당 수신율(KB/s)



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1000	21	5	193	9.98	0.00%	20.1/sec	206.50	4.02	10533.0
TOTAL	1000	21	5	193	9.98	0.00%	20.1/sec	206.50	4.02	10533.0

- Reponse Latencies Over Time



- Transaction Per Seconds (TPS)
  초당 처리할 수 있는 트랜잭션의 양, 서버의 성능평가 기준이 됨.