



Apache JMeter

오픈소스로 대용량 웹 서비스 성능 테스트하기

장재만 지음

Apache JMeter

오픈소스로 대용량 웹 서비스 성능 테스트하기

Apache JMeter **오픈소스로 대용량 웹 서비스 성능 테스트하기**

초판발행 2015년 1월 16일

지은이 장재만 / 펴낸이 김태현

펴낸곳 한빛미디어(주) / 주소 서울시 마포구 양화로7길 83 한빛미디어(주) IT출판부

전화 02-325-5544 / 팩스 02-336-7124

등록 1999년 6월 24일 제10-1779호

ISBN 978-89-6848-723-1 15000 / 정가 14,000원

총괄 배용석 / 책임편집 김창수 / 기획·편집 정지연

디자인 표지 여동일, 내지 스튜디오 [밈], 조판 최송실

영업 김형진, 김진불, 조유미 / 마케팅 박상용

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.

한빛미디어 홈페이지 www.hanbit.co.kr / 이메일 ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

Copyright © 2015 장재만 & HANBIT Media, Inc.

이 책의 저작권은 장재만과 한빛미디어(주)에 있습니다.

저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

지금 하지 않으면 할 수 없는 일이 있습니다.

책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.

한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_ 장재만

포털과 CDN 서비스 업체를 거쳐 현재는 (주)에임투지의 플랫폼연구소 팀장으로 근무하고 있으며 대용량 트래픽 제어 솔루션(NetFUNNEL)을 개발하고 있다. Apache JMeter를 이용하여 다수의 성능 테스트를 한 경험이 있으며 컨설팅 및 오픈소스에 많은 관심이 있다.

취미로 밴드에서 베이스를 연주하며 ‘안전제일’이라는 밴드에서 활동하며 2장의 음반을 발표했다.

- 홈페이지: www.jacojang.com

저자 서문

시간이 갈수록 웹 사이트의 QoS^{Quality of Service}가 중요해지고 있습니다. 비슷한 많은 서비스가 서로 경쟁을 하다 보니 웹 서비스에 방문한 사용자들은 응답을 기다리며 시간을 낭비하는 것을 점점 더 싫어하게 되었습니다. 많은 연구기관에서는 이러한 사용자의 반응에 기반을 둔 QoS와 매출 간의 연관성에 관한 자료를 쏟아내고 있습니다. 하지만 서비스 운영자/개발자 입장에서는 점점 더 복잡해져 가는 IT 인프라와 다양해진 접속 환경을 모두 만족하게 하기란 쉽진 않습니다.

QoS를 항상시키는 가장 좋은 방법은 성능 테스트를 통해서 QoS에 악영향을 미치는 병목 지점을 하나씩 제거해 나가는 것입니다. 성능 테스트를 위해서 ab(Apache HTTP 서버 벤치마킹 툴)와 같은 단순한 커맨드라인 툴부터 Load Runner와 같은 복잡한 상용 툴까지 매우 많은 종류의 툴이 있지만, 이 책에서 제가 소개하려는 툴은 Apache JMeter입니다. 10년 이상 된 오픈소스 프로젝트로, 현재도 활발히 개발 중입니다.

처음 성능 테스트를 진행하면서 Apache JMeter를 선택한 이유는 비용 때문이었습니다. 상용 툴을 사용하고 싶지만, 비용 문제로 원하는 테스트를 하기 힘든 경우가 많아서 어쩔 수 없이 JMeter와 같은 오픈소스 툴을 이용했습니다. 그러나 지금은 비용뿐 아니라 성능과 기능 면에서 충분히 활용할 수 있는 수준이라고 생각합니다.

이 책에는 JMeter를 이용하면서 제가 찾아낸 성능 테스트 노하우와 주의사항이 정리되어 있습니다. JMeter를 이용해서 성능 테스트를 하고자 하는 분들에게 많은 도움이 되었으면 합니다.

마지막으로, 책을 집필하는 동안 많은 도움을 주신 회사 동료들과 한빛미디어의 정지연 님, 무엇보다 큰 힘이 되어준 아내 조혜영에게 감사의 말을 전합니다.

대상 독자

초급

초중급

중급

중고급

고급

이 책은 Apache JMeter를 이용해서 웹 서비스의 성능 테스트를 하려는 시스템 운영자와 웹 개발자를 위한 책입니다. 자바와 웹 서비스 인프라, 웹 프로토콜에 대한 기초적인 지식을 가지고 있는 분이라면 쉽게 읽을 수 있습니다.

기존의 상용 성능 테스트 툴에서 오픈소스 성능 테스트 툴로 변경하고 싶은 테스트 담당자에게도 좋은 자료가 될 것입니다.

이 책에 사용한 예제 코드는 다음에서 다운로드할 수 있습니다.

- 깃헙 자료실: <https://github.com/jacojang/JMeterBookSource>
- 한빛미디어 자료실: <http://hanbit.co.kr/exam/2723>

차례

1	들어가기	1
2	0 1 JMeter와 성능 테스트	2
2	1.1 JMeter란.....	2
3	1.2 성능 테스트.....	3
7	1.3 대용량 성능 테스트.....	7
7	1.4 주요 용어 및 개념.....	7
11	성능 테스트 사전 작업	11
12	0 2 JMeter 설치와 환경 설정	12
12	2.1 실행 환경.....	12
14	2.2 JVM 옵션 설정.....	14
14	2.3 Overriding Property.....	14
15	2.4 non-GUI Mode.....	15
17	0 3 간단한 Test Plan 작성하기	17
17	3.1 Test Plan 작성.....	17
25	3.2 테스트 실행 및 결과 확인.....	25

4.1 Thread Group.....	30
4.2 HTTP Request Default.....	34
4.3 HTTP Request.....	38
4.4 CSV Data Set Config.....	49
4.5 HTTP Cookie Manager.....	53
4.6 Regular Expression Extractor.....	57
4.7 Logic Controller.....	65
4.8 Timer.....	72
4.9 Assertions.....	81
4.10 BeanShell의 활용.....	88
4.11 Proxy 서버를 이용한 리코딩.....	94
4.12 플러그인 활용.....	100
4.13 TCP Sampler의 TCPClient 확장하기.....	107

5.1 요구사항 분석 및 목표 설정.....	112
5.2 테스트 일정	114
5.3 테스트 계획서 예제.....	115

6.1 부하발생기 설치.....	117
6.2 투닝.....	117
6.3 분산 테스트 환경 구축.....	120
6.4 네트워크 구성.....	133

7.1 로직 구성도.....	138
7.2 테스트 방법.....	140
7.3 테스트 케이스 작성.....	141
7.4 테스트 데이터 준비.....	142
7.5 Test Plan 작성.....	145

8.1 사전 테스트.....	169
8.2 테스트 수행.....	173
8.3 결과 수집.....	183

9.1 결과 분석.....	193
9.2 리포트 작성.....	201

Windows 환경.....	206
Linux/Unix 환경.....	206
Test WAR 파일 배포.....	208

들어가기

JMeter를 이용하여 실제 성능 테스트를 실행하기 전에 JMeter와 성능 테스트의 기본 개념, 그리고 JMeter를 사용하기 위한 주요 용어와 그 개념을 알아본다.

1 | JMeter와 성능 테스트

1.1 JMeter란

Apache JMeter는 웹 애플리케이션처럼 클라이언트-서버 구조로 된 소프트웨어의 성능 테스트를 위해서 만들어진 100% 순수 자바 프로그램이다. 스텤파노 마조끼Stefano Mazzocchi가 개발했으며, 이는 현재 톰캣Tomcat으로 이름이 바뀐 Apache JServ의 테스트를 위한 코드에서 시작됐다. 이후 이 코드에 GUI와 기능을 추가하여 JMeter가 만들어졌다.

JMeter는 단위/성능/스트레스 테스트 등 많은 곳에서 활용할 수 있다. 프로토콜Protocol도 계속 추가되어 TCP, HTTP(S), FTP, JDBC, LDAP, SMTP, SOAP/XML RPC 등 현재 범용으로 사용되는 프로토콜 대부분을 지원한다.

JMeter는 통신 프로토콜 단계에서만 동작하고 웹 브라우저에서는 동작하지 않는다. 즉, 통신규약에 맞도록 클라이언트와 서버 간 메시지만 송수신할 뿐이고 클라이언트 자체에서 행해지는 연산 동작은 하지 않는다. 가장 대표적인 예가 ActiveX를 이용하여 암호화나 연산 작업을 하는 사이트다. 이러한 사이트는 ActiveX 로직을 모두 이해하고 자바를 이용하여 별도로 JMeter 내부 모듈을 구현하지 않는 이상 테스트가 불가능하다.

JMeter는 2001년에 1.0을 발표한 후 10여 년 동안 꾸준히 기능과 성능을 향상하여 2011년에는 Apache Software Foundation에서 Top Level Apache Project에 선정되기도 하였다. 또한, 자바 가상 머신JVM, Java Virtual Machine과 H/W의 성능이 향상되면서 초기에 문제가 되던 성능이나 기능 면에서 부족함이 많이 해소되어 현재는 웬만한 성능 테스트에서 핵심으로 활용하기에 부족함이 없는 상태에 도달한 것으로 보인다.

기능이 다양하고 성능이 좋음에도 JMeter는 오픈소스라는 이유로 성능 테스트 현장에서 많이 활용되지 못하고 있는 것이 현실이다. LoadRunner와 같은 외산 상용 솔루션이 가장 많이 사용되지만 라이선스 비용 문제로 현장에서 충분히 활용되지 못하며, 국내 몇몇 솔루션은 엔지니어 층이 얇거나 제대로 검증되지 않아서 신뢰성이 떨어지는 경우가 많다.

필자가 JMeter를 이용해서 많은 사이트의 성능 테스트를 진행하면서 도달한 결론은 불과 몇 년 전만 해도 다소 부족했지만, 현재는 JMeter로도 대부분 사이트에서 원하는 테스트를 수행할 수 있으며 성능이나 기능 면에서 부족함을 느끼기 어렵다는 것이다.

JMeter는 IT 업계에서 종사하는 엔지니어가 사용하기에 그리 복잡하지 않은 프로그램이다. 하지만 이 프로그램이 테스트 환경과 결합하면 많은 변수를 고려해야 하는 상황이 발생한다. 그 변수가 JMeter 자체 문제인지 네트워크 환경 문제인지 그리고 애플리케이션 서버 구성 또는 애플리케이션 자체 문제인지를 적절하게 구별하고 JMeter가 보여주는 결과 수치로 얼마나 의미 있는 값을 찾아내는가가 중요하다.

이 책에서는 JMeter를 사용하는 기초적인 방법부터 실제 성능 테스트 과정에서 일어날 수 있는 문제점에 대한 해결 방안과 JMeter로 대용량 테스트 환경을 구축하기 위한 최적의 방법까지 설명한다.

1.2 성능 테스트

JMeter를 시작하기 전에 성능 테스트에 대한 기본적인 지식을 알고 있는 것이 좋다. 이는 테스트를 계획하고 결과를 분석하는 데 중요한 역할을 한다.

이 책에서 말하는 ‘성능 테스트’란 서비스 및 서비스 시스템의 성능을 확인하기 위해서 실제 사용 환경과 비슷한 환경에서 테스트를 진행하는 것을 말한다. 이를 통

해서 응답시간 Response Time과 처리량 Throughput, 병목구간 등을 확인할 수 있고, 성능 테스트로 얻은 정보로 서비스나 서비스 시스템의 문제점을 확인하고 이를 개선 Tuning하여 보완할 수 있다.

성능 테스트는 쓰임에 따라 다음과 같이 나뉜다.

- **Load 테스트** : 시스템의 성능을 벤치 마크하기 위한 테스트를 의미한다. 이 테스트는 부하 Load를 순차적으로 증가시키면서 응답시간이 급격히 증가하거나 더는 처리량이 증가하지 않거나 시스템의 CPU와 Memory 등이 기준값 이상으로 증가하는 등 비정상 상태가 발생하는 임계점을 찾아내고 이를 바탕으로 성능 이슈에 대한 튜닝과 테스트를 반복한다.
- **Stress 테스트** : 임계값 이상의 요청이나 비정상적인 요청을 보내 비정상적인 상황의 처리 상태를 확인하고 시스템의 최고 성능 한계를 측정하기 위한 테스트를 의미한다.
- **Spike 테스트** : 이 테스트는 예를 들어 빌딩에 화재 경보가 발생했을 때 빌딩에 있는 직원들이 동시에 안전한 장소를 향해서 이동할 경우 시간이 얼마나 걸리며 어떤 문제가 발생하는지를 테스트하는 것과 같다. 즉, 갑자기 사용자가 몰렸을 때 요청이 정상적으로 처리되는지 그리고 그 업무 부하 Workload가 줄어들 때 정상적으로 반응하는지를 확인하기 위한 테스트를 의미한다.
- **Stability 테스트 / Soak 테스트** : 긴 시간 동안 테스트를 진행해서 테스트 시간에 따른 시스템의 메모리 증가, 성능 정보의 변화 등을 확인하는 테스트를 의미한다. 짧게는 한두 시간부터 길게는 며칠 동안 진행하기도 한다.

성능 테스트 프로세스는 [표 1-1]과 같이 진행되며 단계별로 담당자가 바뀐다.

[표 1-1] 성능 테스트 프로세스

단계	프로세스	내용
1	요구사항 분석	<ul style="list-style-type: none">• 테스트 목적과 범위를 정하는 단계로, 효율적인 테스트를 위해서는 목적을 정확히 설정해야 한다.• 구 시스템과 신규 시스템의 비교 테스트, 신규 시스템 오픈 전 사전 임계치 테스트, 장애 발생을 대비한 Failover-Failback 테스트 등 테스트 범위와 우선순위를 결정해야 한다. 모든 서비스를 테스트하면 좋겠지만, 보통 서비스 개발이나 유지 보수 때는 테스트를 위한 시간이 그다지 충분하지 않다. 그러므로 중요도와 테스트 목적에 맞는 우선순위와 그 범위를 정한다.• 범위가 정해지면 해당 시스템의 소프트웨어적인 구조와 하드웨어적인 구조를 분석한다.
2	테스트 계획	<ul style="list-style-type: none">• 언제, 누가, 어떤 방법으로, 어디서 테스트할 것인지 정하는 단계다.• 테스트 수행에는 많은 내/외부 인력이 필요하며 많은 준비사항이 있으므로 테스트 계획이 필수다.• 테스트에 필요한 인력과 역할은 [표 1-2]를 참고한다.
3	테스트 환경 구축	<ul style="list-style-type: none">• 테스트 단계 내에서 테스트 환경 구축을 언제 수행할지는 그리 중요하지 않을 수 있다. 자주 테스트를 수행하는 곳에서는 테스트 전용 서버팜 (Serverfarm)이 이미 구성되어 있기 때문이다. 요즘은 클라우드 환경의 테스트 팜을 구성하는 경우도 있다.• 테스트 환경을 구축할 때 가장 중요한 것은 부하발생기와 테스트 대상 서버 사이의 네트워크가 최단 구간 안에 존재하게 하는 것이다. 중간에 많은 보안 장비와 스위치/라우터(Switch/Router) 등을 거치면 예상하지 못한 결과가 발생할 수 있다.
4	테스트 설계	<ul style="list-style-type: none">• 테스트 절차 및 테스트 시나리오를 작성하고 테스트 케이스 작성 및 스크립트를 구현하며 테스트에 필요한 데이터 셋(Dataset)을 준비하는 단계다.• 테스트에 필요한 데이터 셋을 준비하는 과정은 상당히 중요하다. 테스트에 필요한 충분히 많은 데이터가 준비되지 않는다면 의도하지 않은 결과가 나올 가능성이 높기 때문이다. 예를 들어, 어떤 시스템에서 실제로는 DB의 I/O에 의해 성능 저하가 발생한다고 가정했을 때, DB의 테스트 데이터가 충분히 입력되지 않고 접근 데이터가 고르지 않으면 실제 서비스 때보다 성능이 좋게 나올 가능성이 높다. 즉, 실제 환경과 비슷한 수준의 많은 데이터를 준비할수록 테스트 결과가 좀 더 실제 값과 비슷해진다.

단계	프로세스	내용
5	테스트 수행 및 결과 수집	<p>작성된 스크립트로 실제 테스트를 수행하는 단계다. 테스트 수행은 크게 두 부분으로 나누어진다.</p> <ul style="list-style-type: none"> • Pre-Test: Main-Test 전에 스크립트가 제대로 작성되었는지, 테스트 환경(서버/네트워크/보안 시스템/외부 연동 등)과 준비된 데이터 셋에 문제가 없는지 확인하기 위한 테스트다. Main-Test에는 많은 인력이 투입되므로 Pre-Test가 정상적으로 이루어지지 않으면 많은 인력이 불필요하게 대기하는 상황이 발생할 수 있으니 사전에 꼭 수행해야 한다. • Main-Test: 실제 테스트를 수행하는 단계로, 테스트 분석에 필요한 시스템 성능 자료를 수집한다. 통합 SMS 솔루션이 있으면 OS의 CPU, Memory, I/O 등의 정보는 쉽게 수집할 수 있다. 그렇지 않다면 별도의 시스템 정보 수집 스크립트를 이용해서 수집해야 한다. AP(Application Server)는 APM(Application Performance Management)과 같은 전용 모니터링 도구를 이용하면 많은 정보를 편리하게 수집할 수 있다.
6	테스트 분석	테스트 결과 자료와 시스템 성능 자료를 모아서 테스트 결과를 분석한다. 분석된 자료를 통해서 성능에 영향을 미치는 문제점을 찾는다.
7	문제점 수정 및 재테스트	테스트 분석에서 발견된 문제점을 개발팀(Development Team)이나 시스템 운영팀(System Engineering Team)에 전달하여 문제점을 수정하고 다시 한 번 테스트를 수행한다.
8	결과 리포트 작성	테스트 리포트는 테스트 목적에 따라 해당 목적을 가장 잘 표현할 수 있는 방식으로 작성하는 것이 좋다. 요약 리포트와 상세 리포트를 분리해서 상세 결과를 필요로 하는 부서와 요약 결과만을 필요로 하는 부서에 별도로 리포트를 제출하는 것도 좋은 방법이다.

[표 1-2] 테스트 인력 및 역할

테스트 인력	역할
Test Leader(PM)	전체적인 테스트 계획, 목적 수립, 시나리오 작성, 일정 관리와 인력 배치를 담당한다.
Test Scripter(Designer)	정의된 테스트 목적 및 범위에 따라 작성된 시나리오로 사이트(서비스)를 분석하고 이를 바탕으로 상세 케이스를 작성한다.
Test Operator	작성된 테스트 스크립트를 이용해서 실제 테스트를 수행한다.
Development Team	테스트에 필요한 데이터를 준비하고 애플리케이션을 모니터링하며 발견된 문제점과 개선점을 찾아낸다.
System Engineer(SE) Team	테스트 환경을 구축하고 시스템(OS, 네트워크, 스토리지 등)을 모니터링하여 발견된 문제점과 개선점을 찾아낸다.

테스트 인력	역할
외부 업체 지원 인력	시스템 운영팀(System Engineer Team)에서 모든 정보를 수집하고 분석하면 좋겠지만, 특정 솔루션이나 시스템은 외부 엔지니어의 도움을 받아야 하는 경우가 생긴다. 외부 엔지니어는 해당 솔루션(또는 시스템)의 전문가이므로 테스트 결과 분석에 많은 도움이 되지만 외부 인력이므로 일정 관리를 잘해야 한다.

1.3 대용량 성능 테스트

이 책에서 의미하는 ‘대용량 성능 테스트’란 매우 많은 가상 사용자Virtual User, Thread가 필요한 테스트나 매우 높은 웹 트랜잭션 처리량을 테스트하는 것을 의미한다. 즉, 대규모 장비가 필요한 테스트다. 대용량 성능 테스트를 진행하다 보면 결과에 많은 영향을 주는 요소들이 발생한다. 이것은 현재 수행한 테스트 결과가 믿을만한지, 어떤 요소가 한계점에 다다라서 결과가 왜곡되지 않았는지에 대한 고민에 빠지게 한다. 예를 들어, 여러 대의 부하발생기가 연결되면 부하발생기의 네트워크 구성에 따라 결과가 달라질 수 있으며, 애플리케이션 서버의 능력보다 네트워크 스위치의 성능이나 OS 설정값에 의해 결과가 매우 달라질 수 있다.

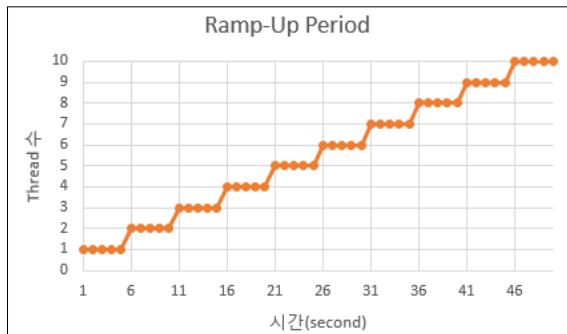
그러므로 좀 더 신뢰성 높은 결과값을 얻기 위해서는 결과에 영향을 미치는 요소들을 최대한 제거해야만 한다. 이 책에서는 기본적인 JMeter의 사용법과 함께 대용량 성능 테스트를 수행할 때 발생할 수 있는 문제점과 그에 대한 해결 방법을 실무에서 겪은 다양한 경험과 시행착오를 바탕으로 테스트 담당자들이 이런 시행착오를 겪지 않도록 하기 위한 팁과 노하우를 다룬다. 또한, 이를 바탕으로 테스트의 신뢰성을 높이고 테스트 결과값에서 의미 있는 내용을 찾는 방법을 설명한다.

1.4 주요 용어 및 개념

성능 테스트와 관련하여 자주 사용되는 용어와 그 개념을 간략하게 정리해 보자.

- **Active User** : 실제 서버에 연결된 상태로 요청을 처리 중인 사용자를 말한다.
- **InActive User** : 웹 브라우저에 결과 화면이 출력된 상태에서 화면의 내용을 읽거나 정보를 입력하고 있는 사용자다. 서버와의 세션Session 정보를 가지고 있지만 직접 접속하여 요청을 주고받는 상태가 아닌 사용자를 의미한다.
- **Concurrent User(Active User + InActive User)** : 보통 ‘동시 접속 사용자 수’라고 표현한다. 일반적으로 사용자 수의 많고 적음을 표현하는 값으로, 성능 테스트에서 가상 사용자 수를 결정하는 기준이 된다. 서비스 유형과 시간에 따라 그 비율이 달라지긴 하지만, 일반적으로 Active User와 InActive User 비율이 1:10 정도다.
- **Virtual User** : 가상 사용자 수로, JMeter에서는 Thread 수로 표현하기도 한다.
- **Ramp-Up Period** : Thread(Virtual User) 생성에 걸리는 시간을 의미한다. Ramp-Up Period 동안 차례대로 Thread를 생성한다. [그림 1-1]은 Ramp-Up Period를 이해하기 쉽도록 작성한 그래프다.

[그림 1-1] Ramp-Up Period에 따른 시간별 Thread 수 변화

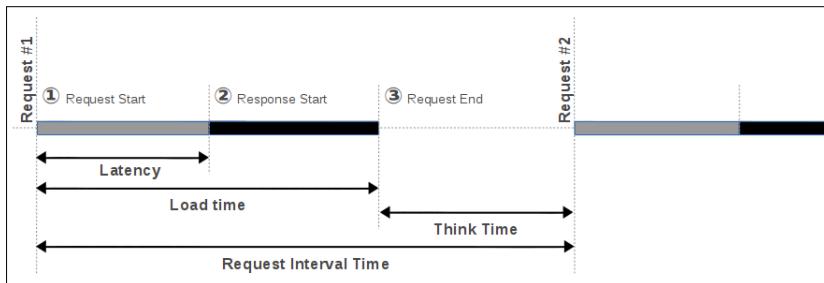


$$\text{Thread 수} = 10(\text{개}) / \text{Ramp-Up Period} = 50(\text{초})$$

이 그래프는 ‘10개의 Thread를 50초 동안 차례대로 생성하라’는 의미다. 즉, 5초(50초/10개)마다 Thread를 하나씩 생성하는 것과 같은 의미다.

- **Throughput** : 단위 시간당 대상 서버(웹서버, WAS, DB 등)에서 처리되는 요청의 수를 말한다. JMeter에서는 시간 단위를 보통 TPS(Transaction Per Second)로 표현한다.
- **Response Time/Load Time** : 응답시간 또는 처리시간이라고 표현한다. 요청을 보낸 후 응답이 완료되어 사용자 화면에 출력될 때까지의 시간을 나타낸다. 시스템의 성능을 평가하는 지표로 주로 사용한다.
- **Latency** : 요청을 보낸 후 데이터를 받기 시작할 때까지 시간이다.
- **Think Time** : 하나의 요청에 응답을 수신하고 다음 요청을 보낼 때까지 시간을 의미한다. 테스트에서 실제 사용자의 사용 패턴과 유사한 패턴을 구현하기 위해서는 이 Think Time을 적절히 적용해야 한다.
- **Request Interval Time** : 요청을 보낸 후 다음 요청을 보낼 때까지 시간을 의미한다.
- **Load Time vs Latency** : [그림 1-2]는 Load Time/Latency/Think Time/Request Interval Time의 관계를 이해하기 쉽도록 그림으로 나타낸 것이다.

[그림 1-2] 시간 관계도



[그림 1-2]를 보면 항상 $\text{Load Time} \geq \text{Latency}$ 가 성립된다. 두 개를 왜 나눠 놨을까? 이것은 Latency와 Load Time을 구분함으로써 성능을 분석할 때 요긴하게 사용할 수 있다.

A와 B 사이트에 동일한 크기(10MB 정도)의 파일을 올려놓고 다운로드 테스트를 진행한다고 가정하자. A 사이트와 B 사이트의 결과를 비교해 보니 B 사이트의 Load Time이 2배 이상 컸다. 하지만 Latency는 거의 비슷했다. 이렇게 차이가 나는 이유는 무엇일까?

Load time에서 Latency를 빼면 데이터를 전송받는 데 걸리는 시간을 나타낸다. 즉, B 사이트가 A 사이트보다 데이터를 내려받는 속도가 느리다고 볼 수 있다. 따라서 B 사이트는 처리량을 늘리기 위해 웹 서버를 튜닝하기보다는 네트워크의 대역폭(Bandwidth)을 늘리는 것을 고려해야 한다.

성능 테스트 사전 작업

실제 대용량 웹 성능 테스트를 수행하는 데 필요한 사전 준비 작업을 알아본다. 웹 성능을 테스트하기 위한 Test Plan을 만들어 보고, 웹 성능 테스트를 하기 위해 꼭 필요한 JMeter의 Element 사용법과 특징을 간단한 예제를 통해서 자세히 알아본다.

2 | JMeter 설치와 환경 설정

JMeter는 아파치 프로젝트 Apache Project에 속한 오픈소스로, [아파치 소프트웨어 재단 홈페이지](#)⁰¹에서 내려받을 수 있다. 다운로드 페이지로 가면 Binaries와 Source 두 가지 버전을 확인할 수 있다. JMeter를 변경 없이 그대로 사용하려면 Binaries 버전을 내려받고, 소스 코드를 수정 또는 추가하거나 컴파일해서 사용하려면 Source 버전을 내려받는다. Source 버전을 이용해서 컴파일한 후 사용하는 방법은 나중에 알아보고 우선 Binaries 버전을 내려받자.

2.1 실행 환경

JMeter는 순수 자바 애플리케이션이므로 Java JDK(JRE)만 설치되어 있으면 구동하는 데 문제없다. 최신 버전의 JMeter는 Java JDK 6 이상⁰²이 필요하다.

JMeter는 별도의 설치 과정 없이 압축 파일을 풀고 apache-jmeter-2.xx/bin 디렉터리에 있는 시작 명령어만 실행하면 바로 구동할 수 있다.

[표 2-1] 디렉터리 구조

디렉터리	설명
apache-jmeter-2.11\bin	JMeter를 실행하기 위한 실행 파일과 설정 파일이 있는 디렉터리다.
apache-jmeter-2.11\docs	API 관련 문서 디렉터리다.
apache-jmeter-2.11\extras	추가 유ти리티가 있는 디렉터리다.
apache-jmeter-2.11\lib	JMeter Components나 플러그인을 실행하는 데 필요한 유ти리티와 Dependency Jars가 있는 디렉터리다.
apache-jmeter-2.11\lib\ext	JMeter에서 사용하는 Components와 플러그인이 있는 디렉터리로, 기본으로 제공되는 Components 외에 추가로 설치된 Component나 플러그인도 이 디렉터리에 두면 JMeter가 구동될 때 자동으로 참조한다.

01 : https://jmeter.apache.org/download_jmeter.cgi

02 : <http://www.oracle.com/technetwork/java/javase/downloads>

디렉터리	설명
apache-jmeter-2.11\licenses	non-ASF 소프트웨어의 라이선스 정보가 담겨 있는 디렉터리다.
apache-jmeter-2.11\printable_docs	도움말 문서가 있는 디렉터리다.

[표 2-2] Linux/Unix 실행 명령어

명령어	설명
jmeter	GUI 모드로 실행하기 위한 명령어다(Default).
jmeter-server	Server 모드로 실행된다(Server 모드는 분산 테스트 설정에서 자세히 설명한다).
shutdown.sh	non-GUI 모드로 실행할 때 정상 종료(Gracefully)하게 하는 명령어다.
stoptest.sh	non-GUI 모드로 실행할 때 즉시 종료(Abruptly)하게 하는 명령어다.

[표 2-3] Windows 실행 명령어

명령어	설명
jmeter.bat	• GUI 모드로 실행하기 위한 명령어다(Default).
jmeter.cmd	• jmeter.bat으로 실행하면 cmd 창이 뜬 상태로 실행되고, jmeterw.cmd로 실행하면 cmd 창 없이 실행된다. • 중간에 출력되는 JMeter 메시지를 보려면 jmeter.bat으로 실행한다.
jmeter-n.cmd	• non-GUI 모드로 실행하기 위한 명령어다. • non-GUI 모드로 실행해야 할 때 추가 인자로 실행될 JMX 파일명을 입력한다. 예) c:\> jmeter-n.cmd test.jmx
jmeter-n-r.cmd	jmeter-n.cmd와 기능은 같지만, local에서 실행되는 것이 아니라 remote_hosts에 등록된 jmeter-server를 이용해서 실행된다.
jmeter-t.cmd	jmeterw.cmd와 같이 GUI 모드로 실행된다. 단, 입력 인자로 JMX 파일을 입력해야 한다.
jmeter-server.bat	Server 모드로 실행된다.
shutdown.cmd	non-GUI 모드로 실행했을 때 정상 종료하게 하는 명령어다.
stoptest.cmd	non-GUI 모드로 실행했을 때 즉시 종료하게 하는 명령어다.

2.2 JVM 옵션 설정

JVM 옵션을 변경해서 실행하려면 JVM_ARGS 변수값을 설정한다.

Windows

jmeter.bat 파일에 다음을 추가하고 실행한다.

```
set JVM_ARGS="-Xms1024m -Xmx1024m -Dpropname=propvalue"
```

Linux/Unix

항상 같은 설정값을 이용한다면 jmeter 파일에 변수값을 설정해도 되지만, 실행할 때마다 설정값을 조금씩 수정해야 하는 상황이라면 다음과 같이 실행 시 변수값을 설정하고 실행할 수도 있다.

```
JVM_ARGS="-Xms1024m -Xmx1024m" jmeter -t test.jmx
```

2.3 Overriding Property

Java System Property와 JMeter의 Property, Logging Property는 jmeter.properties 파일에 설정되어 있다. JMeter를 실행하면 해당 파일을 읽어 들여서 Property를 설정한다. 하지만 변경이 잦을 때는 파일을 수정하기보다는 커맨드라인에서 Property 값을 재정의하는 것이 편하다.

[표 2-4] Property 옵션

옵션	설명
-D<Prop_name>=<value>	Java System Property 값을 정의한다.
-J<Prop_name>=<value>	Local JMeter Property를 정의한다.
-G<Prop_name>=<value>	모든 remote_hosts에 전달될 Property를 정의한다.

옵션	설명
-G<property_file>	Property 내용이 저장된 파일을 remote_hosts에 전송한다.
-L<category>=<priority>	카테고리(Category)별로 로깅 수준(Level)을 결정한다.
-L<priority>	최상위 로깅 수준으로 설정할 수 있다.

[예제]

```
jmeter -Duser.dir=/home/mstover/jmeter_stuff -Jremote_hosts=127.0.0.1
-Ljmeter.engine=DEBUG
```

2.4 non-GUI Mode

non-GUI 모드는 커맨드라인 모드라고도 한다. Linux/Unix에서는 GUI를 실행할 수 없는 환경일 때가 종종 있다. 이럴 때 non-GUI 모드를 사용한다. 그러나 결과를 그래프나 수치로 바로 확인하면서 작업하면 테스트 도중에 발생할 수 있는 문제점이나 비정상 결과값을 즉시 확인할 수 있으므로 가능하면 GUI 모드를 사용하는 것이 좋다.

[표 2-5] non-GUI 옵션

옵션	설명
-n	non-GUI 모드를 실행하는 옵션이다.
-t <testplan name>	테스트에 사용될 Test plan 파일명을 입력한다.
-l <logfile name>	결과(Sample Result)가 저장될 로그 파일명을 입력한다.
-j <jmeter log name>	JMeter 로그 정보가 저장될 파일명을 입력한다. (Default: jmeter.log)
-r	JMeter Property 중 remote_hosts에 설정된 jmeter-server를 실행한다.
-R <list of remote servers>	remote_hosts에 설정된 jmeter-server가 아니라 직접 jmeter-server를 지정해서 테스트할 수 있다.
-H <proxy server host>	프록시 서버(Proxy Server)를 이용해서 접속할 때 해당 프록시 서버의 호스트(host)를 설정한다.
-P <proxy server port>	프록시 서버를 이용해서 접속할 때 해당 프록시 서버의 포트를 설정한다.

[예제]

```
jmeter -n -t test.jmx -l log.jtl -j my_jmeter.log -H my.proxy.server -P 8000
```

[실행 결과]

```
jacojang@jacojang-Desktop:~/apache-jmeter-2.11/bin$ ./jmeter -n -t test.jmx
Creating summariser <summary>
Created the tree successfully using test.jmx
Starting the test @ Sat Apr 05 17:50:42 KST 2014 (1396687842690)
Waiting for possible shutdown message on port 4445
summary +    72 in    17s =   4.3/s Avg:   653 Min:   489 Max:  1006 Err:      0
(0.00%) Active: 3 Started: 3 Finished: 0
summary +   102 in  24.3s =   4.2/s Avg:   660 Min:   603 Max:  1008 Err:      0
(0.00%) Active: 0 Started: 3 Finished: 3
summary =   174 in    41s =   4.3/s Avg:   657 Min:   489 Max:  1008 Err:      0
(0.00%)
Tidying up ...  @ Sat Apr 05 17:51:23 KST 2014 (1396687883952)
... end of run
```

3 | 간단한 Test Plan 작성하기

JMeter에서는 테스트 스크립트를 ‘Test Plan’이라고 표현한다. 이 장에서는 웹 서버를 테스트하기 위한 간단한 Test Plan을 만들어 본다. 설명한 대로 따라 하면 쉽게 만들 수 있다.

JMeter는 부하 발생을 목적으로 하는 프로그램이어서 외부 웹 서버에 접속해서 테스트하면 외부 서버에 부하가 발생하거나 내부 네트워크 트래픽에 과부하를 줄 수 있으므로 자신의 PC에 테스트 타깃 서버(Target Server)를 만들어 놓고 테스트를 진행하는 것이 좋다.

테스트에 필요한 URL을 테스트할 수 있는 WAR 파일은 다음 주소에서 내려받을 수 있다. 톰캣을 설치한 후 내려받은 파일을 배포하면 deploy 이 책에 사용하는 모든 테스트 URL을 사용할 수 있다.

<http://www.jacojang.com/jmeter/jmeter.war>

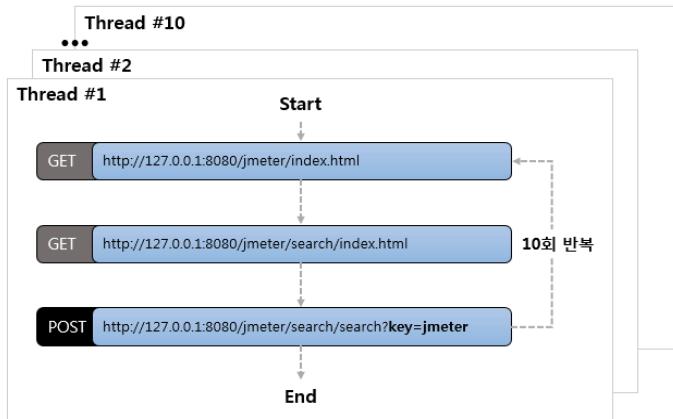
톰캣 설치 방법과 WAR 파일 배포 방법은 [부록](#)을 참고하기 바란다.

3.1 Test Plan 작성

[그림 3-1]과 같이 10명의 사용자가 다음 테스트 페이지를 10번 반복 요청한다고 가정한다.

- <http://127.0.0.1:8080/jmeter/index.htm>(Method: GET)
- <http://127.0.0.1:8080/jmeter/search/index.html>(Method: GET)
- <http://127.0.0.1:8080/jmeter/search/search?key=jmeter>(Method: POST)

[그림 3-1] 테스트 구성도



Test Plan은 다음 순서로 작성한다.

- Thread Group 추가 및 설정 : 가상 사용자(Thread)의 숫자와 반복 횟수, 반복 시간을 설정한다.
- Config Element⁰¹ 추가 및 설정 : 이번 장에서는 HTTP Request Defaults만을 사용해서 작업한다.
- HTTP Request Sampler⁰² 추가 및 설정 : 테스트 페이지 목록에 해당하는 세 개의 Sampler를 추가한다.
- Listener⁰³ 추가 및 설정 : 테스트 결과를 보기 위해서는 Listener를 꼭 추가해야 한다. View Result Tree와 Summary Report를 추가한다.

모든 작업이 완료되면 [그림 3-2]와 같은 형태가 된다.

01 JMeter에서는 Test Plan 아래에 추가되는 노드를 Element라고 한다.

02 Sampler는 실제로 서버에 요청을 보내는 Element를 말한다.

03 Listener는 테스트 결과를 보기 위한 Element를 말한다.

[그림 3-2] Test Plan 계층 구조

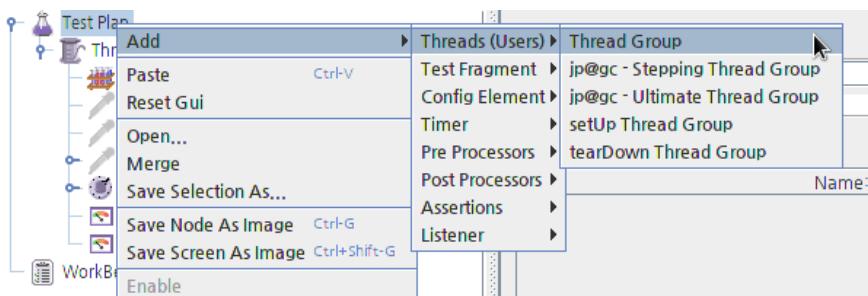


3.1.1 Thread Group 추가 및 설정

Thread Group 추가

Test Plan에서 ‘Add → Threads (Users) → Thread Group’을 선택하여 추가한다.

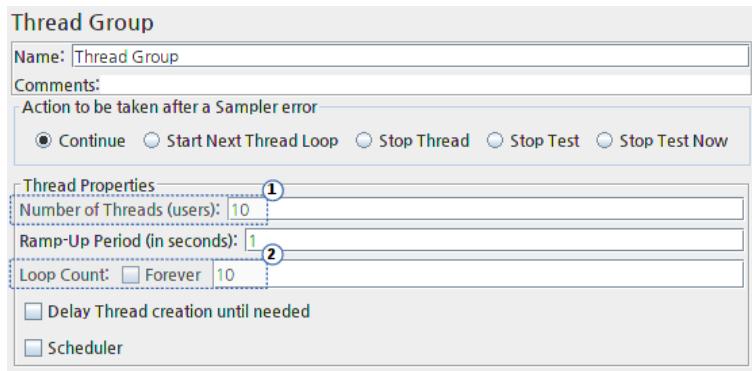
[그림 3-3] Thread Group 추가



Thread Group 설정

- ① Number of Threads(users): 10은 10개의 Thread를 생성하라는 의미다.
- ② Loop Count: 10은 10명이 10번씩 Test Plan을 반복하라는 의미이므로 ‘10(명)
× 10(반복) = 100회’를 수행한다.

[그림 3-4] Thread Group 설정

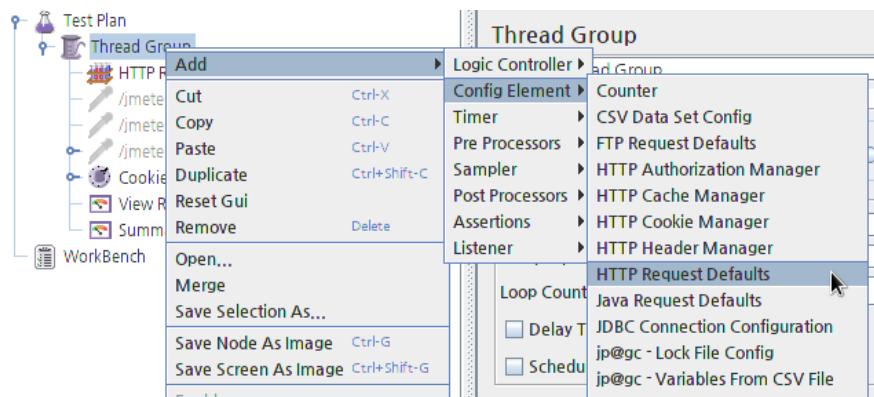


3.1.2 Config Element(HTTP Request Defaults) 추가 및 설정

Config Element 추가

Thread Group에서 'Add → Config Element → HTTP Request Defaults'를 선택하여 추가한다.

[그림 3-5] Config Element 추가



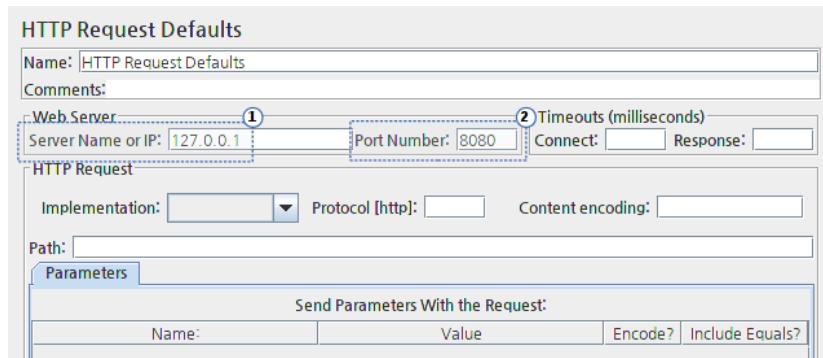
Config Element 설정

HTTP Request Sampler에 설정되는 정보 중에 중복되는 부분을 HTTP Request Defaults에 설정하면 다음에 나올 HTTP Request Sampler의 설정을 간소화할 수 있고, 변경 사항이 생겼을 때 작업량이나 오류 발생이 줄어드는 장점이 있다.

① Server Name of IP: 127.0.0.1

② Port Number: 8080

[그림 3-6] HTTP Request Defaults 설정



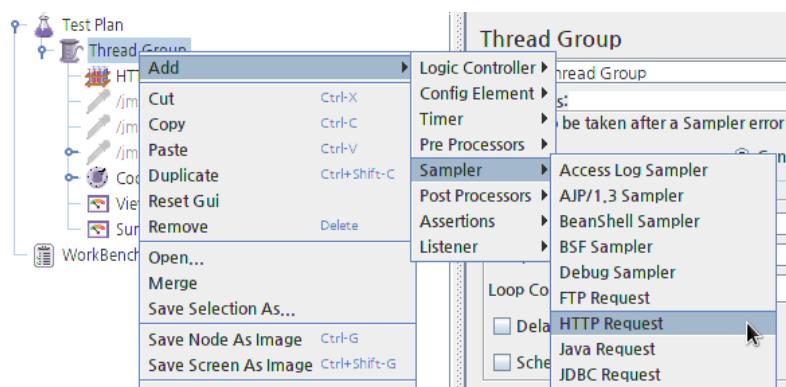
3.1.3 HTTP Request Sampler 추가 및 설정

웹 서버에 요청을 보낼 경우에는 HTTP Request라는 Sampler를 이용한다.

HTTP Request Sampler 추가

Thread Group에서 ‘Add → Sampler → HTTP Request’를 선택하여 추가한다. 테스트 페이지 목록에 3개의 URL이 있으므로 3개의 Sampler가 필요하다. 따라서 이 작업을 3번 반복한다.

[그림 3-7] HTTP Request Sampler 추가



HTTP Request Sampler 설정

① Name: Name은 알맞은 값으로 수정한다. 다음에 설명할 Listener에서 결과를 정리할 때 이 Name을 기준으로 보여주므로 구분되는 이름으로 설정해야 한다.

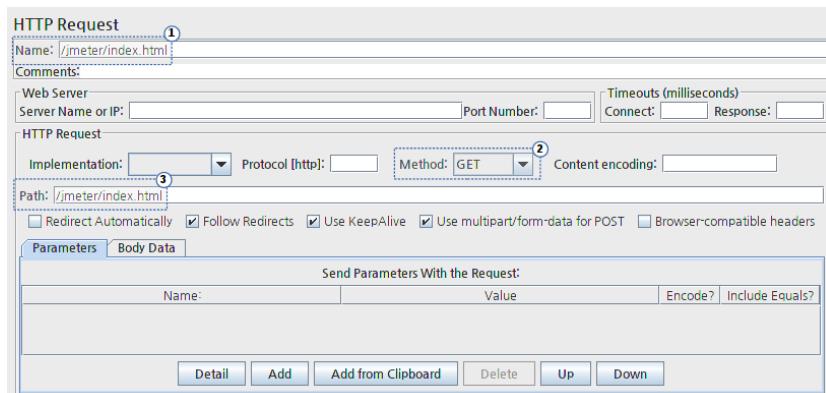
- /jmeter/index.html
- /jmeter/search/index.html
- /jmeter/search/search

② Method: HTTP Request가 생성될 때 기본적으로 GET으로 설정된다. /jmeter/search/search만 POST 방식을 사용하므로 이 Sampler만 POST로 변경한다.

③ Path: 세 개의 HTTP Request Sampler의 Path에 각각 다음 값을 입력한다.

- /jmeter/index.html
- /jmeter/search/index.html
- /jmeter/search/search

[그림 3-8] HTTP Request Sampler 추가



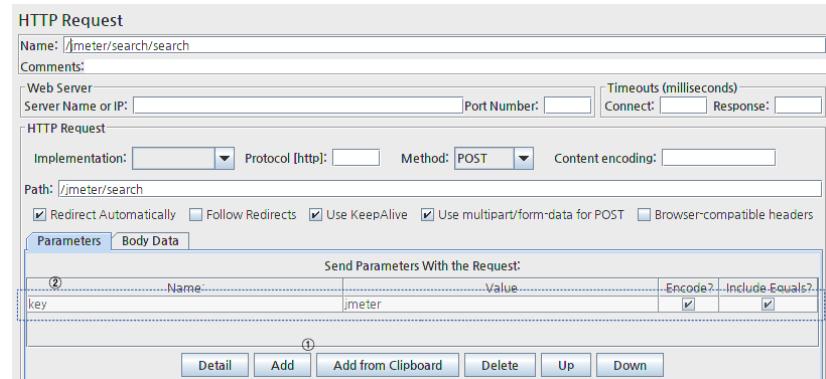
Parameter 입력

/jmeter/search/search에는 검색 키워드를 함께 보내기 위해서 Parameters를 추가한다.

① Add 버튼을 누른다.

② Name : key / Value : jmeter

[그림 3-9] HTTP Request Parameter 추가



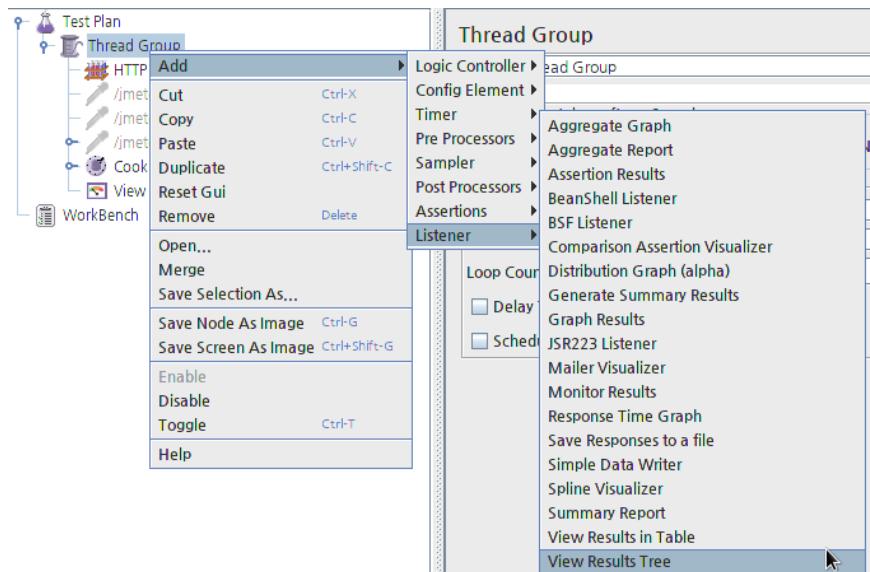
3.1.4 Listener 추가 및 설정

Listener는 Sampler의 요청에 대한 결과를 수집해서 그 결과값을 보여주는 Element를 의미한다. 요청을 보낸 후 성공/실패, 응답시간, 응답 메시지 등을 확인하려면 반드시 추가해야 한다. 여기서는 View Results Tree와 Summary Report를 추가한다.

View Results Tree 추가

Thread Group에서 ‘Add → Listener → View Results Tree’를 선택하여 추가한다.

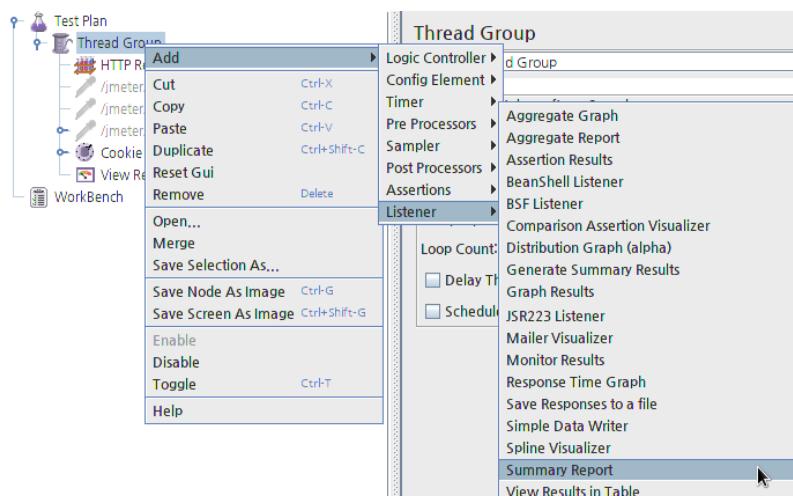
[그림 3-10] View Results Tree 추가



Summary Report 추가

Thread Group에서 ‘Add → Listener → Summary Report’를 선택하여 추가한다.

[그림 3-11] Summary Report 추가



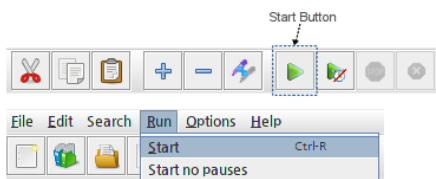
View Results Tree와 Summary Report는 별도의 설정이 필요 없다.

3.2 테스트 실행 및 결과 확인

GUI 모드에서 테스트를 실행하는 방법은 다음 세 가지가 있다.

- 메뉴 바에서 'Start' 버튼을 클릭한다.
- 메뉴에서 'Run → Start'를 선택한다.
- 단축키 'Ctrl-R'를 누른다.

[그림 3-12] 테스트 실행 화면

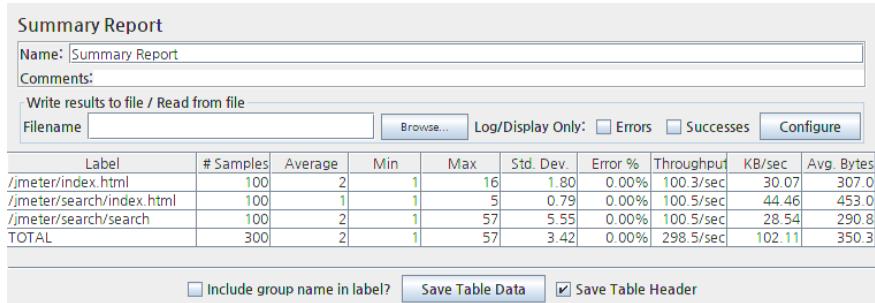


여기에서는 결과보는 방법을 간단히 설명하고 9.1 결과 분석에서 자세한 내용을 다루기로 한다.

3.2.1 Summary Report

테스트 결과를 요약Summary해서 보여준다. 통합된 요청량Sample Count, 응답시간 Response Time/Load Time, 오류율, 단위 시간당 처리량Throughput 등을 확인할 수 있다. Label 부분이 HTTP Request Sampler에서 설정한 Name이다. 동일한 Name을 사용하면 구별이 어려우므로 주의한다. 응답시간은 Average, Min, Max 부분으로 1/1000초 단위로 표시된다.

[그림 3-13] Summary Report 화면



3.2.2 View Results Tree

각 결과의 요청/응답Request/Response를 상세하게 살펴볼 수 있는 Listener다. 초기에 스크립트를 만들고 정상적으로 처리되는지 확인할 때 용이하다. Pre-Test 할 때 사용하므로 기능에 대해서 잘 알아두는 것이 좋다.

[그림 3-14]는 각 Sampler의 결과를 Tree 형태로 보여준다.

- ① Sampler 목록 : 이중에서 하나를 선택하면 오른쪽 패널에 해당 Sampler의 상

세 정보가 출력된다. 상세 정보 패널은 세 개의 탭으로 구성된다.

[그림 3-14] View Results Tree 화면

Name: View Results Tree
Comments:
Write results to file / Read from file
Filename
Browse... Log/Display Only: Errors Successes Configure

Sampler result Request Response data

Thread Name: Thread Group 1-1
Sample Start: 2014-02-07 15:03:59 KST
Load time: 57
Latency: 57
Size in bytes: 287
Headers size in bytes: 173
Body size in bytes: 114
Sample Count: 1
Error Count: 0
Response code: 200
Response message: OK

Response headers:
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Language: ko-KR
Content-Length: 114
Date: Fri, 07 Feb 2014 06:03:59 GMT

Raw Parsed

② Sampler result : 해당 Sampler의 요청 결과를 보여준다. 성공/실패 여부를 포함해서 응답시간과 크기Size 등을 보여준다.

[그림 3-15] Sampler result 화면

Sampler result Request Response data

Thread Name: Thread Group 1-2
Sample Start: 2014-04-06 16:37:30 KST
Load time: 2
Latency: 2
Size in bytes: 453
Headers size in bytes: 173
Body size in bytes: 280
Sample Count: 1
Error Count: 0
Response code: 200
Response message: OK

Response headers:
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Language: ko-KR
Content-Length: 280
Date: Sun, 06 Apr 2014 07:37:30 GMT

HTTPSampleResult fields:
ContentType: text/html;charset=ISO-8859-1
DataEncoding: ISO-8859-1

Raw Parsed

③ Request : 해당 Sampler가 웹 서버에 보낸 Request 정보를 볼 수 있다.

[그림 3-16] Request 화면

Sampler result Request Response data

GET http://127.0.0.1:8080/jmeter/search/index.html

[no cookies]

Request Headers:

Connection: keep-alive
Host: 127.0.0.1:8080
User-Agent: Apache-HttpClient/4.2.6 (java 1.5)

④ Response data : 해당 Sampler의 요청에 대한 응답 메시지를 보여준다. 웹 서버로 요청을 보냈으므로 Response data는 html 문서로 출력된다.

[그림 3-17] Response data 화면 뷰

Sampler result Request Response data

```
<html>
<head>
    <title>Search Index Page</title>
</head>
<body>
<h1>
    Search Index Page
</h1>

<form id="search_form" name="search_form" method="POST" action="search">
    Search Keyword : <input type="text" name="key" type="text" /> <input type="submit" value="se
    arch" />
</form>

</body>
</html>
```

4 | Element의 사용법과 특징

이 장에서는 웹 성능 테스트를 하기 위해 꼭 필요한 Element의 사용법과 특징을 간단한 예제를 통해서 자세히 알아본다.

JMeter에는 여러 그룹의 Element가 존재하는데, 총 9개 그룹으로 나눌 수 있다. 각 그룹은 [표 4-1]과 같다.

[표 4-1] Element 그룹

그룹	아이콘	설명
Threads (Users)		가상 사용자의 숫자와 테스트 시간을 결정한다. - Thread Group, setUp Thread Group, tearDown Thread Group 등
Config Element		변수나 Sampler의 기본값을 설정하거나 파일로부터 정보를 읽어 들일 때 사용한다. - CSV Dataset Config, HTTP Request Defaults, Random Variable 등
Logic Element		반복, 분기, 랜덤 선택 등 Test Plan에 로직을 적용할 때 사용한다. - If Controller, Random Controller, Loop Controller 등
Timer		Think Time을 적용할 때 사용한다. - Constant Timer, Constant Throughput Timer 등
Pre Processors		Sampler가 실행되기 전에 실행되는 Element로, Sampler 실행에 필요한 변수를 설정하거나 URL을 Re-Write할 때 사용한다. - BeanShell PreProcessor, User Parameters 등
Post Processors		Sampler가 실행된 후에 바로 실행되는 Element다. 주로 Sampler의 결과값에서 특정 문자열을 찾아서 변수에 저장하는 용도로 사용한다. - Regular Expression Extractor, Bean Shell PostProcessor 등
Samplers		웹 서버로 요청을 보내는 역할을 한다. - HTTP Request, TCP Sampler, Bean shell Sampler 등
Assertions		Sampler를 통해 받아온 결과값이 정상인지 확인하는 Element다. HTTP 응답은 "200 OK"라는 정상 메시지를 받았어도 실제 본문 내용은 실패인 경우가 많다. 이 때 Assertion을 이용하면 오류로 처리할 수 있다. - Size Assertion, Duration Assertion 등
Listeners		결과를 수집해서 통계나 그래프로 보여주는 Element다. - Summary Report, View Results Tree 등

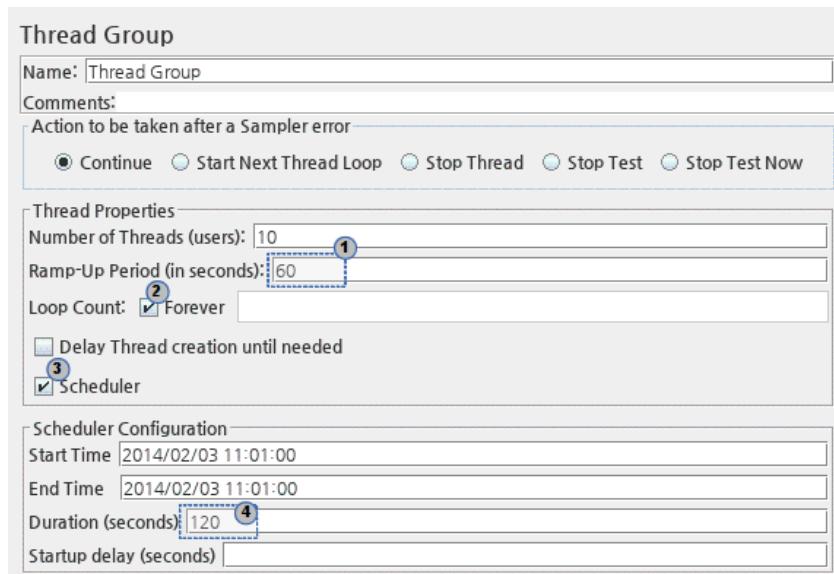
4.1 Thread Group

Test Plan을 작성할 때 가장 먼저 추가한다. Test Plan 바로 하위 단계에만 존재하며 여러 개의 Thread Group이 공존할 수 있다. Thread Group 하위에 또 다른 Thread Group을 추가할 수는 없다.

4.1.1 예제

총 2분(120초) 동안 테스트를 수행하고, Thread는 10개를 생성한다. 한 번에 생성하지 않고 1분(60초) 동안 나누어서 차례대로 생성하고 나머지 1분 동안은 생성된 Thread 숫자를 유지하며 테스트한다.

[그림 4-1] Thread Group 설정



[그림 4-1]은 시나리오대로 Thread Group에 값을 설정한 것으로, 세부 내용은 다음과 같다.

- ① Ramp-Up Period의 60은 Number of Threads에 설정된 숫자만큼의 Thread를 60초 동안 나눠서 차례대로 생성하라는 의미다.
- ② 반복 숫자가 아닌 시간으로 설정하기 위해 Loop Count의 Forever를 체크한다.
- ③ Scheduler를 체크하면 Scheduler Configuration을 설정할 수 있다.
- ④ Duration을 120으로 설정하면 총 120초 동안 테스트한다.

4.1.2 설정 레퍼런스

Thread Group의 설정 요소들을 좀 더 자세히 살펴보자.

Action to be taken after a Sampler error

Thread Group 하위에 있는 Sampler에서 오류가 발생했을 때 어떻게 동작할지에 대한 옵션으로, 보통 Continue 옵션을 사용한다. 테스트 도중 오류가 발생했을 때 나머지 동작을 하지 않고 다시 처음으로 돌아가서 테스트하고 싶다면 Start Next Thread Loop를 선택한다.

Number of Threads

가상 사용자(Thread)의 수를 뜻하는 것으로, 얼마나 많은 가상 사용자를 생성할지 설정한다.

Ramp-Up Period

Ramp-Up Period 값을 설정한다. Ramp-Up Period에 대해서는 뒤에서 자세히 설명하겠다.

Loop Count

테스트를 반복하는 횟수다. 사용자가 Stop 버튼을 누르기 전까지 테스트를 반복하거나 Scheduler에 설정된 시간 동안 계속 반복하고 싶을 때는 Forever를 체크한다.

Delay Thread creation until needed

Number of Threads에 설정된 수만큼 Thread가 생성되면 테스트를 시작하게 하는 옵션이다.

Scheduler Configuration

Scheduler를 체크하면 Scheduler Configuration 정보가 활성화된다.

Start Time/End Time

특정 시간에 테스트를 진행하게 하는 옵션이다. Duration 값을 비워 놓아야 적용된다.

Duration

테스트 시간을 초 단위로 설정할 수 있게 한다. 이 값을 설정하면 Start Time/End Time보다 우선순위로 적용된다.

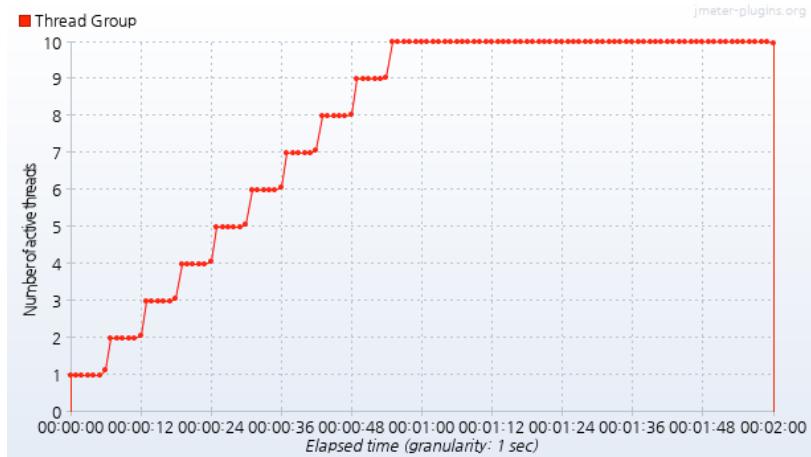
Startup Delay

Start 버튼을 누른 시점이 아닌 Startup Delay 설정 시간 이후에 테스트를 시작하게 한다. 여러 개의 Thread Group이 존재하고 Thread Group마다 시작되는 시점을 다르게 하고 싶을 때 유용하다.

4.1.3 Ramp-Up Period

앞의 Thread Group 예제에서 테스트 시간에 따른 Thread 숫자를 그래프로 그려보면 [그림 4-2]와 같다. 기본으로 제공되는 Thread Group의 Ramp-Up Period는 그다지 정교하지 못해서 목표 Thread 숫자를 설정된 시간 안에 차례대로 생성하는 방법만을 제공한다. 앞의 예제에서 10개의 Thread를 60초 동안 생성했으므로 계단처럼 단계적으로 생성된 것처럼 보이지만, Thread 숫자를 10개에서 100개로 늘리면 [그림 4-3]과 같이 선형으로 바뀐다.

[그림 4-2] 테스트 시간에 따른 Thread 숫자(Thread: 10, Ramp-Up: 60)



[그림 4-3] 테스트 시간에 따른 Thread 숫자(Thread: 100, Ramp-Up: 60)

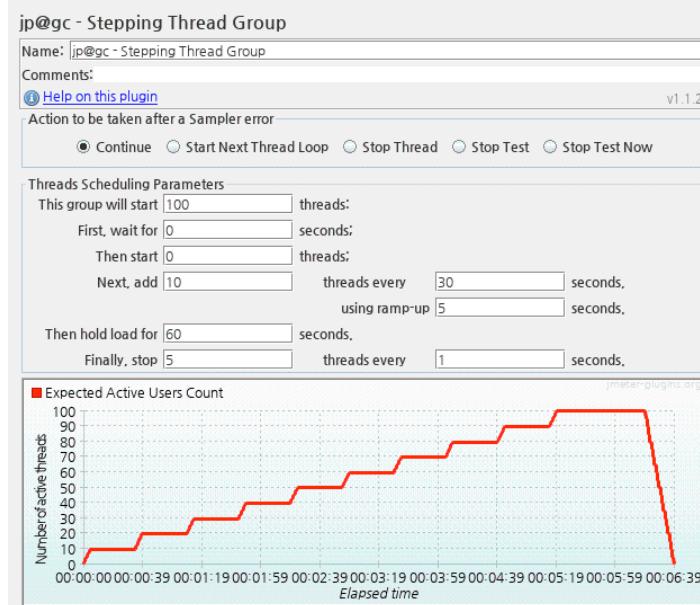


예를 들어, 10개의 Thread로 1분 동안 테스트하고, 다시 10개의 Thread를 추가해서 20개의 Thread로 1분 동안 테스트하도록 설정한다면 Thread Group의 Ramp-Up Period로는 테스트할 수 없다. 약간의 꼼수를 사용하면 여러 개의

Thread Group을 생성해서 Startup Delay를 각각 1분씩 추가하는 방법도 가능하지만, Test Plan이 불필요하게 복잡해진다.

이럴 때 좀 더 정교하게 Thread의 생성과 소멸을 설정하려면 JMeter Plugin⁰¹을 추가로 설치하고 jp@gc - Stepping Thread Group을 사용해 보길 추천한다. Ramp-Up Period뿐 아니라 초기 시간 및 일정 시간 간격 Thread 추가 등 좀 더 정교한 Thread 숫자 조절 기능을 제공한다.

[그림 4-4] jp@gc - Stepping Thread Group



4.2 HTTP Request Default

하나의 Test Plan을 만들면 보통 하나 이상의 HTTP Request Sampler를 사용하

01 · <http://jmeter-plugins.org>

는데, 각각의 Sampler에 IP, Port 등의 정보를 일일이 적어서 Test Plan을 작성할 수 있다. 하지만 일반적으로 패스나 파라미터 이외에 IP, Port 같은 항상 같은 값을 가지므로 중복해서 값을 쓰는 것은 상당히 비효율적이다.

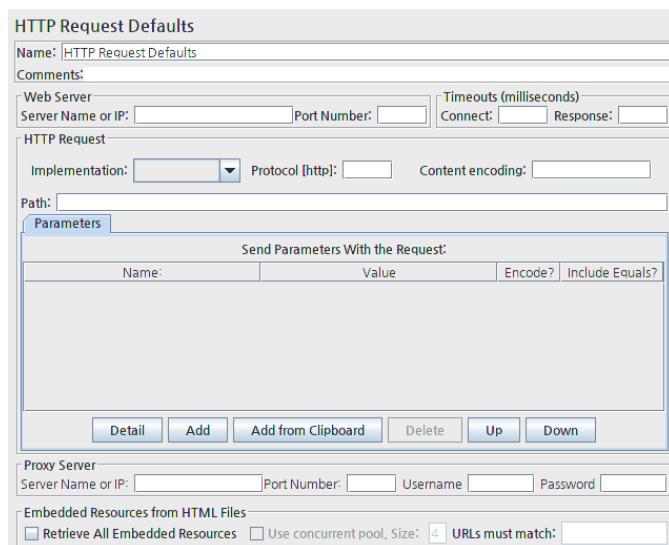
이러한 문제를 해결하기 위해서는 다음 두 가지 방법이 있다.

- 고정된 값을 변수 Variable로 처리한다.
- HTTP Request Defaults를 사용한다.

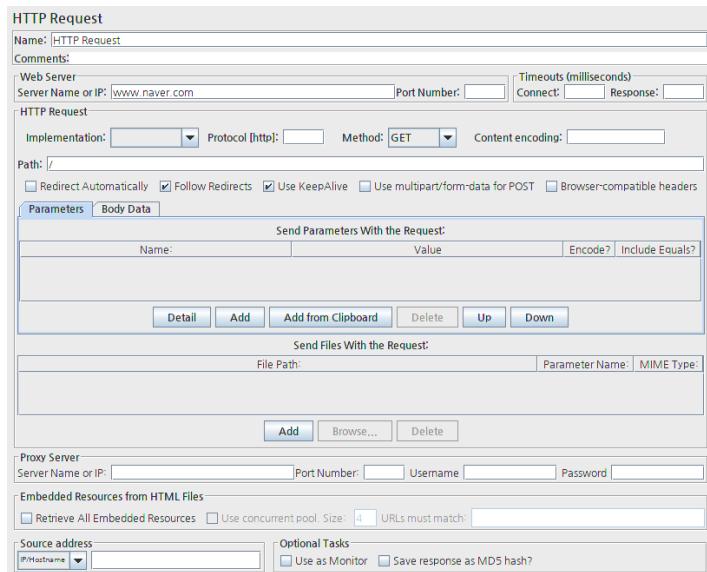
여기서는 HTTP Request Defaults를 이용하는 두 번째 방법을 살펴본다.

HTTP Request Defaults는 Element 그룹 중 Config Element에 속한다. 실제 HTTP Request를 보내는 것이 아니라 HTTP Request Sampler의 기본값을 설정하는 역할을 한다. [그림 4-5]와 [그림 4-6]처럼 HTTP Request Defaults와 HTTP Request는 형태가 비슷하다.

[그림 4-5] HTTP Request Defaults



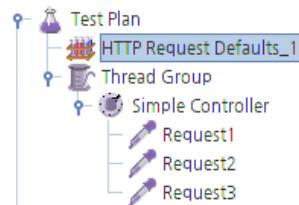
[그림 4-6] HTTP Request



적용 범위

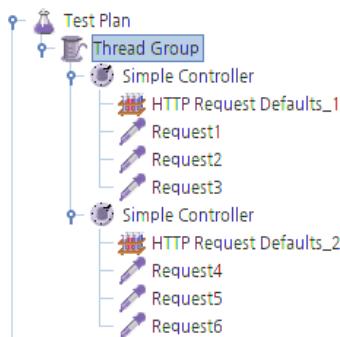
HTTP Request Defaults의 적용 범위는 HTTP Request Defaults가 존재하는 노드 이하에 영향을 미친다. 이 적용 범위 규칙은 대부분의 Config Element에 동일하게 적용된다. Test Plan 전체에 적용하려면 [그림 4-7]과 같이 HTTP Request Defaults를 최상위 노드에 위치시킨다. 따라서 Request1, Request2, Request3은 HTTP Request Default_1의 설정을 사용하게 된다.

[그림 4-7] 최상위 HTTP Request Defaults 적용



그룹별로 HTTP Request Defaults를 적용하고 싶다면 Simple Controller를 이용한다. [그림 4-8]과 같이 두 개의 HTTP Request Defaults를 생성하고 Simple Controller로 그룹을 만든다. 각 그룹에 HTTP Request Defaults를 삽입하면 해당 그룹의 HTTP Request Sampler에만 영향을 미친다. 따라서 Request1, Request2, Request3은 HTTP Request Default_1의 설정을 사용하고, Request4, Request5, Request6은 HTTP Request Default_2의 설정을 사용한다.

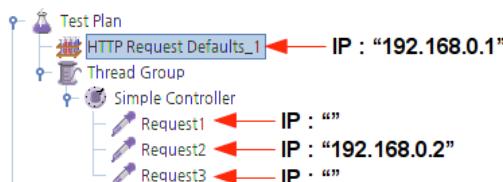
[그림 4-8] 그룹별 HTTP Request Defaults 적용



적용 우선순위

HTTP Request Defaults와 HTTP Request에 동일 필드값이 설정되어 있다면 HTTP Request에 설정된 값이 우선순위가 있다. 예를 들어, [그림 4-9]와 같이 설정되어 있을 때 Request1과 Request3은 192.168.0.1로 요청하고 Request2는 192.168.0.2로 요청하게 된다.

[그림 4-9] HTTP Request Defaults 우선순위



4.3 HTTP Request

웹 성능 테스트를 하면서 가장 많이 사용하게 되는 Element는 HTTP Request Sampler다. 이것은 웹 서버로 요청을 보내고 응답을 받는 역할을 한다. 기본 기능 및 사용법은 [3. 간단한 Test Plan 작성하기](#)에서 다루었으므로 참고하기 바란다. 여기에서는 자주 사용되지는 않지만, 특정 테스트 환경에서 많은 도움을 받을 수 있는 몇 가지 기능을 살펴본다.

4.3.1 예제

웹 브라우저에서 개발자 도구의 활용

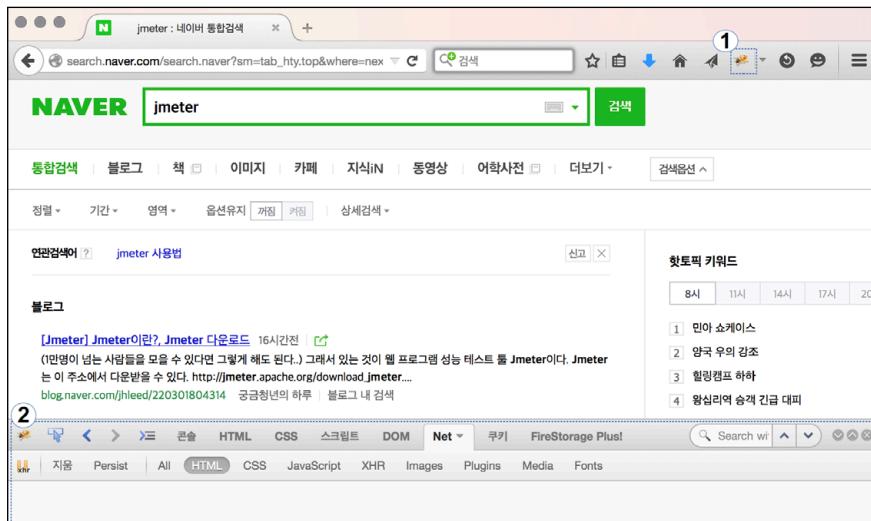
요즘 대부분 웹 브라우저에서는 웹 서버와 주고받는 내용을 분석하거나 자바스크립트 디버깅, DOM 분석 등을 할 수 있는 개발자 도구를 제공한다. 이를 이용하면 HTTP Request Sampler를 추가할 때도 많은 도움이 된다. 여기에서는 파이어폭스 브라우저의 플러그인인 '[Firebug](#)'⁰²라는 개발자 도구를 이용해 유명 포털 사이트의 검색 명령을 수행하는 HTTP Request Sampler 추가 방법을 알아본다.

웹 브라우저에서 기본으로 제공하는 개발자 도구도 유용하지만 필자는 Firebug의 Persistent 기능과 매개변수 템의 정보를 JMeter의 Parameters로 바로 복사해 넣을 수 있는 기능 때문에 Firebug를 주로 사용한다.

파이어폭스를 실행하고 ‘메뉴 → 도구 → 웹 개발 도구 → Firebug’를 선택하거나 [그림 4-10]의 ①에 보이는 벌레 모양 아이콘을 클릭하면 [그림 4-10]의 ②와 같이 Firebug 창을 확인할 수 있다.

02 <http://getfirebug.com/>

[그림 4-10] Firebug 개발자 도구 추가



이때 검색창에 검색어를 'JMeter'로 입력하고 검색하면 [그림 4-11]과 같은 화면을 볼 수 있다.

[그림 4-11] JMeter 검색 결과

NAVER							로그인	로그아웃																																																							
통합검색							검색	도움말	원본방역																																																						
정렬 · 기간 · 영역 · 음선유지							검색																																																								
연관검색어							신고																																																								
블로그							핫토픽 키워드																																																								
[JMeter] JMeter이란?, JMeter 다운로드 16시간전							8시	11시	14시	17시	20																																																				
(인생이 넘는 사람들을 모을 수 있다면 그렇게 해도 된다.) 그래서 있는 것이 웹 프로그램 성능 테스트 툴 JMeter이다. JMeter는 이 주소에서 다운받을 수 있다. http://jmeter.apache.org/download_jmeter.cgi ... blog.naver.com/jhleed/220301804314 궁금청년의 하루 블로그 내 검색							1 민아 쇼케이스	2 양국 우의 강조	3 할링캠프 하하	4 왕십리역 승객 긴급 대피																																																					
2 < > X 콘솔 HTML CSS 스크립트 DOM Net 키wi FireStorage Plus! Search within Net panel																																																															
[JMeter] JMeter 사용 가이드 2014.11.27																																																															
JMeter 사용 가이드 - 폐 차 - 1. 성능 테스트의 개요 1.1 성능테스트의 특징 1.2 성능테스트 의... [JMeter 2.1] JMeter 설치 2.1.1 JDK 6 다운로드 및 설치 2.1.2 JMeter 다운로드 및 설치 2.1.3 JMeter... blog.naver.com/smilemin/220193769549 Easy Peasy Lemon Squeezie 블로그 내 검색																																																															
www.naver.com																																																															
<table border="1"> <thead> <tr> <th>URL</th> <th>상태</th> <th>도메인</th> <th>사이즈</th> <th>Remote IP</th> <th>타이머린</th> </tr> </thead> <tbody> <tr> <td>▶ GET search.naver?sm=tab_0212</td> <td>200 OK</td> <td>search.naver.net</td> <td>53.5 KB</td> <td>125.209.230.167.80</td> <td>110ms</td> </tr> <tr> <td>▶ GET search1_0212.css</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>19.4 KB</td> <td>182.162.92.20.80</td> <td>47ms</td> </tr> <tr> <td>▶ GET search1_1127.css</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>18.5 KB</td> <td>182.162.92.20.80</td> <td>38ms</td> </tr> <tr> <td>▶ GET api_atcmp_0415.css</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>4.5 KB</td> <td>182.162.92.20.80</td> <td>33ms</td> </tr> <tr> <td>▶ GET sdyn.js?f=search/jss/</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>67.2 KB</td> <td>182.162.92.20.80</td> <td>57ms</td> </tr> <tr> <td>▶ GET nhnopen_map_2013</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>1.9 KB</td> <td>182.162.92.20.80</td> <td>30ms</td> </tr> <tr> <td>▶ GET maps_openapi_2013</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>6.6 KB</td> <td>182.162.92.20.80</td> <td>29ms</td> </tr> <tr> <td>▶ GET sn_image_0829.css</td> <td>200 OK</td> <td>sstatic.naver.net</td> <td>1.7 KB</td> <td>182.162.92.20.80</td> <td>34ms</td> </tr> </tbody> </table>							URL	상태	도메인	사이즈	Remote IP	타이머린	▶ GET search.naver?sm=tab_0212	200 OK	search.naver.net	53.5 KB	125.209.230.167.80	110ms	▶ GET search1_0212.css	200 OK	sstatic.naver.net	19.4 KB	182.162.92.20.80	47ms	▶ GET search1_1127.css	200 OK	sstatic.naver.net	18.5 KB	182.162.92.20.80	38ms	▶ GET api_atcmp_0415.css	200 OK	sstatic.naver.net	4.5 KB	182.162.92.20.80	33ms	▶ GET sdyn.js?f=search/jss/	200 OK	sstatic.naver.net	67.2 KB	182.162.92.20.80	57ms	▶ GET nhnopen_map_2013	200 OK	sstatic.naver.net	1.9 KB	182.162.92.20.80	30ms	▶ GET maps_openapi_2013	200 OK	sstatic.naver.net	6.6 KB	182.162.92.20.80	29ms	▶ GET sn_image_0829.css	200 OK	sstatic.naver.net	1.7 KB	182.162.92.20.80	34ms			
URL	상태	도메인	사이즈	Remote IP	타이머린																																																										
▶ GET search.naver?sm=tab_0212	200 OK	search.naver.net	53.5 KB	125.209.230.167.80	110ms																																																										
▶ GET search1_0212.css	200 OK	sstatic.naver.net	19.4 KB	182.162.92.20.80	47ms																																																										
▶ GET search1_1127.css	200 OK	sstatic.naver.net	18.5 KB	182.162.92.20.80	38ms																																																										
▶ GET api_atcmp_0415.css	200 OK	sstatic.naver.net	4.5 KB	182.162.92.20.80	33ms																																																										
▶ GET sdyn.js?f=search/jss/	200 OK	sstatic.naver.net	67.2 KB	182.162.92.20.80	57ms																																																										
▶ GET nhnopen_map_2013	200 OK	sstatic.naver.net	1.9 KB	182.162.92.20.80	30ms																																																										
▶ GET maps_openapi_2013	200 OK	sstatic.naver.net	6.6 KB	182.162.92.20.80	29ms																																																										
▶ GET sn_image_0829.css	200 OK	sstatic.naver.net	1.7 KB	182.162.92.20.80	34ms																																																										

검색 결과에는 이미지, CSS, JS 등의 결과도 함께 출력된다. 이중에서 HTTP Request에 추가될 내용은 ‘HTML’ 요청이므로 [그림 4-12]와 같이 필요한 요청만 추려낸다.

[그림 4-12] 검색 결과에서 HTML 요청 선택

URL	상태	도메인	사이즈	Remote IP	타임라인
▶ GET search.naver?sm=tab	200 OK	search.naver.com	53.5 KB	125.209.230.167:80	110ms

요청에 대한 상세 정보를 보려면 해당 라인을 클릭한다.[그림 4-13]처럼 해당 요청의 상세 정보를 볼 수 있다.

[그림 4-13] 검색 결과의 상세 정보

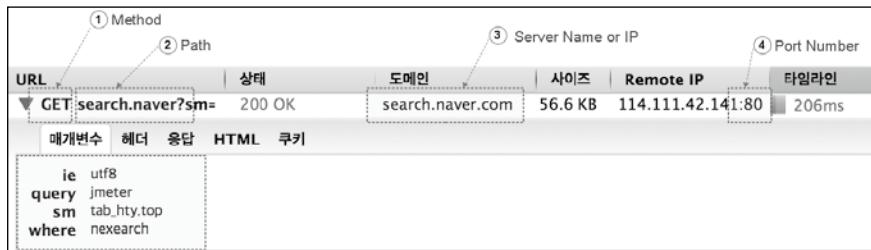
URL	상태	도메인	사이즈	Remote IP	타임라인
▼ GET search.naver?sm=tab	200 OK	search.naver.com	53.5 KB	125.209.230.167:80	110ms

Request parameters:

- ie utf8
- query JMeter
- sm tab_htt_top
- where nexearch

개발자 도구의 정보를 HTTP Request의 각 필드에 해당하는 정보로 변경하면 [그림 4-14]와 같이 매칭한다. 각 필드를 [그림 4-14]처럼 입력하면 HTTP Request Sampler가 하나 완성된다. ‘④ Parameters’는 직접 입력해도 되지만 마우스로 드래그해서 복사한 다음 ‘Add from Clipboard’를 버튼을 사용하면 손쉽게 입력할 수 있다.

[그림 4-14] 개발자 도구 정보와 HTTP Request 필드 매칭



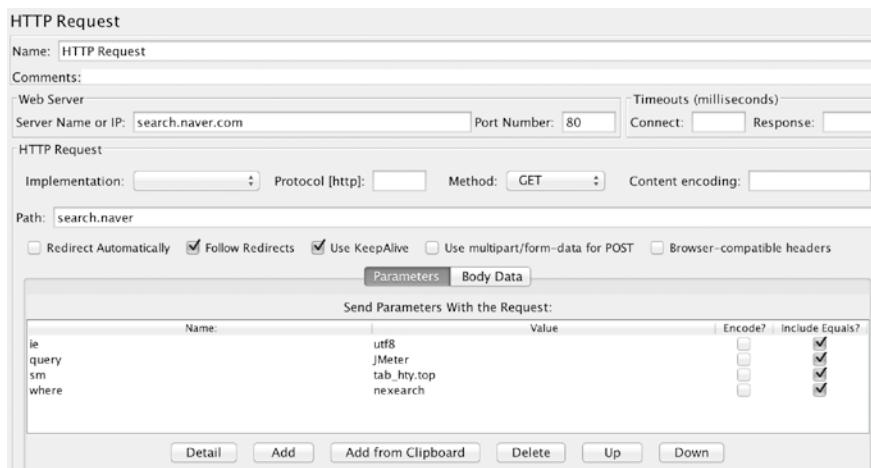
URL	상태	도메인	사이즈	Remote IP	타임라인
GET search.naver?sm=	200 OK	search.naver.com	56.6 KB	114.111.42.141:80	206ms

매개변수 헤더 응답 HTML 쿠키

ie utf8
query jmeter
sm tab_hty.top
where nexearch

입력된 HTTP Request Sampler 결과는 [그림 4-15]와 같다.

[그림 4-15] HTTP Request Sampler 화면



HTTP Request

Name: HTTP Request

Comments:

Web Server

Server Name or IP: search.naver.com Port Number: 80

Timeouts (milliseconds)

Connect: Response:

HTTP Request

Implementation: Protocol [http]: Method: GET Content encoding:

Path: search.naver

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters Body Data

Send Parameters With the Request:

Name:	Value	Encode?	Include Equals?
ie	utf8	<input type="checkbox"/>	<input checked="" type="checkbox"/>
query	jMeter	<input type="checkbox"/>	<input checked="" type="checkbox"/>
sm	tab_hty.top	<input type="checkbox"/>	<input checked="" type="checkbox"/>
where	nexearch	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Detail Add Add from Clipboard Delete Up Down

이렇게 추가된 HTTP Request Sampler를 실행하면 아마 에러가 발생할 것이다. ‘search.naver.com’의 경우에는 HTTP Header 정보 중에 Referer 값을 확인해서 ‘naver.com’에서의 요청이 아니면 ‘403 Forbidden’ 에러를 발생시킨다. 이러한 문제를 제거하기 위한 방법은 여러 가지가 있는데, HTTP Header Manager를 추가해서 별도의 HTTP Header를 서버로 전송하는 방법이 가장 쉽다.

HTTP Header Manager는 추가하려는 HTTP Request Sampler에서 마우스 오른쪽 버튼을 누르고 ‘Add → Config Element → HTTP Header Manager’를 선택하면 추가된다. 추가한 다음 [그림 4-16]과 같이 설정하면 문제가 해결된다.

[그림 4-16] HTTP Header Manager 추가와 설정

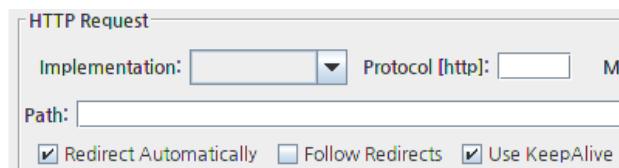


Redirect Automatically와 Follow Redirects

HTTP 응답의 30X 응답코드는 콘텐츠가 다른 곳으로 이동했거나 로그인하지 않고 접근했을 때 로그인 주소로 리다이렉트 Redirect하는 역할을 한다.

이러한 응답이 왔을 때 적절한 주소로 이동시켜주는 JMeter의 기능이 Redirect Automatically와 Follow Redirects 옵션이다. 이 두 옵션은 동시에 활성화할 수 없으며 하나를 체크하면 다른 하나는 체크가 풀린다. 이 두 옵션이 모두 체크되어 있지 않으면 30X 응답코드가 왔을 때 어떤 처리도 하지 않는다.

[그림 4-17] Redirect 옵션



두 옵션은 비슷해 보이나 다른 처리 결과를 나타낸다. 예를 들어, redirect_test.php를 요청하면 redirect_test2.php로 리다이렉트하는 페이지가 있다고 가정했을 때 실행 후 Result Tree View에서 결과를 살펴보면 [그림 4-18]과 [그림 4-19]처럼 매우 다르게 나타난다.

[그림 4-18] Redirect Automatically 옵션 체크 시

— redirect_test.php

[그림 4-19] Follow Redirects 옵션 체크 시

└ redirect_test.php
 └ http://jmeter.test.com/jmeter/redirect_test.php
 └ http://jmeter.test.com/jmeter/redirect_test2.php

즉, Redirect Automatically는 내부적으로 리다이렉트를 처리하여 결과값만 전달하고, Follow Redirects는 redirect_test.php와 redirect_test2.php를 각각의 Sampler로 간주하고 결과를 보여준다.

그렇다면 결과 화면만 다르게 나타날까? 그렇지 않다. 만약 redirect_test.php에 쿠키를 추가하거나 삭제하는 코드가 삽입되어 있다면 Redirect Automatically일 때는 정상적으로 작동되지 않고, Follow Redirects에서는 정상적으로 작동된다. 대표적인 예가 바로 SSO^{Single Sign-On, 통합인증}이다. SSO를 처리하려면 SSO 서버로 리다이렉트하고 이 과정에서 쿠키를 생성하고 삭제하는 경우가 빈번히 발생하기 때문이다.

HTML 내의 모든 파일 받아오기

대부분의 웹 성능 테스트는 WAS, DB에서 구동되는 로직의 성능을 측정하는 테스트지만, 간혹 대역폭을 테스트해야 하는 경우도 있다. 이때 유용하게 사용할 수 있는 옵션이 Embedded Resources from HTML Files다.

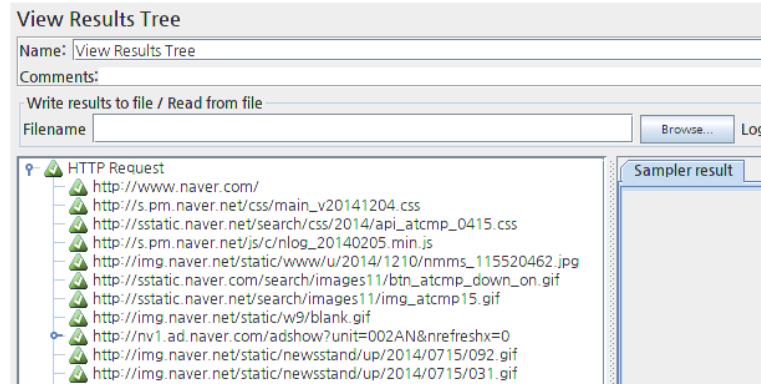
웹 페이지는 많은 이미지와 자바스크립트, CSS 파일로 구성된다. 이러한 자료를 테스트하기 위해 일일이 HTTP Request를 만든다면 매우 비효율적일 것이다. 또한, 수시로 변경되거나 추가되는 자료를 업데이트해서 Test Plan에 반영하는 것은 더더욱 어렵다. 하지만 Embedded Resources from HTML Files의 Retrieve All Embedded Resources 옵션을 체크하면 이를 어려움 없이 처리할 수 있다.

[그림 4-20]과 [그림 4-21]은 국내 유명 포털사이트의 메인 페이지를 받아오는 예제로, 옵션이 체크되지 않은 경우와 체크된 경우의 차이를 View Results Tree에서 확인한 모습이다.

[그림 4-20] Retrieve All Embedded Resources 미적용



[그림 4-21] Retrieve All Embedded Resources 적용

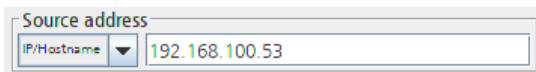


Source address 적용

Source address는 요청을 보내는 JMeter의 IP를 특정 IP로 지정해서 통신하기 위한 기능이다. 대부분 Socket 프로그램에서 별다른 설정을 하지 않으면 시스템의 기본 IP 주소를 가지고 웹 서버와 통신하는데, JMeter가 설치된 PC에 여러 개의 NIC(Network Interface Controller)가 존재하거나 한 NIC에 여러 개의 IP가 설정되어 있을 때 이 기능을 사용할 수 있다.

그렇다면 이 기능은 왜 필요한 것일까? 예를 들어, 웹 서버의 로직에서 Source address 값에 따라 서로 다르게 분기하는 로직이 있거나 L4 Switch에서 Source address에 따라 분기할 때 하나의 Source address로만 요청하면 한쪽 서버로 요청이 몰리는 문제가 있다. 대표적인 성능 테스트 도구인 LoadRunner에는 IP Spoofing 기능이 있어서 이를 이용해서 Source address를 변조할 수 있다. JMeter에는 아직 이러한 기능은 없지만, Source address 기능으로 어느 정도 해결할 수 있다.

[그림 4-22] Source address 설정



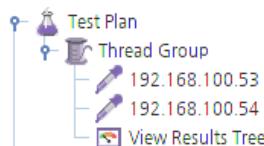
Linux PC에 다음처럼 2개의 IP(192.168.100.53, 192.168.100.54)가 추가로 맵핑되어 있다고 가정한다.

[ifconfig 명령 화면의 일부]

```
eth0:1    Link encap:Ethernet HWaddr 6c:f0:49:54:c6:a7
          inet addr:192.168.100.53 Bcast:192.168.100.255 Mask:255.255.255.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

eth0:2    Link encap:Ethernet HWaddr 6c:f0:49:54:c6:a7
          inet addr:192.168.100.54 Bcast:192.168.100.255 Mask:255.255.255.0
                  UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

[그림 4-23] Source address Test Plan



[그림 4-23]과 같이 2개의 HTTP Request를 추가하고 각각의 Source address를 192.168.100.53과 192.168.100.54로 설정하고 요청을 보내면 웹 서버의 로그에는 다음과 같이 Source address의 값이 기록된다.

```
192.168.100.53 -- [22/Apr/2014:12:02:59 +0900] "GET / HTTP/1.1" 200 793 "-"
"Apache-HttpClient/4.2.6 (java 1.5)"

192.168.100.54 -- [22/Apr/2014:12:02:59 +0900] "GET / HTTP/1.1" 200 793 "-"
"Apache-HttpClient/4.2.6 (java 1.5)"
```

4.3.2 설정 레퍼런스

HTTP Request Defaults와 HTTP Request는 거의 동일한 형태이므로 HTTP Request를 기준으로 설명한다. ([\[그림 4-6\]](#) 참고)

Web Server

요청을 보내려는 웹 서버의 IP와 Port를 입력하는 부분이다. HTTP Request Defaults에 값을 입력하고 각 HTTP Request Sampler에는 이 값을 입력하지 않는 경우가 많다.

Timeouts

접속과 응답 Timeout을 1/1000초 단위로 입력하는 부분이다. Connect는 최초 접속을 위한 Timeout 값이고, Response는 연결이 이루어진 후 자료를 주고받을 때 사용되는 Timeout 값이다.

HTTP Request

- **Implementation** : 어떤 클래스로 HTTP 요청을 주고받을지 결정하는 부분이다. 설정하지 않으면 HttpClient4를 기본으로 사용한다. Implementation

의 종류에 따라 다음에 나오는 특정 기능이 동작하지 않을 수도 있다.

- **Protocol [http]** : http와 https를 선택하는 부분이다. 입력하지 않으면 http로 동작한다.
- **Method** : GET, POST, HEAD 등 http 프로토콜에서 지원하는 요청 방식 중 하나를 선택하는 부분이다. HTTP Request를 생성하면 GET으로 기본 설정된다.
- **Content encoding** : 응답 데이터의 인코딩 Encoding 방식을 지정한다. 자바는 응답 헤더 Header에 인코딩 방식을 지정하지 않으면 기본으로 UTF-8을 사용한다. EUC-KR을 사용하는 페이지의 View Results Tree에서 한글이 깨져 보인다면 인코딩을 EUC-KR로 설정해야 한다.
- **Path** : 서버에서 어떤 경로의 파일을 요청할지 결정한다.

[표 4-2] Path 옵션

옵션	설명
Redirect Automatically/ Follow Redirects	HTTP 응답코드 302, 303 등의 리다이렉트 응답을 받았을 때 어떻게 동작할지에 대한 옵션이다. Redirect Automatically를 선택하고 View Results Tree Listener로 결과를 보면 여러 곳으로 리다이렉트되어도 하나의 요청으로 보인다([그림 4-18] 참고). Follow Redirects를 선택하면 리다이렉트되는 각 과정을 볼 수 있다([그림 4-19] 참고).
Use KeepAlive	웹 서버에서 KeepAlive 기능을 지원한다면 요청 후 접속을 끊지 않고 다음 요청에서 연결을 재사용하여 처리할 수 있다. 일반적으로 이 옵션을 켜면 연결을 자주 요청하지 않으므로 처리량은 늘어난다.
Use multipart/form-data for POST	Method를 POST로 사용하는 경우에 파라미터를 multipart/form-data 형태로 인코딩해서 보낼 수 있다.
Browser-compatible headers	multipart/form-data 형태로 요청할 때 Content-Type, Content-Transfer-Encoding 헤더를 쓰지 않고 Content-Disposition 헤더로만 보낸다.

- **Parameters** : Path와 함께 전달될 파라미터의 Name과 Value를 한 쌍으로 입력한다.

- **Body Data** : POST/PUT 요청 시에 전달될 파라미터를 문자열로 만들어서 한번에 보낼 때 사용한다.

Proxy Server

서버에 직접 접속하지 않고 프락시를 통해 접속해야 하는 환경에서 사용한다.

Embedded Resources from HTML Files

응답 메시지의 내용을 구문 분석 Parsing해서 이미지/JS/CSS 파일 등을 동시에 받아오게 하는 옵션이다. 서비스 로직 테스트가 아닌 웹 서버의 대역폭 등을 종합적으로 테스트할 때 유용하다.

Source address

JMeter를 구동하려는 PC에 여러 개의 네트워크 인터페이스가 있거나 하나의 네트워크 인터페이스에 여러 개의 IP가 설정되어 있을 때 Source IP를 어떤 것으로 지정할지를 강제로 정하는 설정이다. 여러 대의 웹 서버를 L4 Hash 방식으로 분산할 때 1대의 JMeter로 테스트하면 여러 대의 웹 서버 중 한 곳으로만 요청이 몰릴 수 있다. 이 설정을 이용해서 서로 다른 IP에서 요청하는 것처럼 설정하면 여러 대의 웹 서버로 요청을 보낼 수 있다. HTTPClient Implementation에서만 동작한다.

4.3.3 Implementation 종류

Implementation은 HTTP Request를 웹 서버와 주고받기 위해 사용할 자바 클래스를 의미한다.

지정하지 않은 경우

jmeter.properties 파일의 jmeter.httpsampler 옵션을 참조한다. 이 값을 설정하지 않으면 HttpClient4를 기본으로 사용한다.

Java

JVM에서 제공하는 기능을 사용하는데, 일부 기능 제약이 있다. Connection Reuse의 제약이 있고, HTTPS를 사용하면 버그가 존재한다.

HTTPClient3.1

Apache Commons HttpClient 3.1을 사용한다. 더는 개발되지 않고 있어서 나중에는 JMeter에서 빠질 가능성이 높다.

HTTPClient4

Apache HttpComponents HttpClient 4.x.를 사용한다.

4.4 CSV Data Set Config

JMeter에서 테스트 데이터를 전달하는 가장 쉬운 방법은 바로 CSV Data Set Config다. ‘CSV Data Set Config’란 일련의 데이터를 CSV⁰³ 형태의 파일에 저장해 놓고 이를 차례대로 불러들여서 변수로 사용하는 Element를 말한다. 가장 많이 활용되는 부분은 사용자 계정 정보와 검색 키워드 입력이다. 동일한 변수를 이용해서 Loop Iteration마다 서로 다른 값을 입력할 수 있어서 JMeter가 좀 더 실제 서비스 상황에 근접할 수 있도록 도와주는 유용한 도구다.

4.4.1 예제

CSV파일을 통한 검색 키워드 입력

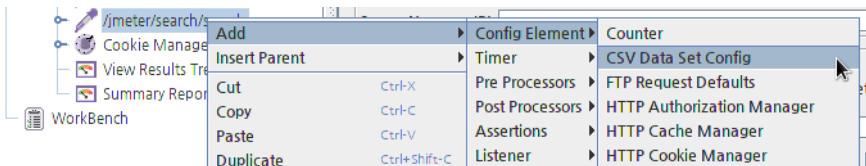
앞 장에서 작성한 Test Plan에 CSV 파일로부터 검색 키워드를 받아서 입력하도록 수정한다. 키워드가 입력된 파일은 ‘search_keyword.csv’라고 가정한다.

[그림 4-24]처럼 /jmeter/search/search 부분에 마우스 오른쪽 버튼을 눌러서

03 · Comma-Separated Values, Character-Separated Values

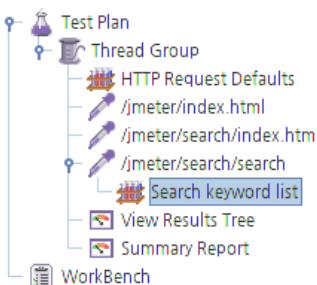
Config Element 그룹에 속한 CSV Data Set Config를 추가한다.

[그림 4-24] CSV Data Set Config 추가



추가되면 [그림 4-25]와 같은 형태가 된다. 이처럼 특정 Sampler의 자식 노드^{Child Node}로 추가하는 것은 해당 Element의 적용 범위를 해당 Sampler로 한정하는 좋은 방법이다. 하지만 CSV Data Set Config는 변수에 값을 입력하는 작업이 부모 노드^{Parent Node}의 Sampler가 동작할 때 이루어지는 것이 아니라 Loop Iteration이 시작할 때 일괄적으로 동작하는 형태다. 그러므로 동일한 변수명으로 여러 개의 CSV Data Set Config를 각 Sampler의 자식 노드 형태로 추가하더라도 그중 하나의 값으로만 적용되므로 이를 주의해서 변수명을 설정해야 한다.

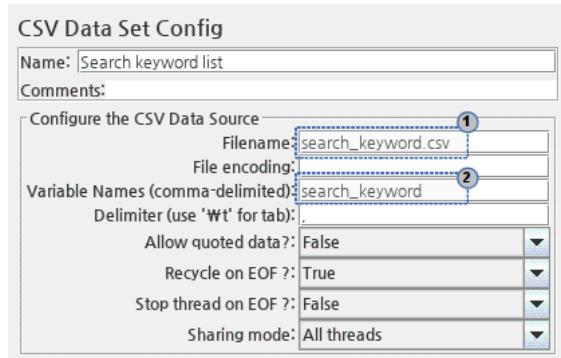
[그림 4-25] CSV Data Set Config 추가 후



- ① Filename에 준비한 CSV 파일명을 입력한다. 파일 경로는 상대 경로와 절대 경로로 모두 입력할 수 있다. 상대 경로로 입력하면 Test Plan 파일이 위치한 경로부터 상대 경로로 인식하게 된다. 즉, 파일명만 입력하면 Test Plan 파일이 위치하는 경로와 같은 경로의 파일을 참조한다.

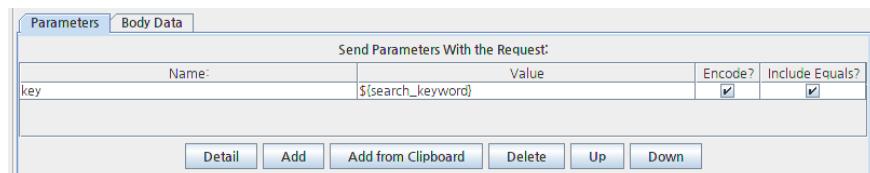
② Variable Names는 사용할 변수명을 설정하는 부분이다. 다른 변수와 구별될 수 있는 이름을 콤마(.)로 구분해서 CSV파일에 존재하는 필드 수만큼 입력해 준다. 이 예제에서는 한 개의 변수만 사용해서 콤마가 사용되지 않았다.

[그림 4-26] CSV Data Set Config 설정



CSV Data Set Config의 추가 및 설정이 끝나면 /jmeter/search/search Sampler의 내용을 수정해야 한다. HTTP Request Sampler에서 Parameters의 Value 값을 \${search_keyword}로 변경한다. JMeter에서 변수를 사용할 때에는 변수명 앞 뒤를 '{' 와 '}'로 감싼다.

[그림 4-27] CSV Data Set Config 변수 활용



이제 Loop Iteration이 반복될 때마다 search_keyword.csv 파일을 차례대로 읽어 들여서 새로운 값을 입력하게 된다. Threads Group의 Loop Count를 100으로 설정했는데, CSV 파일의 Data가 50개밖에 없으면 어떻게 동작할까? 어떤 설

정도 변경하지 않았다면 51번째 반복에서는 CSV 파일의 첫 번째 라인으로 이동하여 다시 반복하게 된다.

4.4.2 설정 레퍼런스

Filename

CSV 파일이 존재하는 위치와 파일명을 입력한다. 절대 경로와 상대 경로를 모두 입력할 수 있다. 상대 경로는 Test Plan 파일이 있는 위치부터 시작한다.

File encoding

파일에 저장된 내용의 인코딩 정보를 입력한다. 입력하지 않으면 UTF-8로 인식한다.

Variable Names

다른 Sampler에서 참조하는 변수 이름을 설정한다. 2개 이상의 변수가 있을 때에는 콤마(,)로 구분해서 적는다.

Delimiter

CSV 파일 내의 데이터를 구분할 구분자를 설정한다. 기본값은 콤마(,)다.

Allow quoted data?

큰따옴표를 어떻게 처리할 것인가 대한 설정으로, CSV 파일에 큰따옴표가 붙은 상태로 “jmeter”로 저장되어 있다면 이 옵션이 False일 때는 변수에 “jmeter”라는 값이 그대로 입력되고, True일 때 변수에 jmeter라는 값으로 설정된다.

Recycle on EOF?

파일 내의 모든 데이터를 사용하고 난 후에도 계속해서 데이터를 요청하면 어떻게 처리할 것인가에 대한 설정이다. 데이터가 9개고 10번을 반복하는 테스트 상황일

때 이 옵션을 True로 설정하면 10번째에는 다시 처음으로 돌아가서 첫 번째 데이터를 전달하고, False로 설정하면 10번째에는 변수에 <EOF>라는 값이 들어간다.

Stop thread on EOF?

파일 내의 모든 데이터를 사용하면 Thread가 멈춰 더는 테스트가 진행되지 않게 하는 설정이다. 데이터가 9개고 10번을 반복하는 테스트 상황일 때 이 옵션을 True로 설정하면 9번째 반복 후 Thread가 종료되어 10번째는 실행되지 않는다.

Sharing mode

CSV 파일의 공유 범위를 설정하는 옵션이다.

[표 4-3] Sharing Mode 옵션

옵션	설명
All threads	전체 Test Plan에 1개의 파일이 생성되어 전체 Thread Group 내의 모든 Thread가 이를 공유한다. 이 설정이 기본값이다.
Current thread group	CSV Data Set Config가 존재하는 Thread Group별로 1개의 파일이 생성되어 해당 Thread Group의 모든 Thread가 공유한다.
Current thread	Thread별로 각각 CSV 파일을 생성해서 사용한다.

4.5 HTTP Cookie Manager

보통의 웹 서비스는 자료 저장이나 세션 유지를 위해 쿠키를 사용한다. 로그인된 사용자에게만 허가된 웹 페이지는 쿠키에 저장된 세션 정보로 사용자를 인증하고 해당 페이지에 대한 권한을 부여한다. JMeter에서 쿠키를 저장할 때 사용하는 Element가 바로 HTTP Cookie Manager다.

HTTP Cookie Manager는 Config Element의 하나로, 가상 사용자별로 서버에서 받은 쿠키 값을 저장하며 사용자에 의해 정의된 쿠키 값을 웹 서버로 전달한다.

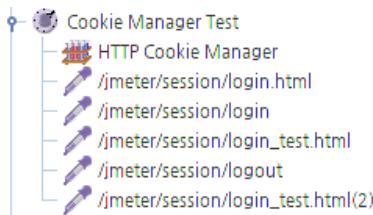
4.5.1 예제

로그인 유지하기

하나의 HTTP Cookie Manager와 5개의 HTTP Request로 테스트를 진행한다.

[그림 4-28]과 같은 형태로 구성되고, /jmeter/session/login.html과 /jmeter/session/login은 로그인을 처리하는 웹 페이지다. 로그인에 성공하면 세션 정보를 쿠키에 저장하도록 설정되어 있다. /jmeter/session/login_test.html, /jmeter/session/login_test.html(2)는 로그인하지 않고 접근하면 오류(403 Forbidden)가 발생하는 웹 페이지다. /jmeter/session/logout은 로그아웃을 처리하는 웹 페이지로, 쿠키에 저장된 세션 정보를 삭제하는 역할을 한다.

[그림 4-28] HTTP Cookie Manager



HTTP Cookie Manager가 있을 때와 없을 때를 비교하기 위해 HTTP Cookie Manager를 Enable과 Disable로 설정하면서 결과를 View Result Tree를 통해 확인해 본다. Enable된 상태의 특정 Element를 Disable 상태로 바꾸려면 원하는 Element를 마우스로 선택한 후 단축키 ‘Ctrl-t’를 누르거나 메뉴에서 ‘Edit → Disable’을 선택한다.

HTTP Cookie Manager가 Enable된 상태에서 테스트를 진행하면 결과는 [그림 4-29]와 같다. 로그인이 필요한 /jmeter/session/login_test.html는 로그아웃 이전에는 정상적인 결과가 나오지만, 로그아웃 후에는 오류가 발생하는 것을 확인할 수 있다.

[그림 4-29] HTTP Cookie Manager Enable 결과

The screenshot shows the JMeter Test Plan tree on the left with several requests listed. On the right, the 'Sampler result' tab of the 'View Results Tree' dialog is selected, displaying the following details:

Thread Name	Thread Group 1-1
Sample Start	2014-02-20 16:48:13 KST
Load time	3
Latency	3
Size in bytes	487
Headers size in bytes	175
Body size in bytes	312
Sample Count	1
Error Count	1
Response code	403
Response message	Forbidden

[그림 4-30]과 같이 HTTP Cookie Manager가 Disable된 상태로 테스트를 진행하면 [그림 4-31]과 같이 로그인이 필요한 모든 웹 페이지에서 오류가 발생하는 것을 확인할 수 있다.

[그림 4-30] HTTP Cookie Manager Disable 상태

The screenshot shows the JMeter Test Plan tree on the left. A 'Cookie Manager Test' element is present, containing an 'HTTP Cookie Manager' element and two 'HTTP Request' elements for 'login.html' and 'login_test.html'. The 'HTTP Request' for 'login_test.html' is highlighted with a red warning icon.

[그림 4-31] HTTP Cookie Manager Disable 결과

The screenshot shows the JMeter Test Plan tree on the left with the same structure as in [그림 4-30]. On the right, the 'Sampler result' tab of the 'View Results Tree' dialog is selected, displaying the following details:

Sample Start	2014-02-20 17:02:16 KST
Load time	5
Latency	5
Size in bytes	613
Headers size in bytes	257
Body size in bytes	356
Sample Count	1
Error Count	1
Response code	403
Response message	Forbidden

4.5.2 설정 레퍼런스

Clear Cookies each Iteration

한번 저장된 쿠키 값을 Loop Iteration이 반복될 때마다 삭제할지 말지에 대한 설정입니다.

정으로, Loop를 시작할 때 로그인되지 않은 상태여야 한다거나 Loop Iteration이 반복될 때마다 사용자 정보가 변경되어야 한다면 이 옵션을 체크한다.

Cookie Policy

쿠키 관리 정책으로, 기본값은 compatibility고 거의 모든 경우에 정상 동작한다.

Implementation

HC3CookieHandler(HttpClient 3.1 API)와 HC4CookieHandler(HttpClient 4 API)에 각각 대응하는 값으로, 디폴트 값은 HC3CookieHandler다. IPv6 주소로 테스트한다면 HC4CookieHandler를 선택해야 한다.

User-Defined Cookies

이 부분에 값을 추가하면 웹 서버에서 전달받아 생성된 쿠키뿐 아니라 JMeter 사용자가 임의로 쿠키를 생성해서 웹 서버로 보낼 수 있다.

[그림 4-32] HTTP Cookie Manager 설정화면



4.6 Regular Expression Extractor

HTTP Request Sampler로 웹 서버에서 받아온 결과값을 파싱 Parsing해서 그 내용 중 의미 있는 값을 찾아내는 가장 쉽고 유용한 방법이 바로 Regular Expression Extractor다. 이름에서 알 수 있듯이 정규표현식 Regular Expression으로 특정 문자열을 추출하는 기능을 담당한다.

Regular Expression Extractor는 Post Processors에 해당한다. ‘Post Processors’란 하나의 Sampler(HTTP Request와 같은)가 요청을 보내고 응답을 받은 후 다음 Sampler로 진행하기 전에 실행되는 Element로, 해당 Sampler의 결과값을 이용해서 특정 정보를 변수에 저장하거나 해당 Thread를 종료하는 등의 역할을 수행한다. 이를 통해 Sampler 결과값에 따라 다음 Sampler의 행동이 변경되는 좀 더 다이나믹한 Test Plan을 만들 수 있다.

4.6.1 예제

이번 예제는 3장에서 작성했던 Test Plan을 좀 더 확장해서 작성한다.

검색 결과에서 URL 얻어오기

[그림 4-33]은 /jmeter/search/search 결과의 일부분이다. 이번 예제에서는 <A> 태그의 href 값에 해당하는 URL(그림 4-33]의 ①, ②)에서 무작위로 하나의 URL을 선택하여 next_url이라는 변수에 저장하고 이 URL로 이동하는 HTTP Request Sampler를 만든다. 이는 사용자가 웹 화면에서 여러 개의 링크 중 하나를 무작위로 선택해서 클릭하는 것과 같은 효과를 줄 수 있다.

HTTP Request에서는 IP, Port, Path, Protocol을 별도로 입력해야 하므로 정규표현식을 작성할 때에는 URL의 IP, Port, Path, Protocol을 분리해서 각 그룹으로 추출해야 한다.

[그림 4-33] /jmeter/search/search의 결과값 일부

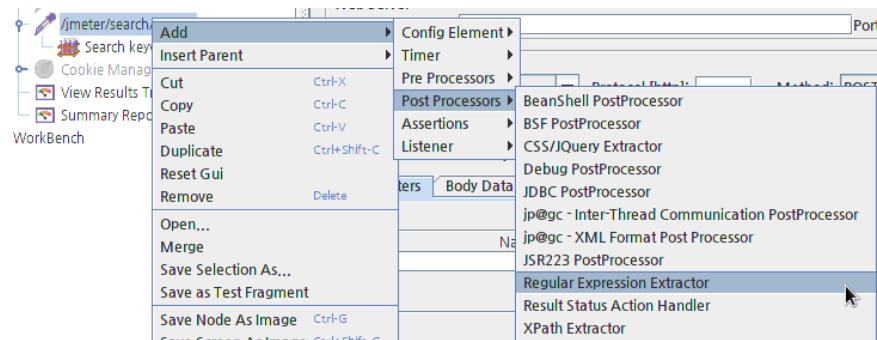
```
<div class="keyword">
    <b>Search Keyword</b> : "application", <b>Result Count</b> : 2
</div>

<div class="title">
    1 - java <span class="url">(<a href="http://www.java.com/" target="_blank">http://
www.java.com</a> )</span>
</div>
<div class="contents">
    Java programming language,application
</div>

<div class="title">
    2 - InterBase <span class="url">(<a href="http://www.embarcadero.com/products/interbase">
target="_blank">http://www.embarcadero.com/products/interbase</a> )</span>
</div>
<div class="contents">
    InterBase is a full-featured, high performance and scalable relational database for
software developers who are looking to embed a low cost, zero-admin, lightweight database into
applications on Android, iOS, Windows, OS X, Linux and Solaris.
</div>
```

[그림 4-34]처럼 /jmeter/search/search Sampler에서 ‘Add → Post Processors → Regular Expression Extractor’를 선택하여 추가한다.

[그림 4-34] Regular Expression Extractor 추가

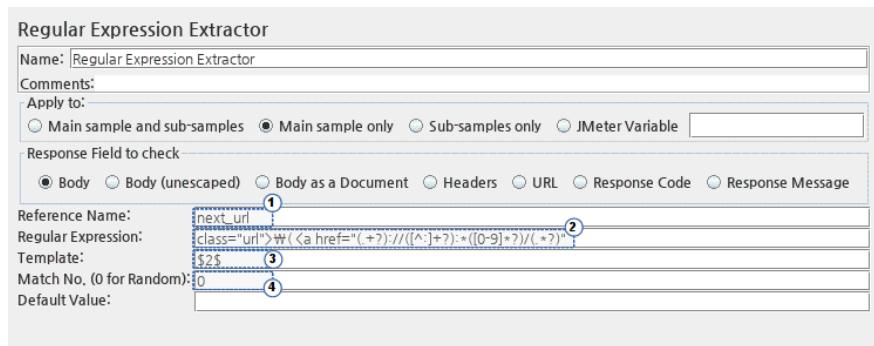


Element를 다음과 같이 설정한다.

① Reference Name은 변수로 사용될 변수명을 입력하는데, 여기서는 next_url로 설정한다.

- ② 가장 중요한 설정값으로, 원하는 문자열을 얻어올 때 사용될 정규표현식을 입력한다. 각 그룹은 팔호로 묶어준다('\'에서 '\'은 escape 기능을 수행하는 표현식으로, 팔호를 그룹의 시작이 아닌 팔호 자체로 인식하게 한다). 여기서는 `class="url">\(\a href="(.)?://([^\:]+?)?:*([0-9]*?)?/(.*?)"`로 설정한다.
- ③ Template은 ①에서 설정한 Reference Name 변수명을 그대로 사용했을 때 보여질 값을 설정하는 것으로, \$2\$를 입력한다. 이것은 다음 설정 레퍼런스에서 좀 더 자세하게 설명하겠다.
- ④ 정규표현식에 의해 1개 이상의 문자열이 매칭되었을 때 어떠한 값을 선택할 것인지에 대한 설정으로, 이번 예제에서는 여러 매칭 중 무작위로 하나를 선택하므로 0으로 설정한다.

[그림 4-35] Regular Expression Extractor 설정

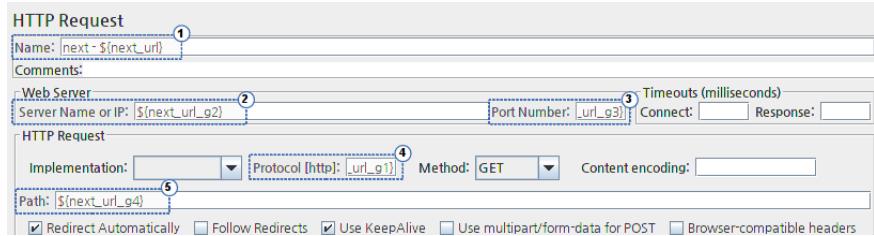


다음으로 Regular Expression Extractor로 추출된 URL을 요청하는 HTTP Request Sampler를 추가하고 설정한다.

- ① Name은 next - \${next_url}로 설정해서 Summary Report에서 URL별로 정 보가 출력되게 한다.
- ② Server Name or IP는 두 번째 그룹에 해당하므로 \${next_url_g2}로 설정한다.

- 설정된 변수명 뒤에 추가로 붙는 _g2는 두 번째 그룹의 값이라는 것을 의미한다.
- ③ Port Number는 있거나 없을 수도 있다. 세 번째 그룹이므로 \${next_url_g3}로 설정한다.
 - ④ Protocol은 첫 번째 그룹이므로 \${next_url_g1}로 설정한다.
 - ⑤ Path는 네 번째 그룹이므로 \${next_url_g4}로 설정한다

[그림 4-36] HTTP Request Sampler 설정



Test Plan 작성이 모두 끝났다. 이제 Thread Group에서 Thread = 1, Loop Count = 10으로 설정한 후 테스트를 진행하고 Summary Report를 이용해서 결과값을 확인한다. [그림 4-37]과 같이 Label에 next-xxxxxxxxx과 같은 형식으로 정규표현식에 의해 추출된 URL을 호출한 것을 확인할 수 있다.

[그림 4-37]에서 Error가 1건 발생했는데(①), 이는 /jmeter/search/search의 결과 중 정규표현식에 부합하는 문자열이 없기 때문이다. 즉, 검색 결과에 URL이 하나도 없다는 의미다. 이 문제의 해결 방법은 다양하며 다음 장에서 다루겠다.

[그림 4-37] Regular Expression Extractor 결과

Summary Report									
Name: <input type="text" value="Summary Report"/> Comments: Write results to file / Read from file <input type="text" value="Filename"/> <input type="button" value="Browse..."/> Log/Display Only: <input checked="" type="checkbox"/> Errors <input type="checkbox"/> Successes <input type="button" value="Configure"/>									
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Through...	KB/sec	Avg. By...
/jmeter/search/search	10	7	5	11	1.48	0.00%	31.3/min	0.82	1604.
next - www.embarcadero.com	1	2394	2394	2394	0.00	0.00%	25.1/min	13.49	33077.
next - www.postgresql.org	1	5506	5506	5506	0.00	0.00%	10.9/min	3.02	17053.
next - www.python.org	2	821	244	1398	577.00	0.00%	26.3/min	19.13	44766.
next - cocoon.apache.org	2	2398	1899	2898	499.50	0.00%	12.2/min	4.58	23023.
next - www.java.com	1	729	729	729	0.00	0.00%	1.4/sec	8.85	6608.
next - dlang.org	1	3986	3986	3986	0.00	0.00%	15.1/min	4.88	19933.
Error:	1	0	0	0	0.00	100.00%	0/hour	0.00	654.
next - www.adalit.org	1	3390	3390	3390	0.00	0.00%	17.7/min	8.88	30812.
TOTAL	20	1125	0	5506	1622.25	5.00%	1.2/sec	14.69	12988.

Include group name in label? Save Table Header

4.6.2 설정 레퍼런스

[표 4-4] Regular Expression Extractor 설정 레퍼런스

옵션	설명
Apply to	Main sample은 HTTP Request 자체를 의미하며 Sub-sample은 HTTP Request에서 'Embedded Resources from HTML Files' 옵션을 설정했을 때 추가로 받아오는 자료를 의미한다. <ul style="list-style-type: none"> Main sample and sub-samples : Main과 Sub-sample에서 검색한다. Main sample only : Main sample에서만 검색한다. Sub-samples only : Sub-sample에서만 검색한다.
Response Field to check	<ul style="list-style-type: none"> Body : header를 제외한 웹 페이지 내용이다. Body (unesaped) : header를 제외한 웹 페이지 내용의 HTML escape 코드를 모두 변환한 것을 의미한다(테스트 성능에 영향을 줄 수 있으므로 주의해서 사용한다). Body as a Document : Apache Tika를 이용해서 파싱된 내용이다(테스트 성능에 영향을 줄 수 있으므로 주의해서 사용한다). Headers : HTTP Header의 내용이다. URL : 요청한 URL 자체다. Response Code : 200, 400, 500 등의 HTTP Response Code다. Response Message : OK, Internal Server Error 등의 HTTP Response Message다.

옵션	설명
Reference Name	정규표현식으로 선택된 문자열이 저장될 변수명이다.
Regular Expression	정규표현식이다(정규표현식 규칙에 대해서는 뒤에서 좀 더 자세히 다룬다).
Template	정규표현식에 여러 Group이 존재할 때 Reference Name 변수를 사용할 경우 보여지는 내용을 설정한다. <ul style="list-style-type: none"> • \$0\$: 정규표현식에 의해 선택된 문자열 전체를 의미한다. • \$n\$: n번째 Group의 내용을 나타낸다.
Match No. (0 for Random)	정규표현식으로 문서의 Match를 검사하면 여러 개의 Match가 나올 수 있다. 이를 때 몇 번째 문자열을 선택할지에 대한 설정이다. <ul style="list-style-type: none"> • 0 : Match된 여러 case 중 하나를 무작위로 골라서 Reference Name에 설정한다. • 0보다 큰 경우($n > 0$) : n번째 Match된 case를 Reference Name에 설정한다. • 0보다 작은 경우($n < 0$) : 모든 Match된 Case에 접근할 수 있다.
Default Value	Match된 문자열이 하나도 없을 때 변수에 저장될 문자열이다.

4.6.3 Match Case와 Group에 따른 변수명

정규표현식에서 Group은 괄호로 감싸진 부분을 의미한다. 앞의 예제에서 정규표현식 `class="url">\((<a href="(.)?://([:]+?)\:*(\d+)?)(.*?)`에는 총 4개의 Group이 존재한다.

Match No. ≥ 0

- \${ReferenceName} : Template에 지정된 값이 들어간다.
- \${ReferenceName_gn} : n에는 0, 1, 2, …이 올 수 있으며, 각각 n번째 Group의 값을 나타낸다. 0은 Match된 문자열 전체를 의미한다.
- \${ReferenceName_g} : Group의 개수다.

Match No. < 0

- \${ReferenceName_matchNr} : 정규표현식이 Match된 횟수다(0일 수도 있다).
- \${ReferenceName_n} : n에는 1, 2, 3, …이 올 수 있으며, Template에 의해 지정된 값이 들어간다.

- \${ReferenceName_n_gm} : m에는 0, 1, 2, …이 올 수 있으며, 각각 m번째 Group의 값을 나타낸다. 0은 Match된 문자열 전체를 의미한다.
- \${ReferenceName} : 항상 디폴트 값이 들어간다.

4.6.4 정규표현식 요약

JMeter에서 제공하는 정규표현식은 Perl의 문법과 매우 유사하므로 Perl의 정규표현식 관련 문서를 참조해도 무방하다. 단, Perl처럼 ‘//’로 감싸지 않아도 된다.

Meta Characters

[표 4-5] Meta Character 종류

Character	분류	설명
()	grouping	Group을 생성한다.
[]	character classes	[0-9] : 숫자를 의미한다. [a-z] : 영문 소문자를 의미한다. [ab^c] : c를 포함하지 않는 a, b 문자만을 의미한다.
{ }	repetition	반복 숫자를 지정할 수 있다. [a-z]{4,5} : 영문 소문자가 4~5번 반복된다. [0-9]{3}과 [0-9][0-9][0-9]는 같은 의미다.
* + ?	repetition	* : 0번 이상 반복을 의미한다. + : 1번 이상 반복을 의미한다. ? : 첫 번째 Match가 되면 더는 검색하지 말 것을 의미한다. 예 : http://abc.com 정규표현식 : style=".+?" 결과 : text-align:center 정규표현식 : style=".+" 결과 : text-align:center" ><a href="http://abc.com"
.	wild-card character	모든 Character를 의미한다.
\	escape character	Meta Character에 해당하는 문자를 쓰려면 ‘\’로 escape해야 한다. 앞의 예제에서 Group의 시작을 의미하는 '('를 문자열로 사용하기 위해 '\('로 입력한 것을 확인할 수 있다.

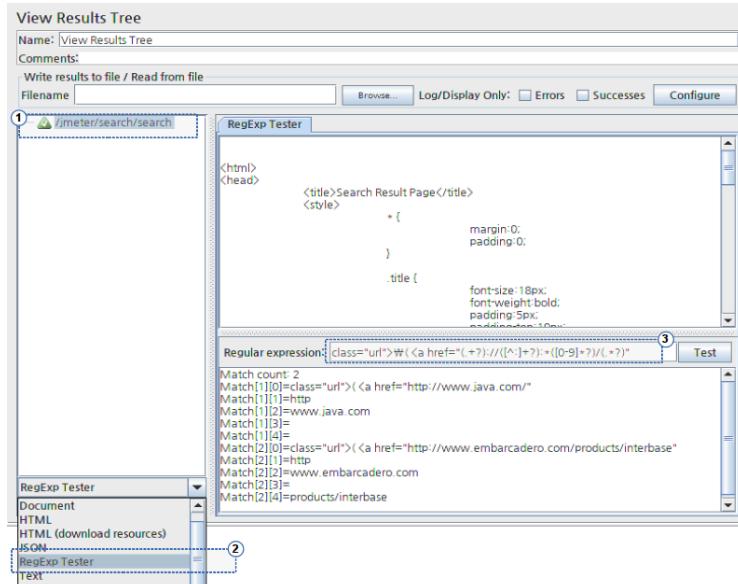
Character	분류	설명
	alternatives	선택적인 적용을 의미한다. (gray grey) : gray와 grey 둘 중 한 문자와 일치하는 경우다.
^ \$	start and end of	^는 라인의 시작, \$는 라인의 끝을 의미한다.

RegExp Tester를 이용한 테스트

정규표현식을 정확히 작성했는지 확인하는 가장 쉬운 방법은 View Result Tree에서 제공하는 'RegExp Tester'를 이용하는 것이다. [그림 4-38]과 같이 하면 RegExp Tester를 활성화할 수 있다.

- ① RegExp Tester를 이용하려는 Sampler를 결과 리스트에서 선택한다.
- ② 내용보기 포맷 선택에서 RegExp Tester를 선택한다.
- ③ 원하는 정규표현식을 입력하고 실제로 결과가 어떻게 나오는지 확인한다.

[그림 4-38] RegExp Tester 활성화



4.7 Logic Controller

Logic Controller는 프로그래밍 언어에서 자주 사용하는 if, for, while 반복문처럼 변수값이나 설정에 따라 특정 작업을 반복하거나 분기하는 역할을 하는 Controller다. Test Plan에 Sampler을 추가한 후에 Test Plan을 좀 더 실제 사용자 패턴과 비슷하게 만들기 위해서는 Logic Controller를 사용하는 것이 필수다.

JMeter에는 많은 종류의 Logic Controller가 있지만, If, Loop, Random, Random Order, Interleave, Controller를 가장 많이 사용한다. 여기서는 앞에서 언급한 Logic Controller들의 사용법을 간단한 예제를 통해서 알아보겠다.

4.7.1 If/Loop Controller

If Controller는 조건에 따라 로직을 분기하는 역할을 하고, Loop Controller는 로직을 반복적으로 수행하는 역할을 한다.

앞의 Regular Expression Extractor 예제의 결과에서 정규표현식에 부합하는 문자열을 찾지 못하면 URL 정보를 가져오지 못해서 오류가 발생한 것을 볼 수 있었다. 이런 오류가 발생하지 않게 하려면 정규표현식에 부합하는 문자열이 없을 때에는 그 밑의 로직을 실행하지 않아야 한다. 이처럼 조건에 의한 분기는 If Controller를 이용하면 쉽게 해결할 수 있다.

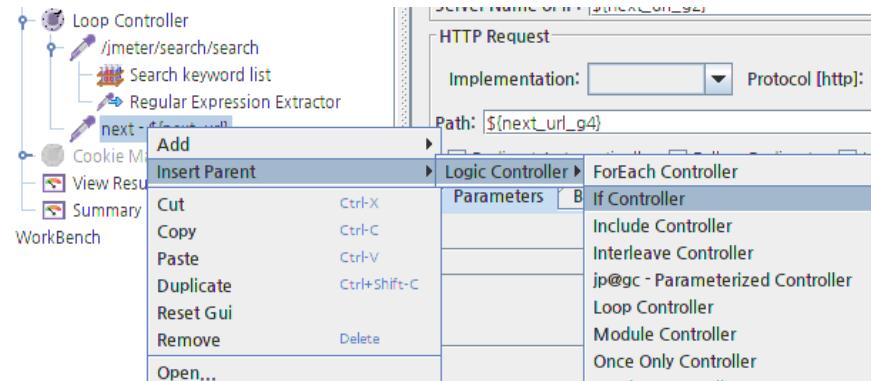
앞의 예제에서 ‘10회 반복’을 Thread Group의 Loop Count 값을 이용해서 처리했는데, 여기서는 Thread Group의 Loop Count 값을 1로 설정하고 그 대신 Loop Controller를 이용해서 ‘10회 반복’을 구현해 본다.

If Controller 추가 및 설정

If Controller는 Insert Parent 메뉴를 이용해서 추가한다. 기존 Sampler들을 자식 노드로 보내고 신규로 추가되는 If Controller를 부모 노드로 설정하는데, 이 경

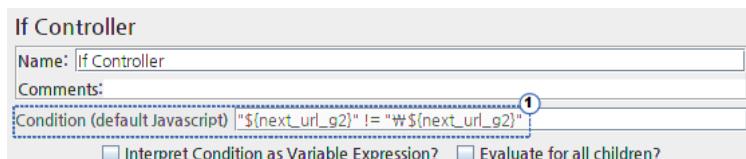
우에 유용한 메뉴가 ‘Insert Parent’다. Ctrl 키를 누른 상태에서 /jmeter/search/search와 next - \${next_url} Sampler를 선택한 후 마우스 오른쪽 버튼을 누르고 ‘Insert Parent → Logic Controller → If Controller’를 추가하면 /jmeter/search/search와 next - \${next_url} Sampler가 If Controller의 자식 노드로 내려간 것을 확인 할 수 있다

[그림 4-39] If Controller 추가



추가한 후 [그림 4-40]의 ①처럼 Condition을 "\${next_url_g2}" != "\\${next_url_g2}"로 설정한다.

[그림 4-40] If Controller 설정



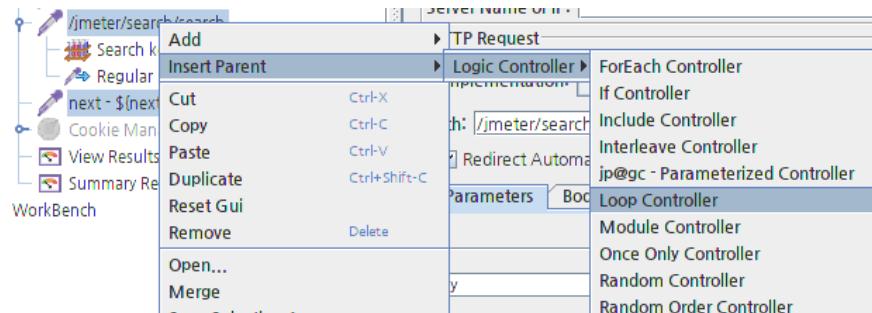
JMeter에서는 A라는 변수가 존재할 때 그 값에 접근하기 위해 PHP나 Shell Script와 유사하게 \${A} 같은 방식을 이용한다. 변수 A가 존재하지 않으면 \${A}는 단순히 \${A}라는 문자열로 처리한다.

Regular Expression Extractor는 'next_url'이라는 Reference Name을 가진 정규표현식에 부합하는 문자열이 존재하면 그룹별로 next_url_g1, next_url_g2, next_url_g3, next_url_g4 이렇게 4개의 변수가 생성되고, 문자열이 존재하지 않으면 변수는 생성되지 않는다. 변수가 생성되었는지 비교하는 식이 "\${next_url_g2}" != "\${next_url_g2}"다. 단, "\${next_url}" != "\${next_url}"로 비교하면 안 된다. next_url은 매칭되는 문자열이 존재하지 않더라도 이전 Loop에서 설정된 값이 있으면 그 값을 계속 유지하기 때문이다. \${next_url}을 사용하고 싶다면 Regular Expression Extractor 설정에서 디폴트 값에 특정 문자열("default_value"로 설정했다고 가정한다)을 지정하고 "\${next_url}" != "default_value" 같은 방식으로 비교해야 한다. 디폴트 값은 매칭되는 문자열이 없을 때 Reference Name에 설정되는 값이다.

Loop Controller 추가 및 설정

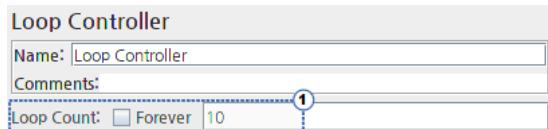
Loop Controller도 If Controller처럼 Insert Parent 메뉴를 이용해서 추가한다.

[그림 4-41] Loop Controller 추가



추가된 후 Loop Count를 [그림 4-42]의 ①처럼 10으로 설정하고, Thread Group의 Loop Count는 1로 변경한다.

[그림 4-42] Loop Controller 설정



결과 및 분석

Loop Controller의 Loop Count를 10으로 설정했으므로 [그림 4-43]을 보면 /jmeter/search/search Sampler가 총 10회 수행된 것을 알 수 있다. 그러나 TOTAL #Samples는 20이 아니라 19다. 이것은 10번 중 1번은 Regular Expression에 부합하는 문자열이 존재하지 않는다는 것을 의미한다. 하지만 오류는 발생하지 않고 정상적으로 처리된 것을 확인할 수 있다.

[그림 4-43] If/Loop Controller 수행 결과

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throug...	KB/sec	Avg. Bytes
/jmeter/search/search	10	5	4	7	0.90	0.00%	1.8/sec	2.78	1604.3
next - www.java.com	1	448	448	448	0.00	0.00%	2.2/sec	14.40	6608.0
next - www.postgresql.org	1	1642	1642	1642	0.00	0.00%	36.5/min	10.20	17156.0
next - www.python.org	2	201	177	225	24.00	0.00%	4.5/sec	202.23	46075.0
next - www.jacobjang.com	1	17	17	17	0.00	0.00%	58.8/sec	25.16	438.0
next - www.php.net	1	1575	1575	1575	0.00	0.00%	38.1/min	17.19	27726.0
next - clang.org	1	862	862	862	0.00	0.00%	1.2/sec	22.73	20066.0
next - jmeter.apache.org	1	553	553	553	0.00	0.00%	1.8/sec	20.12	11393.0
next - www.adacig.org	1	1912	1912	1912	0.00	0.00%	31.4/min	15.74	30825.0
TOTAL	19	393	4	1912	616.84	0.00%	2.5/sec	28.75	11705.5

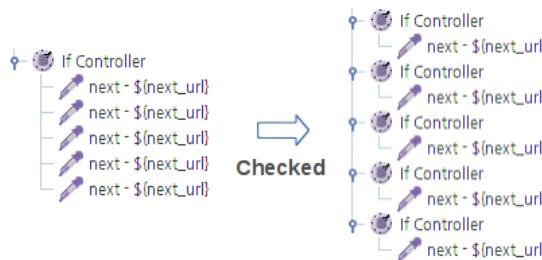
Include group name in label? Save Table Header

설정 레퍼런스

[표 4-6] If/Loop Controller 설정 레퍼런스

Controller	옵션	설명
If Controller	Condition	별도의 함수 표시가 없는 연산식을 사용하면 자바스크립트 형태로 인식된다. vars, log, ctx 변수도 사용할 수 있다. 예 : \${COUNT} < 10, \${JMeterThread.last_sample_ok}, "\${VAR}" == "abcd"
	Interpret Condition as Variable expression?	이 옵션이 체크되면 Condition 필드에 True로 평가된 표현식이 들어와야만 한다. 즉, Condition이 True인지만 확인한다. 예 : \${__jexl(\${COUNT} < 10), \${RESULT}}
	Evaluate for all children?	자식 노드 엔트리 전체를 실행하기 위해 Condition을 평가할 것인지, 자식 노드 각각을 실행하기 위해 각 Condition을 평가할 것인지에 대한 설정이다. [그림 4-44]와 같이 If Controller의 자식 노드에 여러 개의 Sampler가 존재할 때 이 옵션이 체크되면 각 Sampler에 별도로 If Controller를 추가한 것과 같은 효과를 줄 수 있다.
Loop Controller	Loop Count	자식 노드의 Element를 몇 번 수행할 것인지에 대한 설정으로, Forever를 체크하면 입력창이 Disable되면서 무한 반복한다.

[그림 4-44] Evaluate of all children? 옵션 체크 결과



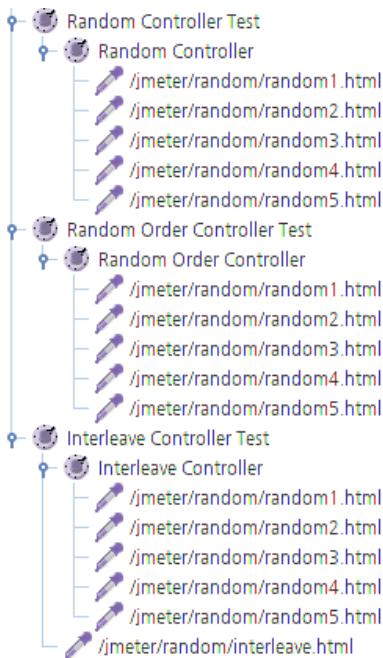
4.7.2 Random/Random Order/Interleave Controller

Test Plan을 작성하다 보면 여러 항목 중 하나를 무작위로 고른다거나 여러 항목의 순서를 바꾸고 싶은 경우가 발생한다. 이럴 때 사용할 수 있는 것이 Random/Random Order/Interleave Controller다. 여기서는 간단한 예제로 각각의 사용법과 결과를 알아본다.

[그림 4-45]처럼 Random/Random Order/Interleave Controller를 추가하고 각각의 자식 노드에 5개의 Sampler를 추가한다. 그리고 5번을 반복 수행해서 처리 순서가 어떻게 되는지 알아본다.⁰⁴

Random/Random Order/Interleave Controller는 추가하고 나면 별도의 설정은 필요 없다.

[그림 4-45] Random/Random Order/Interleave Controller



결과 및 분석

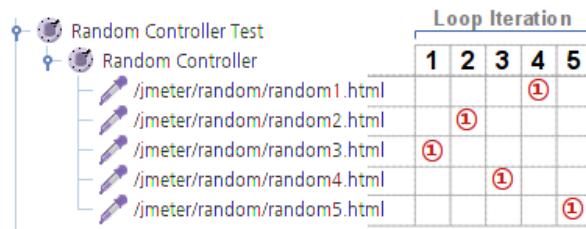
Random Controller에 추가된 5개의 HTTP Request Sampler 중에서 임의로 1

04 · Random/Random Order/Interleave Controller Test는 Simple Controller로를 이용해서 각각의 그룹으로 분리해 주는 역할만을 수행한다. 없어도 상관없는 Controller다.

개를 선택해서 실행한다. Thread Group에서 Thread는 1로, Loop Count는 5로 설정했을 때 각각의 Loop Iteration마다 5개의 Sampler 중 무작위로 1개가 선택된다. 무작위로 선택되므로 실행할 때마다 그 결과는 달라진다.

[그림 4-46]에서 5번 동안 모두 다른 Sampler가 선택된 것으로 나타나지만, 실제 상황에서는 같은 Sampler가 선택될 수도 있다. 즉, 앞뒤에 연관성이 있는 것이 아니라 Iteration마다 별도의 확률이라고 볼 수 있다.

[그림 4-46] Random Controller 결과



이번에는 Random Order Controller에 추가된 5개의 HTTP Request Sampler를 임의의 순서로 실행한다. Thread는 1로, Loop Count는 5로 설정했을 때 Loop Iteration마다 5개의 Sampler가 실행되지만, 순서는 그때마다 달라진다.

[그림 4-47] Random Order Controller 결과



마지막으로 Interleave Controller에 추가된 5개의 HTTP Request Sampler를 순서대로 1개씩 선택해서 실행한다. Thread는 1로, Loop Count는 5로 설정했을

때 각각의 Loop Iteration마다 5개의 Sampler 중 1개의 Sampler를 차례대로 선택한다.

‘1작업 → 저장 → 2작업 → 저장 → 3작업 → 저장 → … → N작업 → 저장’과 같이 서로 다른 N개의 작업이 있고 해당 작업을 수행한 후에 항상 ‘저장’하는 로직이 있다고 가정해 보자. Interleave Controller는 자주 사용하지는 않지만, ‘Interleave Controller(1작업 → 2작업 → 3작업 → … → N작업) → 저장’과 같이 Interleave Controller를 사용하면 같은 Sampler를 중복해서 복사하지 않고 깔끔하게 처리할 수 있는 좋은 방법이다.

[그림 4-48] Interleave Controller 결과



4.8 Timer

좀 더 정교한 Test Plan이라는 것은 다르게 표현하면 사용자가 서비스를 이용하는 패턴과 좀 더 유사하다는 것을 의미한다. 컴퓨터는 반복 작업을 빠르게 수행할 수 있지만, 일반 사용자는 그렇지 못하다. 화면상의 버튼 또는 링크를 누르고 내용을 입력하거나 글을 읽다 보면 중간 중간에 서버에 보내지 않고 대기하는 시간이 발생 한다. 이 시간을 보통 ‘Think Time’이라고 표현한다. 이 Think Time을 잘 표현해야만 좀 더 정교한 Test Plan이 완성된다. JMeter에서 이 Think Time을 표현하는 방법이 Timer다.

JMeter에서 가장 많이 사용하는 Timer는 설정한 일정 시간 동안 작업을 지연하는

Constant Timer다. Gaussian Random Timer, Poisson Random Timer처럼 평균값으로 일정 범위 내의 값을 무작위로 지연해주는 Timer도 있으며, 사용법은 Constant Timer와 거의 동일하다. 그 외에도 많은 종류의 Timer를 기본으로 제공한다.

여기서는 시간을 기준으로 지연해주는 Constant Timer, TPS를 기반으로 지연해주는 Constant Throughput Timer, 동시수행 숫자를 기반으로 지연해주는 Synchronizing Timer를 간단한 예제로 알아보자.

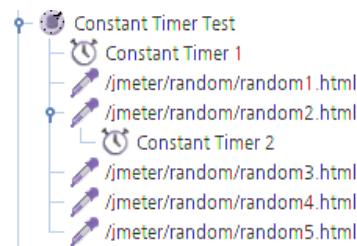
4.8.1 Constant Timer

Constant Timer는 앞에서 설명했듯이 설정된 일정 시간을 지연해주는 기능을 한다. 즉, 지연 시간 기반 Timer다.

추가 및 설정

5개의 HTTP Request Sampler를 추가하고 각각에 2개의 Constant Timer를 추가한다. 2개의 Timer를 서로 다른 위치에 추가하면 범위에 따라 어떻게 영향을 주는지를 보기 위해서다.

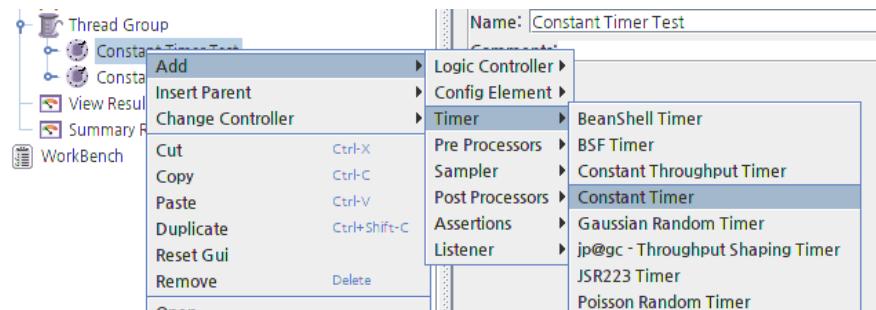
[그림 4-49] Constant Timer



Constant Timer Test(Simple Controller)에서 마우스 오른쪽 버튼을 누르고 ‘Add → Timer → Constant Timer’(Constant Timer 1)를 선택한다. 5개의 HTTP

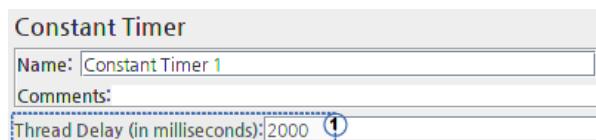
Request Sampler를 추가하고 /jmeter/random/random2.html Sampler에 Constant Timer를 하나 더 추가한다(Constant Timer 2).

[그림 4-50] Constant Timer 추가



각 Timer의 Thread Delay를 2000으로 설정한다. 1/1000초 단위이므로 2000은 2초를 의미한다.

[그림 4-51] Constant Timer 설정



결과 및 분석

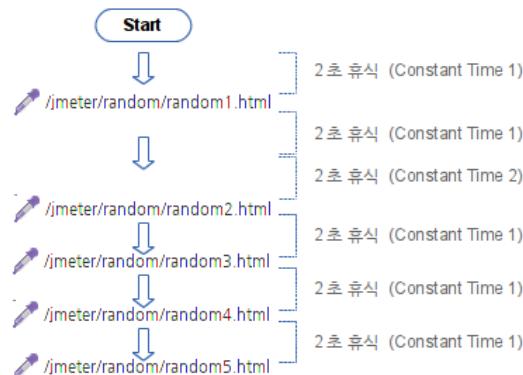
Constant Timer는 [그림 4-52]와 같이 위치에 따라 적용 범위가 결정된다. Constant Timer 1은 같은 레벨과 그 하위 레벨 Sampler에 영향을 주므로 random1.html부터 random5.html까지 모두 영향을 주고, Constant Timer 2는 random2.html의 자식 노드에 속해 있으므로 부모 Sampler에만 영향을 준다.

[그림 4-52] Constant Timer 결과



Timer를 적용할 때 가장 흔하게 범하는 실수는 “지연이 언제 발생하는가?”에 대한 것이다. Sampler가 실행된 후에 지연이 발생한다고 생각하지만 그 반대다. 지연이 발생한 후에 Sampler가 실행된다. Test Plan 작성 시 꼭 주의해야 한다. 예를 들어, A와 B Sampler가 있을 때 그 사이에 2초의 지연 시간을 삽입하려면 B Sampler에 Timer를 추가해야 한다.

[그림 4-53] Constant Timer 결과 그래프



4.8.2 Constant Throughput Timer

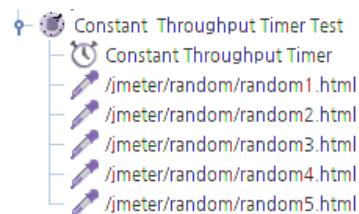
Constant Throughput Timer는 이름에서도 나타나듯이 처리량(Throughput)을 기준으로 하는 Timer다. 즉, 처리량을 일정하게 유지하기 위해 지연시키는

Timer다. 예를 들어, 응답시간이 100ms인 Sampler를 1초에 1번씩 실행되게 하고 싶다면 한 번 실행하고 900ms를 쉬면 된다. 하지만 응답시간이 100ms로 고정된 것이 아니라 늘었다 줄었다 한다면 몇 초를 쉬어야 할까? 이럴 때 사용하는 Timer가 Constant Throughput Timer다. 응답시간 100ms이면 900ms를 지연하고 응답시간이 500ms로 증가하면 500ms를 지연하는 방식으로 처리량을 일정하게 유지시켜 준다.

추가 및 설정

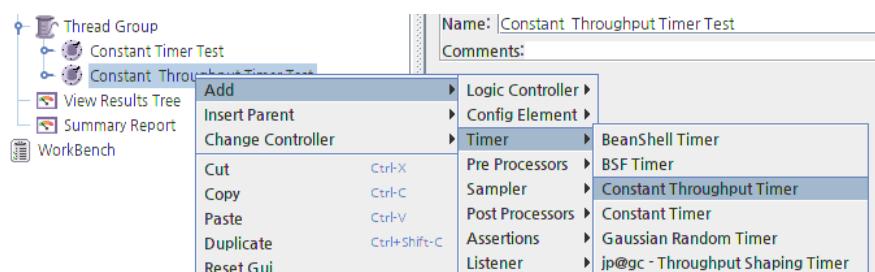
Constant Throughput Timer를 하나 추가하고 5개의 HTTP Request Sampler를 추가한다.

[그림 4-54] Constant Throughput Timer



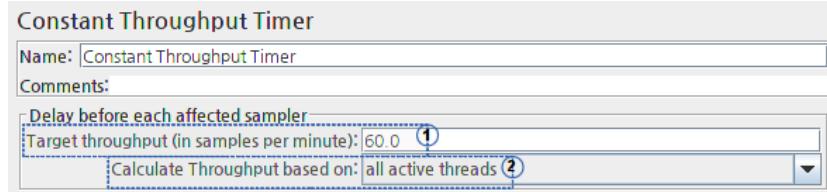
추가하려는 위치에서 마우스 오른쪽 버튼을 누르고 ‘Add → Timer → Constant Throughput Timer’를 선택한다.

[그림 4-55] Constant Throughput Timer 추가



Target throughput의 단위는 samples per minute이므로 60이면(①) 분당 60번 즉, 초당 1번 요청한다는 뜻이다. Calculate Throughput base on은 ①에서 입력한 Throughput 값을 Thread별로 계산할 것인지 모든 Thread를 합쳐서 계산할 것인지에 대한 설정이다. 여기서는 all active threads로 설정한다(②).

[그림 4-56] Constant Throughput Timer 설정



결과 및 분석

설정에서 Target throughput을 60, Calculate Throughput을 모든 Thread 기반으로 설정했으므로 Throughput은 1tps가 나와야만 한다.

Thread는 1, Loop Count는 10으로 테스트하면 결과는 [그림 4-57]과 같다. ①에서 보듯이 TOTAL Throughput이 1.0/sec임을 확인할 수 있다($1.0/\text{sec} = 60.0/\text{minute}$).

[그림 4-57] Constant Throughput Timer 결과

The screenshot shows the 'Summary Report' dialog with the following data:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
/jmeter/random/random1.html	10	2	2	3	0.46	0.00%	13.3/min	0.08	370.0
/jmeter/random/random2.html	10	2	2	3	0.46	0.00%	13.3/min	0.08	370.0
/jmeter/random/random3.html	10	1	1	3	0.54	0.00%	13.3/min	0.08	370.0
/jmeter/random/random4.html	10	2	1	3	0.67	0.00%	13.3/min	0.08	370.0
/jmeter/random/random5.html	10	2	2	3	0.46	0.00%	13.3/min	0.08	375.0
TOTAL	50	2	1	3	0.56	0.00%	1.0/sec ①	0.37	371.1

설정 레퍼런스

[표 4-7] Constant Throughput Timer 설정 레퍼런스

옵션	설명
Target throughput (in samples per minute)	목표 Throughput 값을 입력한다. 다른 Element와는 다르게 단위가 samples/minute다. sample/second로 환산하려면 60으로 나누어야 한다(1 sample/second = 60 sample/minute).
Calculate Throughput base on	Target throughput을 각각의 Thread에 부여할지 모든 Thread나 Thread Group 내 Thread의 합으로 계산할지에 대한 설정이다. <ul style="list-style-type: none">• this thread only : 각각의 Thread가 Target throughput을 유지하려고 한다. 따라서 전체 Throughput은 'Target throughput × Active thread 수'가 된다.• all active threads in current thread group : Target Throughput이 Thread Group 내의 Active Thread 수만큼 나뉘어서 각 Thread는 이전 수행을 기초로 지연 시간을 결정한다.• all active threads : Target throughput이 모든 Active Thread 수만큼 나뉘어서 각 Thread는 이전 수행을 기초로 지연될 시간을 결정한다. Thread Group 별로 Constant Throughput Timer 설정이 동일하거나 Test Plan 최상위에 하나만 설정하는 것이 좋다.• all active threads in current thread group (shared) : all active threads in current thread group과 대부분 같고, 자신을 포함한 다른 Thread의 이전 수행을 기초로 지연 시간을 결정한다.• all active threads (shared) : all active threads와 대부분 같고, 자신을 포함한 다른 Thread의 이전 수행을 기초로 지연 시간을 결정한다.

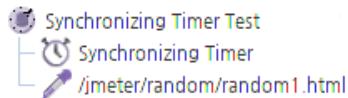
4.8.3 Synchronizing Timer

이 Timer는 앞의 두 유형의 Timer와는 다른 형태의 Timer로, 동시에 요청하기 위해 지연하는 역할을 수행한다. 여러 Thread가 존재할 때 각각의 Thread는 같은 요청을 하더라도 응답시간이 다르게 마련이다. 이때 응답시간이 빠른 Thread가 가장 느린 Thread에 맞춰서 기다렸다가 동시에 요청하게 하는 역할을 한다. 간단한 테스트 스크립트로 그 결과를 비교해 보자.

추가 및 설정

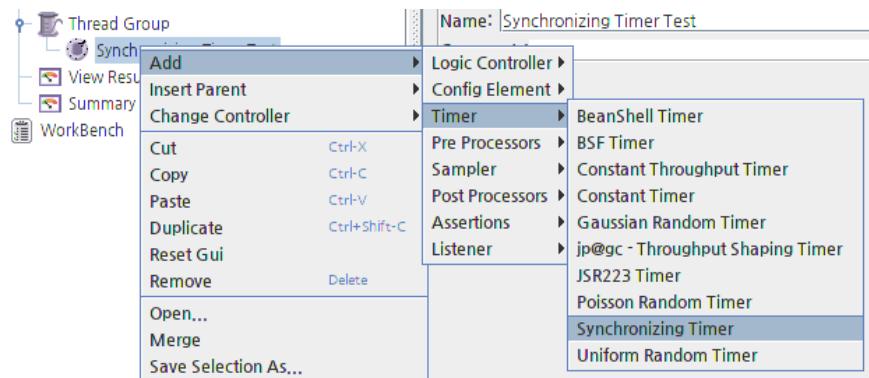
[그림 4-58]과 같이 Synchronizing Timer와 HTTP Request Sampler를 하나씩 추가한다.

[그림 4-58] Synchronizing Timer



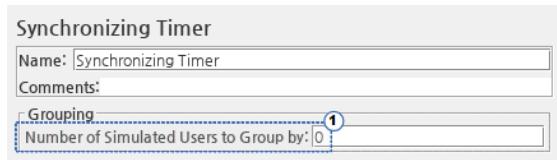
추가하려는 위치에서 마우스 오른쪽 버튼을 누르고 ‘Add → Timer → Synchronizing Timer’를 선택한다.

[그림 4-59] Synchronizing Timer 추가



Number of Simulated Users to Group by를 0으로 설정한다(①). 0은 전체 Thread 숫자를 의미하는데, 전체가 아닌 일부 Thread만 동기화하고 싶다면 이 값에 원하는 숫자를 입력한다.

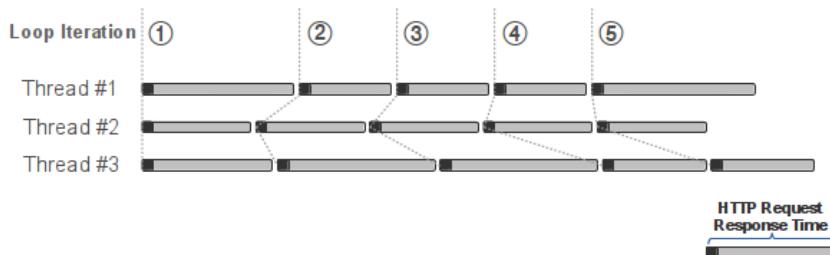
[그림 4-60] Synchronizing Timer 설정



결과 및 분석

Thread Group에서 Thread는 3, Loop Count는 5로 설정하고. Synchronizing Timer를 사용했을 때와 사용하지 않았을 때를 비교하여 테스트한다. Synchronizing Timer를 사용하지 않았을 때는 [그림 4-61]과 같이 각각의 Iteration이 Thread별로 응답 종료시점에 바로 시작하고, Synchronizing Timer를 사용하면 [그림 4-62]와 같이 모든 Thread가 종료된 후 동시에 요청이 시작된다.

[그림 4-61] Synchronizing Timer 사용하지 않은 결과



[그림 4-62] Synchronizing Timer 사용 결과



적용 범위는 Constant Timer와 마찬가지로 부모 노드와 자기 자신 이하의 자식 노드에 영향을 미친다. 이 예제에서는 Synchronizing Timer Test(Simple Controller) 아래의 모든 Sampler가 동기화된다.

4.9 Assertions

Test Plan 실행 시 HTTP Request가 성공했는지, 실패했는지는 HTTP 응답코드로 나뉜다. 즉, 20X나 30X 응답코드는 성공이고, 40X나 50X은 오류로 처리된다. 하지만 20X나 30X 응답코드를 받았다고 모두 성공이라고 볼 수 있을까? 그렇지 않은 경우가 상당히 많다. HTTP 응답코드는 “200 OK”를 받았지만, 본문 내용에는 “처리에 실패했습니다.”라는 메시지를 받는 경우도 있다. 아무런 처리를 하지 않으면 이런 경우는 성공으로 기록된다.

이번에는 실제로 오류로 처리해야 하지만 “200 OK” 응답을 받아서 성공으로 처리된 경우를 구분하는 Assertions이라는 기능을 살펴보자. 테스트 과정에서 가장 많이 하는 실수의 하나가 HTTP 응답코드만 보고 성공/오류를 판단해서 정상적인 테스트를 수행하지 못하는 경우다.

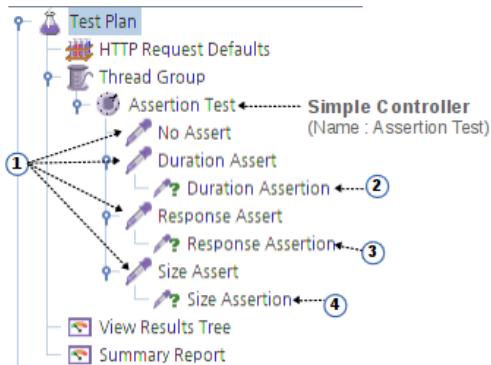
JMeter에는 많은 종류의 Assertion이 있는데, 여기에서는 Duration Assertion, Response Assertion, Size Assertion을 이용한 예제와 사용법을 살펴본다.

4.9.1 예제

추가 및 설정

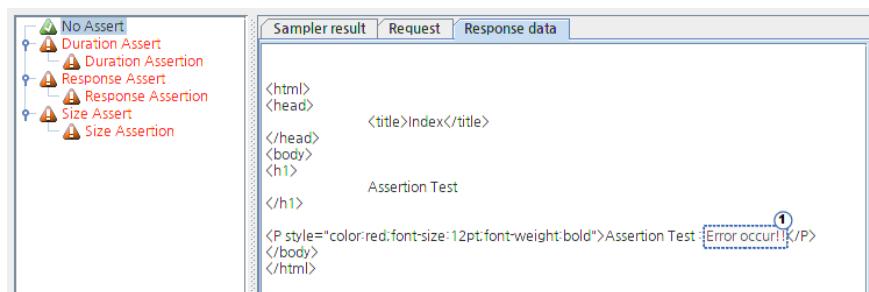
4개의 HTTP Request Sampler를 추가한다([그림 4-63]의 ①). 4개의 Sampler는 이름만 다를 뿐 같은 URL을 나타낸다(/jmeter/assert/index.html).

[그림 4-63] Assertions



이 URL은 HTTP 응답코드가 “200 OK”이지만 내용이 오류다([그림 4-64]의 ①).

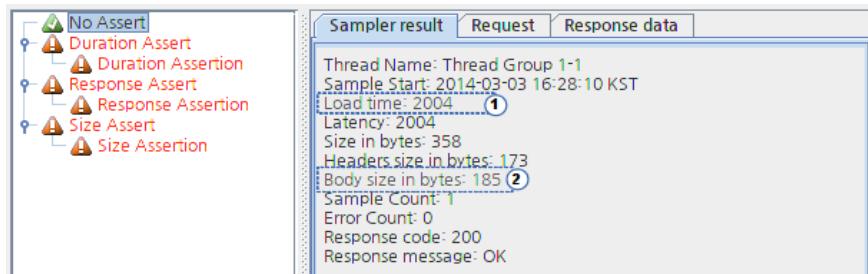
[그림 4-64] 오류 URL 본문 내용



오류가 발생하는 이유는 다음 세 가지다.

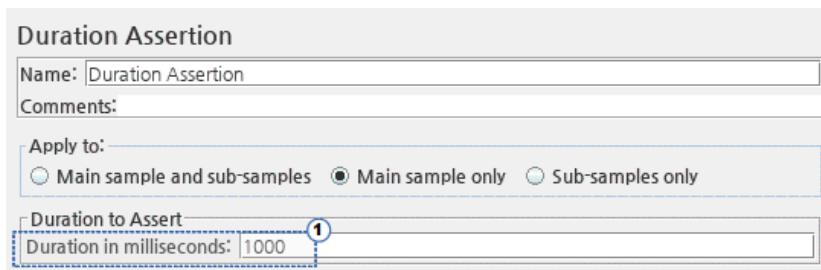
- 내용에 “Error occur!!”를 포함하고 있다.
- 응답시간이 1000ms보다 작아야 하지만, 이 URL은 그보다 큰 응답시간을 나타낸다([그림 4-65]의 ①).
- 응답 본문 사이즈가 500Byte 이상이어야 하지만, 이 URL은 500Byte보다 작다([그림 4-65]의 ②).

[그림 4-65] 오류 URL 응답 헤더



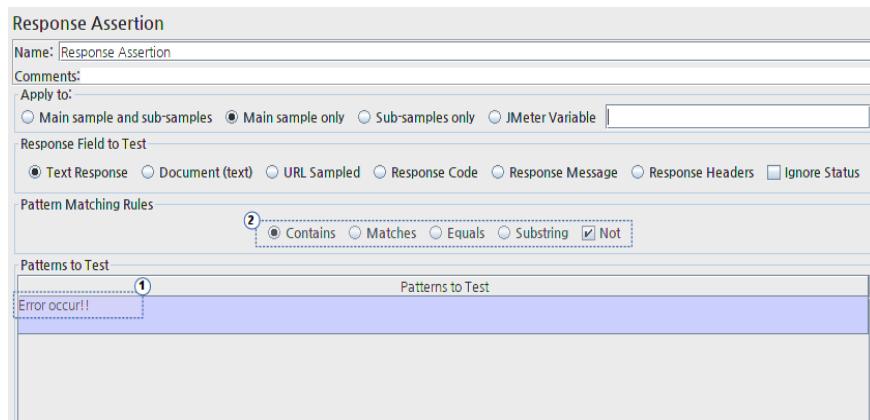
Duration Assertion을 추가하려면 적용하려는 Sampler에서 마우스 오른쪽 버튼을 누르고 ‘Add → Assertions → Duration Assertion’을 선택한다. 그리고 Duration in milliseconds 값을 1000으로 설정한다([그림 4-66]의 ①). 이는 Duration이 1000ms보다 크면 오류로 처리하라는 의미다.

[그림 4-66] Duration Assertion 설정



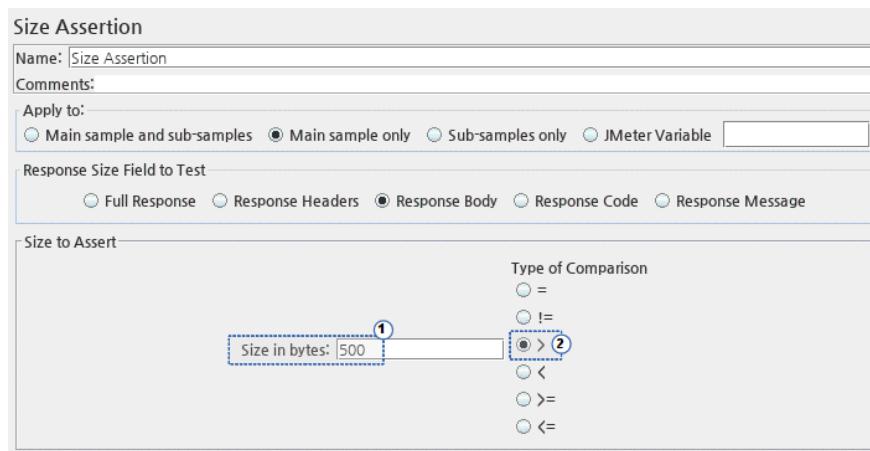
Response Assertion을 추가하려면 적용하려는 Sampler에서 마우스 오른쪽 버튼을 누르고 ‘Add → Assertions → Response Assertion’을 선택한다. 그 다음 Add 버튼을 누르고 Patterns to Test에 Error occur!!를 입력한다([그림 4-67]의 ①). 그리고 Patterns Matching Rules는 Contains을 선택하고([그림 4-67]의 ②) Not을 체크한다. 이는 응답 내용이 “Error occur!!”를 포함하고 있으면 오류로 처리하라는 의미다.

[그림 4-67] Response Assertion



Size Assertion을 추가하려면 적용하려는 Sampler에서 마우스 오른쪽 버튼을 누르고 ‘Add → Assertions → Size Assertion’을 선택한다. 그리고 Size in bytes에 500을 입력하고([그림 4-68]의 ①) Type of Comparison은 >을 선택한다([그림 4-68]의 ②). 이는 응답 본문의 크기가 500bytes보다 작으면 오류로 처리하라는 의미다.

[그림 4-68] Size Assertion



결과 및 분석

Thread Group에서 Thread는 1, Loop Count는 1로 설정해서 테스트한다. 테스트를 수행하고 결과를 View Results Tree Listener를 이용해서 살펴보면 어떤 Assertion에 의해 오류로 처리되었는지 결과를 확인할 수 있다. 따라서 Test Plan을 작성한 후 정상적으로 동작하는지 View Results Tree로 확인하는 것이 좋다.

[그림 4-69], [그림 4-70], [그림 4-71]은 세 가지 Assertion에 의해 오류로 처리된 결과를 보여준다. Assertion이 적용되지 않은 No Assert만 성공으로 처리되고 나머지는 모두 오류로 처리되었다.

[그림 4-69] Duration Assertion 결과

The screenshot shows the JMeter View Results Tree Listener. On the left, there is a tree view with nodes: No Assert, Duration Assert (highlighted in blue), Duration Assertion, Response Assert, Response Assertion, Size Assert, and Size Assertion. On the right, under the 'Assertion result' tab, the output is:

```
Assertion error: false  
Assertion failure: true  
Assertion failure message: The operation lasted too long: It took 2,004 milliseconds, but should not have lasted longer than 1,000 milliseconds.
```

[그림 4-70] Response Assertion 결과

The screenshot shows the JMeter View Results Tree Listener. On the left, there is a tree view with nodes: No Assert, Duration Assert, Duration Assertion, Response Assert (highlighted in blue), Response Assertion, Size Assert, and Size Assertion. On the right, under the 'Assertion result' tab, the output is:

```
Assertion error: false  
Assertion failure: true  
Assertion failure message: Test failed: text expected not to contain /Error occur!//
```

[그림 4-71] Size Assertion 결과

The screenshot shows the JMeter View Results Tree Listener. On the left, there is a tree view with nodes: No Assert, Duration Assert, Duration Assertion, Response Assert, Response Assertion (highlighted in blue), Size Assert, and Size Assertion. On the right, under the 'Assertion result' tab, the output is:

```
Assertion error: false  
Assertion failure: true  
Assertion failure message: The result was the wrong size: It was 185 bytes, but should have been greater than 500 bytes.
```

Assertion 역시 기준의 다른 Element와 동일하게 부모 노드와 자기 자신의 노드에 영향을 미친다. 앞 예제에서도 적용 범위를 각 HTTP Request Sampler의 자식 노드에 적용했으므로 각각의 Sampler에만 적용된 상태다.

설정 레퍼런스

[표 4-8] Duration Assertion 설정 레퍼런스

옵션	설명
Apply to	Main sample은 HTTP Request 자체를 의미하며 Sub-sample은 HTTP Request에서 “Embedded Resources from HTML Files” 옵션을 사용했을 때 추가로 받아오는 자료를 의미한다. <ul style="list-style-type: none">● Main sample and sub-samples : Main과 Sub-sample에서 검색한다.● Main sample only : Main sample에서만 검색한다.● Sub-samples only : Sub-sample에서만 검색한다.
Duration to Assert (Duration in milliseconds)	1/1000 초 단위로 기준 시간을 정한다. Load time이 설정값 이상이 되면 오류로 처리된다.

[표 4-9] Response Assertion 설정 레퍼런스

옵션	설명
Apply to	Duration Assertion과 동일하다.
Response Field to Test	전체 응답 내용 중에서 Patterns to Test에 등록된 검색 패턴을 적용해서 검색 할 자료를 지정한다. <ul style="list-style-type: none">● Text Response : HTTP header를 제외한 Body 내용을 의미한다● Document (text) : Apache Tika로 추출한 문서의 내용이다.● URL sampled : 요청 URL의 내용을 의미한다.● Response Code : e.g. 200● Response Message : e.g. OK● Response Headers : Set-Cookie headers를 포함한 모든 헤더의 내용이다.● Ignore Status : Sample의 성공/실패는 HTTP 응답코드와 Assertion 결과에 의해서 결정된다. 하지만 이 옵션을 체크하면 HTTP 응답코드에 의한 실패를 무시하고 무조건 성공으로 처리하고, Assertion으로만 성공/실패를 판단한다.

옵션	설명
Pattern Matching Rules	<p>검색 패턴이 어떤 식으로 매칭될 때 오류로 처리할지에 대한 설정이다.</p> <ul style="list-style-type: none"> Contains : 텍스트에 정규표현식 형태의 패턴 내용이 포함되어 있으면 True다. Text : "abCdefg" Pattern : b[a-z]d ==> "bCd" (True) Matches : 모든 텍스트 내용이 정규표현식 형태의 패턴과 일치하면 True다. Text : "abCdefg" Pattern : a[a-z]{5}g ==> "abcdefg" (True) Equals : 모든 텍스트 내용이 패턴 문자열과 같으면 True다(case-sensitive). Text : "abCdefg" Pattern : abCdefg ==> "abCdefg" (True) Substring : 텍스트 내용에 패턴 문자열이 포함되어 있으면 True다(case-sensitive). Text : "abCdefg" Pattern : bCd ==> "bCd" (True) Pattern : bcd ==> (False) Not : 결과를 반대로 뒤집는 것을 의미한다. Pattern Matching Rule에 의해서 True가 선택되면 False로 처리한다.

[표 4-10] Size Assertion 설정 레퍼런스

옵션	설명
Apply to	Duration Assertion과 동일하다.
Response Size Field to Test	<p>크기(Size)를 체크하기 위한 대상이 되는 자료를 선택한다.</p> <ul style="list-style-type: none"> Full Response : 응답 전체(Header + Body) 내용을 확인한다. Response Headers : Header 내용만 확인한다. Response Body : Body 내용만 확인한다. Response Code : HTTP 응답코드만 확인한다(e.g. 200). Response Message : HTTP 응답 메시지만 확인한다(e.g. OK).
Size to Assert	Size in bytes에 기준이 되는 크기를 입력하고 Type of Comparison을 이용해서 기준을 정한다. 앞의 예제는 “Response Body(응답 본문)의 크기가 500bytes보다 작으면 True”라는 의미다.

4.10 BeanShell의 활용

일반적인 자바 소스 코드는 자바 컴파일러를 이용해서 바이트코드^{Bytecode} 형태로 바꾼 후에 자바 명령어로 실행한다. 하지만 BeanShell은 이런 과정 없이 자바 코드를 Perl이나 자바스크립트처럼 동적으로 실행할 수 있게 하는 자바 소스 인터프리터다.

BeanShell은 작고 프로그램에 임베디드하기 쉬워서 많은 자바 애플리케이션에서 런타임에 자바 코드를 동적으로 실행하거나 확장성을 제공하기 위해 활용한다. BeanShell은 자바로 작성되었고, 같은 임베디드 애플리케이션과 가상머신을 사용하므로 애플리케이션의 객체를 Beanshell로 보내고 결과를 전달받을 수 있다.

JMeter에서 BeanShell은 Sampler/PreProcessor/PostProcessor/Assertion으로 다양하게 활용된다. BeanShell은 자바 가상머신 위에 자바 코드 인터프리터 계층이 하나 더 존재하여 많이 사용하면 JMeter 자체의 성능이 떨어질 수 있으므로 대용량 테스트를 할 때에는 꼭 필요한 부분이 아니라면 사용하지 않는 것이 좋다.

4.10.1 변수 설정 및 사용

BeanShell 내에서 사용되는 변수가 아닌 HTTP Request Sampler 등의 다른 Element에서 사용할 수 있는 변수를 설정하고 접근하는 방법이다. vars는 미리 정의된 변수로, JMeter에서 사용하는 변수를 설정하고 가져오는 데 사용된다.

다음 예는 counter라는 변수를 설정하거나 값을 가져오는 방법을 보여준다. 이렇게 설정된 변수는 HTTP Request Sampler에서 \${counter} 형태로 볼 수 있다.

【읽어오기】

```
String counter = vars.get("counter");
```

[설정하기]

```
vars.put("counter","12345");
```

4.10.2 log 변수

미리 정의된 log 변수는 \${jmeter_path}/bin/jmeter.log 파일에 로그 내용을 추가하는 역할을 한다.

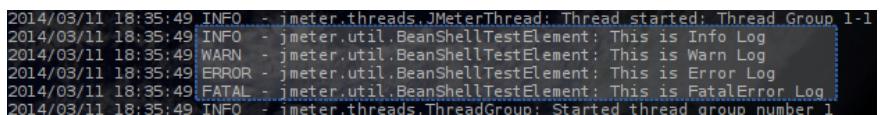
[로그 출력]

```
log.debug("This is Debug Log");
log.info("This is Info Log");
log.warn("This is Warn Log");
log.error("This is Error Log");
log.fatalError("This is FatalError Log");
```

jmeter.log 내용

[그림 4-72]를 보면 This is Debug Log라는 내용이 없는 것을 확인할 수 있다. 이는 bin/jmeter.properties 파일에 log_level.jmeter=INFO로 설정되어 있기 때문이다. Debug 내용을 보려면 log_level.jmeter=DEBUG로 수정한다.

[그림 4-72] BeanShell log 출력



```
2014/03/11 18:35:49 INFO  - jmeter.threads.JMeterThread: Thread started: Thread Group 1-1
2014/03/11 18:35:49 INFO  - jmeter.util.BeanShellTestElement: This is Info Log
2014/03/11 18:35:49 WARN  - jmeter.util.BeanShellTestElement: This is Warn Log
2014/03/11 18:35:49 ERROR - jmeter.util.BeanShellTestElement: This is Error Log
2014/03/11 18:35:49 FATAL - jmeter.util.BeanShellTestElement: This is FatalError Log
2014/03/11 18:35:49 INFO  - jmeter.threads.ThreadGroup: Started thread group number 1
```

4.10.3 Properties 접근

프로퍼티Properties 객체(props)는 JMeter의 설정값을 읽어오거나 설정할 때 사용된다. 프로퍼티를 설정하면 모든 Thread에 공통으로 적용되므로 다른 Thread 그룹

과 정보를 공유하는 용도로도 사용할 수 있다.

[읽어오기]

```
String log_level = props.get("log_level.jmeter");
```

[설정하기]

```
props.put("log_level.jmeter", "ERROR");
```

이 예에서 "log_level.jmeter"의 프로퍼티를 동적으로 설정했지만, JMeter 자체의 log_level이 변경되어 ERROR level로 즉시 동작하지는 않는다. JMeter는 실행 시 설정된 log_level로 계속 동작한다. 이는 단순히 값을 설정할 수 있다 정도의 예로 생각하기 바란다.

4.10.4 Test Plan/Thread 컨트롤

BeanShell을 이용하면 해당 Test Plan을 종료 또는 다음 번 Loop Iteration으로 넘어가거나 Thread 번호를 확인하는 등의 처리를 할 수 있다.

HTTP Request의 응답 내용 중 특정 문자가 포함된 경우 테스트하기

다음은 BeanShell PostProcessor를 이용한 소스다. BeanShell PostProcessor의 prev 변수는 BeanShell Sampler의 SampleResult(org.apache.jmeter.samplers.SampleResult)와 같다. 즉, PostProcessor의 대상이 되는 Sampler를 나타낸다.

[그림 4-73] BeanShell을 이용한 테스트 종료

```
2 | String resdata = new String(data);
3 |
4 |if(resdata.contains("Stop")){
5 |    log.info("Stop Occur!!");
6 |    prev.setStopTestNow(true); // "Stop" 문자열이 포함되어 있으면
7 |} // Test를 바로 종료 한다.
```

Test를 멈추는 것이 아니라 다음 Loop Iteration을 실행하려면 setStartNextThreadLoop(true)를 이용한다.

[그림 4-74] BeanShell을 이용하여 다음 Iteration으로 이동하기

```
2 String resdata = new String(data);
3
4 if(resdata.contains("Stop")){
5     log.info("Stop Occur!!");
6     //prev.setStopTestNow(true);
7     prev.setStartNextThreadLoop(true);
8 }
```

Thread 번호에 따라 변수에 서로 다른 값 설정하기

다음은 BeanShell Sampler를 이용한 소스다.

[그림 4-75] Thread 번호에 따른 분기

```
12 int num = ctx.getThreadNum()%3;           Thread 번호를 가져온 후 3으로 나머지 연산을 해서 변수에 저장한다.
13
14 switch(num){
15     case 0:
16         vars.put("host","top.cafe.daum.net");
17         vars.put("port","80");
18         vars.put("path","/_c21/_bestcafe");
19         break;
20     case 1:
21         vars.put("host","section.cafe.naver.com");
22         vars.put("port","80");
23         vars.put("path","/RandomPowerCafeList.nhn");
24         break;
25     case 2:
26         vars.put("host","news.nate.com");
27         vars.put("port","80");
28         vars.put("path","/recent?mid=n0100");
29         break;
30 }
```

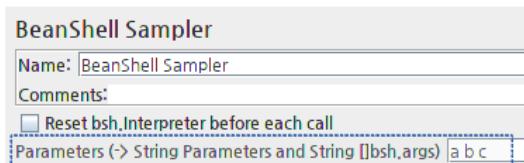
4.10.5 미리 정의된 변수

BeanShell을 사용하는 Element마다 미리 정의된 변수가 있다. 이는 BeanShell이 호출될 때 JMeter의 Context에서 전달받은 변수들이라고 보면 된다. 이를 이용해서 변수를 설정하거나 로그 쓰기 등의 작업을 수행할 수 있다.

BeanShell Sampler

- Parameters, bsh.args : 스크립트에 전달되는 파라미터에 접근하는 변수다. BashShell Sampler나 Pre/PostProcessor의 Parameters 부분에 a b c라고 설정하고 스크립트 영역에 [그림 4-77]의 코드를 입력하면 jmeter.log의 결과는 [그림 4-78]과 같다. 즉, Parameters는 Parameters 필드에 입력된 내용 전체가 되고, bsh.args은 Parameters 필드의 내용을 하나의 공백 문자로 분리한 String 배열이 저장된다.

[그림 4-76] Parameter 설정



[그림 4-77] Parameter 사용 코드

```
log.info("bsh.args[1]="+bsh.args[1]);
log.info("Parameters="+Parameters);
```

[그림 4-78] Parameter 사용 결과 로그

```
2014/03/13 19:28:17 INFO - jmeter.util.BeanShellTestElement: bsh.args[1]=b
2014/03/13 19:28:17 INFO - jmeter.util.BeanShellTestElement: Parameters=a b c
```

- SampleResult : 현재 Sampler(BeanShell Sampler)의 SampleResult를 나타낸다. (Class : org.apache.jmeter.samplers.SampleResult)
- ResponseCode : 응답코드를 설정하거나 읽어 들일 수 있는 변수(String)다. (예 : “200”)
- ResponseMessage : 응답 메시지를 설정하거나 읽어 들일 수 있는 변수(String)다. (예 : “OK”)

- **IsSuccess** : Sampler의 성공과 실패를 설정하거나 읽어 들일 수 있는 변수 (Boolean)다. (예: true/false)
- **Label** : Sampler의 Label을 설정하거나 읽어 들일 수 있는 변수(String)다.
- **FileName** : 스크립트 파일에 입력된 파일명을 읽어 들일 수 있는 변수(String)다.
- **ctx** : JMeter의 Context에 접근할 수 있게 하는 변수다. Thread, Thread Group 관련 정보와 Loop, JMeterEngine에 접근할 수 있다. (Class : org.apache.jmeter.threads.JMeterContext)
- **vars** : 다른 Element에서 설정되거나 사용될 변수를 설정하고 읽어 들일 수 있게 하는 변수다. (Class: org.apache.jmeter.threads.JMeterVariables)
- **props** : 설정 정보에 접근할 수 있게 하는 변수다. (Class: java.util.Properties)
- **log** : jmeter.log 파일에 로그를 쓸 수 있게 하는 변수다. (Class: org.apache.log.Logger)

BeanShell PostProcessor

- **prev** : 바로 직전에 실행된 Sampler의 SampleResult를 나타낸다. 즉, PostProcessor가 적용되는 Sampler의 SampleResult를 의미한다. (Class: org.apache.jmeter.samplers.SampleResult)
- **data** : 현재의 PostProcessor가 적용된 Sampler의 ResponseData를 나타낸다.
- 나머지 변수는 BeanShell Sampler와 동일하다.

BeanShell PreProcessor

- **prev** : 바로 직전에 실행된 Sampler의 SampleResult를 나타낸다. (Class: org.apache.jmeter.samplers.SampleResult)

- **sampler** : 현재의 PreProcessor가 적용된 Sampler를 나타낸다. HTTP Sampler라면 org.apache.jmeter.protocol.http.sampler.HTTPSamplerBase을 참조한다.
- 나머지 변수는 BeanShell Sampler와 동일하다.

4.11 Proxy 서버를 이용한 리코딩

LoadRunner와 같은 대부분의 상용 성능 테스트 솔루션에는 스크립트 레코딩 툴이 포함되어 있다. 하나하나 손으로 작성하는 것이 아니라 사용자가 웹 브라우저를 사용함과 동시에 요청 정보가 기록되어 이를 다시 재현해 줄 수 있는 툴이다.

상용 툴보다 다소 복잡하기 하지만 Recording Controller와 HTTP(S) Test Script Recorder를 이용하면 JMeter에서도 이런 기능을 사용할 수 있다.

HTTP(S) Test Script Recorder는 일종의 프락시 서버로, 이 Element를 이용하는 요청 중 원하는 패턴의 요청을 Recording Controller에 저장하는 방식으로 동작한다. 여기에서는 Recording Controller와 HTTP(S) Test Script Recorder로 손쉽게 Test Plan을 작성하는 방법을 알아본다.

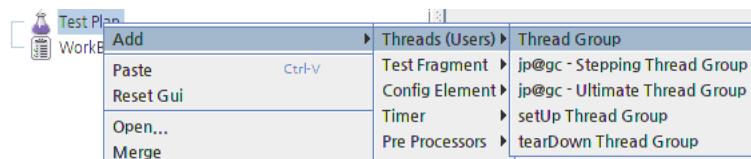
4.11.1 Proxy를 통한 레코딩 예제

이번 예제에서는 JMeter를 이용해서 <http://jmeter.apache.org> 사이트의 *.html 요청을 레코딩해 본다.

Thread Group 추가

Test Plan에서 마우스 오른쪽 버튼을 누르고 ‘Add → Threads (Users) → Thread Group’을 선택하여 추가한다.

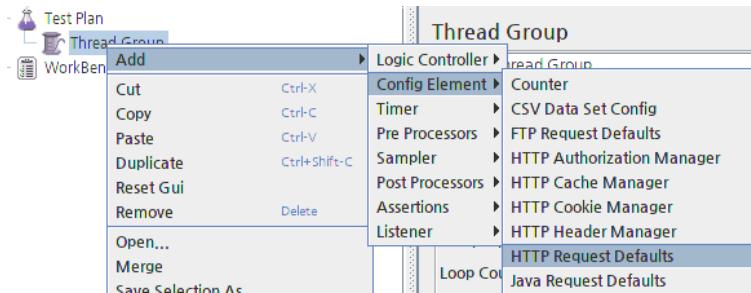
[그림 4-79] Thread Group 추가



HTTP Request Defaults 추가 및 설정

꼭 필요한 Element는 아니지만 나중에 Test Plan을 다듬고 손질하는 데 매우 편리하다. Thread Group에서 ‘Add → Config Element→ HTTP Request Defaults’를 선택하여 추가한다. 설정 창에서 Server Name or IP를 jmeter.apache.org로 설정하고 나머지는 비워 놓는다.

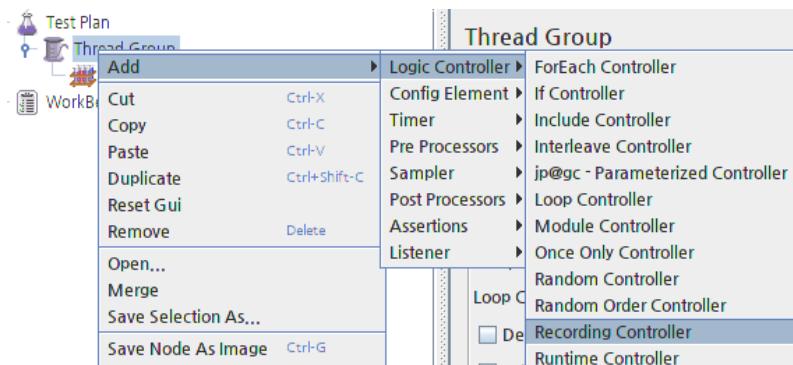
[그림 4-80] HTTP Request Defaults 추가



Recording Controller 추가 및 설정

Thread Group에서 ‘Add → Logic Controller → Recording Controller’를 선택하여 추가한다. 별도의 설정은 필요 없다.

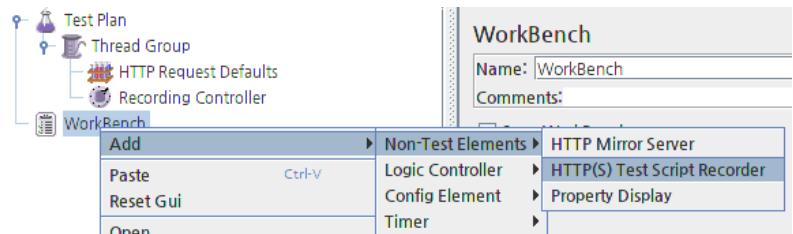
[그림 4-81] Recording Controller 추가



HTTP(S) Test Script Recorder 추가 및 설정

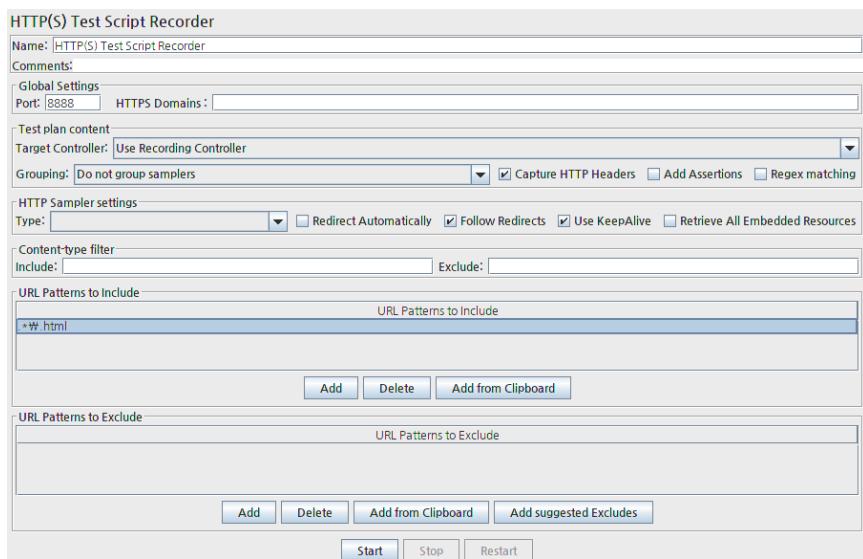
WorkBench에서 ‘Add → Non-Test Elements → HTTP(S) Test Script Recorder’를 선택하여 추가한다.

[그림 4-82] HTTP(S) Test Script Recorder 추가



Port는 8888로 설정한다. 기본값은 8080이지만 로컬에서 WAS를 실행하고 있다면 충돌할 가능성이 높으므로 수정해 준다. URL Patterns to Include의 Add 버튼을 눌러서 `.*\html`을 추가한다. 이는 Script Recorder를 통해서 요청되는 값 중 확장자가 html인 요청만 Recording Controller로 보내라는 의미다.

[그림 4-83] HTTP(S) Test Script Recorder 설정



설정이 완료되면 Start 버튼을 눌러서 HTTP(S) Test Script Recorder를 활성화 한다. 이제 8888 포트로 한 개의 프락시 서버가 실행된다. 어떤 요청들이 HTTP(S) Test Script Recorder를 통해서 오가는지 확인하고 싶으면 HTTP(S) Test Script Recorder의 자식 노드에 View Results Tree를 추가한다. 이때 WorkBench에 추가된 내용은 Test Plan을 저장하고 다시 열면 없어진다는 점을 유의해야 한다.

웹 브라우저의 프락시 설정

웹 브라우저의 프락시 설정을 앞에서 실행한 HTTP(S) Test Script Recorder로 변경한다.

Firefox 설정

'설정 → 고급 → 네트워크 → 프락시 설정 → 프락시 수동 설정'에서 [그림 4-85]와 같이 설정한다.

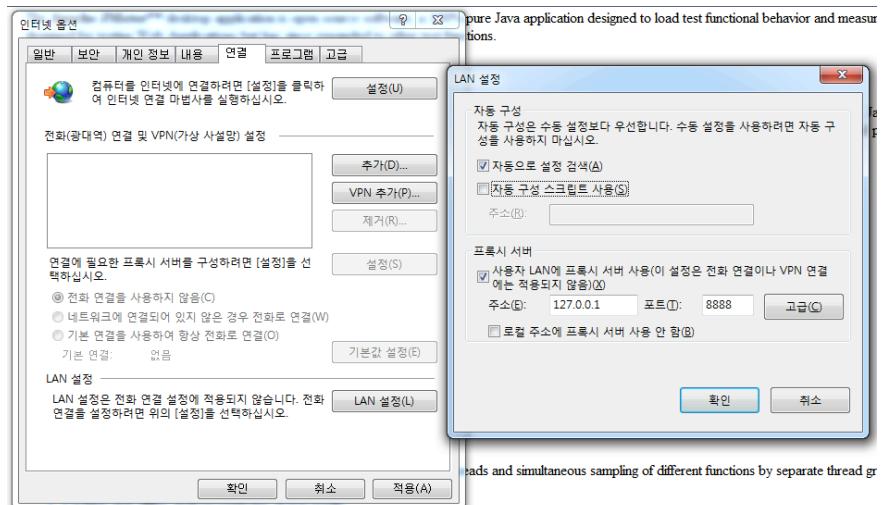
[그림 4-84] Firefox 설정



Internet Explorer 설정

'도구 → 인터넷 옵션 → 연결 → LAN 설정 → 프락시 서버'에서 [그림 4-85]과 같이 설정한다.

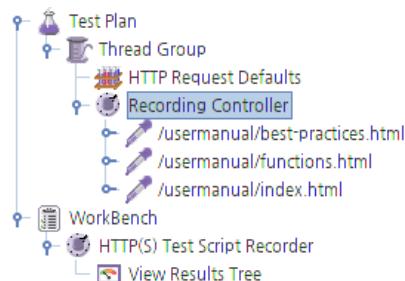
[그림 4-85] Internet Explorer 설정



결과

모든 설정이 끝나면 웹 브라우저에서 <http://jmeter.apache.org>의 이곳 저곳을 클릭한 다음 다시 JMeter로 돌아와서 확인한다. Recording Controller에 [그림 4-86]과 같이 HTTP Request Sampler가 추가된 것을 확인할 수 있다.

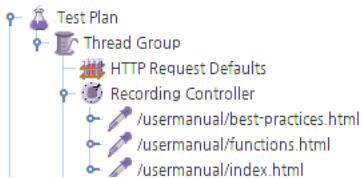
[그림 4-86] 레코딩 결과



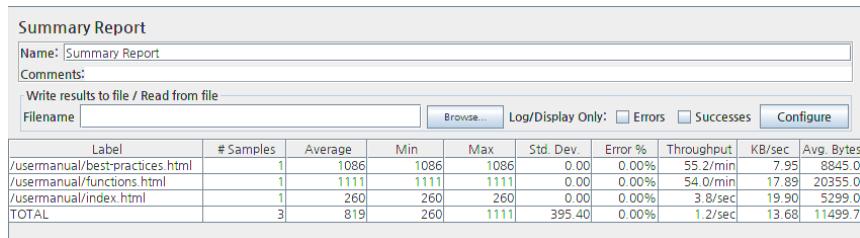
4.11.2 레코딩된 Test Plan 실행

HTTP(S) Test Script Recorder로 레코딩된 내용을 이용해서 바로 테스트할 수 있다. 결과를 살펴보기 위해 Summary Report를 하나 추가한 다음 실행하면 결과는 [그림 4-87]과 같다.

[그림 4-87] Summary Report 추가



[그림 4-88] 레코딩된 Test Plan 실행 결과



4.12 플러그인 활용

JMeter는 그 자체만으로도 훌륭한 툴이지만 다소 아쉬운 부분이 있다. 필자가 가장 아쉬워하는 부분은 Listener다. 테스트 후 보고서를 작성하거나 결과를 분석하기 위한 기능이 매우 빈약하다. Listener를 포함하여 이러한 부족함을 보완해 줄 수 있는 것이 플러그인이다. 다행히 현재 쓸만한 플러그인이 많이 개발되어 도움을 받을 수 있다. 가장 대표적인 플러그인으로는 [JMeter Plugins⁰⁵](#)가 있다.

05 : <http://jmeter-plugins.org>

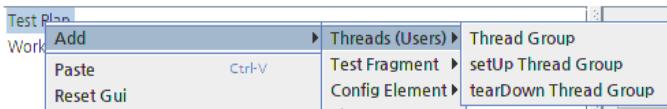
4.12.1 플러그인 설치

플러그인 설치는 상당히 간단하다. 플러그인 파일(jar파일)을 lib\ext 폴더에 옮기고 JMeter를 재시작하면 설치가 끝난다.

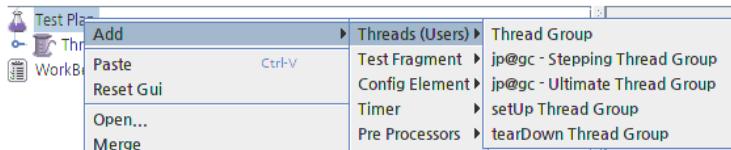
<http://jmeter-plugins.org/downloads/all/>에서 JMeterPlugins-Standard-1.x.x.zip을 내려받는다. 필요하면 JMeterPlugins-Extras-1.x.x.zip도 받는다. 내려받은 압축 파일을 JMeter 설치 디렉터리에서 풀어주면 lib\ext에 JMeterPlugins-Standard.jar 파일이 설치된다.

설치가 완료되면 JMeter를 재실행한다. 별다른 설정은 필요 없으며, JMeter가 로딩되면서 lib와 lib\ext 아래의 모든 파일을 로드하므로 자동으로 추가된다. JMeter를 재실행하면 [그림 4-90]처럼 ‘jp@gc -’ 이름의 Element가 추가된 것을 확인할 수 있다.

[그림 4-89] JMeter Plugin 설치 전



[그림 4-90] JMeter Plugin 설치 후



4.12.2 플러그인 Element를 이용한 예제

이번 예제는 Stepping Thread Group을 이용해서 Thread 숫자를 차례대로 증가시키고, Dummy Sampler로 가상의 결과를 얻어와서 그 결과값을 Response

Times Over Time과 Transaction per Second로 보여주는 간단한 예제다.

[그림 4-91] 플러그인 예제

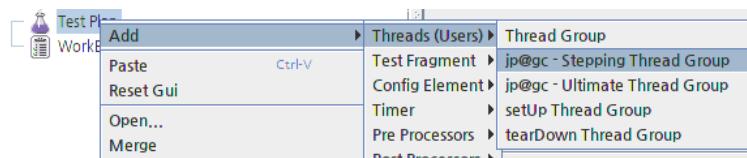


JMeter로 테스트를 하다 보면 다음 네 가지 Element를 자주 사용하게 된다.

Stepping Thread Group 추가 및 설정

Test Plan에서 ‘Add → Threads (Users) → jp@gc - Stepping Thread Group’을 선택하여 추가한다.

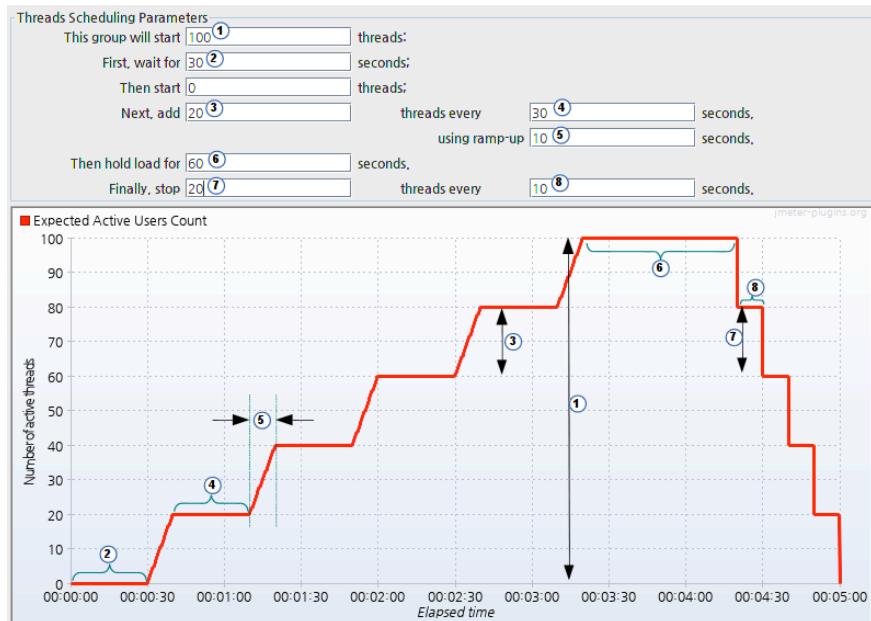
[그림 4-92] Stepping Thread Group 추가



기본으로 제공되는 Thread Group은 Thread 숫자와 Ramp-Up 시간 정도만 설정할 수 있으나 Stepping Thread Group는 증가폭과 감소폭을 좀 더 정밀하게 설정할 수 있다. 또한, 그 결과를 그래프로 보여주므로 설정이 쉽다.

각 항목의 의미는 [그림 4-93]의 위와 아래의 그림 번호를 매칭해서 보면 이해할 수 있으므로 설명은 생략한다.

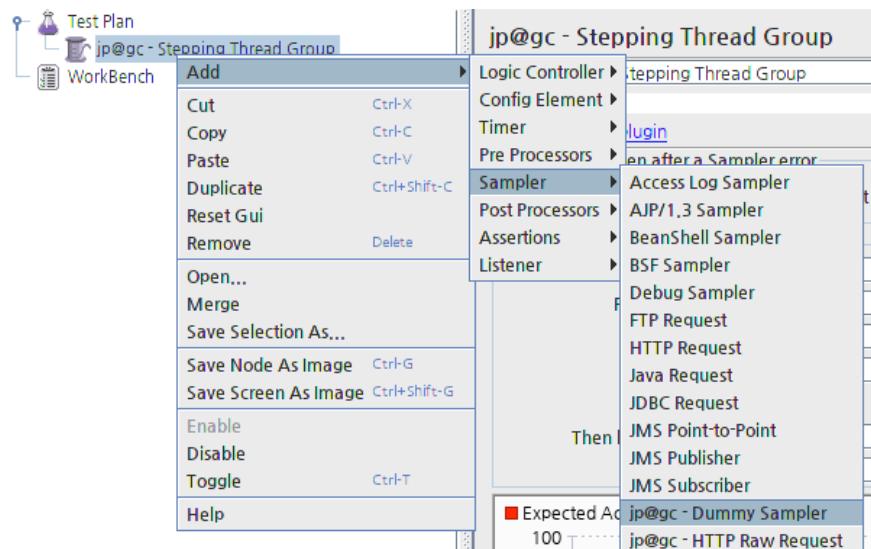
[그림 4-93] Stepping Thread Group 설정



Dummy Sampler 추가 및 설정

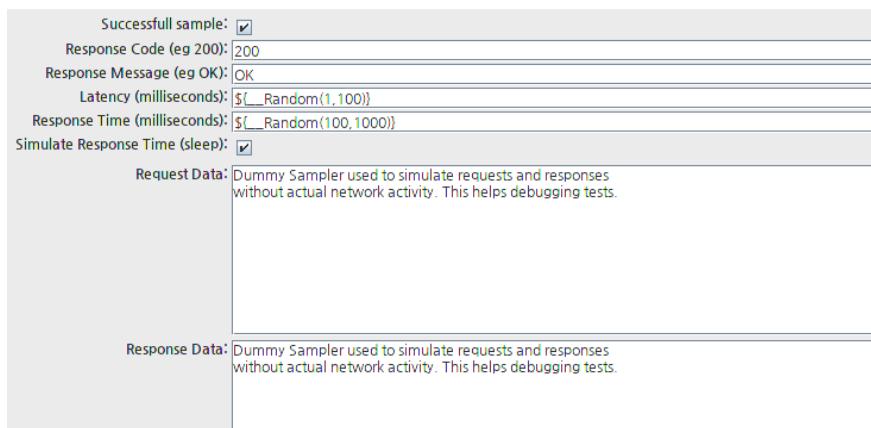
Dummy Sampler는 이름처럼 데미dummy 데이터를 전달해주는 Sampler다. Test Plan 작성 시 결과값에 따라 로직이 제대로 분기되는지를 검증할 때 사용하면 유용하다. Stepping Thread Group에서 'Add → Sampler → jp@gc - Dummy Sampler'를 선택해서 추가한다.

[그림 4-94] Dummy Sampler 추가



응답코드와 응답 메시지, 응답시간 등을 원하는 대로 편집할 수 있다. 이번 테스트에서는 기본값을 사용한다.

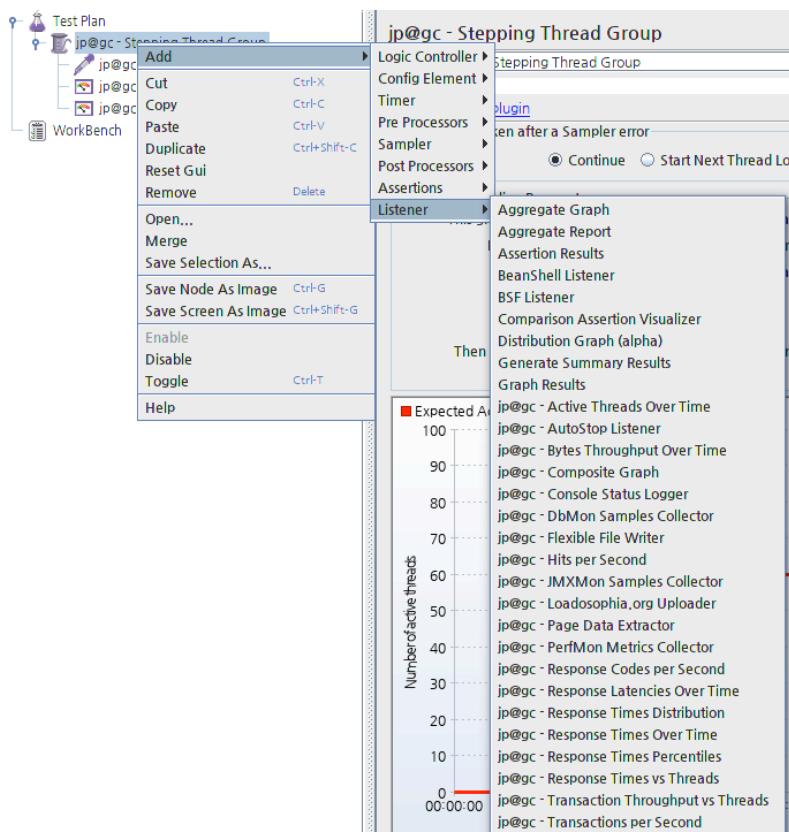
[그림 4-95] Dummy Sampler 설정



Listener 추가

이번 테스트에서는 Response Times Over Time과 Transaction per Second를 추가하고 그 결과를 살펴본다. 추가 방법은 ‘Stepping Thread Group → ‘Add → Listener’에서 jp@gc Response Times Over Time와 jp@gc Transaction per Second를 선택하면 된다. 이 두 가지 Listener는 성능 테스트에서 가장 일반적으로 모니터링하는 응답시간과 처리량을 시간대별 그래프로 보여준다. 보고서 작성이나 결과 분석에 매우 유용하다.

[그림 4-96] Listener 추가



결과 그래프

작성된 예제를 구동해 보자. Thread는 총 100개까지 생성되며 총 5분 동안 구동된다(Stepping Thread Group 참조). [그림 4-97]과 [그림 4-98]은 그 결과 그래프다.

[그림 4-97] Response Times Over Time 결과 그래프



[그림 4-98] Transaction per Seconds 결과 그래프



4.13 TCP Sampler의 TCPClient 확장하기

웹 서비스의 테스트를 하다 보면 간혹 HTTP와 TCP가 혼용된 서비스가 있다. 이런 경우 보통은 웹 브라우저가 아닌 별도의 애플리케이션을 이용해서 서버와 통신하게 된다.

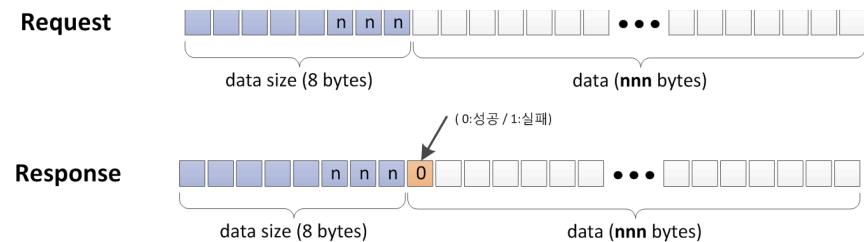
HTTP 프로토콜은 JMeter의 HTTP Request Sampler를 이용하면 대부분 쉽게 구현할 수 있지만, TCP Sampler는 서비스마다 자체적인 통신 규약을 만들어서 사용하는 경우가 많아서 TCPClient를 확장해서 별도의 클래스를 만들어야 한다.

이번에는 간단한 TCPClient 확장 클래스를 만들고 적용하는 방법을 알아본다. 여기서는 자바에 대한 기본 지식이 있다고 가정하고 설명한다.

4.13.1 TCP 데이터 프로토콜(가정)

요청 전문 데이터는 텍스트 기반이며 [그림 4-99]와 같은 형태를 보인다고 가정한다. 요청(Request)/응답(Response) 전문의 마지막에는 별도의 EOF^{End of File}/EOM^{End of Messages} 문자가 없고, 응답 전문의 9번째 문자열은 요청의 성공/실패를 나타내는 플래그^{flag}라고 가정한다. 이 결과 코드의 플래그 값이 '1'이면 Sampler의 요청 결과가 실패로 나타나야 한다. 요청과 응답의 처음 8Bytes는 뒤따라올 데이터의 크기를 Byte 단위로 나타낸다.

[그림 4-99] TCP 데이터 구조



4.13.2 Apache JMeter Source 준비

Apache JMeter 홈페이지에서 (Binary 버전이 아닌) Source 버전을 내려받는다. 자바 빌드 툴인 ant가 설치되어 있지 않으면 ant도 설치해야 한다. 이클립스에 프로젝트를 추가해서 빌드하는 방법도 있지만 ant를 이용하는 방법이 가장 쉽다.

4.13.3 확장 클래스 생성

내려받은 Source 파일의 압축을 풀고 \${Source}\src\protocol\tcp\org\apache\jmeter\protocol\tcp\sampler 폴더로 이동한다. 직접 TCPClient interface로 클래스를 생성해도 되지만, 여기에서는 TCPClient interface를 확장해서 만들어 놓은 AbstractTCPClient 클래스를 이용한다.

TCPClientImpl.java를 MyTCPClientImpl.java로 복사한 다음 Source 안의 클래스 이름도 MyTCPClientImpl로 수정해야 한다.

```
$> cp TCPClientImpl.java MyTCPClientImpl.java
```

MyTCPClientImpl.java의 read 함수 부분을 다음과 같이 수정한다.

```
public String read(InputStream is) throws ReadException{
    ByteArrayOutputStream w = new ByteArrayOutputStream();
    try {
        // Check Response Size
        byte[] bsize = new byte[8];
        int x = is.read(bsize);
        if(x < 0){ throw new ReadException("Size Read Error", null,
w.toString()); }
        w.write(bsize, 0, x);
        int data_size = Integer.parseInt(new String(bsize));
```

```
boolean error = true;
int toread = data_size;

while(true){
    byte[] data = new byte[4096];
    x = is.read(data,0,4096);
    if(x < 0){ throw new ReadException("Data Body Read Error", null,
w.toString()); }

    if(toread == data_size && x > 0){
        // Error Code Check
        // -> Success : 0 (Dec 48) (Hex 30)
        // -> Error   : 1 (Dec 49) (Hex 31)
        if(data[0] == 48){
            error = false;
        }
    }

    toread -= x;
    w.write(data, 0, x);

    if(toread <= 0 || x == 0){
        // Read Done
        break;
    }
}

if(error == true){
    throw new ReadException("Return Code Error", null, w.toString());
}
return w.toString(charset);
} catch (IOException e) {
    throw new ReadException("", e, w.toString());
}
}
```

4.13.4 빌드

빌드 과정은 단순하다. 별도의 설정 없이 다음 명령어를 실행하면 추가된 클래스도 포함하여 빌드된다.

```
$> ant download_jars  
$> ant [install]
```

4.13.5 새로운 클래스 사용

Thread Group 아래 TCP Sampler를 추가한다. TCPClient classname 부분에 추가된 MyTCPClientImpl을 입력한다.

[그림 4-100] TCP Sampler 설정



jmeter.properties에도 설정을 추가해야 한다. '#'으로 주석 처리된 부분을 제거하고 MyTCPClientImpl을 입력한다.

```
tcp.handler=MyTCPClientImpl
```

대용량 웹 성능 테스트

4장까지가 JMeter로 웹 성능 테스트를 하기 위한 준비 작업이었다면, 5장부터는 실제 대용량 웹 성능 테스트에서 JMeter를 어떻게 사용하는지와 그 준비 과정, 그리고 그 결과를 분석하는 방법을 알아본다.

5 | 테스트 계획

테스트 과정에서 가장 먼저 해야 할 것은 ‘무엇을’, ‘언제’, ‘누가’, ‘어떻게’ 테스트 할 것인지 계획을 수립하는 과정이다. 명확한 목표와 이에 맞는 계획을 세워야만 제대로 된 테스트를 할 수 있다. 이 부분은 실제 JMeter로 테스트를 구현하는 부분과는 큰 연관성이 없으므로 몇 가지 구분 방법 및 주의사항만을 정리하겠다.

5.1 요구사항 분석 및 목표 설정

계획 단계에서 먼저 해야 할 작업은 ‘요구사항 분석’이다. 정확한 요구사항이 수집되어야만 이에 맞는 테스트 목적을 설정할 수 있고, 이 목적에 맞는 테스트 방법과 일정을 산정할 수 있다.

5.1.1 테스트 목표

신규 서비스(리뉴얼) 오픈 테스트

목표한 성능이 나오는가? 성능 지표는 보통 처리량, 응답시간, CPU 등의 H/W 리소스 사용량 등으로 판별한다(디버깅: S/W, H/W, 네트워크 구성).

비교 테스트(리뉴얼 시 기존과 비교 테스트)

H/W를 증설하거나 S/W를 변경한 후에 기존과 어떤 차이가 있는지를 확인하는 테스트다.

사이징 테스트

이벤트 등을 대비해서 얼마나 많은 시스템을 구축해야 하는지 사이징을 계산하기 위한 테스트다. 보통 스트레스(Stress) 테스트로 단위 H/W가 처리할 수 있는 최대의 처리량을 계산해서 그 값을 기준으로 전체 사이징을 계산한다.

5.1.2 가상 사용자

필요한 가상 사용자가 몇 명인가에 따라서 테스트에 필요한 부하발생기의 수, 사양, 테스트 방법이 달라질 수 있으므로 계획 단계에서 정리해야 한다.

가상 사용자의 숫자가 매우 많아야 하는 경우

- 보통 1대의 부하발생기로는 300~500명 정도의 가상 사용자를 생성해서 테스트하는 것이 일반적이다. 요즘은 PC 성능이 좋아져서 그 이상도 충분히 처리할 수 있지만, 부하발생기 자체 성능에 부담 없이 테스트가 가능한 수치라고 보면 된다(사실 스크립트 작성 내용에 따라서 많이 달라지는 부분이다).
- 10,000명 이상 많은 수의 가상 사용자가 필요하면 고성능의 부하발생기 1대보다는 성능이 조금 떨어지더라도 여러 대의 부하발생기를 두는 게 좀 더 낫다.
- 필자가 직접 경험한 예로, 모 대학에서는 요구 조건 중의 하나가 “10,000명이 동시에 접속해도 오류 없이 처리해야 한다”여서 20대의 부하발생기를 이용하여 테스트한 경우가 있었다.

처리량(Throughput)이 매우 높아야 하는 경우

- 가상 사용자보다는 처리량에 초점을 맞춘 경우에는 여러 대의 부하발생기보다는 고사양의 부하발생기를 사용하는 것이 좀 더 테스트하기 편하다.

5.1.3 범위

전체 서비스 중에서 어느 부분을 얼마만큼 테스트할지를 고민해야 한다. 크게 하드웨어적 범위와 소프트웨어적 범위로 나눌 수 있다.

하드웨어적 범위

- 네트워크와 보안 장비 포함 여부 : 테스트 환경을 구축할 때 매우 중요한 부분

으로, 테스트 결과에도 큰 영향을 미친다. 테스트 목표가 해당 네트워크와 보안 장비의 성능이나 기능이 아니라면 가능한 우회하는 것이 좋다.

- **실 서비스 서버 vs 테스트 서버** : 실 서비스 서버를 테스트하는 것이 일반적이지만 불가능한 상황이라면 테스트 서버에서 테스트를 진행하고 그 결과를 참고하여 실 서비스 서버의 결과를 유추하는 테스트 방식이다.
- **서버 전체 vs 일부 서버** : 실제 서비스 중 테스트를 진행하는 경우에는 서버 전체를 대상으로 하는 것이 아니라 실 서비스 서버 중에서 일부를 서비스에서 제외하고 이 서버를 이용해서 테스트한 다음 이 결과로 전체 서비스 서버의 결과를 유추하는 테스트 방식이다.

소프트웨어적 범위

- **전체 서비스 vs 일부 서비스** : 전체 서비스를 테스트하면 좋지만 그다지 중요하지 않은(사용자의 요청이 적은) 부분도 있고 비슷한 로직이 중복되는 경우도 있으므로 이럴 때에는 해당 부분은 제외하고 테스트한다.
- **기능적 제약에 따른 분리** : 클라이언트 사이드 암호화 모듈이나 결제 모듈과 같이 ActiveX로 구동되거나 자바스크립트에서 많은 연산을 처리하는 경우에는 테스트할 수 없거나 테스트 스크립트 작성이 매우 복잡해질 수 있다. 이런 경우 해당 부분이 로직에서 매우 중요한 부분이 아니라면 제외하고 테스트할 때가 많다. ActiveX로 구동되는 XecureWeb과 같은 암호화 모듈이 이에 해당한다.

5.2 테스트 일정

테스트 계획과 목표가 설정되고 나면 일정을 조율해야 한다. 목표 설정과 테스트 범위가 결정되면 전반적인 일정과 인력 사용 계획을 확정하고 관련 부서나 외부 업체와의 협의를 시작해야 한다.

문서에 담당 부서 및 담당자를 지정해 주고, 간단한 요구사항 및 주의해야 할 필요 사항 등을 일정표와 함께 제공해 주는 것이 좋다.

테스트에는 많은 인력이 필요하다. 특히 큰 기업은 여러 부서로 업무가 분리되어 있어서 하나의 테스트에 여러 부서의 협조가 필요한 경우가 많다. 특히, 인력 일정 조정을 하려면 내부와 외부의 협의가 필요하다. 외부 인력은 자주, 오랜 시간 협조 요청을 하기 어려우므로 꼭 필요한 시간을 정확히 확인해서 협의해야 한다.

5.3 테스트 계획서 예제

계획서 형식은 업체나 상황에 따라 달라지므로 상황에 맞게 작성하면 된다. 다음 예제는 DB 업그레이드에 따른 성능 비교를 위한 테스트 예제다. 간단하게 목적 및 담당자를 지정하고 일정에 따른 주의사항을 포함하면 좋다.

[그림 5-1] 테스트 계획서 개요 예시 1

1. 성능진단 개요
01. 개요

❖ 목표
신규 Upgrade 서버와 기존 서버의 성능을 비교 테스트 함으로서 신규 Upgrade 이후에 서비스에 문제가 없을지를 판단하기 위한 테스트이다.
서비스 서버의 처리량(Throughput), 응답시간 및 CPU 사용량 등을 통해서 기존 서버와 Upgrade 서버간의 성능 차이를 파악한다.
변경사항은 DB버전 Upgrade(Oracle 10g → 11gR2) / OS버전 Upgrade (AIX 5.3 → 6.1)이다.

❖ 장소 및 일정
• 장소
 일산 IDC 센터 8층
• 일정
 1차 테스트 – 2014년 9월 30일 - 0시
 2차 테스트 – 2014년 10월 2일 - 0시

❖ 참여인력
• 서비스 운영 팀 – 홍OO 과장, 정OO 대리 외 1명
• 개발팀 – 이OO 수석, 김OO 책임
• 테스트 팀 – 장OO 과장
• 외부 지원인력
 • L4 모니터링 – 김OO 대리 (※ XX 네트웍스)
 • DB 모니터링 – 장OO 과장 (※ XX 데이터 시스템즈)

[그림 5-2] 테스트 계획서 개요 예시 2



6 | 테스트 환경 구축

6.1 부하발생기 설치

대용량 성능 테스트를 위해서는 많은 수의 부하발생기와 높은 처리량이 필요하다. 여러 대의 부하발생기를 관리하려면 일괄^{batch} 작업 수행이나 모니터링 같은 작업이 쉬워야 하고 높은 성능을 보여야 한다.

JMeter는 자바로 작성되어서 JVM이 구동되는 대부분의 OS에서 동작한다. 이런 여러 OS 중에서 부하발생기로 사용하기에 가장 적합한 OS는 무엇일까?

필자의 경험으로는 리눅스가 여러 면에서 적합하다고 생각한다. 터미널 환경으로 복수의 서버를 편리하게 관리할 수 있고 커널 파라미터를 튜닝하면 많은 수의 요청을 효과적으로 처리할 수 있다.

이 책에서 사용한 모든 예제 및 샘플 코드는 리눅스(CentOS x86_64)를 기반으로 작성되었다. JMeter는 JVM 기반의 자바 애플리케이션이므로 OS 관련 몇 가지 설정 값 외의 모든 작업은 다른 OS에서도 거의 동일하다고 보면 된다.

부하발생기(jmeter-server)의 설치는 [2.1 실행 환경](#)과 동일하다. 별도의 설치 작업은 필요 없으며 [홈페이지](#)⁰¹에서 프로그램을 내려받아 압축을 풀면 끝난다.

6.2 튜닝

간단한 테스트에서는 문제없지만 대용량 테스트를 하려면 커널 파라미터의 튜닝은 필수다.

01 : https://jmeter.apache.org/download_jmeter.cgi

6.2.1 Open file 수 변경

리눅스나 유닉스에서는 Socket 하나하나가 모두 파일이어서 대용량으로 접속하면 파일 숫자가 모자라서 종종 오류가 발생한다. 기본값은 1024인데, 성능 테스트를 하기에는 터무니 없이 작은 값이다. 이 값을 변경하는 방법은 두 가지다.⁰²

- /etc/security/limits.conf 값을 추가한다.

```
* soft  nofile      60000
* hard  nofile      60000
```

- ulimit 명령어를 이용한다. 영구적 설정은 아니며 해당 터미널 세션에만 적용되는 수치다. 영구적으로 모든 세션에 적용하려면 앞의 방법을 사용해야 한다.

```
$ ulimit -n 60000
```

변경된 정보를 확인하려면 다음 명령어를 실행한다. open files가 64000으로 설정된 것을 볼 수 있다.

```
[root@test] ~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals          (-i) 30381
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files              (-n) 64000
```

02 60000이라는 값은 테스트 환경에 맞게 적당한 값으로 입력한다. 60000정도면 대부분 테스트할 수 있는 수치다.

```
pipe size          (512 bytes, -p) 8
POSIX message queues   (bytes, -q) 819200
real-time priority      (-r) 0
stack size           (kbytes, -s) 10240
cpu time             (seconds, -t) unlimited
max user processes    (-u) 30381
virtual memory        (kbytes, -v) unlimited
file locks            (-x) unlimited
```

6.2.2 Socket 관련 커널 파라미터 설정

설정값이 PC를 재시작해도 적용되게 하려면 \etc\sysctl.conf에서 다음 내용을 수정한다.

- **Port Range** : 웹 서버에 접속하려면 서버의 80포트로 접속한다. 이때 로컬 PC에도 임의의 포트를 할당받는데 이 포트로 사용할 수 있는 범위를 말한다. 이 값이 작으면 더는 접속하지 못하는 상황이 발생한다.

```
net.ipv4.ip_local_port_range = 1024 65535
```

- **Time Wait 소켓 처리** : 정상적인 소켓 접속 종료를 수행하면 해당 소켓은 Time Wait 상태를 일정 시간 동안 유지한다. 이는 접속이 종료되었지만, 잉여 패킷 수신을 위해 커널에서 일정 시간 상태를 유지하는 것이다. 리눅스의 기본값은 60초다. 하지만 소켓 접속을 연결하고 끊는 작업을 짧은 시간에 대량으로 수행하면 결국 Time Wait 소켓으로 가득 차서 더는 접속하지 못하는 상황이 발생한다. 이를 해결하기 위한 옵션은 두 가지다. 필요에 따라 Time Wait 소켓도 접속에 사용할 수 있게 하는 설정(① 1=true, 0=false)과 Time Wait를 유지하는 시간을 줄이는 설정(② 단위: 초)이다.

```
net.ipv4.tcp_tw_recycle = 1          ①  
net.ipv4.tcp_fin_timeout = 1         ②
```

\etc\sysctl.conf 파일에 수정된 값을 바로 적용하려면 sysctl -p 명령을 사용하고, PC를 재시작해도 적용된다. 적용 여부는 다음 명령어로 확인할 수 있다.

```
cat /proc/sys/net/ipv4/ip_local_port_range  
cat /proc/sys/net/ipv4/tcp_tw_recycle  
cat /proc/sys/net/ipv4/tcp_tw_timeout
```

6.3 분산 테스트 환경 구축

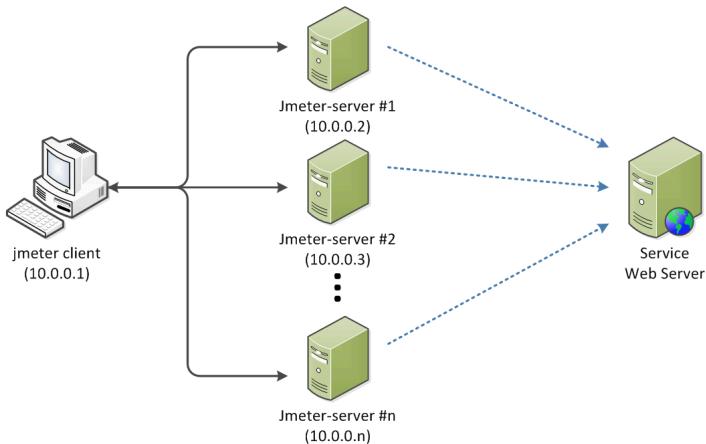
대용량 성능 테스트는 한 대의 부하발생기로는 불가능하고, 여러 대의 부하발생기를 병렬로 연결해서 동시에 테스트를 해야만 한다. JMeter는 이런 환경을 구축하기 위해 복수의 jmeter-server를 원격에서 실행할 수 있는 Remote Testing 기능을 제공한다.

6.3.1 JMeter 분산 환경 구축

여기서는 [그림 6-1]과 같은 구조의 분산 테스트 환경을 구축한다고 가정한다.

- **jmeter controller** : 실제로 부하를 발생시키는 jmeter-server에 명령을 전달하는 Master 역할을 수행하는 노드다.
- **jmeter-server** : 실제로 부하를 발생하는 노드로, jmeter controller에서 받은 명령을 수행하고 그 결과값을 다시 jmeter controller로 전달하는 Slave 역할을 한다.

[그림 6-1] 분산환경 구축



노드 준비

가장 먼저 수행할 작업은 jmeter controller와 jmeter-server에 같은 버전의 JMeter 프로그램을 설치하는 것이다. 서로 다른 버전이 설치되어도 대부분 구동되지만 가능하면 버전을 맞추는 것이 좋다.

Test Plan의 내용은 jmeter controller에서 jmeter-server로 전달되지만, CSV 파일과 같은 데이터는 전달되지 않으므로 Test Plan의 CSV 파일을 사용하려면 모든 jmeter-server에 미리 복사해 놓아야 한다.

불편해 보일 수도 있지만, 실제 테스트에서는 매우 유용하게 사용될 수 있다. 각 jmeter-server에 동일한 CSV 파일을 복사하는 것이 아니라 서로 다른 내용의 CSV 파일을 복사할 수 있으므로 좀 더 다양한 테스트를 할 수 있다. 또한, 중복 로그인이 불가능한 서비스 환경에서 CSV 파일에 아이디/패스워드가 저장되어 있을 때 동일한 파일을 복사해 놓으면 중복 로그인 오류가 발생하므로 꼭 서로 다른 내용의 CSV 파일을 각 부하 발생기에 올려놓아야 문제가 발생하지 않는다.

jmeter-server 실행

환경에 맞는 명령어를 실행한다.

[Linux/Unix]

JMETER_HOME/bin/jmeter-server

[Windows]

JMETER_HOME/bin/jmeter-server.bat

실제 실행하면 다음 정보도 함께 실행된다.

```
# ./jmeter-server  
Created remote object: UnicastServerRef [liveRef: [endpoint:[10.0.0.2:22553](local),objID:[6c1d1081:1490f622fbe:-7fff, 4118357431945944039]]]
```

jmeter controller 설정

JMETER_HOME\bin\jmeter.properties 파일을 편집한다. jmeter-server 접속을 위한 IP 설정과 결과 데이터를 받아오는 방식을 설정한다.

- **remote_hosts** 설정 : jmeter controller에서 접속할 remote jmeter-server의 정보를 입력하는 부분이다. 다음과 같이 콤마(,)로 구분된 IP 리스트를 적어준다. 포트 정보를 별도로 입력해주지 않으면 1099로 설정된다.

```
remote_hosts=10.0.0.2,10.0.0.3,...,10.0.0.n
```

별도의 포트를 지정하고 싶다면 다음과 같이 포트 정보를 입력한다.

```
remote_hosts=10.0.0.2:2099,10.0.0.3:3099, ..., 10.0.0.n:5099
```

- 정보 수집 모드 설정 : jmeter-server의 결과값은 jmeter controller로 보내는데, 보내는 방법은 여러 가지가 있다.

Standard 모드는 Sampler 하나하나의 결과를 바로 jmeter controller로 전달하는 방식이다. jmeter controller의 Listener에 정보가 바로 업데이트되며 Sampler의 모든 결과값이 온전하게 controller로 전달되는 장점이 있지만, 여러 대의 jmeter-server로부터 정보를 받다 보니 CPU나 네트워크 대역폭이 모자라서 테스트가 제대로 되지 않는 경우가 발생할 수 있다.

Standard의 단점을 다소 보완한 모드가 Batch다. Standard 모드와 달리 Sampler의 결과가 바로 전달되는 것이 아니라 100개씩 모아서 전송된다. 하지만 Batch 모드 역시 모든 Sampler의 정보를 전부 받아 오는 방법이므로 controller에 가해지는 부하가 확연히 줄어들지는 않는다.

controller에 가해지는 부하를 줄여주는 가장 좋은 방법은 Statistical 모드다. 이름에서 알 수 있듯이 여러 개의 결과값을 모아서 평균값을 controller로 전달하는 방식이다. CPU 사용률과 네트워크 대역폭이 매우 감소하지만, 평균값만 전달되므로 View Results Tree처럼 하나하나의 정보를 확인하는 Listener는 사용할 수 없는 단점이 있다.

이 외에도 Hold나 DiskStore같이 jmeter-server에 정보를 저장했다가 테스트가 끝났을 때 한꺼번에 정보를 전달하는 방식이 있는데, 이 방식은 그다지 추천하지 않는다. 테스트 종료 후 정보를 받아오는 시간이 만만치 않게 걸린다.

Asynch 모드는 데이터를 수신한 Thread가 직접 jmeter controller로 정보를 전달하는 것이 아니라 asynch batch queue에 정보를 쌓아 놓으면 비동기로 jmeter controller에 정보를 전달하는 방식이다. 하지만 asynch batch

queue가 꽉 차면 일정 시간 동안 대기하는 것은 Standard 모드와 동일하다.
asynch.batch.queue.size 값으로 크기를 조절할 수 있다.

이름에 ‘Stripped’, ‘StrippedBatch’, ‘StrippedDiskStore’처럼 ‘Stripped’라는 수식어가 붙은 모드가 있는데, 이는 Sampler의 결과 정보를 전달할 때 본문을 제외하고 보내는 방식이다. controller의 대역폭이 문제라면 이 모드도 사용해볼 만하다. JMeter 2.9 버전 이전에는 Standard 모드가 기본값이었으나 2.9 버전부터는 StrippedBatch가 기본값이다.

[정보 수집 모드 설정](jmeter.properties)

```
# Remote batching support
# Since JMeter 2.9, default is MODE_STRIPPED_BATCH, which returns samples
in
# batch mode (every 100 samples or every minute by default)
# Note also that MODE_STRIPPED_BATCH strips response data from
SampleResult, so if you need it change to
# another mode
# Hold retains samples until end of test (may need lots of memory)
# Batch returns samples in batches
# Statistical returns sample summary statistics
# hold_samples was originally defined as a separate property,
# but can now also be defined using mode=Hold
# mode can also be the class name of an implementation of
org.apache.jmeter.samplers.SampleSender
mode=Standard
#mode=Batch
#mode=Hold
#mode=Statistical
#Set to true to key statistical samples on threadName rather than
threadGroup
#key_on_threadname=false
```

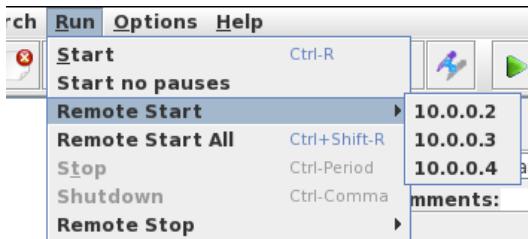
```
#mode=Stripped
#mode=StrippedBatch
#mode=org.example.load.MySampleSender
#
#num_sample_threshold=100
# Value is in milliseconds
#time_threshold=60000
#
# Asynchronous sender; uses a queue and background worker process to
return the samples
#mode=Asynch
# default queue size
#asynch.batch.queue.size=100
# Same as Asynch but strips response data from SampleResult
#mode=StrippedAsynch
#
# DiskStore: as for Hold mode, but serialises the samples to disk, rather
than saving in memory
#mode=DiskStore
# Same as DiskStore but strips response data from SampleResult
#mode=StrippedDiskStore
```

테스트 실행

노드 설정이 모두 완료되면 테스트를 실행한다. 실행 방법은 한 대의 JMeter를 이용할 때와 약간 다르다. 실행 및 정지 방법 외에 모니터링 방법은 기존 방법과 동일하다.

등록된 remote_hosts의 리스트는 메뉴에서 ‘Run → Remote Start’를 선택하면 [그림 6-2]와 같이 확인할 수 있다.

[그림 6-2] 분산 테스트 실행



- 실행 : 각 jmeter-server를 별도로 실행하려면 메뉴에서 ‘Run → Remote Start → [IP]’를 선택하고, 모든 jmeter-server를 동시에 실행하려면 ‘Run → Remote Start All’을 선택한다. 메뉴바에서 아이콘을 선택해도 모든 jmeter-server가 동시에 실행된다.
- 정지 : 각 jmeter-server를 별도로 정지하려면 메뉴에서 ‘Run → Remote Stop → [IP]’를 선택하면 되고, 모든 jmeter-server를 동시에 정지하려면 ‘Run → Remote Stop All’을 선택한다. 메뉴에서 아이콘을 선택해도 동시에 정지된다.

통신 포트 설정 변경

jmeter controller와 jmeter-server 사이에 방화벽이 존재한다면 특정 포트를 별도로 설정해야 한다. 그래야만 방화벽에 해당 포트를 등록하고 통신할 수 있다. JMeter는 jmeter controller와 jmeter-server 사이에 양방향 통신이 가능해야 한다.

Remote Start 버튼을 누르면 처음에는 jmeter controller가 jmeter-server의 서버 포트를 이용해서 접속하지만, 실제로 테스트가 실행되면 서로 RMI^{Remote Method Invocation}⁰³ 포트를 이용해서 정보를 주고받는다. 서버 포트는 RMI Registry 포트라고 보면 된다. 이는 등록된 원격 객체^{remote object}의 정보를 주고받는 역할을

03. 로컬에서 원격 컴퓨터의 메서드를 호출하는 기술이다.

한다. 그러므로 방화벽을 설정할 때에는 서버 포트뿐 아니라 각각의 RMI 포트도 설정해야 한다.

[그림 6-3] 포트 설정



- **jmeter-server 설정** : 서버 포트와 RMI 포트 설정은 다음과 같이 변경할 수 있다.

```
server_port=2099  
server.rmi.localport=4000
```

- **jmeter controller 설정** : 서버 포트를 2099로 설정했다면 remote_hosts 설정에서 IP 주소 뒤에 포트 정보를 함께 입력해야 한다.

```
remote_hosts=10.0.0.2:2099  
client.rmi.localport=4001
```

모든 변경이 완료되면 [그림 6-4]와 같은 구성이 된다.

[그림 6-4] 포트 변경



- SSH Tunnel 설정 : jmeter-server에서 jmeter controller 방향의 통신이 안 되는 경우가 종종 발생한다. 이런 경우에는 SSH Tunneling을 이용해서 해결 할 수 있다.

[그림 6-5] SSH Tunnel 구성



jmeter controller에서 다음과 같이 jmeter controller에서 생성된 원격 객체의 RMI 주소를 localhost로 설정한다.

```
$ export JVM_ARGS="-Djava.rmi.server.hostname=localhost"
```

그리고 다음 명령으로 SSH Tunnel을 생성한다. 10.0.0.2 이외의 서버가 더 있다면 서버가 있는 만큼 생성해 준다.

```
$ ssh -R 4001:localhost:4001 10.0.0.2
```

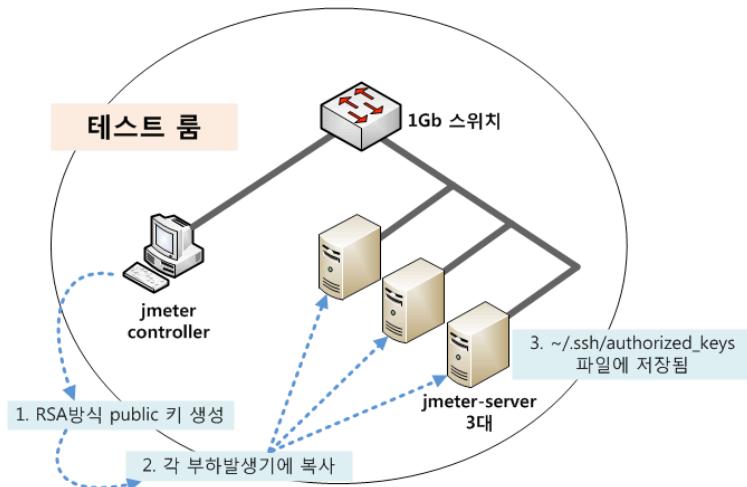
이렇게 설정한 후에 jmeter-server와 jmeter controller를 실행하면 [그림 6-4]와 같이 jmeter-server가 jmeter controller의 RMI 포트에 직접 접속하는 것이 아니라 SSH Tunneling 포트를 통해서 통신하게 된다.

6.3.2 SSH 자동접속 환경 구축

부하발생기로 리눅스를 사용하면 각 부하발생기에 SSH를 사용하여 접속한다. 많은 수의 부하발생기에 접속할 때마다 암호를 입력하려면 업무 처리 능률이 떨어지

게 된다. 이때 유용한 것이 public 키를 미리 생성/배포하여 암호 입력 없이 접속하는 방법이다.

[그림 6-6] SSH 자동 접속



ServerA(jmeter controller)와 ServerB(jmeter-server)가 있다고 가정하고, ServerA에서 ServerB로 암호 없이 SSH 접속을 하려면 다음과 같이 설정한다. 이 예제는 root 계정을 이용하므로 모든 작업은 ServerA에서 이루어진다.

- RSA방식 키 생성(ssh-keygen) : Passphrase 없이 그냥 진행한다.

```
root@ServerA:~# ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
```

```
The key fingerprint is:  
49:7d:30:7d:67:db:58:51:42:75:78:9c:06:e1:0c:8d root@aneesh-pc  
The key's randomart image is:  
+--[ RSA 2048]----+  
|       ooo+=B |  
|       . E=.o+B |  
|       . . .+.*o |  
|       . . . . . |  
|       S         |  
|               |  
|               |  
|               |  
|               |  
+-----+
```

- Public 키 복사(ssh-copy-id)

```
root@ServerA:~# ssh-copy-id -i ~/.ssh/id_rsa.pub root@ServerB
```

- **authorized_keys 확인** : 두 번째 작업을 수행하면 ServerB에 ~\.ssh\authorized_keys라는 파일이 생성된 것을 확인할 수 있다.

이와 같이 설정하면 ServerA에서 ServerB로 SSH 접속할 때 암호 입력 없이 바로 접근할 수 있다.

6.3.3 jmeter-server 일괄 On/Off 스크립트

SSH 자동 접속 환경이 구축되었다면 이제 원격에서 부하발생기를 On/Off하는 스크립트를 작성해 보자. 부하발생기를 계속 켜놓고 사용하면 좋겠지만, 테스트하다 보면 비정상적으로 종료되거나 멈춰버리는 경우가 종종 발생해서 가끔 On/Off해

야 하는 상황이 발생한다. 좀 더 편한 환경을 구축하기 위해 일괄 On/Off 스크립트를 만들어본다. 이 스크립트는 나중에 여러 대의 서버에서 동일한 작업을 수행할 때 사용할 수 있는 일괄 작업 스크립트로도 활용할 수 있다.

```
#!/bin/bash

## Configure
## -----
-- 
SERVER_LIST="10.0.0.2 10.0.0.3 10.0.0.4"
JMETER_HOME=/data/jmeter

## Main
## -----
-- 
for server in ${SERVER_LIST}
do
    # Check Process IDs
    pids=$(ssh root@$server ps -ef | grep jmeter-server | grep -v grep | awk '{print $2}')
    # Restart jmeter-server
    ssh root@$server "kill `echo ${pids}` 2>/dev/null ; sleep 1 ; cd ${JMETER_HOME}/bin ; nohup ./jmeter-server >/dev/null &"
done
```

스크립트 파일이 restart_jmeter_server.sh라면 다음과 같이 실행한다.

```
$ sh restart_jmeter_server.sh
```

6.3.4 시간 동기화

여러 대의 부하발생기를 설치할 경우 모든 부하발생기의 시간을 반드시 동기화해야 한다. 그렇지 않으면 원치 않는 결과를 받게 된다. 매우 중요한 부분이므로 꼭 확인한다. 이는 jmeter-server에서 보내주는 시간을 기준으로 jmeter controller가 정보를 수집하고 평균값을 보여주므로 현재 시간보다 이전 시간의 정보가 도착하면 jmeter controller는 이를 예전 정보로 간주한다. 이렇게 되면 실제 처리량보다 낮은 처리량이 화면에 보여지게 된다.

동기화 방법

- ntpdate 명령어 이용(`ntpdate <ntp server>`)

```
$ ntpdate 192.168.100.234
```

- rdate 명령어 이용(`rdate -s <time server>`)

```
$ rdate -s time.bora.net
```

타임 서버가 존재하지 않을 때

폐쇄 구조의 네트워크일 때 내부에 타임 서버가 별도로 존재하지 않으면 기존 방식을 이용한 동기화가 불가능하다. 이 경우에는 부하발생기 중 한 대에 ntp 서버를 구축하거나 여러 대의 부하발생기 중에서 한 대에서 현재 시간을 배포하는 방법으로 동기화할 수 있다. 다음과 같이 간단하게 시간을 공유하는 스크립트를 만들어서 공유할 수 있다. SSH 자동 접속을 설정해 놓으면 좀 더 쉽게 처리할 수 있다.

```
#!/bin/bash
## Configure
```

```
## -----
SERVER_LIST='10.0.0.2 10.0.0.3 10.0.0.4'

## Main
## -----
for server in ${SERVER_LIST}
do
    cdate=`date +%m%d%H%M%Y.%S`
    ssh root@$server "date ${cdate}"
done
```

6.4 네트워크 구성

테스트 목적이 무엇이냐에 따라 네트워크 구성은 매우 중요한 포인트가 된다. 웹 서버, WAS, DB 서버 등 서비스 서버의 성능을 보려면 서버와 최대한 가까운 네트워크 위치에 부하발생기가 설치되어야 한다. 네트워크의 대역폭과 성능까지 고려하는 테스트라면 상황에 따라 부하발생기 위치를 달리해야 한다.

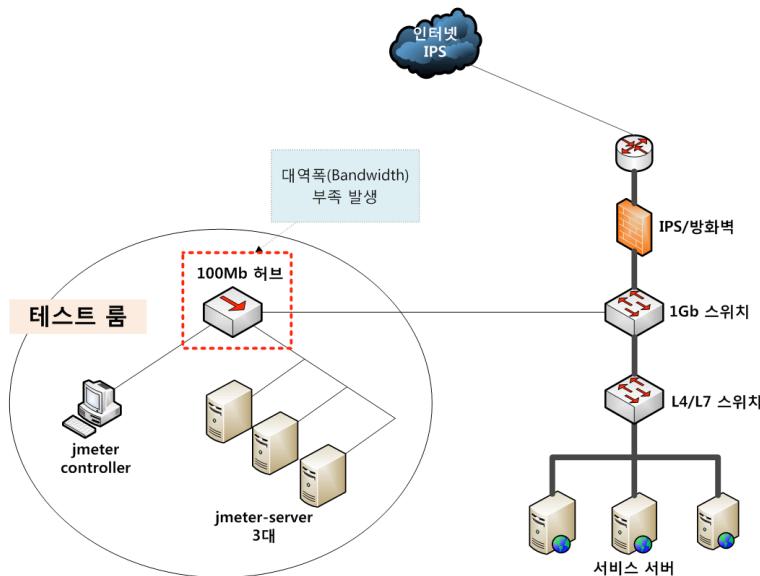
6.4.1 서비스 서버의 성능을 테스트할 때 네트워크 구성 주의사항

서버의 성능을 테스트하기 위해서는 네트워크 제약을 최대한 제거해야 한다. 즉, 테스트 대상 서버와 부하발생기가 네트워크에서 최대한 가까운 위치에 있는 것이 좋다.

네트워크 구간 중간에 대역폭에 영향을 주는 스위치가 존재하지는 않는가?

가끔 IDC 내의 테스트 룸(모니터링 룸) 등에서 테스트를 진행하는 경우가 있다. 이때 테스트 룸 내부의 스위치가 100MB 대역폭의 스위치가 아닌지 꼭 확인해야 한다. 테스트 과정에서 대역폭의 영향을 받으면 처리량이 더는 증가하지 않고 응답시간만 증가하여 서비스 서버의 CPU 등 리소스는 여유로운 상태를 유지하는 비정상적인 현상이 발생할 수 있다.

[그림 6-7] 네트워크 구성도 - 대역폭 문제일 때



방화벽/IPS/IDS 등의 네트워크 장비를 경유하는가?

최대한 IPS/IDS/방화벽 등을 거치지 않도록 해야 한다. 부하발생기와 서비스 서버 사이에 이러한 보안 장비가 존재하면 부하발생기로부터 요청량이 증가함에 따라 요청을 차단하거나 처리량이 주기적으로 떨어지는 현상이 발생할 수 있다.

[그림 6-8]은 IPS 기능이 있는 방화벽 장비가 존재하는 상태에서 진행한 테스트와 해당 장비를 제거하고 실시한 테스트의 결과 그래프다. D대학교의 실제 테스트 결과로, 그래프에서 보는 바와 같이 보안 장비가 주기적으로 트래픽을 차단한 것을 확인할 수 있다.

[그림 6-8] 보안 장비 유무에 따른 테스트 결과

▪ Case.1 Firewall On



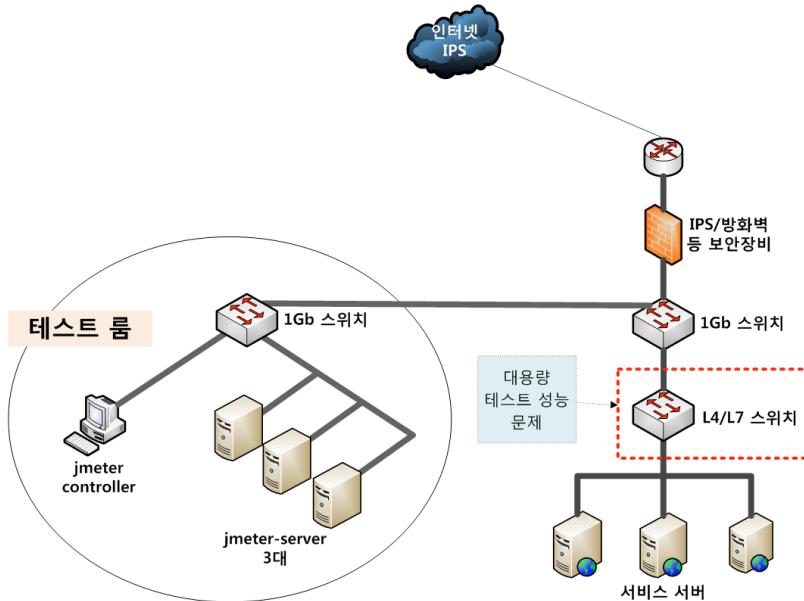
▪ Case.2 Firewall Off



스위치 성능(초당 처리량)은 충분한가?

스위치의 성능은 항상 서비스 서버의 처리량을 감당하기에 충분하다고 생각하는 경우가 많다. L2나 L3 스위치를 사용하는 구간에서는 그리 틀린 말은 아니지만, L4나 L7 스위치를 사용하는 구간에서는 상당히 조심해야 한다. L4나 L7 스위치를 통해서 테스트를 진행할 때에는 꼭 해당 스위치의 CPU 상태를 파악해야 한다. 그렇지 않으면 알 수 없는 오류가 주기적으로 발생할 수 있다. 테스트할 때 항상 중요 네트워크 장비의 상태를 확인할 수 있는 엔지니어와 함께해야만 원활하게 테스트 할 수 있다.

[그림 6-9] 네트워크 구성도 - 스위치 성능 문제일 때

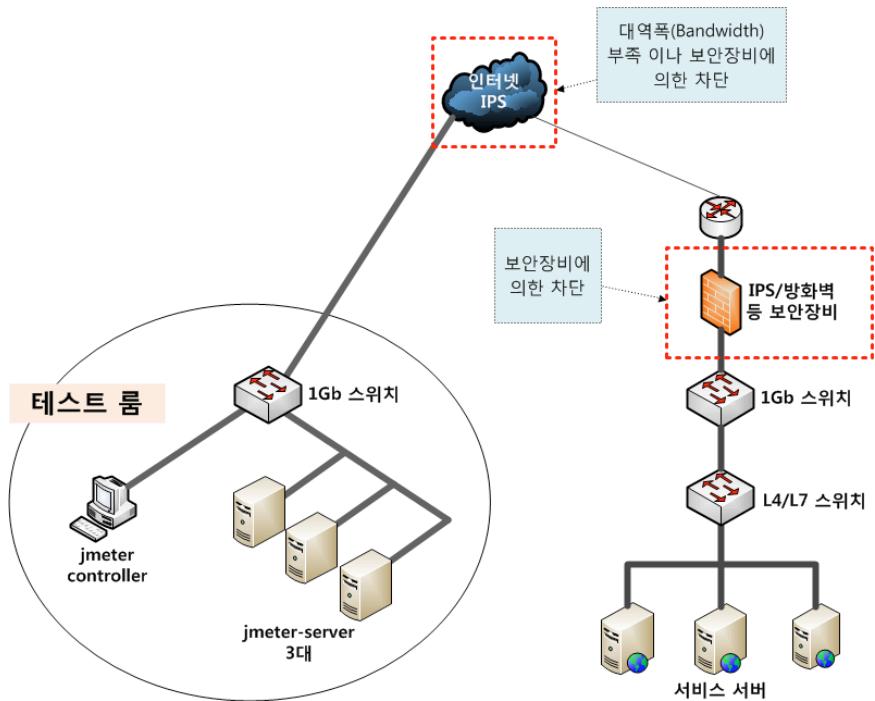


6.4.2 네트워크 구간을 고려한 성능 테스트를 할 때 네트워크 구성 주의사항

가끔 일반 사용자와 동일한 환경에서 테스트해야만 정상적인 테스트라고 생각하는 경우가 있다. 이때에는 외부에 부하발생기를 설치하고 인터넷(ISP)를 거쳐서 테스트를 진행하는데, 이 경우에는 테스트 결과에 영향을 미치는 요소가 너무 많아서 실제로 어떤 부분이 문제인지 판단하기가 어렵다. 특히 ISP 내부에서 패킷이 유실되거나 ISP 내부 보안장비에 의해서 차단되는 경우도 있어서 매우 복잡한 테스트가 되고 만다.

이러한 네트워크 구성은 성능 테스트용이라기보다는 서비스 구간 중간에 오류가 없는지를 확인하기 위한 테스트로 사용하는 것이 적절하다.

[그림 6-10] 네트워크 구성도 - 네트워크 구간 문제일 때



7 | 테스트 설계 및 Test Plan 작성

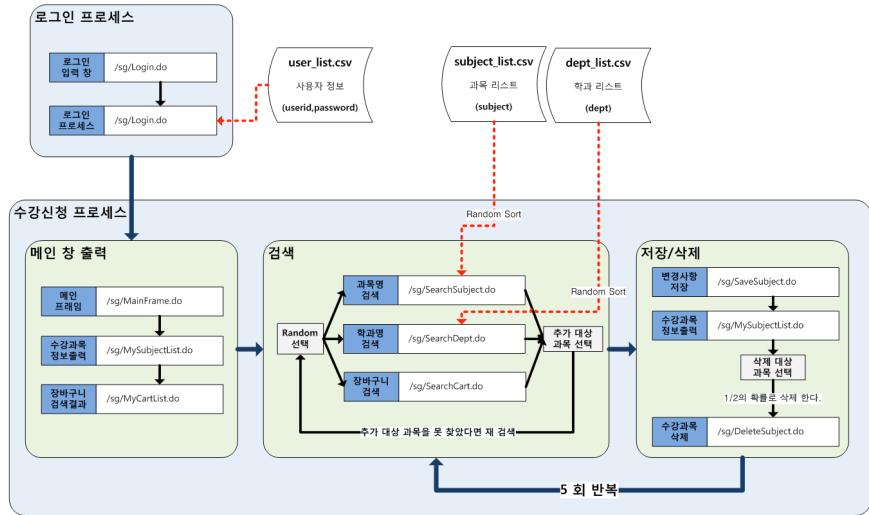
테스트 계획과 테스트 환경이 구축되면 이제 JMeter에서 사용할 Test Plan 스크립트를 작성해야 한다. 이번 예제는 대학교의 수강신청 시스템을 테스트한다고 가정한다.

7.1 로직 구성도

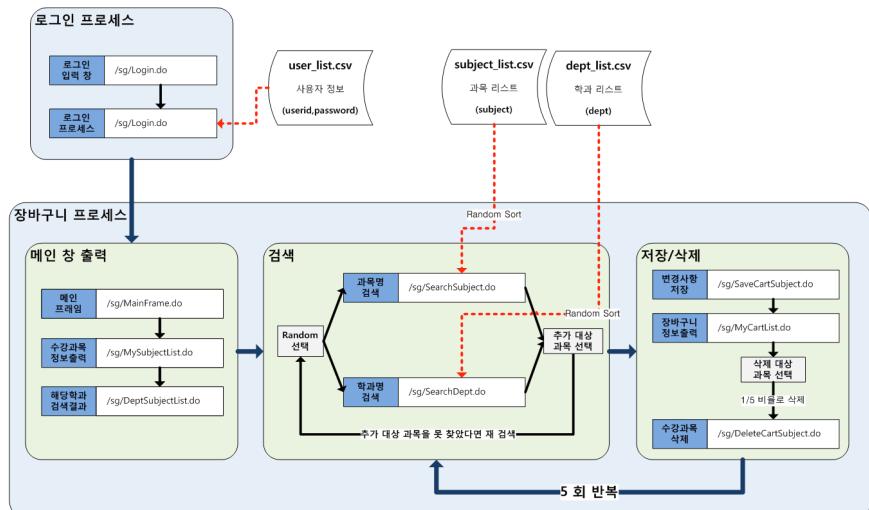
간단한 테스트일 때는 바로 Test Plan 스크립트를 작성해도 되지만, 좀 더 체계적인 작업을 위해서는 테스트하려는 서비스의 로직 구성도(시나리오)를 작성하는 것이 좋다. 이는 서비스 구성에 대한 이해를 높이고, 빠진 영역을 추가하거나 불필요한 영역을 걸러내는 데 유용하며, 다른 부서와의 커뮤니케이션에도 효과적이다.

[그림 7-1]과 [그림 7-2]는 대학교 수강신청 시스템의 로직 구성도 예시다. 사용자가 수강신청을 할 때 서비스에 접근하는 동선을 그대로 구성했고, 이 과정에서 필요한 데이터 파일의 이름과 접근 형태, 반복 수행되는 작업, 결과값 추출 등 스크립트 작성에 필요한 내용도 포함하였다. 이 그림에는 두 가지 로직 구성도가 나타나는데, ‘장바구니’ 테스트와 ‘수강신청’ 테스트라는 두 가지 시나리오다.

[그림 7-1] 수강신청 로직 구성도



[그림 7-2] 장바구니 로직 구성도



7.2 테스트 방법

로직 구성도를 완성하면 이 로직을 어떤 방법으로 테스트할지를 결정해야 한다. 테스트 방법에는 여러 가지 방식이 있으며 테스트 목표에 가장 적합한 것을 선택하면 된다. 보통은 한 가지 방법만 쓰지 않고, 두세 가지 방법을 함께 사용한다.

가상 사용자 증가 방식

가장 일반적인 테스트 방법으로, 가상 사용자 수를 순차적으로 증가시키면서 성능의 변곡점을 찾는 방식이다. 성능의 변곡점은 응답시간, CPU 사용률 등의 성능 지표가 가상 사용자 숫자 증가에 따라 일정하게 유지하거나 선형적으로 증가하다가 갑자기 급격하게 변화하는 지점을 의미한다.

요청량 증가 방식

요청이 서비스 서버로 유입될 때 시스템과 서비스 상태가 어떠한지를 요청량을 순차적으로 증가시키면서 확인하는 테스트 방식이다. 응답시간과 처리량의 변곡점을 찾고 이를 기반으로 사용자 수를 제어하거나 시스템을 증설할 수 있다.

구간 테스트

전체 서비스 서버의 구성 구간 중 어떤 부분에서 병목이 발생하는지를 판별하기 위한 테스트 방법이다. ‘스위치’, ‘스위치 → 웹 서버’, ‘스위치 → 웹 서버 → WAS’, ‘스위치 → 웹 서버 → WAS → DB’ 순서로 서비스 구간을 변경하면서 어느 부분이 전체 서비스 품질에 가장 많은 영향을 미치는지 확인한다.

장시간 테스트

서비스 초기에는 정상으로 동작하지만, 시간이 지날수록 성능이 저하되는 경우가 종종 발생한다. 예를 들어, DB에 정보가 쌓일수록 느려지거나 WAS의 메모리가 증가함에 따라 성능이 차이가 현저하게 발생할 때 단시간의 테스트로는 현상을 파악

하기 힘든 경우가 있다. 이럴 때 장시간 테스트로 그 영향을 파악할 수 있다. 장시간의 기준은 서비스 환경이나 목적에 따라 달라질 수 있으며 짧게는 한두 시간, 길게는 며칠에서 몇 달이 될 수도 있다.

7.3 테스트 케이스 작성

보통 대학교의 수강신청은 많게는 한 학년에 5~6천 명, 전체는 2~3만 명의 학생이 동시에 진행한다. 이때 현재 시스템으로 동시에 모든 학생이 수강신청이 가능한지 아니면 학년별로 기간을 나눠서 진행해야 하는지를 판단해야만 원활한 수강신청을 할 수 있다. 이 경우에는 가상 사용자를 증가시키면서 테스트를 진행하면 현 시스템에 맞는 값을 찾을 수 있다. 테스트로 시스템에서 처리할 수 있는 적정 사용자 기준을 찾으면 수강신청을 대비해서 시스템을 증설하거나 수강신청 방법을 변경하는 등의 의사결정을 할 수 있으며 성공적인 서비스가 될 수 있다.

이번 예제에서는 대학교 수강신청을 '가상 사용자 증가 방식'으로 테스트하고, 한 학년의 인원은 각 1,000명, 전체는 4,000명이라고 가정한다. 이를 바탕으로 테스트 케이스를 뽑으면 [표 7-1]과 같은 케이스가 도출된다(실제 테스트 목적과 내용에 따라 테스트 케이스는 달라질 수 있다).

[표 7-1] 테스트 케이스

시나리오	테스트 케이스	가상 사용자(명)	Think Time(초)	테스트 시간(초)	체크 항목 및 비고
수강신청 (S1)	TC-S1-10	1,000	1	300	WEB/WAS : CPU, Session, Error DB : CPU, Session, Load, Error JMeter : 처리량, 응답시간
	TC-S1-20	2,000	1	300	
	TC-S1-30	3,000	1	300	
	TC-S1-40	4,000	1	300	
장바구니 (S2)	TC-S2-10	1,000	1	300	JMeter : 처리량, 응답시간
	TC-S2-20	2,000	1	300	
	TC-S2-30	3,000	1	300	
	TC-S2-40	4,000	1	300	

7.4 테스트 데이터 준비

[그림 7-1]과 [그림 7-2]를 보면 user_list.csv, subject_list.csv, dept_list.csv라는 3가지 데이터 파일이 필요하다. user_list.csv 파일은 내용을 userid, password 형태로 저장하고 나머지는 검색에 필요한 각 항목을 라인 단위로 나열 한다.

7.4.1 파일 나누기

시스템에 따라 중복 로그인이 불가능한 경우가 종종 있다. 이럴 때 여러 대의 부하 발생기를 이용하거나 가상 사용자 수보다 user_list.csv의 테스트 아이디 수가 적으면 중복으로 로그인되어 오류가 발생할 수 있다. 가상 사용자 수보다 user_list.csv의 테스트 아이디 수가 적으면 불가피하게 user_list.csv의 테스트 아이디 수를 가상 사용자 수에 맞춰야 한다.

여러 대의 부하발생기로 테스트할 때에는 부하발생기 수에 맞춰 파일을 나눠야 한다. 텍스트 편집기를 이용해서 라인 단위로 잘라서 파일을 만들 수도 있지만, 리눅스를 사용한다면 split 명령어로 간단하게 파일을 나눌 수 있다. 다음은 split 명령어로 파일을 나누는 셀 스크립트다.

[split_file.sh]

```
#!/bin/bash

infile=$1
split_count=$2
tmpfile=`tempfile`


function usage()
{
    echo ""
    echo "Usage: $0 infile split_count"
    echo "Example: $0 user_list.csv 10000"
}
```

```
echo "Usage : $0 infile split_count"
echo ""
}

if [ ! -f "${infile}" ] ; then
    echo " ***** Error : Invalid input file "
    usage
    exit -1
fi

if [ "x${split_count}" = "x" ] ;then
    echo " ***** Error : Invalid split count "
    usage
    exit -1
fi

ext=${infile##*.}
if [ "x${ext}" = "x${infile}" ] ; then
    ext=""
fi
prefix=`basename ${infile} .${ext}`

split -n ${split_count} --additional-suffix=".${ext}" --numeric-suffixes=1
${infile} "${prefix}_"
```

user_list.csv 파일을 세 개의 파일로 나누고 싶다면 다음 명령을 실행하면 된다.

```
# sh split_file.sh user_list.csv 3
```

결과는 user_list_01.csv, user_list_02.csv, user_list_03.csv로 나뉘어 저장된다.

7.4.2 랜덤 정렬

JMeter에 입력된 CSV 파일은 첫 라인부터 차례대로 참조되는데, 테스트 케이스가 반복되어도 항상 순서는 똑같아진다. 데이터를 좀 더 실제 환경과 비슷하게 구현 하려면 데이터에 무작위로 접근하게 하는 것이 좋다. sort 명령어를 이용하면 파일 내용의 라인 순서를 바꿀 수 있다.

여러 대의 부하발생기마다 서로 다른 순서로 정렬된 파일을 사용하거나 테스트 케이스가 변경될 때마다 서로 다른 순서로 정렬된 파일을 사용하고 싶다면 sort 명령어로 쉽게 작업할 수 있다. 다음은 sort 명령어를 이용한 간단한 랜덤 정렬 셸 스크립트다.

[random_sort.sh]

```
#!/bin/bash

infile=$1
outfile=$2
tmpfile=`tempfile`'

function usage()
{
    echo ""
    echo "Usage : $0 infile [outfile]"
    echo ""
}

if [ ! -f "${infile}" ] ; then
    echo "***** Error : Invalid input file "
    usage
    exit -1
fi

if [ "x${outfile}" = "x" ] ; then
```

```
outfile=${infile}  
fi  
  
sort -R ${infile} > ${tmpfile}  
cat ${tmpfile} > ${outfile}
```

subject_list.csv 파일을 섞고 싶다면 다음 명령을 실행하면 된다.

```
# sh random_sort.sh subject_list.csv
```

7.5 Test Plan 작성

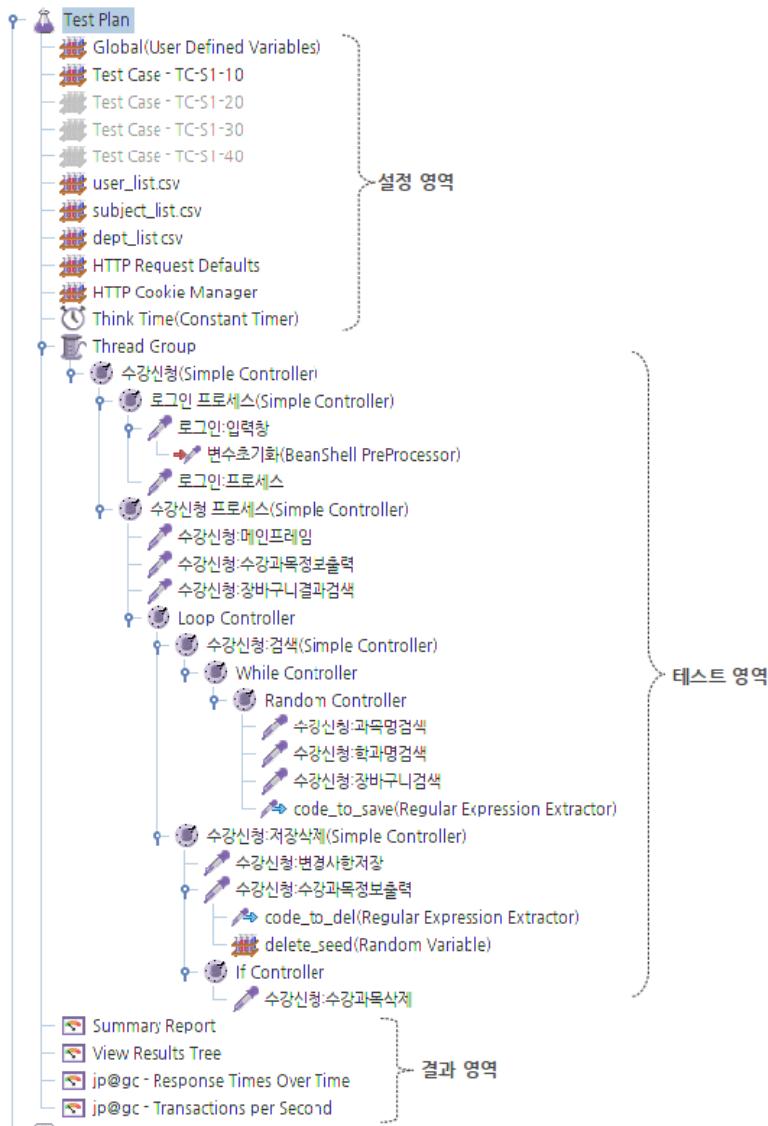
이제 데이터도 준비되었고 테스트 시나리오와 테스트 케이스도 준비되었으니 실제 Test Plan 스크립트를 작성해 보자. 여기서는 ‘수강신청’ 시나리오를 기준으로 작성한다. “장바구니” 시나리오는 ‘수강신청’ 시나리오가 완성되면 이를 참고해서 어렵지 않게 구현할 수 있을 것이다.

[표 7-2]와 [그림 7-3]을 보면 시나리오는 크게 세 가지 영역으로 구분된다. 꼭 이렇게 구성해야 하는 것은 아니고, 보기 쉽고 작업하기 쉽게 하려는 필자의 구분 방식이다. 좀 더 편안한 자신만의 구분 방식이 있다면 어떻게 구분해도 상관없다.

[표 7-2] 영역 구분

영역	비고
설정 영역	<ul style="list-style-type: none">테스트 영역에서 사용될 변수값을 설정한다. 테스트 전체에 사용될 변수와 테스트 케이스별로 사용될 변수를 미리 정의해서 테스트 케이스에 따른 설정값 변경을 이 부분에서만 처리되게 한다.데이터 파일을 불러온다. 필요한 데이터 파일의 참조 변수를 미리 설정해서 테스트 영역에서 변수값을 가져다 쓸 수 있게 한다.기타 기본 설정 : 세션 유지를 위한 Cookie Manager 등의 도구를 놓는다.
테스트 영역	실제 서버로 요청을 전송하는 Sampler과 이러한 Sampler를 연결해주는 Logic Controller를 배치해서 실제 시나리오를 구현하는 영역이다.
결과 영역	테스트 결과 정보를 볼 수 있는 Listener를 위치시키는 영역이다.

[그림 7-3] Test Plan



7.5.1 설정 영역

Global

테스트 과정에서 사용될 Global 변수를 설정하는 부분이다. [그림 7-4]를 보면 테스트 대상 서버의 정보, 데이터나 스크립트의 저장 경로 등의 정보를 담고 있다. 여기에 정의된 변수들은 \${target_host}(target_host의 경우)와 같은 방식으로 Test Plan 스크립트 내의 어떠한 Element에서도 사용할 수 있다. target_host는 HTTP Request Defaults에서 사용된다.

사실 간단한 테스트라면 굳이 변수로 미리 정의할 필요는 없다. 하지만 테스트가 복잡해지고 Sampler의 수가 증가할 경우 변수로 설정해 놓지 않으면 값을 변경할 때 많은 시간이 소요되거나 일관성이 맞지 않아서 테스트에 실패할 수 있다. 따라서 항상 변수로 설정해 놓으면 실수를 줄이고 시간을 아낄 수 있다.

[그림 7-4] Global

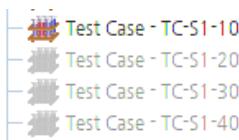
User Defined Variables		
Name:	Value	Description
target_host	xxxxxx.jmeter.ac.kr	
target_port	80	
connect_timeout	30000	
response_timeout	30000	
script_path	/jmeter_test/script/	
log_path	/jmeter_test/log/	
data_path	/jmeter_test/data/	

테스트 케이스

Global 변수 중에서 테스트 케이스마다 달라질 수 있는 변수들은 별도로 분리해서 테스트 케이스별로 파일을 나누고 테스트 케이스에 맞게 설정한다. 실제 테스트할 때에는 [그림 7-5]처럼 테스트 케이스에 맞는 것만 활성화하고 나머지는 비활성화하는 방식을 사용하면 유용하다. 활성화/비활성화 항목을 선택한 다음 ‘Ctrl-t’ 버

튼을 사용해서 토글toggle할 수 있다.

[그림 7-5] 테스트 케이스 활성화



[그림 7-6]은 ‘Test Case - TC-S1-10’의 내용이다. 각 변수는 테스트 영역에서 사용되는데, test_case_id 변수는 log 파일명을 테스트 케이스별로 구분하는 역할을 한다([결과 영역의 Summary Report](#) 참조).

[그림 7-6] Test Case - TC-S1-10

User Defined Variables

Name:	Test Case - TC-S1-10
Comments:	

User Defined Variables

Name:	Value	Description
threads_no	1000	
think_time	1000	milliseconds
test_case_id	tc_s1_10	
loop_cnt_1	5	
duration	300	seconds
rampup_time	150	seconds

데이터 파일

시나리오를 보면 user_list.csv, subject_list.csv, dept_list.csv 이렇게 세 개의 파일이 사용되는데, 이 파일들의 정보에 특정 변수로 접근할 수 있도록 [그림 7-7], [그림 7-8], [그림 7-9]처럼 CSV Data Set Config를 설정한다.

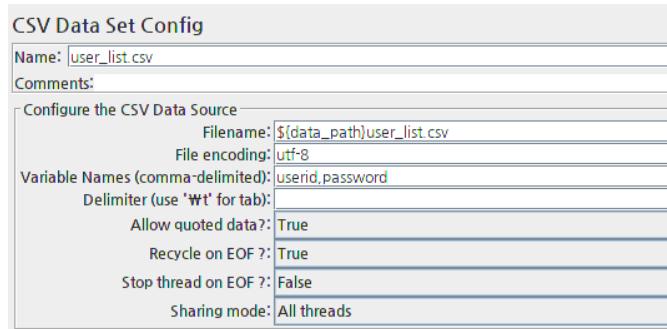
각 그림의 Filename을 보면 \${data_path}라는 Global에서 정의된 변수값을 사용하고 있다. 이렇게 설정하면 CSV 파일의 위치가 달라져도 Global 변수값만 바꿔 주면 쉽게 적용할 수 있다.

user_list.csv의 정보는 \${userid}와 \${password} 변수로 순차적으로 접근할

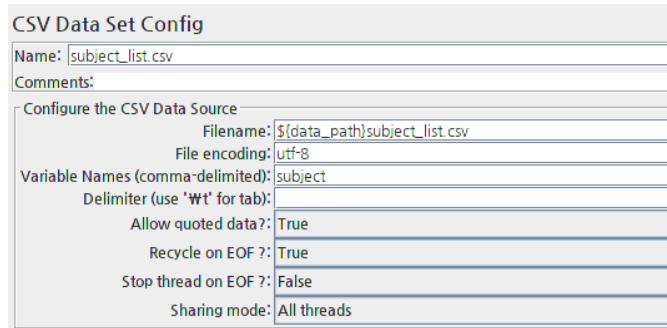
수 있고, subject_list.csv의 정보는 \${subject} 변수, dept_list.csv의 정보는 \${dept} 변수로 접근할 수 있게 설정되어 있다. 변수를 한 번 참조할 때마다 파일의 한 라인씩 아래로 이동하면서 서로 다른 값으로 설정된다.

Allow quoted data?와 Recycle on EOF?는 True, Stop thread on EOF?는 False로 설정하여 파일 내용이 다 소진되면 다시 처음부터 반복할 수 있도록 한다.

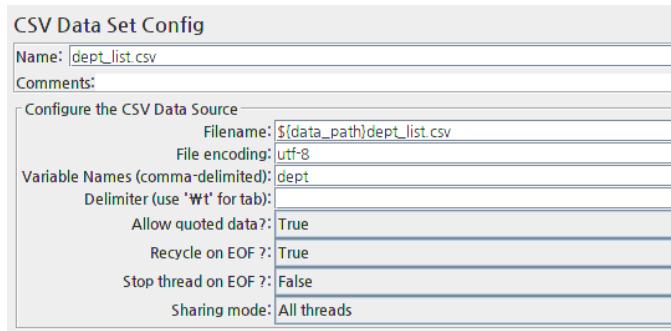
[그림 7-7] user_list.csv



[그림 7-8] subject_list.csv



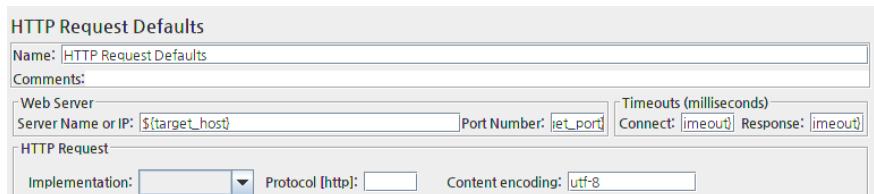
[그림 7-9] dept_list.csv



HTTP Request Defaults

HTTP Request Sampler를 사용할 때는 HTTP Request Defaults를 사용하는 것이 좋다. 이는 Global에서 정의된 \${target_host}, \${target_port}, \${connect_timeout}, \${response_timeout} 값을 각각 Server Name or IP, Port Number, Connect, Response로 설정할 수 있어서 Global에서 설정한 변수값으로 모든 HTTP Request Sampler의 정보를 갱신할 수 있다. 또한, 변수명 자체가 변경되는 경우에도 HTTP Request Defaults의 값만 변경하면 되므로 작업량이 줄고 실수를 방지할 수 있다.

[그림 7-10] HTTP Request Defaults



HTTP Cookie Manager

쿠키 기반의 세션을 유지해야 하는 서비스에는 반드시 추가해야 한다. Clear

cookie each iteration? 옵션을 체크해서 반복 수행할 때마다 쿠키 정보가 초기화 되도록 설정한다.

[그림 7-11] HTTP Cookie Manager



Think Time

테스트 케이스에 Think Time 설정이 있으므로 Constant Timer를 이용해서 Think Time을 적용한다.

[그림 7-12] Think Time



7.5.2 테스트 영역

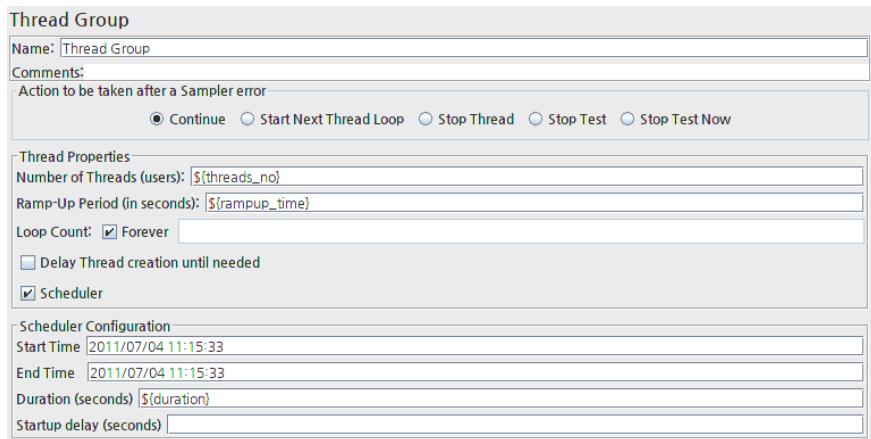
Thread Group

이번 테스트는 테스트 수행 시간이 300초(5분)이므로([표 7-1] 참고) Loop Count의 Forever와 Scheduler를 체크하고, Duration을 \${duration}으로 설정해서 테스트 케이스(Test Case - TC-S1-10, ...)에 설정된 시간만큼 테스트가 수행되도록 설정한다.

Number of Threads와 Ramp-Up Period도 테스트 케이스에 설정된 \${threads_no}와 \${rampup_time}으로 설정한다. Test Case - TC-S1-10에 설정된 threads_no는 1000, rampup_time은 150, duration은 300이므로 0부터 150

초까지는 가상 사용자 수가 0부터 1000까지 순차적으로 증가하고 150초부터 300초까지는 1000을 유지한다.

[그림 7-13] Thread Group



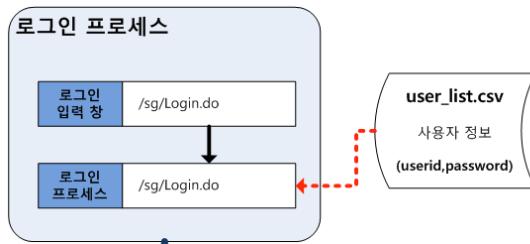
Simple Controller

테스트 영역에서는 여러 개의 Simple Controller가 사용되는데, 별다른 기능은 없고 단순히 각각의 Sampler나 Controller를 그룹으로 묶어주는 역할을 한다. 시나리오에 나타난 ‘로그인 프로세스’, ‘수강신청 프로세스’ 등의 그룹을 이 Simple Controller로 묶어주면 시나리오와 Test Plan 스크립트를 매칭시켜 분석하기가 쉽고 복사, 삭제, 활성화/비활성화 등도 쉬워진다.

로그인 프로세스

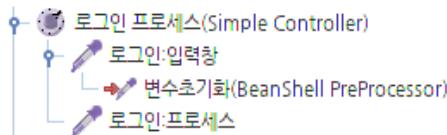
로직 구성도에서 로그인 프로세스는 [그림 7-14]처럼 두 개의 Sampler(로그인 입력창, 로그인 프로세스)와 1개의 데이터 파일(user_list.csv)로 구성되어 있다.

[그림 7-14] 로직 구성도의 로그인 프로세스

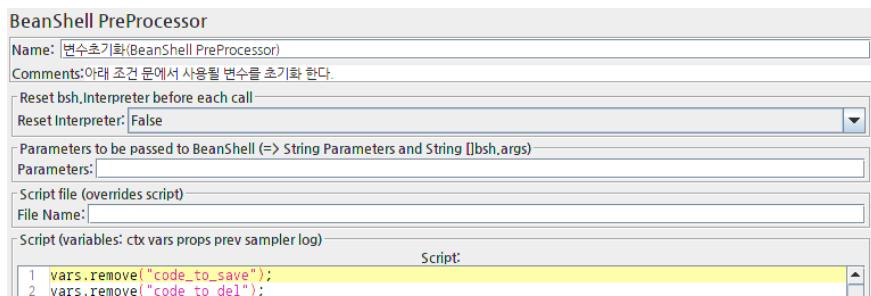


로그인 프로세스에서는 보이지 않지만 user_list.csv는 설정 영역에서 이미 추가 했으므로 \${userid}와 \${password}라는 변수로 접근할 수 있다. 로직 구성도에는 없지만 [그림 7-15]를 보면 ‘변수초기화(BeanShell PreProcessor)’가 있다. 이는 이후에 사용될 code_to_save와 code_to_del 변수를 테스트 Iteration 시작 시 초기화하는 과정이다. 별도의 BeanShell Sampler를 이용해서 구현할 수도 있지만, 그렇게 되면 Summary Report 등의 Listener에 기록이 남으므로 PreProcessor를 이용한다.

[그림 7-15] 로그인 프로세스



[그림 7-16] 변수초기화



로그인: 입력창에는 시나리오에 정의된 URL 외에 별다른 설정이 없고, 로그인: 프로세스에는 [그림 7-17]과 같이 \${userid}와 \${password}를 txt_user_id와 txt_passwd 파라미터에 적용하는 설정과 Method를 POST 방식으로 변경하는 설정이 추가되어 있다.

[그림 7-17] 로그인:프로세스

HTTP Request

Name: 로그인:프로세스
Comments:

Web Server
Server Name or IP: _____ Port Number: _____ Connect: _____ Response: _____
Timeouts (milliseconds)

HTTP Request

Implementation: _____ Protocol [http]: _____ Method: POST Content encoding: _____
Path: /sg/Login.do
 Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters **Body Data**

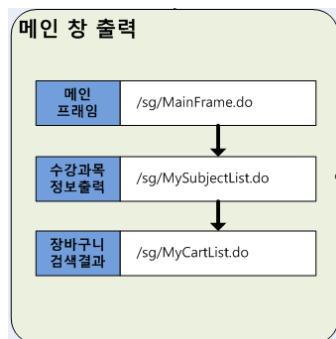
Send Parameters With the Request:

Name:	Value	Encode?	Include Eq.
txt_passwd	\$(password)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
txt_user_id	\$(userid)	<input type="checkbox"/>	<input checked="" type="checkbox"/>

메인창 출력

로직 구성도의 메인창 출력 영역을 보면 별다른 로직이나 데이터 입력이 없는 단순한 HTTP Request Sampler의 리스트다.

[그림 7-18] 로직 구성도의 메인창 출력



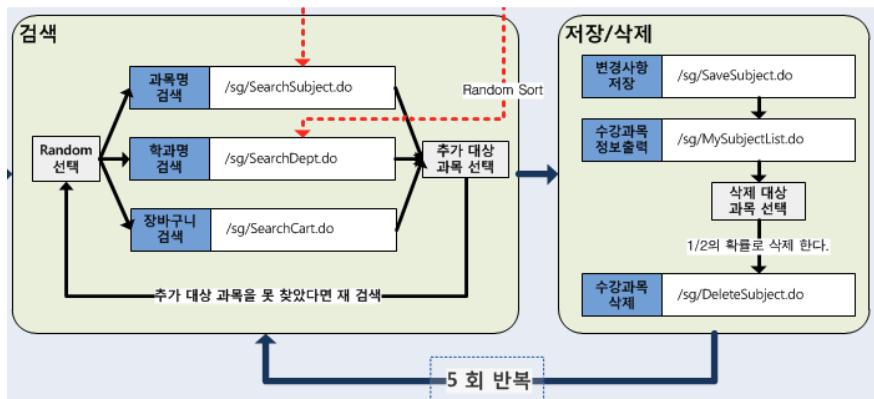
[그림 7-19] 메인창 출력

- 수강신청:메인프레임
- 수강신청:수강과목정보출력
- 수강신청:장바구니결과검색

Loop Controller

[그림 7-20]은 로직 구성도에서 검색과 저장/삭제 부분이다. 그림 아래쪽을 보면 검색과 저장/삭제를 ‘5회 반복’ 하라는 시나리오를 확인할 수 있다. 이런 반복 작업은 Loop Controller로 구현할 수 있는데, 그 내용은 [그림 7-21]과 같다.

[그림 7-20] 로직 구성도의 반복



[그림 7-21]에서 Loop Count에 설정된 \${loop_cnt_1}은 Test Case - TC-S1-10에 설정된 변수값이다.

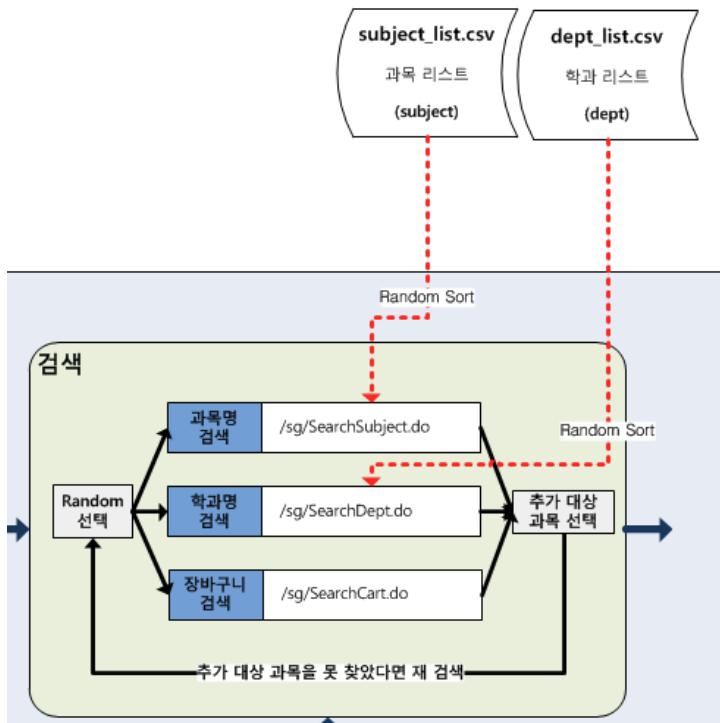
[그림 7-21] Loop Controller

Loop Controller	
Name:	Loop Controller
Comments:	
Loop Count:	<input type="checkbox"/> Forever \${loop_cnt_1}

검색

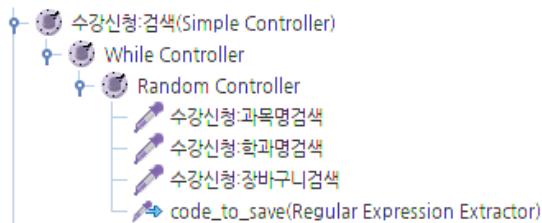
[그림 7-22]의 검색 부분 로직을 살펴보면 세 가지 검색 방법(과목명검색, 학과명검색, 장바구니검색)이 있고, 이중에서 하나를 무작위로 선택한 후 검색 결과에서 ‘추가 대상 과목 선택’을 한다. 선택된 과목이 있으면 다음 단계로 이동하고 그렇지 않으면 검색 작업을 반복한다.

[그림 7-22] 로직 구성도의 검색



[그림 7-23]은 검색 로직을 Test Plan 스크립트로 구현한 것이다.

[그림 7-23] 검색



세 가지 검색 중 한 가지를 무작위로 선택할 때는 Random Controller를 사용한다. 특별한 설정은 없으며 Random Controller에 존재하는 Sampler 중 하나를 무작위로 선택하게 된다.

code_to_save는 Sampler가 아닌 PostProcessor이므로 무작위 선택의 대상이 아니고, Sampler를 실행한 후에 추가 대상 과목을 선택하는 부분이다.

각 검색 결과는 다음 구조를 가진다. 이 내용에서 회색 부분을 Regular Expression Extractor로 추출한다.

```
<table cellpadding=2>
<tr>
    <th>과목 코드</th>
    <th>과목 명</th>
    <th>학과</th>
    <th>교수</th>
    <th>수업 일시</th>
    <th>강의실</th>
    <th></th>
</tr>
<tr>
    <td>132456</td>
    <td>자료구조</td>
    <td>컴퓨터공학과</td>
```

```

<td>홍길동</td>
<td>수3-4, 목1</td>
<td>학술관 203호</td>
<td><input type="button" onclick="sugang_save(132456,1122);"
class="btn_sugang_save" value="신청"></td>
</tr>
<tr>
<td>135426</td>
<td>이산수학</td>
<td>수학과</td>
<td>김세훈</td>
<td>월1-2, 수3</td>
<td>본관 303호</td>
<td><input type="button" onclick="sugang_save(135426,2134);"
class="btn_sugang_save" value="신청"></td>
</tr>
</table>

```

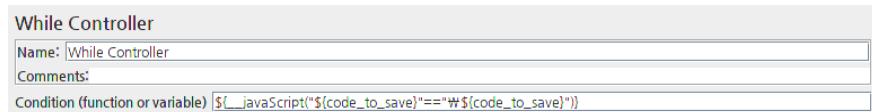
PostProcessor에 Regular Expression Extractor를 추가하고 [그림 7-24]와 같이 설정한다. 정규표현식과 일치하는 정보가 여러 개면 이 중에서 무작위로 하나가 선택된다. 정규표현식에 두 개의 그룹이 있으므로 두 개의 변수 \${code_to_save_g1}, \${code_to_save_g2}가 설정된다.

[그림 7-24] code_to_save

The screenshot shows the 'Regular Expression Extractor' configuration in JMeter's PostProcessor settings. The 'Name' field is set to 'code_to_save'. The 'Regular Expression' field contains the pattern for the save button: '<input type="button" onclick="sugang_save(\d{0-9}\d{0-9}),(\d{0-9}\d{0-9});" class="btn_sugang_save" value="신청"'. The 'Template' field is set to '\$1\$'. The 'Match No. (0 for Random)' field is set to '0'. The 'Default Value' field is empty. Other options like 'Main sample only' and 'Sub-samples only' are also visible.

정규표현식에 의해 선택된 정보가 없다면 다시 검색을 반복한다. 이는 While Controller로 구현할 수 있는데, [그림 7-25]와 같이 설정한다. 이는 “\${code_to_save}의 값이 정의되어 있지 않으면 다시 반복 수행하라”라는 의미다. 앞의 로그인 프로세스에서 ‘변수초기화’를 했는데, 그 부분에서 변수를 초기화한 이유가 바로 이 조건식을 만족하기 위해서다. 이는 초기에 변수를 삭제하지 않으면 한번 선택된 \${code_to_save} 변수값이 Iteration이 반복되어도 계속 사용되어서 더는 While Controller 안으로 들어오지 않기 때문이다.

[그림 7-25] While Controller]



과목명검색과 학과명검색에서 사용될 \${subject}와 \${dept} 변수는 설정 영역에서 추가한 subject_list.csv와 dept_list.csv로 접근할 수 있다. 각 변수는 arg_curi_nm과 CuriDept 파라미터에 적용되어 있다.

[그림 7-26] 수강신청:과목명검색

HTTP Request

Name: 수강신청:과목명검색

Comments:

Web Server

Server Name or IP: Port Number: Timeouts (milliseconds)

Connect: Response:

HTTP Request

Implementation: Java Protocol [http]: Method: POST Content encoding:

Path: /sg/SearchSubject.do

Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters Body Data

Send Parameters With the Request:				
Name:	Value	Encode?	Include Equ	
arg_curi_nm	\${subject}	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
arg_curi_num		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
sconf	curi	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

[그림 7-27] 수강신청:학과명검색

HTTP Request

Name: 수강신청:학과명검색
Comments:

Web Server
Server Name or IP: Port Number: Timeouts (milliseconds)
Connect: Response:

HTTP Request

Implementation: Java Protocol [http]: Method: POST Content encoding:
Path: /sg/SearchDept.do
 Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters Body Data

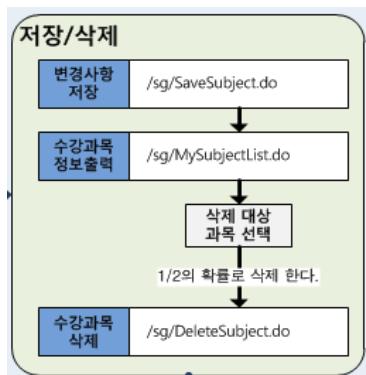
Send Parameters With the Request:

Name:	Value	Encode?	Include Equ.
CuriDept	\${dept}	<input type="checkbox"/>	<input checked="" type="checkbox"/>
sconf	dept	<input type="checkbox"/>	<input checked="" type="checkbox"/>
class_div	1	<input type="checkbox"/>	<input checked="" type="checkbox"/>
class_div1	01	<input type="checkbox"/>	<input checked="" type="checkbox"/>
groupNum	01	<input type="checkbox"/>	<input checked="" type="checkbox"/>

저장/삭제

테스트 영역의 마지막 부분은 저장/삭제다. [그림 7-28]을 보면 검색은 세 개의 Sampler(변경사항저장, 수강과목정보출력, 수강과목삭제)로 구성되며 ‘수강과목정보 출력’에서 삭제 대상 과목을 선택해서 1/2의 확률로 삭제 Sampler를 실행하는 로직이 추가되어 있다.

[그림 7-28] 로직 구성도의 저장/삭제



이 로직 구성도의 내용을 JMeter의 Test Plan 스크립트로 구현하면 [그림 7-29]의 형태로 나타낼 수 있다.

[그림 7-29] 저장/삭제 영역



'수강신청:변경사항저장' Sampler는 검색 영역에서 선택한 과목정보를 저장하는 부분이다. 이는 [그림 7-30]의 구조를 가지며, 검색 영역에서 선택한 과목정보를 변수 \${code_to_save_g1}과 \${code_to_save_g2}를 이용해서 서버로 전달한다. 대부분 이런 정보 저장 작업은 POST 방식을 이용한다. 실 서버에서 POST와 GET 방식을 모두 동일하게 처리하면 상관없지만, 보안이나 구현상의 이유로 POST 방식만 처리되는 경우가 있으므로 허용하는 방식에 맞는 형식을 선택해야 한다. 여기에서는 POST 방식을 이용한다.

[그림 7-30] 수강신청:변경사항저장

The screenshot shows the configuration for an 'HTTP Request' sampler in JMeter. The 'Name' field is set to '수강신청:변경사항저장'. The 'Server Name or IP' field is empty, and the 'Port Number' field is also empty. Under the 'HTTP Request' tab, the 'Implementation' is set to 'Java', 'Protocol' is 'http', 'Method' is 'POST', and 'Content encoding' is 'euc-kr'. The 'Path' is set to '/sg/SaveSubject.do'. Under the 'Parameters' tab, there are two parameters: 'subject_code' with value '\${code_to_save_g1}' and 'dept_code' with value '\${code_to_save_g2}'. Both parameters have 'Encode?' checked and 'Include Eq' checked.

‘변경사항저장’이 완료되면 지금까지 신청한 과목의 리스트를 출력하는 ‘수강신청:수강과목정보출력’ Sampler를 실행해서 삭제 대상 과목을 선정해야 한다. 여기서는 선택된 과목이 있을 때 무조건 삭제하는 것이 아니라 1/2의 확률, 즉 두 번 중 한 번은 삭제하고 한 번은 삭제하지 않는 방식으로 구현해야 한다.

이런 확률적인 구현은 여러 가지 방식으로 구현할 수 있지만 여기에서는 랜덤 변수를 생성하는 Random Variable과 If Controller를 사용한다. Random Variable로 0~9까지의 숫자를 생성한 다음 If Controller를 이용해서 앞에서 받아온 숫자가 5보다 작은 경우에만 삭제를 수행하는 방식으로 확률적 제어를 구현한다.

먼저 ‘수강과목정보출력’ 결과에서 삭제될 과목정보를 추출한다. 이는 Regular Expression Extractor PostProcessor를 이용해서 처리하는데, 결과는 다음과 같다.

```
<table>
  <tr>
    <th>과목코드</th>
    <th>과목명</th>
    <th>학과</th>
    <th>교수</th>
    <th>수업일시</th>
    <th>강의실</th>
    <th></th>
  </tr>
  <tr>
    <td>132456</td>
    <td>자료구조</td>
    <td>컴퓨터공학과</td>
    <td>홍길동</td>
    <td>수3-4, 목1</td>
    <td>학술관 203호</td>
    <td><input type="button" onclick="sugang_del(132456,1122);"
      class="btn_sugang_del" value="삭제"></td>
```

```

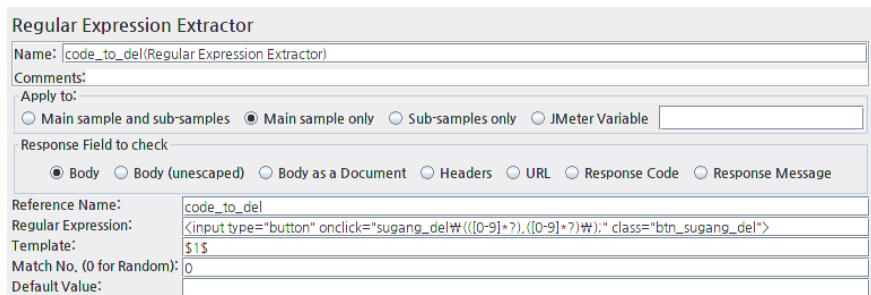
</tr>
<tr>
    <td>135426</td>
    <td>이산수학</td>
    <td>수학과</td>
    <td>김세훈</td>
    <td>월1-2, 수3</td>
    <td>본관 303호</td>
    <td><input type="button" onclick="sugang_del(135426,2134);"
class="btn_sugang_del" value="삭제"></td>
</tr>
</table>

```

회색으로 표시된 부분에서 `sugang_del` 함수에 인자로 입력되는 두 개의 숫자를 정규표현식으로 찾는데, 이때 Regular Expression Extractor의 내용은 [그림 7-31]과 같다. 결과에 접근할 변수명은 `code_to_del`이고, 두 그룹이 존재하므로 `${code_to_del_g1}`과 `${code_to_del_g2}`로 접근할 수 있다. 그리고 정규표현식 `<input type="button" onclick="sugang_del\(([0-9]*?),([0-9]*?)\);"` class="btn_sugang_del">

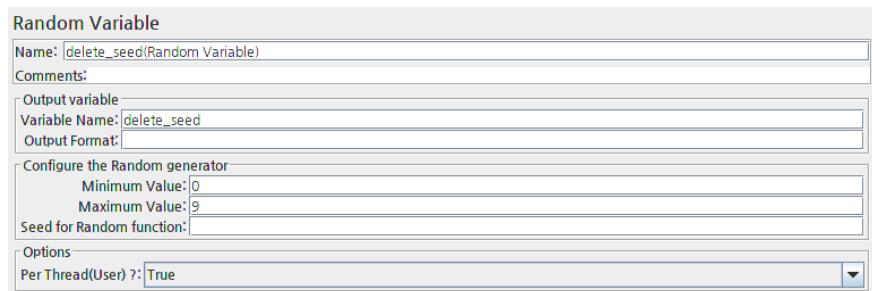
로 두 그룹의 정보를 받아온다. 여러 개의 매칭이 발생하면 무작위로 하나를 선택하도록 Match No.를 0으로 설정한다.

[그림 7-31] code_to_del



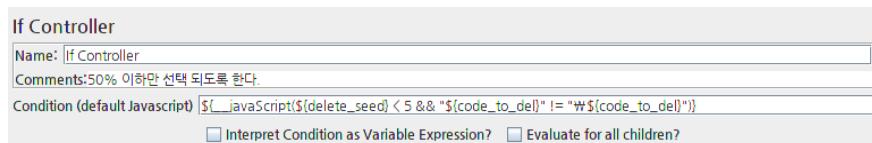
삭제할 과목이 선정되면 1/2의 확률을 계산하기 위한 Random Variable을 추가한다. [그림 7-32]처럼 변수명은 delete_seed로 정하는데, 이 변수에 선택될 값은 0~9 사이 값 중 하나가 된다. 결과값은 0, 5, 7과 같이 정수로 출력된다.

[그림 7-32] delete_seed



`${delete_seed}`와 `${code_to_del}`이 설정되었으므로 50%의 확률을 계산한다. [그림 7-33]과 같이 If Controller를 추가하고 Condition 부분에 `${__javaScript(${delete_seed} < 5 && "${code_to_del}" != "\${code_to_del}")}`을 입력한다. 이는 `${delete_seed}`가 5보다 작고 `${code_to_del}` 변수가 설정된 경우(삭제과목이 존재할 경우)에 If Controller의 자식 노드 내용을 실행하라는 의미다.

[그림 7-33] If Controller



모든 조건이 충족되면 정규표현식에 의해 선택된 정보를 이용해서 삭제 명령을 수행한다. 두 그룹이 있으므로 `${code_to_del_g1}`, `${code_to_del_g2}`로 파라미터를 입력한다.

[그림 7-34] 수강신청:수강과목삭제

HTTP Request

Name: 수강신청:수강과목삭제
Comments:

Web Server
Server Name or IP: Port Number: Timeouts (milliseconds)
Connect: Response:

HTTP Request

Implementation: Java Protocol [http]: Method: POST Content encoding: euc-kr
Path: /sg/DeleteSubject.do
 Redirect Automatically Follow Redirects Use KeepAlive Use multipart/form-data for POST Browser-compatible headers

Parameters Body Data

Send Parameters With the Request:

Name	Value	Encode?	Include Equ.
subject_code	\$(code_to_del_g1)	<input type="checkbox"/>	<input checked="" type="checkbox"/>
dept_code	\$(code_to_del_g2)	<input type="checkbox"/>	<input checked="" type="checkbox"/>

7.5.3 결과 영역

결과 영역은 테스트 영역에서 수행된 테스트의 결과를 볼 수 있는 Listener 영역이다. 테스트 과정에서 가장 많이 사용될 네 가지 Listener를 추가한다.

Summary Report

결과를 한눈에 수치로 볼 수 있는 Listener로, 가장 많이 사용된다. 여기서 주의 깊게 볼 부분은 Filename이다. 각 테스트 케이스의 내용을 로그 파일에 저장해 놓으면 나중에 테스트 결과를 다시 불러와서 복기할 수 있으므로 매우 유용하다. \${log_path}/\${test_case_id}_\${TESTSTART.MS}.jtl로 설정되어 있으며 미리 정의된 변수값을 이용해서 \${log_path} 아래에 \${test_case_id}와 테스트 시작시간 (\${TESTSTART.MS})으로 구분하여 저장된다. 테스트 케이스마다 직접 파일을 지정해도 되지만, 이렇게 해놓으면 자동으로 파일이 생성되므로 실수로 다른 곳에 정보를 저장하는 문제가 발생하지 않는다.

[그림 7-35] Summary Report

Summary Report																		
Name:	[Summary Report]																	
Comments:																		
Write results to file / Read from file																		
Filename:	\$[log_path]/\$(test_case_id)_\$(TESTSTART_MS).jtl					Browse...	Log/Display Only:	<input type="checkbox"/> Errors	<input type="checkbox"/> Successes									
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes									
로그인 일련상	1515	15	0	397	34.87	0.00%	15.5/sec	26.65	1761.0									
로그인 프로세스	1492	16	0	405	36.54	0.00%	15.4/sec	26.52	1761.0									
수강신청 메인프레임	1467	15	0	408	34.10	0.00%	15.3/sec	25.29	1690.0									
수강신청_수강과목별로출석체크	6878	16	0	404	35.25	0.00%	72.6/sec	119.81	1690.0									
수강신청_수강구니_결과검색	1414	16	0	379	34.90	0.00%	15.1/sec	24.90	1690.0									
수강신청_수강구니_검색	441	17	0	240	39.90	0.00%	4.8/sec	7.85	1690.0									
수강신청_변경사항저장	5565	859	500	1417	229.89	0.00%	60.7/sec	100.18	1691.0									
수강신청_학과명검색	463	1504	1000	2133	312.41	0.00%	5.0/sec	8.30	1692.0									
수강신청_과목명검색	442	3492	2000	5075	904.41	0.00%	4.8/sec	7.88	1691.0									
수강신청_수강과목삭제	2645	514	200	1125	204.39	0.00%	29.8/sec	49.25	1691.0									
TOTAL	22322	385	0	5075	626.71	0.00%	228.3/sec	379.02	1700.0									

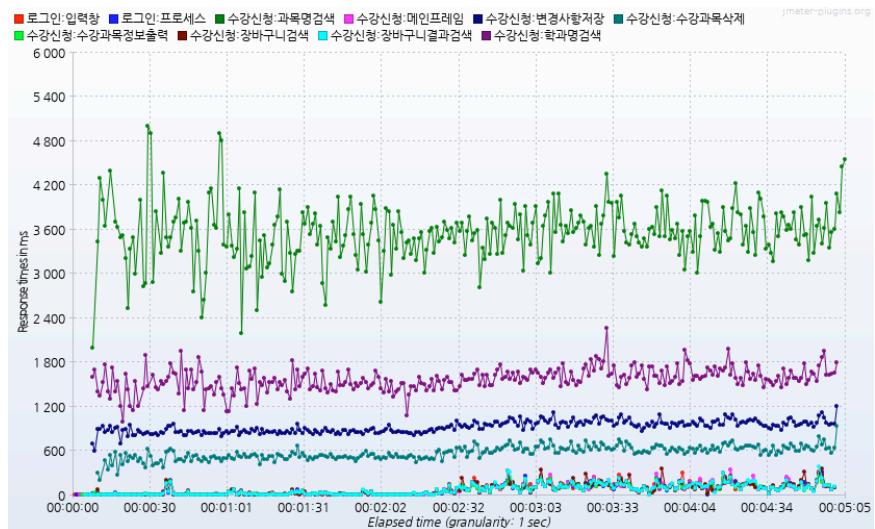
View Results Tree

성능에 매우 많은 영향을 미치므로 실 테스트에서는 활성화해서 사용하면 안 된다. 사전 테스트(Pre-Test)나 스크립트 검증 때에만 사용하도록 한다. (jp@gc - Response Times Over Time/ jp@gc - Transactions per Second)

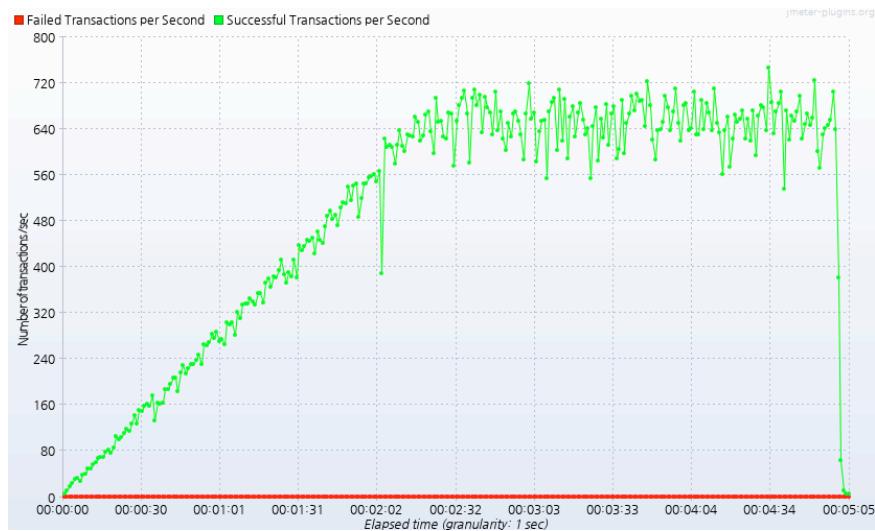
테스트 과정에서 가장 중요한 결과는 응답시간과 처리량 정보다. Summary Report에서는 테스트 내내 누적된 최종평균값만을 확인할 수 있어서 각 값의 변화 추이를 보려면 이 Listener를 추가하는 것이 좋다.

[그림 7-36]과 [그림 7-37]는 각각 jp@gc - Response Times Over Time 과 jp@gc - Transactions per Second의 실행 결과다. [그림 7-36]처럼 각 Sampler의 결과를 따로 볼 수 있고, [그림 7-37]처럼 모든 Sampler의 결과를 통합해서 하나로 볼 수도 있다.

[그림 7-36] Response Times Over Time



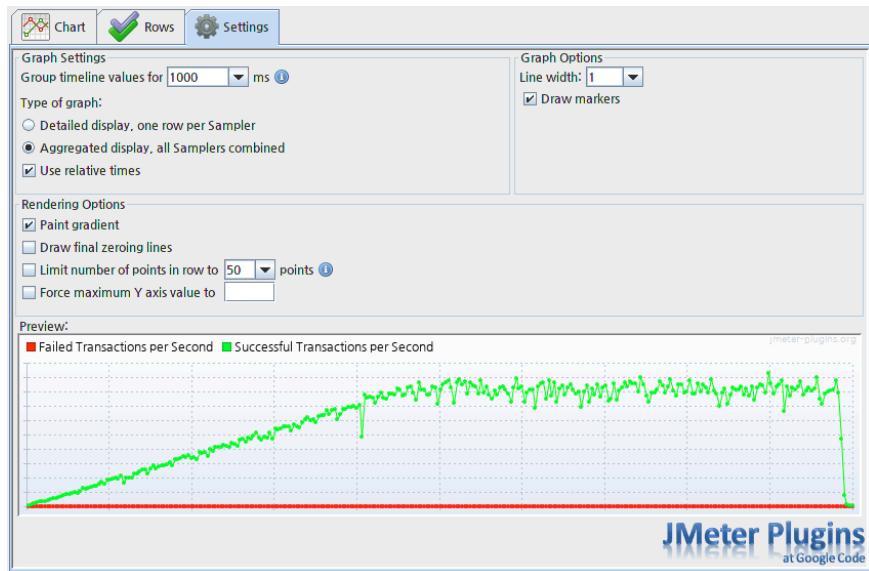
[그림 7-37] Transactions per Second



설정 정보는 Settings 탭에서 변경할 수 있다. Sampler를 각각 볼지 통합해서 볼지는 Type of graph 옵션으로 변경한다.

Group timeline values for는 가장 중요한 옵션의 하나로, Sampler 결과를 취합해서 하나의 점을 그래프에 찍을 때 그 기간을 얼마로 할지에 대한 설정이다. 작게 하면 할수록 그래프가 빠죽 빠죽 거칠어지고 점이 좀 더 자주 찍히며, 크게 하면 할수록 완만해지며 점이 덜 자주 찍히게 된다.

[그림 7-38] Transactions per Second – Settings



8 | 테스트 수행 및 결과 수집

8.1 사전 테스트

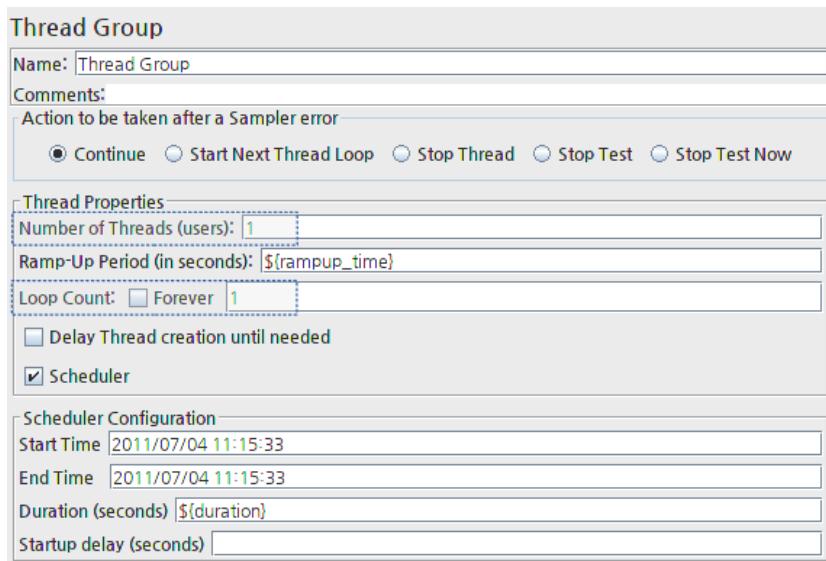
‘사전 테스트(Pre-Test)’란 실제 테스트가 진행되기 전에 테스트에 문제가 없는지 점검하는 테스트다. 사전 테스트를 진행하지 않고 바로 테스트를 진행하면 테스트 관련된 내부 및 외부 인력을 모두 모아놓은 상황에서 불필요한 대기시간이 발생하거나 급하게 테스트 일정을 변경해야 하는 불상사가 종종 발생하곤 한다. 이러한 문제가 발생하지 않으려면 사전 테스트로 다음 내용을 미리 점검해야 한다.

8.1.1 JMeter로 사전 테스트하기

사전 테스트는 본격적인 테스트를 하기 전에 테스트가 정상적으로 수행될 수 있을지 검사하는 과정이므로 많은 수의 가상 사용자로 오랫동안 테스트하는 것이 아니라 적은 수의 가상 사용자로 스크립트를 실행해 보는 것이다. 여러 대의 부하발생기로 테스트할 예정이라면 부하발생기도 정상 동작하는지 확인해야 한다.

가장 먼저 할 일은 Thread Group의 설정을 변경하는 것이다. [그림 8-1]처럼 Number of Threads와 Loop Count를 1로 변경해서 1번만 실행되도록 설정하고 테스트한다.

[그림 8-1] 사전 테스트를 위한 Thread Group 설정



결과는 View Results Tree를 활성화해서 확인한다. View Results Tree는 사전 테스트에서 사용하려고 Test Plan에 추가한 것이므로 사전 테스트가 끝난 후에는 반드시 비활성화시켜야 한다.

[그림 8-2]는 Thread Group 설정으로 사전 테스트를 실행한 결과다. ‘수강신청: 학과명검색’에서 오류가 발생하여 Sample result에 HTTP/1.1 404 Not Found라는 메시지가 뜬 것을 볼 수 있다. 이는 Test Plan 스크립트에서 URL 경로가 잘못 작성되었거나 서버 애플리케이션에 문제가 있는 것이므로 확인이 필요하다.

[그림 8-2] View Results Tree 사전 테스트 결과

The screenshot shows the 'View Results Tree' listener in JMeter. On the left, a tree view lists various test samples under categories like '로그인_입력창', '로그인_프로세스', and '수강신청_메인프레임'. A specific sample under '수강신청_화면검색' is selected, highlighted with a yellow warning icon. The right panel displays detailed information for this selected sample:

- Sampler result** tab:
 - Thread Name: Thread Group 1-1
 - Sample Start: 2014-11-21 19:00:16 KST
 - Load time: 1
 - Latency: 1
 - Size in bytes: 479
 - Headers size in bytes: 180
 - Body size in bytes: 299
 - Sample Count: 1
 - Error Count: 1
 - Response code: 404
 - Response message: 'Not Found'
- Response headers:**
 - HTTP/1.1 404 Not Found
 - Date: Fri, 21-Nov-2014 10:00:16 GMT
 - Server: Apache/2.4.10 (Ubuntu)
 - Content-Length: 299
 - Connection: close
 - Content-Type: text/html; charset=iso-8859-1
- HTTPSSampleResult fields:**
 - ContentType: text/html; charset=iso-8859-1
 - DataEncoding: iso-8859-1

At the bottom, there are tabs for **Raw** and **Parsed**.

8.1.2 점검사항

Test Plan 스크립트는 정상 동작하는가?

때로는 Test Plan 스크립트를 작성한 시점과 실제 테스트하는 시점에서 애플리케이션의 로직이 변경되어 오류가 발생하곤 한다. 사전 테스트에서 문제가 발견되면 개발팀과 상의해서 변경사항을 Test Plan 스크립트에 추가 또는 변경해야 한다.

Test Plan 스크립트를 작성한 PC와 부하발생기의 JMeter 버전이나 확장 모듈 내

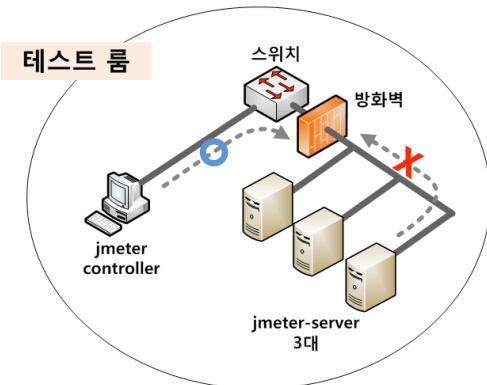
용이 달라서 작성한 Test Plan 스크립트가 구동되지 않는 경우도 있으므로 꼭 버전을 맞춰준다.

테스트 환경은 정상적으로 구축되었는가?

개인 PC에서 Test Plan을 작성할 때에는 정상 동작했으나 실제로 부하를 발생시킬 jmeter-server에서 Test Plan 스크립트를 실행하면 네트워크 연결 문제로 오류가 종종 발생한다.

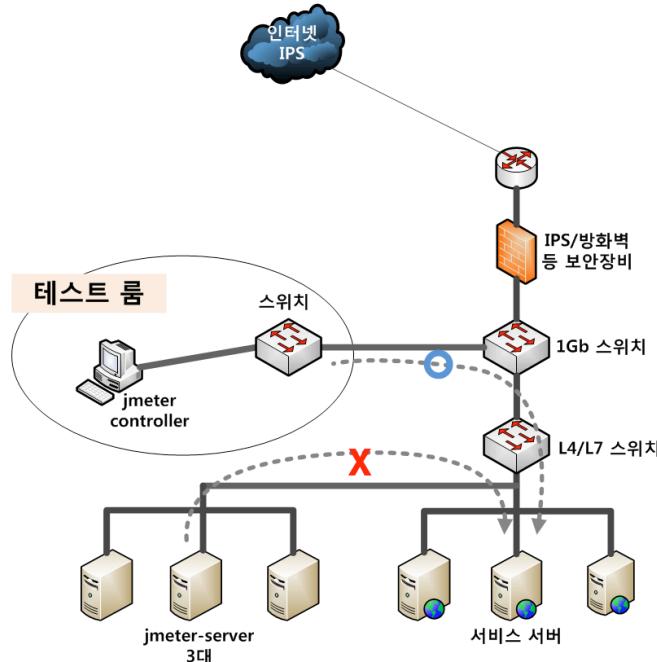
jmeter controller와 jmeter-server 사이에 방화벽이나 보안 장비가 존재하면 JMeter의 RMI 포트가 차단되거나 보안 규칙에 걸려 차단되는 경우가 있다.

[그림 8-3] 방화벽에 의한 차단



JMeter controller와 서비스 서버 구간에서는 정상적으로 페이지가 로딩되지 만, jmeter-server와 서비스 서버 구간에서는 페이지가 로딩되지 않는 경우에는 jmeter-server의 네트워크 구성을 다시 한 번 살펴보아야 한다. 예를 들어, [그림 8-4]처럼 L4에 직접 jmeter-server가 연결되면 같은 L4 스위치 내의 서비스 서버에 접근하지 못하는 경우도 있다. 이때는 상위 스위치로 네트워크 구성을 변경해야 한다.

[그림 8-4] L4 스위치에 의한 차단



WAS/WEB/DB 서버 등의 서비스 서버에 H/W 문제가 없는지 확인해야 한다. 가장 흔히 발생하는 문제는 디스크가 가득 차서 평소와는 다른 비정상적인 결과를 나타내는 경우다.

8.2 테스트 수행

8.2.1 부하발생기 수와 가상 사용자

한 대의 JMeter로 테스트하는 경우에는 Thread Group에 설정된 Number of Threads의 수가 실제 생성되는 가상 사용자의 수와 동일하지만 복수의 부하발생기를 이용하는 경우에 'Number of Threads × 부하발생기 수' 만큼의 가상 사용자

자가 생성된다. 그러므로 네 대의 부하발생기를 이용할 때 1000명의 가상 사용자를 생성하려면 Number of Threads를 250으로 설정해야 한다. 즉, 분산 테스트 시 Thread Group에 입력될 가상 사용자 수는 다음과 같다.

Thread Group 입력 가상 사용자 수 = 목표 가상 사용자 수 / 부하발생기 수

8.2.2 실행

JMeter를 실행하고 정지하는 방법은 앞에서 설명했으므로 앞 부분을 참고하기 바란다. 몇 개의 Thread가 실행되고 있는지는 JMeter 화면 우측 상단, Log Viewer를 실행시키는 느낌표 버튼의 오른쪽에 표시된다. jmeter controller에서 실행했을 때와 jmeter-server에서 실행했을 때 이 숫자가 다소 다르게 나타난다. controller에서만 실행하면 전체 실행해야 할 Thread 숫자가 함께 표시되고, jmeter-server에서 분산 테스트 형태로 실행하면 현재 실행되고 있는 Thread 숫자만 출력된다. 숫자 오른쪽 연두색 사각형은 현재 테스트가 실행 중인지를 나타내는 아이콘으로, 정지 상태일 때는 회색이고 실행 중일 때에는 [그림 8-5]와 같이 연두색으로 표시된다.

[그림 8-5] jmeter controller에서 실행



[그림 8-6] jmeter-server에서 실행



8.2.3 테스트가 정상적으로 수행되는가?

테스트를 실행하면 단순히 결과가 나올 때까지 기다리면 되는 것이 아니다. 정상적으로 동작하는지 항상 점검해야 한다. 성능 테스트 과정에서 문제를 파악하지 못하

면 밤샘을 하면 테스트한 것이 허사로 돌아가거나 많은 부하발생기와 인력(내부, 외부)을 다시 소집해야 하는 불상사가 발생할 수도 있다.

Summary Report 모니터링

테스트가 정상적으로 수행되는지는 Summary Report를 모니터링해서 어느 정도 파악할 수 있다.

특정 Sampler에서 오류가 발생하거나 오류가 주기적인 패턴으로 발생하는 경우가 종종 있다. 서버 문제일 수도 있고 부하발생기의 문제일 수도 있다. 초기에는 오류가 발생하지 않다가 요청량이 높아지면서 갑자기 오류가 발생하기 시작한다면 소켓 수 등의 시스템 설정 제한에 걸린 것이 아닌지 확인해야 한다.

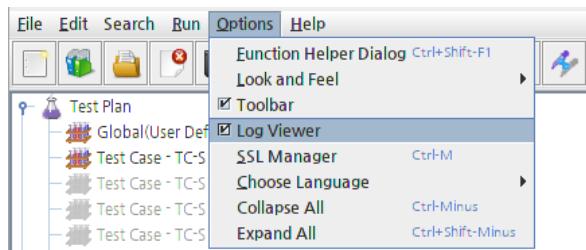
또한, 결과 수치가 터무니없이 작거나 큰 현상도 발생할 수 있다. 10KB 정도 크기의 파일을 요청했는데, 100B도 안 되는 응답이 오는 경우가 있다(Summary Report의 Avg Bytes 필드로 확인) 이는 보통 서버에서 오류 메시지를 정상적인 HTTP 응답(200 OK)으로 보낼 때 발생한다. 이 또한 오류이므로 이런 오류를 처리할 수 있는 Assertion을 추가해야 한다.

대역폭도 확인하는 것이 좋다. TOTAL KB/sec 값이 스위치나 서비스 서버에서 허용된 최고치에 가까운지를 확인한다. 테스트 장비가 1Gbits/sec 스위치에 연결되어 있을 때 TOTAL KB/sec가 131,072KB/sec에 가깝게 접근한다면 대역폭 제한에 의한 제약을 의심해 봐야 한다. 1Gbits/sec의 장비라 해도 현실적으로는 이 수치의 7~80%만 사용되어도 전체를 거의 사용했다고 판단하는 것이 일반적이므로 TOTAL KB/sec가 104,857KB/sec 근처라면 충분히 의심해볼 만하다. 대역폭 제약이 의심된다면 사이즈가 큰 파일을 테스트에서 제외하거나 네트워크나 서버의 용량을 올려서 테스트해야 한다.

JMeter 로그 확인

jmeter controller에서 로그를 확인하는 가장 쉬운 방법은 Log Viewer를 이용하는 것이다. Log Viewer를 활성화하는 방법은 두 가지다. 메뉴에서 ‘Options → Log Viewer’를 선택하면(그림 8-7) 각 노드의 아래쪽에 [그림 8-8]처럼 Log Viewer 창이 보인다.

[그림 8-7] Log Viewer 활성화



[그림 8-8] Log Viewer

The screenshot shows the JMeter Log Viewer interface. At the top, there's a 'Summary Report' section with fields for 'Name' (Summary Report), 'Comments', 'Filename' (\$log_path/\$test_case_id_\${TESTSTART_MS}.jtl), and buttons for 'Browse...', 'Log/Display Only', 'Errors', 'Successes', and 'Configure'. Below this is a table with columns: Label, # Samples, Average, Min, Max, Std. Dev., Error %, Throughput, KB/sec, and Avg. Bytes. The table data is as follows:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
로그인_입력창	1	1	1	1	0.00	0.00%	1000.0/sec	1719.73	1761.0
로그인_프로세스	1	1	1	1	0.00	0.00%	1000.0/sec	1719.73	1761.0
수강신청_메인페이지	1	1	1	1	0.00	0.00%	1000.0/sec	1650.39	1690.0
수강신청_수강과목정보보기	6	1	1	1	0.00	0.00%	20.5/min	0.56	1690.0
수강신청_장바구니결과검색	1	0	0	0	0.00	0.00%	0/hour	0.00	1690.0
수강신청_학과별과목검색	1	3	3	3	0.00	100.00%	333.3/sec	155.92	479.0
수강신청_변경사항저장	5	781	502	1101	231.28	0.00%	22.1/min	0.61	1691.0
수강신청_수강과목삭제	2	601	601	601	0.00	0.00%	24.9/min	0.69	1691.0
TOTAL	18	284	0	1101	378.95	5.56%	48.6/min	1.29	1631.0

Below the table is a checkbox for 'Include group name in label?' and a 'Save Table Data' button. The main log viewer area shows a list of log entries from November 22, 2014, at 13:05:55. The log entries are as follows:

```
7 2014/11/22 13:05:55 INFO - jmeter.engine.StandardJMeterEngine: Thread will continue on error
8 2014/11/22 13:05:55 INFO - jmeter.threads.Threads: Starting thread group 1 threads 1 ramp-up 150 perThread 150000.0 de...
9 2014/11/22 13:05:55 INFO - jmeter.threads.ThreadGroup: Started thread group number 1
10 2014/11/22 13:05:55 INFO - jmeter.engine.StandardJMeterEngine: All thread groups have been started
11 2014/11/22 13:05:55 INFO - jmeter.threads.JMeterThread: Thread started: Thread Group 1-1
12 2014/11/22 13:05:55 WARN - jmeter.config.CSVDataSet: Empty delimiter converted to ','
13 2014/11/22 13:05:55 INFO - jmeter.services.FileServer: Stored: /jmeter_test/data/dept_list.csv
14 2014/11/22 13:05:55 ERROR - jmeter.config.CSVDataSet: java.io.FileNotFoundException: /jmeter_test/data/dept_list.csv (그린 파일이거나...)
15 2014/11/22 13:05:55 WARN - jmeter.config.CSVDataSet: Empty delimiter converted to ','
16 2014/11/22 13:05:55 INFO - jmeter.services.FileServer: Stored: /jmeter_test/data/subject_list.csv
17 2014/11/22 13:05:55 ERROR - jmeter.config.CSVDataSet: java.io.FileNotFoundException: /jmeter_test/data/subject_list.csv (그린 파일이거나...)
18 2014/11/22 13:05:55 WARN - jmeter.config.CSVDataSet: Empty delimiter converted to ','
19 2014/11/22 13:05:55 INFO - jmeter.services.FileServer: Stored: /jmeter_test/data/user_list.csv
20 2014/11/22 13:05:55 ERROR - jmeter.config.CSVDataSet: java.io.FileNotFoundException: /jmeter_test/data/user_list.csv (그린 파일이거나...)
21 2014/11/22 13:06:18 INFO - jmeter.threads.JMeterThread: Thread finished: Thread Group 1-1
22 2014/11/22 13:06:18 INFO - jmeter.engine.StandardJMeterEngine: Notifying test listeners of end of test
23 2014/11/22 13:06:18 INFO - jmeter.gui.util.JMeterMenuBar: setRunning(false, "local")
```

다른 활성화 방법은 [그림 8-9]와 같이 상단 툴 바 오른쪽 끝에 있는 숫자와 느낌표 경고 표시 부분을 클릭하는 것이다. 경고 표시 왼쪽의 숫자는 로그에 오류가 발생한 수를 나타낸다.

[그림 8-9] Log Viewer 활성 버튼



Log Viewer에서 보여지는 내용은 \${JMETER_HOME}\bin 디렉터리에 있는 jmeter.log 파일에 기록되는 내용으로, 다음 명령어로 확인할 수 있다.

```
# tail -f jmeter.log
```

분산 테스트를 할 때에는 jmeter controller와 jmeter-server 간의 오류만 출력되므로 jmeter controller에서는 이 오류만 볼 수 있고, jmeter-server와 서비스 서버 간 테스트 관련 오류는 각 jmeter-server의 로그 파일(jmeter-server.log)에 별도로 남는다.

[그림 8-10]은 로그 파일의 한 예로, CSV Data Set Config에서 CSV 파일의 경로가 잘못 설정되었거나 해당 경로에 CSV 파일을 옮겨놓지 않았을 때 발생하는 오류를 보여준다. CSV 파일 설정이 잘못되면 실제 HTTP Request Sampler 요청 및 결과에는 오류가 나타나지 않을 수 있으므로 오류 내용을 꼭 확인해야 한다.

[그림 8-10] jmeter-server.log 예제 – No such file or directory

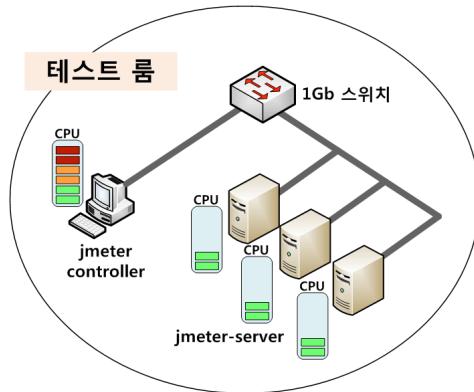
```
2014/11/22 13:36:38 INFO  - jmeter.threads.JMeterThread: Thread started: Thread Group 1-1
2014/11/22 13:36:38 WARN  - jmeter.config.CSVDataSet: Empty delimiter converted to ','
2014/11/22 13:36:38 INFO  - jmeter.services.FileServer: Stored: /jmeter_test/data/dept_list.csv
2014/11/22 13:36:38 ERROR - jmeter.config.CSVDataSet: java.io.FileNotFoundException: /jmeter_test/data/dept_list.csv (No such file or directory)
2014/11/22 13:36:38 WARN  - jmeter.config.CSVDataSet: Empty delimiter converted to ','
2014/11/22 13:36:38 INFO  - jmeter.services.FileServer: Stored: /jmeter_test/data/subject_list.csv
2014/11/22 13:36:38 ERROR - jmeter.config.CSVDataSet: java.io.FileNotFoundException: /jmeter_test/data/subject_list.csv (No such file or directory)
```

JMeter의 CPU 사용률

테스트 과정에서 꼭 확인할 부분은 jmeter controller와 jmeter-server의 CPU 사용률이다. top, iostat, vmstat과 같은 명령어로 간단히 확인할 수 있다.

[그림 8-11]과 같이 jmeter controller의 CPU 사용률이 매우 높을 때는 언제일까? 분산 테스트를 할 때 jmeter controller는 Test Plan 스크립트의 로직을 수행하는 것이 아니라 단순히 jmeter-server에서 전달되는 결과를 취합해서 화면에 보여주는 역할을 한다. 그러므로 CPU 사용률이 높다면 결과 취합 과정에서 어떤 문제가 발생한 것이다.

[그림 8-11] jmeter controller CPU의 과부하



이런 문제는 결과 취합 모드가 Standard일 때 자주 발생한다. Standard 모드일 때에는 테스트 결과를 실시간으로 전달받아서 바로 화면에 결과를 업데이트하므로 결과를 손쉽게 확인할 수 있지만, 여러 대의 jmeter-server의 결과를 한 대의 jmeter controller에서 취합하다 보니 용량이 부족할 때가 많다.

가장 쉬운 해결 방법은 결과 취합 모드를 Statistical 모드로 변경하는 것이다. Statistical 모드일 때에는 jmeter-server에서 결과를 취합한 후 평균과 합만

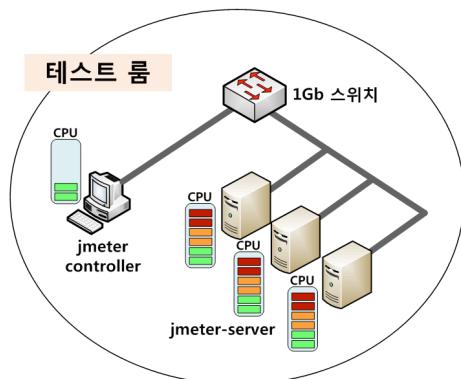
jmeter controller로 보내므로 jmeter controller의 부하가 눈에 띄게 줄어든다. 다만, 평균값과 합만 전달되어서 결과를 저장하는 로그 파일이 정상적으로 작성되지 않는 문제가 있다. 로그 파일을 정상적으로 작성하고 싶다면 Batch나 Stripped Batch 모드를 시도해 보는 것도 좋다.

JMeter의 결과 수집 모드는 \${JMETER_HOME}\bin\jmeter.properties 파일의 mode 변수에서 변경할 수 있다.

그럼 [그림 8-12]와 같이 jmeter-server의 CPU 사용률이 높을 때는 언제일까? jmeter-server는 실제로 Test Plan 스크립트의 로직이 실행되는 부분이므로 CPU를 많이 사용하는 로직이 스크립트에 포함되어 있다.

가장 쉬운 해결 방법은 jmeter-server를 증설하는 것이다. 서버 증설로 한 대에서 구동되는 가상 사용자의 수를 줄여주면 CPU 부하도 함께 줄어든다. 증설이 힘든 상황이라면 BeanShell이나 If Controller 등 CPU를 많이 사용하는 요소를 제거하고 Test Plan 스크립트를 재구성해야 한다.

[그림 8-12] jmeter-server CPU의 과부하

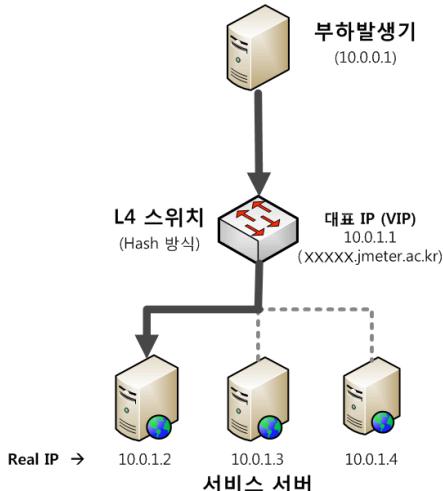


7.2.4 여러 대의 서비스 서버 중 한 대로 요청이 몰리고 있지는 않은가?

한 대의 서비스 서버로 테스트 요청이 몰리는 현상은 L4 스위치를 사용하는 환경에서 흔히 발생하는 문제다. [그림 8-13]은 `xxxxx.jmeter.ac.kr`에 HTTP Request를 요청할 때 요청을 수신한 L4 스위치가 세 대의 서비스 서버(리얼 서버) 중 한 대로 요청을 전달하는 모습을 보여준다.

대부분 L4의 Hash 방식은 요청 클라이언트의 IP를 기준으로 서비스 서버를 할당하므로 한 대의 부하발생기에 아무리 많은 가상 사용자가 생성되어 있어도 모두 하나의 서버로 몰릴 수밖에 없게 된다.

[그림 8-13] Hash 방식의 L4 구성



이 문제를 해결하는 방법은 세 가지다. 첫 번째는 서비스 서버보다 많은 수의 부하발생기(jmeter-server)를 준비하는 것이다. 서비스 서버 대비 2배 이상의 부하발생기를 설치하면 한쪽으로 요청이 몰리는 현상을 그리 걱정하지 않아도 된다.

두 번째는 부하발생기에 가상의 추가 IP를 설정하고 HTTP Request의 Source IP

를 가변으로 변경하는 것이다. 이때 여러 개의 IP를 준비할 수 있는 상황이어야 한다. 가장 먼저 부하발생기에 IP Alias를 생성한다. 관리자 권한의 아이디(root)로 다음 명령을 실행한다. 다수의 네트워크 인터페이스(NIC)가 설치된 PC라면 각 NIC에 서로 다른 IP를 할당하고 모든 NIC를 스위치에 연결하는 것도 하나의 방법이지만, 좀 더 복잡한 작업이므로 IP Alias를 생성하는 방법을 이용한다(eth0 부분에는 현재 사용 중인 인터페이스 이름을 넣는다).

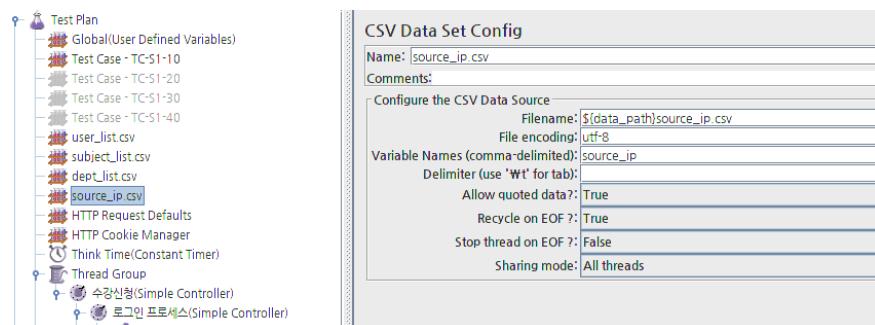
```
$ ifconfig eth0:1 10.0.0.2  
$ ifconfig eth0:2 10.0.0.3
```

그 다음으로 source_ip.csv 파일을 생성하고 다음 내용을 입력한다.

```
10.0.0.1  
10.0.0.2  
10.0.0.3
```

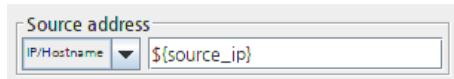
생성한 파일을 사용할 수 있도록 CSV Data Set Config를 추가하고 [그림 8-14]처럼 설정한다. 이때 데이터에 접근할 변수명은 source_ip로 설정한다.

[그림 8-14] source_ip.csv



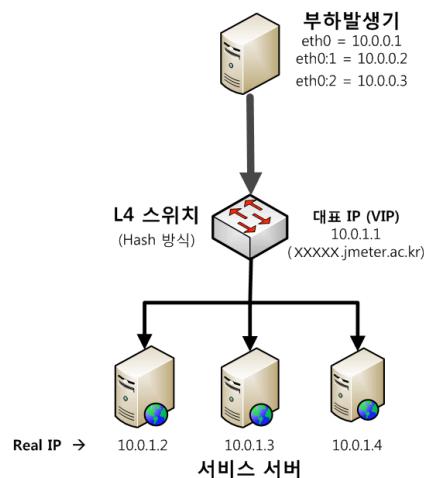
source_ip.csv가 추가되면 각 HTTP Request Sampler의 왼쪽 아래에 있는 Source address에 \${source_ip}를 값으로 입력한다.

[그림 8-15] HTTP Request의 Source address 설정



이렇게 하면 IP Alias로 L4에서 부하를 분산하여 테스트를 진행할 수 있다.

[그림 8-16] IP Alias를 이용한 Hash 방식 L4 문제 해결

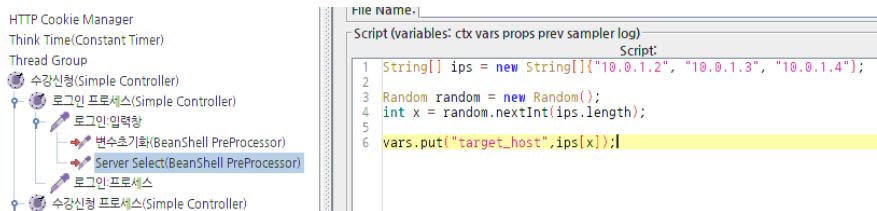


세 번째는 L4를 통하지 않고 직접 서비스 서버의 IP에 접근할 수 있다면 CSV Data Set Config나 BeanShell Sampler로 요청할 서비스 서버의 IP를 지속적으로 변경하는 방법이다.

변수초기화 PreProcessor 아래에 Server Select라는 BeanShell PreProcessor를 추가하고 [그림 8-17]과 같이 입력한다. target_host 변수는 Global 부분에서 sugang.jmeter.ac.kr로 설정된 변수로, BeanShell PreProcessor를 실행하면

target_host의 내용이 10.0.1.2, 10.0.1.3, 10.0.1.4 중 하나가 무작위로 선택되어 재정의된다.

[그림 8-17] Server Select



8.3 결과 수집

8.3.1 JMeter

로그 파일 저장 & 리로드

JMeter의 결과를 저장하는 가장 쉬운 방법은 Listener에 로그 파일을 지정해서 결과를 저장하는 것이다. [그림 7-35]처럼 Listener에 Filename을 입력하면 해당 파일에 결과가 저장된다(보통 .jtl이라는 확장자를 가진다). 테스트할 때마다 파일을 지우고 다시 쓰는 것이 아니라 파일 뒤에 내용이 계속 추가되는 형태이므로 모든 테스트 케이스에서 동일한 이름으로 설정해도 정보는 모두 저장된다. 대신 알아보기 힘들게 출력되므로 테스트 케이스마다 이름을 다르게 저장할 것을 추천한다.

한 번 저장된 로그 파일은 테스트가 끝난 후 Summary Report뿐 아니라 다른 Listener에서 열어도 정보를 확인할 수 있다. 메뉴에서 Browse.. 버튼을 누르고 저장된 jtl 파일을 열면 jtl 파일에 저장된 내용이 Listener에 알맞은 형태로 출력된다.

화면 저장

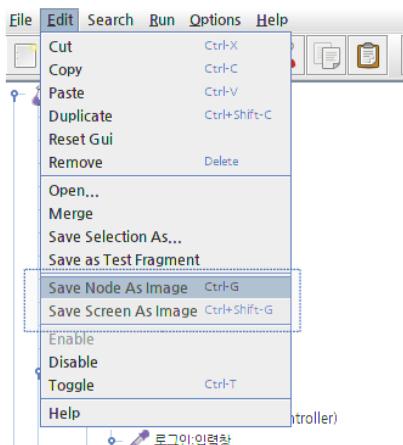
로그 파일만 저장해도 테스트 후에 대부분의 JMeter 정보를 다시 확인할 수 있다.

하지만 ‘분산 테스트 + Statistical’ 방식으로 정보를 수집한 경우에는 로그 파일로 테스트 당시와 동일한 결과를 리로드할 수 없다. 이때에는 테스트 과정에서 화면을 저장해 놓으면 결과 분석 및 리포트 작성 시 매우 유용하다.

JMeter는 기본적으로 전체화면(Screen)과 노드(Node)를 이미지로 저장하는 기능을 제공하는데, 노드는 각 Element의 정보가 출력되는 영역이다.

[그림 8-18]처럼 메뉴에서 ‘Edit → Save Screen As Image’, ‘Edit → Save Node As Image’를 선택하면 화면을 이미지로 저장할 수 있다. 단축키는 각각 ‘Ctrl-Shift-G’와 ‘Ctrl-G’고, PNG 형식으로 저장된다.

[그림 8-18] 화면/노드 저장



Summary Report와 같이 테이블 형태로 제공되는 정보는 CSV 파일로도 저장할 수 있다. [그림 8-19]처럼 Summary Report의 아래쪽에 Save Table Data 버튼이 있는데, 이 버튼을 클릭하면 Summary Report의 정보가 CSV 파일로 저장된다.

[그림 8-19] 테이블 데이터 저장

Summary Report

Name: Summary Report
Comments:

Write results to file / Read from file

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec
로그인_입력창	146	7	2	174	20.65	0.00%	6.3/sec	10.92
로그인_프로	139	6	2	208	19.97	0.00%	6.3/sec	10.87
수강신청_메	132	6	1	223	23.73	0.00%	6.3/sec	10.40
수강신청_수...	353	7	1	201	24.64	0.00%	17.7/sec	29.17
수강신청_장...	120	6	1	208	22.29	0.00%	6.3/sec	10.45
수강신청_학...	47	3	2	16	2.38	100.00%	2.7/sec	1.27
수강신청_장...	31	14	1	209	44.76	0.00%	1.7/sec	2.88
수강신청_변...	282	858	502	1303	214.79	0.00%	15.6/sec	25.84
수강신청_수...	88	526	202	926	213.20	0.00%	6.0/sec	9.91
수강신청_과...	35	3855	2203	5004	966.63	0.00%	1.7/sec	2.74
TOTAL	1373	313	1	5004	697.58	3.42%	49.2/sec	79.99

Include group name in label? Save Table Data Save Table Header

8.3.2 서비스 서버

JMeter-Plugin의 PerfMon 활용

JMeter에 기본으로 포함된 것은 아니지만, JMeter-Plugin에 포함된 PerfMon을 활용하면 쉽게 JMeter를 이용해서 서비스 서버의 성능 정보를 가져올 수 있다.

PerfMon의 ServerAgent를 모니터링하려는 서버에 설치하고 PerfMon Metrics Collector를 사용해서 그 정보를 수집하는 방식으로 동작한다(그림 8-20 참고).

ServerAgent를 설치하려면 JRE 1.4 이상이 설치되어 있어야 한다. JRE는 이미 설치되어 있다고 가정한다. <http://jmeter-plugins.org/downloads/all/>에서 ServerAgent-x.x.x.zip을 내려받는다. 모니터링하려는 서비스 서버에 이를 업로드하고 압축을 풀어주고 다음 명령을 실행하면 설치는 끝난다.

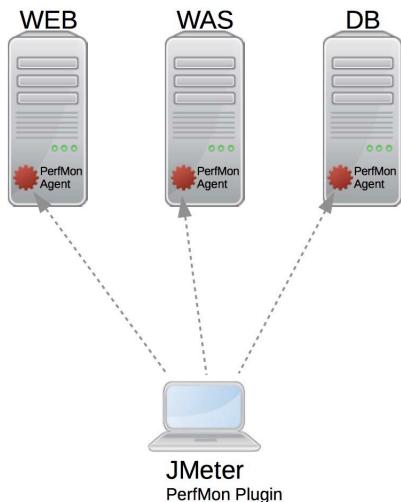
[Linux/Unix]

```
# ./startAgent.sh
```

[Windows]

```
# startAgent.bat
```

[그림 8-20] PerfMon의 동작 구조



사용하는 포트를 변경해서 실행하려면 다음과 같이 실행한다. UDP 포트를 0으로 설정하는 것은 UDP를 사용하지 않겠다는 의미다.

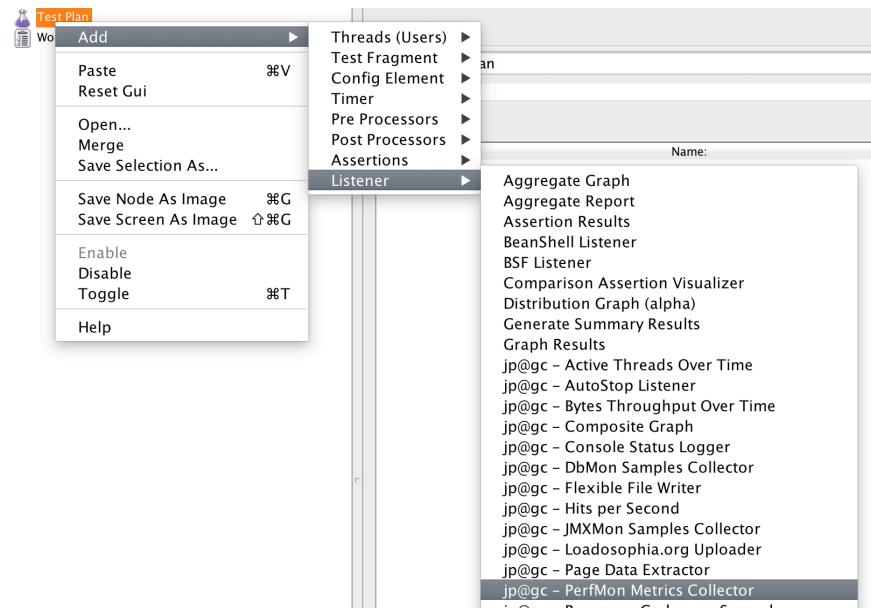
```
# ./startAgent.sh --udp-port 0 --tcp-port 4445
INFO    2014-10-06 01:59:05.470 [kg.apc.p] (): Binding TCP to 4445
INFO    2014-10-06 01:59:05.513 [kg.apc.p] (): JP@GC Agent v2.2.0 started
```

기타 옵션의 의미는 다음과 같다.

- **--udp-port** : UDP Listen 포트 설정(디폴트: 4444)
- **--tcp-port** : TCP Listen 포트 설정(디폴트: 4444)
- **--interval <seconds>** : 수집 주기
- **--auto-shutdown** : 테스트가 종료되면 Agent도 자동으로 종료된다.
- **--sysinfo** : 사용 가능한 시스템 옵션을 보여준다.

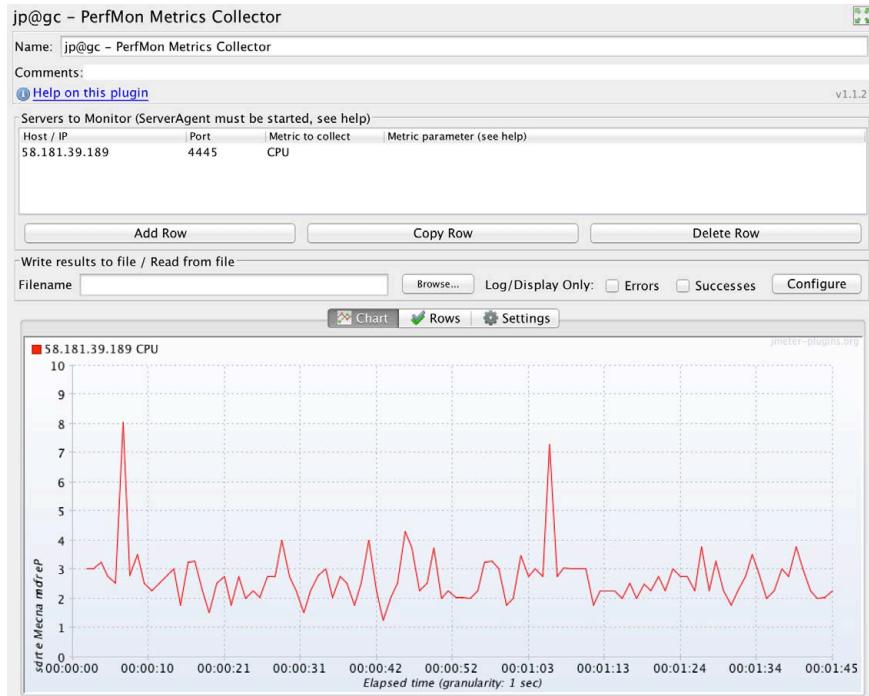
ServerAgent를 설치하고 구동시킨 다음 PerfMon Metrics Collector를 JMeter에 추가하고 정보를 수집하면 된다. PerfMon Metrics Collector는 [그림 8-21]과 같이 ‘Add → Listener → jp@gc - PerfMon Metrics Collector’를 선택하면 추가된다.

[그림 8-21] PerfMon Metrics Collector 추가



PerfMon Metrics Collector를 추가한 후에는 Add Row 버튼을 눌러서 Servers to Monitor 영역에 Row를 추가하고, ServerAgent가 설치된 서버의 IP와 포트를 설정하고 모니터링할 항목을 설정한다. 선택된 항목의 상세 설정은 Metric Parameter에서 한다. [그림 8-22]는 PerfMon로 CPU 정보를 수집한 결과다.

[그림 8-22] PerfMon Metrics Collector 화면



이런 방식의 모니터링에서 항상 주의할 점은 모니터링하는 정보를 수집하는 JMeter의 CPU 사용률이 높거나 네트워크 대역폭이 부족하거나 ServerAgent가 설치된 서비스 서비스의 CPU와 I/O 및 대역폭 사용률이 매우 높으면 모니터링 정보가 일정 시간 동안 누락될 수도 있다는 것이다. 이런 경우를 대비해서 서비스 서비스 내부에 시스템 정보를 간단한 스크립트를 이용해서 로그로 기록해 놓는 것이 좋다.

간단한 스크립트를 이용한 CPU 정보 수집

대상 서버가 Linux/Unix 계열이면 간단하게 스크립트로 정보를 저장할 수 있다. vmstat 나 iostat과 같은 시스템 정보 출력 도구는 서버 설치 시 기본으로 설치되므로 이를 이용한다. 다음은 vmstat의 정보를 가져와서 파일에 저장하는 간단한 스크립트다.

[get_vmstat.sh]

```
#!/bin/bash

## -----
## vmstat logger
##
## Usage : ./get_vmstat.sh [outfile]
## Options:
##         outfile   결과가 저장될 파일 (Default:/dev/null)
##
## @author jacojang<jacojang@jacojang.com>
## -----


OUTFILE=$1;shift;

if [ "x${OUTFILE}" = "x" ] ; then
    OUTFILE=/dev/null
fi

## Print Header
## -----
echo ",,procs,,memory,,,swap,,io,,system,,cpu,,," | tee -a ${OUTFILE}
echo "date,hour,r,b,swpd,free,buff,cache,si,so,bi,bo,in,cs,us,sy,id,wa,st" |
tee -a ${OUTFILE}
```

```
## Print Data
## -----
vmstat 2 | grep --line-buffered [0-9] | while read p_r p_b m_swpd m_free m_buff
m_cache s_si s_so i_bi i_bo s_in s_cs c_us c_sy c_id c_wa c_st
do
    ctime=`date +%Y%m%d,%H%M%S`
    echo
${ctime},${p_r},${p_b},${m_swpd},${m_free},${m_buff},${m_cache},${s_si},${s_so},
${i_bi},${i_bo},${s_in},${s_cs},${c_us},${c_sy},${c_id},${c_wa},${c_st} |
tee -a ${OUTFILE}
done
```

다음을 실행하면 같은 명령어로 각 서버의 정보를 hostname별로 저장할 수 있다.

```
# ./get_vmstat.sh `hostname`_cpu.log
```

APM을 이용한 정보 수집

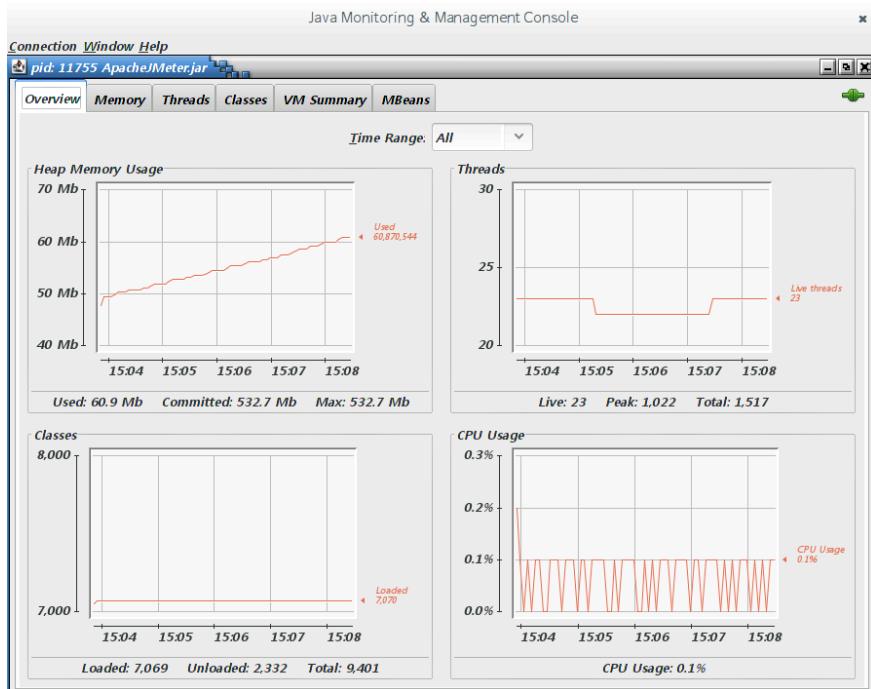
일정 규모 이상의 서비스 업체라면 한 개 이상의 애플리케이션 모니터링 프로그램이 있을 것이다. APM⁰¹은 국내에는 제니퍼(JenniferSoft)나 Sysmaster(TMAX), InterMax(EXAM) 등의 제품이 있고, 해외에서는 AppDynamics, Compuware 등의 회사 제품이 유명하다. 일반적인 APM은 WAS의 성능뿐 아니라 CPU, 네트워크 같은 시스템 정보도 함께 제공한다. 하지만 통계 정보가 최소 1분 단위로 저장되는 경우가 있어서 실제 테스트 자료로는 변별력이 부족할 수 있으므로 통계 정보의 최소 단위가 어떻게 되는지 꼭 미리 확인해야 한다.

APM과 같은 별도의 애플리케이션이 없다면 자바 설치 시 기본으로 설치되는 jconsole을 이용하는 것도 한 방법이다. JVM 정보만 출력되긴 하지만 테스트 과

01 · Application Performance Management

정에서 생길 수 있는 문제에 대한 모니터링과 정보 수집에 유용하다.

[그림 8-23] jconsole



SMS와 NMS를 이용한 정보 수집

SMS⁰²와 NMS⁰³가 구성되어 있다면 이를 이용해서 정보를 수집하는 것도 한 방법이다. 이 또한 APM과 마찬가지로 통계 정보 수집의 최소 시간 단위가 어떻게 되는지 확인해야 한다.

02 · System Management System

03 · Network Management System

8.3.3 CPU 중 한 개의 Core에만 부하가 집중되는가?

APM이나 NMS, SMS 등에는 서비스 서버의 CPU(or Core)별 사용량이 나오지 않는 경우가 많다. 한 CPU(or Core)에만 부하가 집중되면 전체 CPU 사용량은 낮지만 처리량이나 응답시간이 비정상적으로 변화할 수 있으므로 하나의 CPU에 부하가 집중되지는 않는지 확인해야 한다.

나중에 보고서 작성이나 튜닝을 제안할 때 근거 자료가 되므로 이러한 부하 편중 현상이 의심된다면 꼭 해당 정보를 수집한다. 부하 편중을 확인하는 가장 쉬운 방법은 top, nmon, glance, topas, mpstat 등의 명령어를 사용하는 것이다.

[그림 8-24]는 top 명령어를 실행한 후 1 버튼을 누른 모습이다. 1 버튼은 CPU 전체 평균을 출력할지 Core별 정보를 출력할지에 대한 토글 명령이다. 여기에서는 다음 두 부분을 주목해야 한다. Cpu0은 이를 100.0%를 사용해서 idle(id)| 0.0% 상태라는 것과 java라는 프로세스가 100.0% 상태로 실행(R) 중이라는 것이다.

이는 실제로 전체 CPU 사용률은 25%정도지만, java라는 프로세스는 한 개의 Core를 100% 사용하고 있어서 더는 성능을 내지 못하는 상황이다. vmstat로만 정보를 수집하면 이런 부분이 누락될 수 있으니 테스트 과정에서 틈틈이 CPU별 정보를 확인하고 수집해야 한다.

[그림 8-24] top 실행화면

```
top - 15:27:06 up 29 days, 19:09, 4 users, load average: 1.03, 0.93, 0.51
Tasks: 246 total, 2 running, 240 sleeping, 4 stopped, 0 zombie
Cpu(s): 100.0us, 0.0sy, 0.0ni, 0.0id, 0.0wa, 0.0hi, 0.0si, 0.0st
CPU0 : 0.0us, 0.0sy, 0.0ni, 100.0id, 0.0wa, 0.0hi, 0.0si, 0.0st
CPU2 : 0.3us, 0.3sy, 0.0ni, 99.3id, 0.0wa, 0.0hi, 0.0si, 0.0st
CPU3 : 0.0us, 0.0sy, 0.0ni, 100.0id, 0.0wa, 0.0hi, 0.0si, 0.0st
Mem: 3909988k total, 3020204k used, 889784k free, 832240k buffers
Swap: 4046840k total, 704k used, 4046136k free, 1610664k cached

```

PID	USER	PR	Nl	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
25413	root	20	0	3920	332	264	R	100.0	0.0	9:04.51	java
8412	root	20	0	97200	4000	2932	S	0.7	0.1	0:00:02	sshd
8410	root	20	0	97200	4000	2932	S	0.3	0.1	0:00:03	sshd
8496	root	20	0	96948	3844	2912	S	0.3	0.1	0:00:01	sshd
8498	root	20	0	96948	3844	2912	S	0.3	0.1	0:00:01	sshd
12801	root	20	0	2940m	147m	12m	S	0.3	3.9	0:08.50	java
1	root	20	0	19384	1112	880	S	0.0	0.0	0:16.01	init

9 | 결과 분석 및 리포트 작성

9.1 결과 분석

성능 테스트에서 가장 중요한 부분은 결과 분석이라고 생각한다. 같은 결과 그래프나 표를 가지고도 분석하는 사람에 따라 서로 다른 결과를 도출해 내는 경우도 많으므로 항상 가장 많은 고민을 해야 하는 부분이다. 여기에서는 결과 분석을 하는 일반적인 방법과 해당 지표를 JMeter에서 확인하는 방법을 살펴본다.

9.1.1 일반적인 가상 사용자 증가에 따른 성능 지표 변화

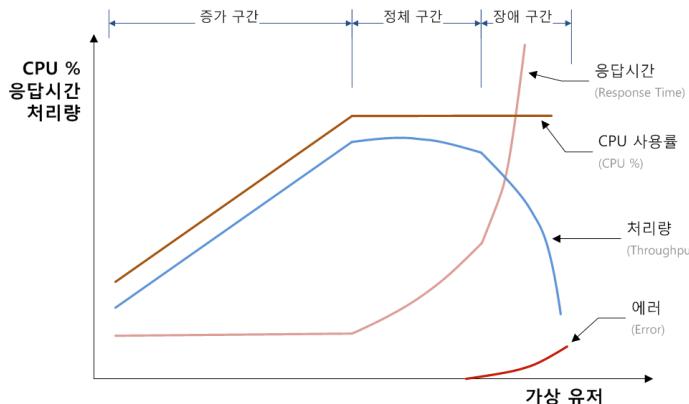
성능 테스트에서 가장 중요시하는 성능 지표는 ‘응답시간’, ‘처리량’, ‘CPU 사용률’이다. 그러므로 가상 사용자의 증가에 따른 이런 성능 지표의 변화는 매우 중요한 정보다.

일반적인 서비스 환경에서 기타 외부 변수가 없다고 가정했을 때 가상 사용자를 증가시키면서 테스트를 진행하면 [그림 9-1]과 같은 성능 지표의 변화를 보일 것이다. 이는 크게 세 구간으로 나누어 볼 수 있다.

- **증가 구간** : 가상 사용자가 증가함에 따라 CPU 사용률과 처리량은 선형적으로 증가하고 응답시간은 크게 변화하지 않는 구간을 의미한다. 정상적인 서비스가 가능한 영역으로, 성능 테스트나 튜닝의 목적은 실제 서비스 환경에서 서비스 서버의 상태가 이 구간에 위치하게 하는 것이다.
- **정체 구간** : 가상 사용자가 증가해도 더는 처리량이나 CPU 사용률이 증가하지 않고 응답시간만 증가하는 구간을 말한다. 이 영역에서도 비교적 정상적인 서비스가 가능하긴 하지만, 언제 장애 구간으로 넘어갈지 모르는 상황이므로 주의가 필요하다.

- 장애 구간 : 가상 사용자가 증가함에 따라 오류가 발생하기 시작하고, 응답시간이 급격하게 증가하며 처리량은 급격하게 떨어지는 구간으로, 정상적인 서비스가 불가능한 상태를 말한다.

[그림 9-1] 가상 사용자 증가에 따른 성능지표 변화



9.1.2 JMeter에서 가상 사용자 증가에 따른 성능 지표 확인

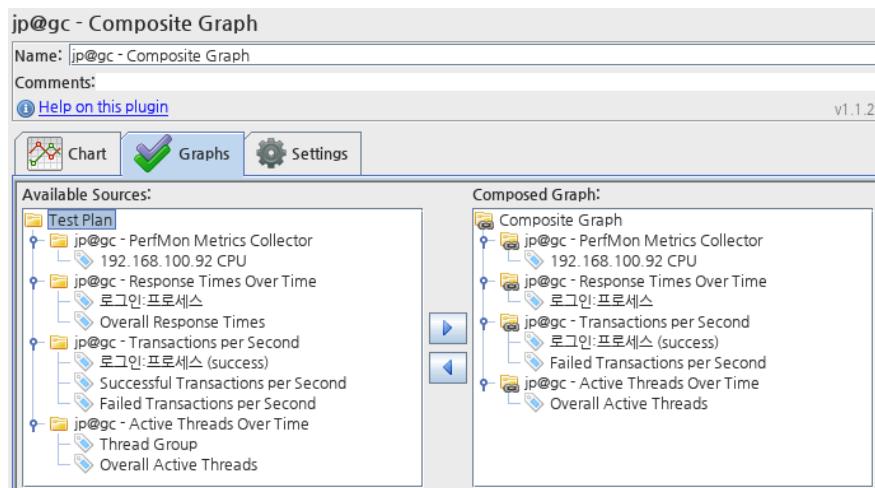
JMeter에서 가상 사용자 증가에 따른 성능 지표를 확인하는 가장 좋은 방법은 jp@gc - Composite Graph로 jp@gc - Active Threads Over Time, jp@gc - Transactions per Second, jp@gc - Response Times Over Time, jp@gc - PerfMon Metrics Collector 정보를 한 그래프 안에서 확인하는 것이다. Composite Graph를 이용하려면 [그림 9-2]와 같이 Listener를 추가한다.

[그림 9-2] Composite Graph 설정을 위한 Listener 추가



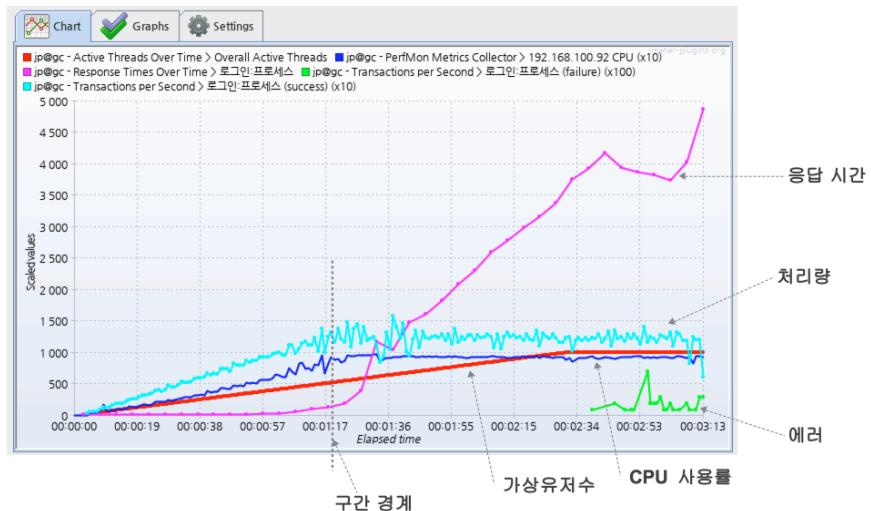
Listener를 추가한 후 Composite Graph를 설정해야 하는데, 이는 Composite Graph에서 어떤 값을 보여줄지 설정하는 것이다. jp@gc - Composite Graph 를 추가한 후에 해당 Listener의 Graphs 템을 클릭해봐도 Available Sources에 아무것도 없다. Available Sources는 Test Plan 스크립트를 한 번 구동해야만 확인할 수 있다. Test Plan 스크립트를 실행하면 [그림 9-3]처럼 Available Sources 부분에 Listener 정보가 추가된다. 그 다음 Composed Graph에 추가하고 싶은 항목을 선택하고 오른쪽 방향 삼각형을 눌러주면 항목이 추가된다. [그림 9-3]의 Composed Graph 부분에서 추가된 모습을 볼 수 있다.

[그림 9-3] Composite Graph 설정



[그림 9-4]는 결과 그래프 화면이다. [그림 9-1]과 유사한 모양을 나타내는 것을 볼 수 있다. x축이 가상 사용자 숫자가 아닌 테스트 시간이긴 하지만 Active Threads Over Time 정보를 그래프에 출력했으므로 쉽게 비교 평가할 수 있다.

[그림 9-4] Composite Graph



이 그래프를 분석해 보면 가상 사용자 500명 정도까지는 응답시간이 일정하게 유지되다가 500명을 지나면서 응답시간이 증가한다. 이 타이밍은 CPU 사용률이 100%에 근접한 지점과 유사하다. 이후 처리량은 증가하지 않고 응답시간만 증가하는 정체 구간으로 접어들었다가 가상 사용자가 1000명에 근접하면서부터 오류가 발생한다. 이는 가상 사용자 500명이 증가 구간과 정체 구간의 경계라고 볼 수 있다.

이 테스트 시나리오를 기반으로 시스템을 분석하면 가상 사용자는 500명 이하까지 정상 서비스가 가능하고 최대 처리량은 약 120tps다. 실제 서비스에서 사용자 수가 500명보다 크거나 처리량이 120tps 이상 필요하다면 장비 증설이나 튜닝이 필요하다고 볼 수 있다.

9.1.3 표준편차

Summary Report에는 응답시간에 대한 표준편차 Standard Deviation 정보를 제공한

다. 이 수치는 결과 분석에 유용하게 쓰인다. 단위는 1/1000초다. 같은 평균 응답 시간이라 해도 표준편차의 값이 크다면 응답시간이 좀 더 들쭉날쭉하다는 의미다. 응답시간이 들쭉날쭉하다는 것은 시스템이 불안정하거나 외부 요인에 의한 영향을 많이 받는다고 볼 수 있다. 이는 정상적인 상태라고 보기 힘들다. 즉, 표준편차가 큰 경우 해당 시스템이나 응용 프로그램을 반드시 점검해 봄야 한다.

그런데 표준편차가 크다는 것은 어느 정도 수치를 말하는 것일까? 절대적인 수치도 중요하지만 상대적인 수치가 좀 더 중요하다고 볼 수 있다. 예를 들어, 10개의 Sampler 중에서 9개는 표준편차가 10이하인데, 나머지 하나만 500을 나타낸다면 의심을 해봐야 한다. 평균 응답시간이 1초일 때 표준편차가 500인 것과 평균 응답시간이 10초일 때 표준편차가 500인 것은 좀 다른 의미로 분석될 수 있다. 후자는 응답시간이 10초이므로 500ms 정도의 편차는 큰 의미가 없다.

[그림 9-5]는 평균 응답시간과 처리량은 같고 표준편차만 차이가 많이 나는 Summary Report의 결과다. 표준편차가 크다는 것은 평균 응답시간을 기준으로 응답시간이 크게 요동치고 있다는 의미다.

[그림 9-5] 표준편차만 다른 Summary Report 결과

Label	# Samples	Average	Min	Max	Std. Dev	Error %	Throughput	KB/sec	Avg. Bytes
2	665	4543	2017	6999	1431.23	0.00%	2.2/sec	0.24	114.0
1	670	4516	4006	4998	279.55	0.00%	2.2/sec	0.25	114.0

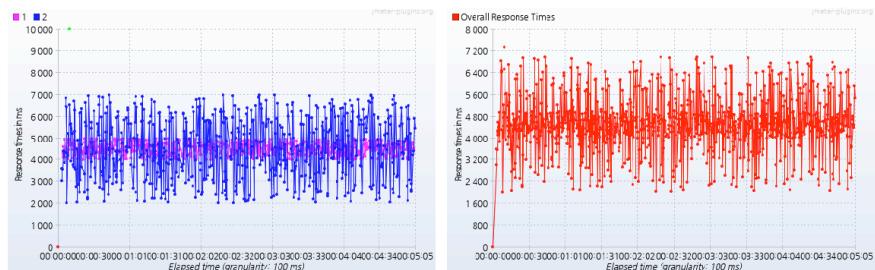
평균 응답 시간은 거의 동일하나.
표준 편차는 1초 이상 차이가 난다.

Label 1은 표준편차가 작고 Label 2는 표준편차가 크다. Label 1과 Label 2의 응답시간을 [그림 9-6]처럼 하나의 그래프로 합쳐서 보면 어떤 차이가 있는지 한눈에 들어오지 않는다.

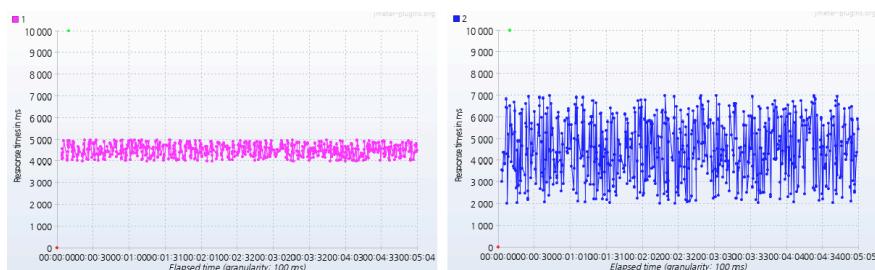
하지만 Label 1과 Label 2를 각각 따로 놓고 보면 확실하게 구분할 수가 있다. [그림

9-7]의 왼쪽 그래프는 Label 1에 대한 그래프로 응답시간 변화폭이 오른쪽 Label 2 보다 적다.

[그림 9-6] 표준편차 관련 응답시간 그래프(통합)



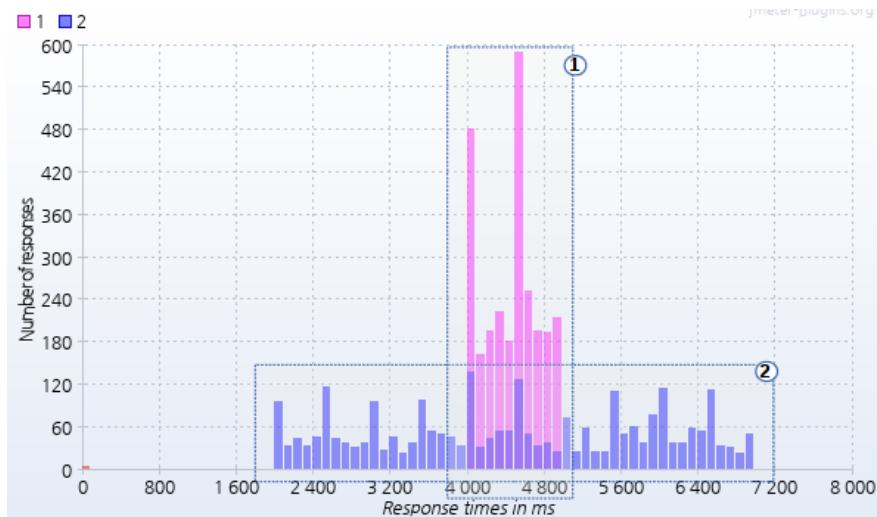
[그림 9-7] 표준편차 관련 응답시간 그래프(개별)



'add → Listener → jp@gc - Response Times Distribution' 그래프를 선택하면 응답시간 분포를 가장 확실하게 알 수 있다. 응답시간이 어느 범위에 집중되고 있는지를 쉽게 확인할 수 있다. [그림 9-8]에서 ①은 가운데 집중되어 있고, ②는 넓게 펴져있는 것을 볼 수 있다.

②는 동일한 평균 응답시간과 처리량을 보이지만 ①보다 불안정하고 외부 요인이나 알 수 없는 요인에 의해 영향을 받고 있다고 판단할 수 있으므로 좀 더 안 좋은 상황이라고 할 수 있다. 이는 튜닝이나 원인 분석이 필요하다.

[그림 9-8] jp@gc – Response Times Distribution 그래프



9.1.4 분석 예제

분석 예제 1

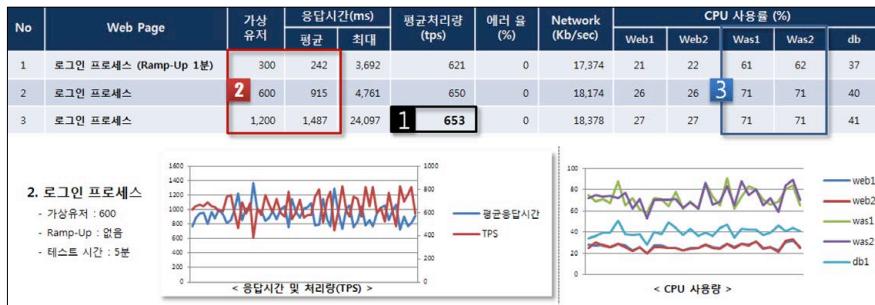
[그림 9-9]는 가상 사용자를 증가시키면서 진행한 실제 테스트 결과표다. Think Time 없이 진행한 테스트로, 서비스 시스템의 최대 처리량을 찾기 위해 Think Time 없는 테스트를 진행하곤 한다.

그림의 1과 2 부분에서 볼 수 있듯이 이 시스템의 최대 처리량은 653tps고, 가상 사용자가 증가함에 따라 응답시간이 선형적으로 변하는 정체 구간에 해당하는 단계라고 볼 수 있다.

이 표에서 주목할 요소는 서비스 서버의 CPU 사용률이다. 그림의 3 부분을 보면 WAS가 가장 높은 CPU 사용률을 나타낸다. 이는 평균처리량이 WAS의 성능 때문에 더는 증가하지 못한다고 볼 수 있다. 하지만 CPU를 100%까지 활용하지 못하

는 이유는 더 확인해 봄야 할 부분이다. 예를 들어, WAS에 설치된 CPU와 메모리 크기보다 서비스 컨테이너가 작게 설치되었다면 서비스 컨테이너를 추가하는 것을 고려해 봄야 한다.

[그림 9-9] 결과표 1



분석 예제 2

[그림 9-10]은 ‘WEB → WAS → DB’로 진행하는 각 서비스 구간을 분리해서 테스트한 결과표다. 그림에서 A 부분은 IPS 기능이 추가된 방화벽 유무에 따른 테스트 결과의 차이를 보여주는 것이다. 그림의 B 부분은 같은 내용의 html 파일과 jsp 파일의 성능 차이를 나타낸다. 성능이 처리량 기준으로 13배 가까이 차이가 나는 것을 볼 수 있다. 이것은 html 페이지로 서비스하는 것이 성능 향상에 더 도움이 된다는 것을 의미한다. 어찌 보면 당연한 결과지만 개발 편의성 때문에 그냥 jsp로 처리하는 경우도 많다. 그림의 C 부분에서는 처리량이 일정한 패턴을 보이지 않고 틀쭉날쭉한 것을 볼 수 있다. 이는 서비스에 영향을 미치는 부분을 다시 분석하고 테스트해야 하는 상황이다. 서비스 페이지의 로직에서 메모리가 비정상적으로 쌓여서 빈번하게 GC^{Garbage Collection}가 일어난다거나 테스트 과정에서 WAS 쪽에 간접이 발생하는 등의 요인이 있을 수 있다.

[그림 9-10] 결과표 2

No	항목	Test Case						결과						
		Web Page		Thread 수	대상 서버				응답 시간 (ms)	응답 처리량 (n/sec)	응답 Error량 (n/sec)	네트워크 사용량 (Kbit/sec)		
		File	Size (Byte)		Web 64F	WAS 24F	DB 14F	NF 14F				Web	WAS	DB
1	방화벽 WEB WAS	Khi_test.html [firewallon]	2832	10	✓				5,366	A 163	14	53	2	
2		Khi_test.html [firewalloff]	2832	10	✓				23	21,234	3	469,802	31	
3		Khi_test.html [direct]	2832	10	✓				24	B 20,159	0	446,017	92	
4		Index.jsp	2830	10	✓				665		0	33,236	7	51
5	DB	Main.jsp	37659	10	✓	✓			1,094	418	0	122,683	5	57
6		Main.jsp	37659	100	✓	✓			4,033	876	0	257,569	7	89
7		200912111401_psssjsp	1302	10	✓	✓	✓		7,582	C 77	0	3,044	2	47
8		200912111401_psssjsp	1302	40	✓	✓	✓		4,735	194	0	7,857	3	71
9		200912111401_psssjsp	1302	50	✓	✓	✓		102,275	D 8	0	210	7	75
10		200912111401_psssjsp	1302	100	✓	✓	✓		5,758	260	0	10,526	3	84
11		200912111401_psssjsp	1302	150	✓	✓	✓		9,682	E 213	0	8,561	3	67

9.2 리포트 작성

성능 테스트의 마지막 단계는 수집된 결과를 가지고 결과 리포트를 작성하는 것이다. 결과 리포트는 단순히 결과를 나열하는 수준에서 머물기보다는 결과 분석을 통해서 성능이 만족할 만한 수준인지 만족할 만한 수준이 아니라면 어느 부분에 문제가 있는지와 이 문제를 개선하려면 어떻게 해야 하는지를 포함하는 것이 좋다.

9.2.1 요약 정리

요약 정리는 엔지니어를 위한 내용이라기보다는 상사에게 보고 하는 용도라고 보면 되는데, 테스트 목적에 맞는 결과값을 한 장으로 짧게 정리하는 것을 말한다. 예를 들어, 수강신청 인원별 테스트 결과로는 “1,000명 이상의 사용자가 동시에 사용하면 응답시간이 급격히 증가하므로 현재 시스템에서는 동시 사용자를 1,000명 이하로 서비스해야 한다”와 같이 간략하게 요약한 내용을 적는다.

9.2.2 결과 자료 정리 및 분석

테스트 케이스별 또는 테스트 시나리오별 결과를 표와 그래프를 이용해서 나타내고 해당 결과가 의미하는 바가 무엇인지를 설명한다. 결과 리포트의 포맷 및 내용은 성능 테스트의 목적이나 결과 리포트의 사용 목적에 따라 그 형태가 달라지므로 목적에 맞게 표현해 준다. [그림 9-11]은 시나리오를 요약한 결과 페이지 샘플로, 목표한 결과를 간단하게 표현한다.

[그림 9-11] 결과 리포트

시나리오	테스트 케이스	가상 유저	응답시간(ms) (가장유저 최대값)	평균처리량 (tps)	에러율 (%)	Network (Kb/sec)	CPU 사용률 (%)	
							WAS	DB
수강신청	TC-S1-10	1,000	3,027	180	0	22,701	92.6	51.6
	TC-S1-20	2,000	8,015	172	2	21,790	91.1	50.8
	TC-S1-30	3,000	19,118	110	8	18,822	96.2	40.0
	TC-S1-40	4,000	23,283	90	25	12,840	97.3	30.6

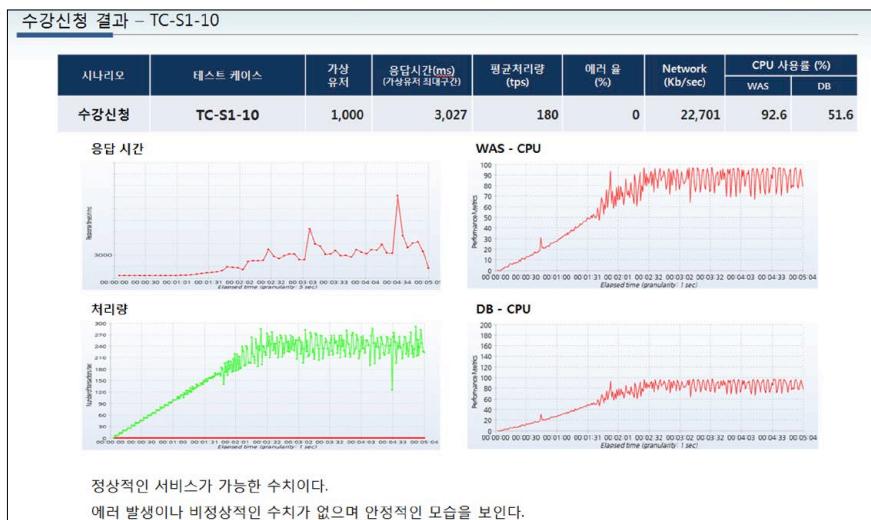
처리량 **180 tps**
응답시간 **3,024 ms**
에러율 **0 %**



한 학년이 1,000명이므로 수강신청은 한 학년씩 진행하는 것이 가장 좋다.

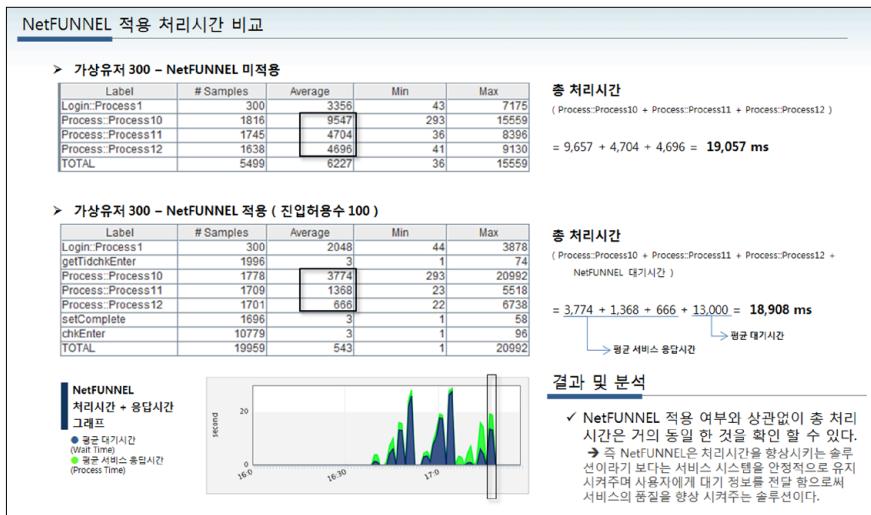
앞 장에서 요약했다면 [그림 9-12]처럼 테스트 케이스별로 좀 더 자세한 자료를 첨부한다.

[그림 9-12] 결과 리포트 - 테스트 케이스



[그림 9-13]은 NetFUNNEL이라는 솔루션을 적용했을 때와 그렇지 않았을 때를 비교한 테스트 결과 리포트의 일부분이다. 비교 대상의 차이점이나 공통점을 부각하여 표현해 주는 것이 좋다.

[그림 9-13] 결과 리포트 – 비교



9.2.3 개선방향 제시

결과 리포트의 마지막에는 가능하다면 개선방향에 관한 내용을 포함하는 것이 좋다. 테스트의 결과 자료를 기반으로 어떤 부분을 어떻게 수정 또는 보완하면 원하는 결과나 좀 더 좋은 결과가 나올 수 있는지를 기술한다. 몇 가지 예로 그 작성법을 살펴보자.

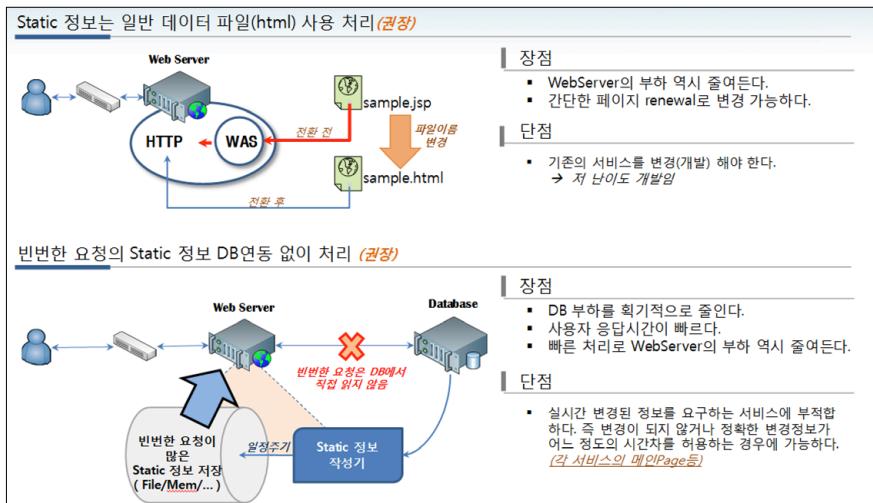
정적(Static) 페이지 활용

사이트의 메인 페이지가 무거우면 전반적으로 서비스가 느려지는 경우가 많다. 메인 페이지가 무거워지는 가장 흔한 이유는 메인 페이지에서 너무 많은 정보를 한

번에 보여주려 하기 때문이다. 이 때문에 많은 DB 쿼리가 발생하고 부하가 일어난다. 하지만 메인 페이지의 내용은 개인 정보보다는 공지사항 등의 정보가 대부분이므로 메인 페이지를 로딩하기 위해 빈번하게 DB에 접속하는 것은 매우 비효율적인 방법이다.

보통 대형 서비스 업체에서는 메인 페이지를 주기적으로 html 파일로 변환해서 이 html 페이지를 활용한다. 이 방법을 이용하면 DB 접속량을 매우 많이 줄일 수 있다.

[그림 9-14] 정적 페이지 활용

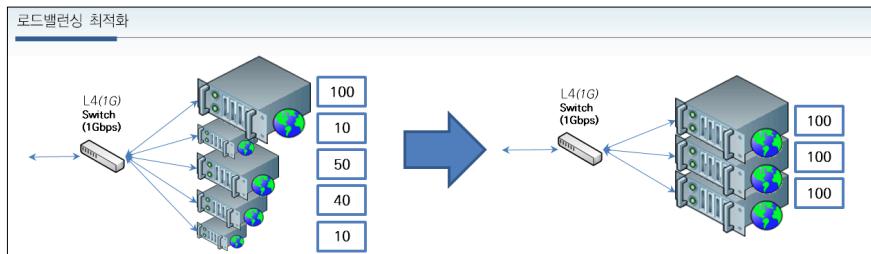


로드밸런싱 최적화

보통 여러 대의 웹 서버로 서비스할 때 L4 스위치를 통해 Hash 방식으로 서비스한다. 웹 서버의 숫자는 많지만 성능 차이가 많이 나는 경우와 웹 서버의 숫자는 다소 적지만 성능 차이가 나지 않는 경우가 있다면 어떤 방식이 더 좋을까?

전체적인 서비스 품질을 위해서는 후자가 훨씬 유리하다. 전자는 성능이 떨어지는 서버가 서비스의 병목 역할을 하게 되고 오류를 발생시킬 가능성성이 높다.

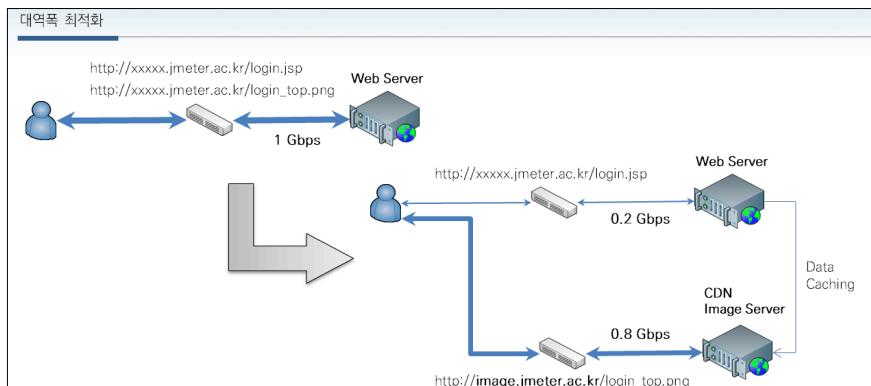
[그림 9-15] 로드밸런싱 최적화



대역폭 최적화

이미지나 동영상 등 대역폭을 많이 사용하는 서비스는 대역폭 문제로 서비스가 원활히 이루어지지 않을 가능성이 높다. 지속적으로 이러한 서비스를 해야 한다면 네트워크 대역폭을 늘려주는 것이 좋지만, 짧은 기간 동안 단발 이벤트 형식으로 진행하는 서비스라면 네트워크 회선을 증설하는 것은 가격면에서 효과적이지 못하고 확보한 대역폭보다 더 많은 사용자가 서비스를 이용할 경우 네트워크 대역폭을 늘린 효과가 없으므로 효과적이지 못하다. 이런 경우에는 CDN과 같은 외부 서비스를 받는 것이 좋다. 단, 외부에 유출되면 안 되는 보안 데이터를 서비스하는 경우에는 조심해야 한다.

[그림 9-16] 대역폭 최적화



부록 | 톰캣 설치와 배포

톰캣 설치 방법과 WAR 파일 배포 방법을 간단히 소개한다.

Windows 환경

홈페이지⁰¹에서 최신 버전의 톰캣 서버를 내려받는다. 다운로드 페이지⁰²에서 ‘Core → 32-bit/64-bit Windows Service Installer’를 선택하고 내려받는다. 파일을 실행하여 서비스 포트와 User Name, Password를 설정한다. 서비스 포트는 디폴트로 8080인데, 필요 시 수정한다. User Name과 Password는 설치 후에 WAR 파일을 배포할 때 필요하므로 꼭 입력한다.

다음으로 JRE 위치와 톰캣이 설치될 위치를 설정한다. JRE는 별로도 지정된 위치가 있다면 경로를 변경한다. 톰캣은 디폴트로 ‘C:\Program Files\Apache Software Foundation\Tomcat 8.0’에 설치되는데, 값을 수정하여 경로를 변경 할 수 있다. 정상적으로 설치되면 ‘Run Apache Tomcat’을 체크해서 바로 실행 한다.

Linux/Unix 환경

Linux/Unix에서는 별도의 설치 과정이 없다. 다운로드 페이지⁰³에서 ‘Core → tar.gz’를 내려받아서 실행하면 된다. 여기서는 \opt\apache-tomcat-8.x.x에 설치한다고 가정한다. 내려받은 apache-tomcat-8.x.x.tar.gz 파일을 설치하려는 위치로 이동하고 압축을 푼다.

01 : <http://tomcat.apache.org>

02 : <http://tomcat.apache.org/download-80.cgi>

03 : <http://tomcat.apache.org/download-80.cgi>

~/.bashrc 파일에 필요한 환경변수를 다음과 같이 설정한 후 '. ~/.bashrc'로 변경된 환경변수를 적용한다.

```
export CATALINA_HOME=/opt/apache-tomcat-8.x.x  
export JAVA_HOME=/usr/lib/jvm/default-java
```

[표 A-1] 환경변수 설정

환경변수	설명
CATALINA_HOME	내려받은 톰캣 서버 binary 배포 파일의 최상위 디렉터리를 의미한다.
JAVA_HOME	자바 런타임 환경(Java Runtime Environment)의 최상위 Root 디렉터리의 위치를 의미한다.

관리자 페이지 Manager Page에 접근하기 위한 계정 정보를 입력한다. \opt\apache-tomcat-8.x.x\conf\tomcat-users.xml에 다음 내용을 추가한다. user name과 password는 원하는 값으로 입력한다.

```
<tomcat-users>  
<user name="jacobjang" password="*****" roles="manager-gui" />  
...  
</tomcat-users>
```

설정이 끝나면 톰캣을 실행한다.

[Tomcat 실행]

```
/opt/apache-tomcat-8.x.x/bin/start.sh
```

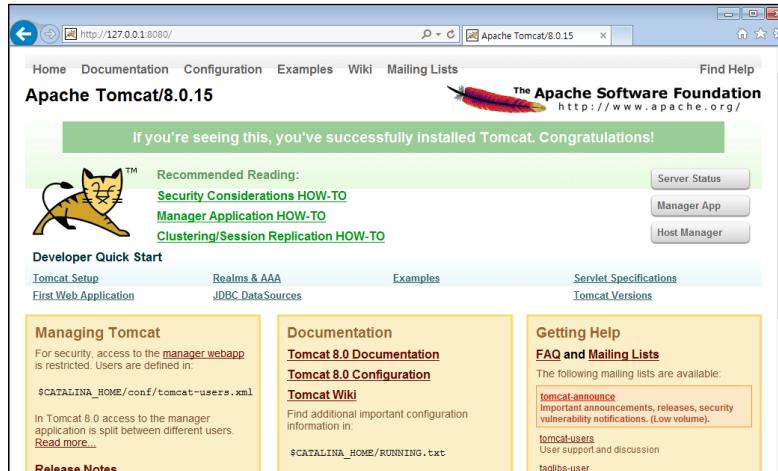
```
Using CATALINA_BASE:   /opt/apache-tomcat-8.x.x  
Using CATALINA_HOME:  /opt/apache-tomcat-8.x.x  
Using CATALINA_TMPDIR: /opt/apache-tomcat-8.x.x/temp
```

```
Using JRE_HOME:      /usr/lib/jvm/default-java
Using CLASSPATH:
/opt/apache-tomcat-8.x.x/bin/bootstrap.jar:/opt/apache-tomcat-8.x.x/bin/
tomcat-juli.jar
Tomcat started.
```

Test WAR 파일 배포

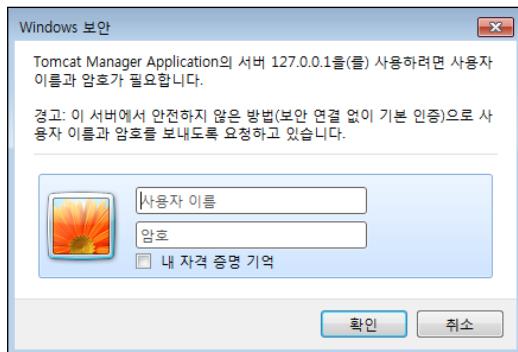
톰캣이 모두 설치되면 <http://127.0.0.1:8080>으로 접속해 보자. 다음과 같은 페이지가 보일 것이다.

[그림 A-1] Tomcat 설치 화면



WAR 파일을 배포하려면 우선 Manager App 버튼을 눌러서 관리자 페이지로 이동한다. 아이디와 암호를 입력하는 창이 뜨면 설치 시 설정했던 User Name과 Password를 입력한다.

[그림 A-2] 관리자 페이지 로그인 화면



User Name과 Password를 입력하고 확인을 누르면 다음과 같은 관리자 페이지 화면을 볼 수 있다.

[그림 A-3] 관리자 화면

http://127.0.0.1:8080/manager/html /manager

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Deploy
Deploy directory or WAR file located on server

Context Path (required):
XML Configuration file URL:
WAR or Directory URL:
Deploy

WAR file to deploy

Select WAR file to upload Deploy

Diagnostics

페이지 중간에 WAR file to deploy 부분이 있다. 앞서 내려받은 WAR 파일을 찾아서 선택하고 Deploy 버튼을 누르면 작업이 끝난다.

정상적으로 배포되었는지 확인하려면 웹 브라우저 주소창에 <http://127.0.0.1:8080/jmeter/search/index.html>을 입력한다. 정상적으로 배포되면 다음과 같은 화면이 나온다.

[그림 A-4] 정상 설치 시 웹 브라우저 화면



한빛 리얼타임

한빛 리얼타임은 IT 개발자를 위한 전자책입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.

500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 접할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. 무료로 업데이트되는 전자책 전용 서비스입니다.

종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 베전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위해 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 편리한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가겠습니다.

지금 보시는 전자책에 소유권한을 표시한 문구가 없거나 타인의 소유권한을 표시한 문구가 있다면 위법하게 사용하고 있을 가능성이 높습니다. 이 경우 저작권법에 의해 불이익을 받으실 수 있습니다.

다양한 기기에 사용할 수 있습니다. 또한 한빛미디어 사이트에서 구입하신 후에는 횟수와 관계없이 내려받으실 수 있습니다.

한빛미디어 전자책은 인쇄, 검색, 복사하여 붙이기가 가능합니다.

전자책은 오탈자 교정이나 내용의 수정·보완이 이뤄지면 업데이트 관련 공지를 이메일로 알려드리며, 구매하신 전자책의 수정본은 무료로 내려받으실 수 있습니다.

이런 특별한 권한은 한빛미디어 사이트에서 구입하신 독자에게만 제공되며, 다른 사람에게 양도나 이전은 허락되지 않습니다.