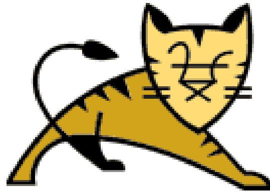
Pippo-Proxy 로
Tomcat의 정적 Content의 성능을 고속화!



Pippo-Proxy

- 옮긴이 : 변두리 개발자 paradozz@paran.com

- http://tomcat.apache.org/tomcat-4.1-doc/proxy-howto.html
- http://www.coreservlets.com/Apache-Tomcat-Tutorial/,
- http://www.javaworld.com/javaworld/jw-02-2005/jw-0228-pippo.html
- http://sourceforge.net/projects/pippoproxy/

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

1. 소 개

기존의 Apache와 Tomcat을 연동하여 정적 contents를 apache로 일관하는 방법은 문제가 있다고 알려져 있습니다. 본 문서는 Tomcat을 분산처리로 이용할 때, 정적 contents의 성능에 불만을 품거나, 딱히 이점을 체험하지 못하는 분들에게 꼭 추천하고 방법입니다.

그러한 문제점으로 나온 대안이 바로 Tomcat용으로 개발된 Pippo-Proxy입니다.

이 문서는, Apache + Tomcat의 연동의 문제점을 지적하고, PippoProxy의 이용법,

그리고 PippoProxy + Tomcat의 이점을 소개하며, 끝으로는 각 방법에 대한 퍼포먼스 테스트로 끝을 맺습니다.

2. Apache/Tomcat 구성의 개요와 문제점

표준 Apache/Tomcat의 구성에서는, Web 어플리케이션의 동작환경이 되는 Tomcat의 앞단, 즉 구체적으로 기업내의 인트라넷과 인터넷과의 중간 지대 (이른바 : 비무장지대 DMZ : Demilitarized Zone)에 , Web서버의 Apache가 배치됩니다. 이 경우, Apache는, Web페이지를 구성하는 static-contents(정적 컨텐츠-html,htm,xml)의 입출력을 load-balance할 뿐만 아니라. DMZ로 동작하는 Tomcat에 대해 , Http-Proxy기능도 제공합니다.(그림1)

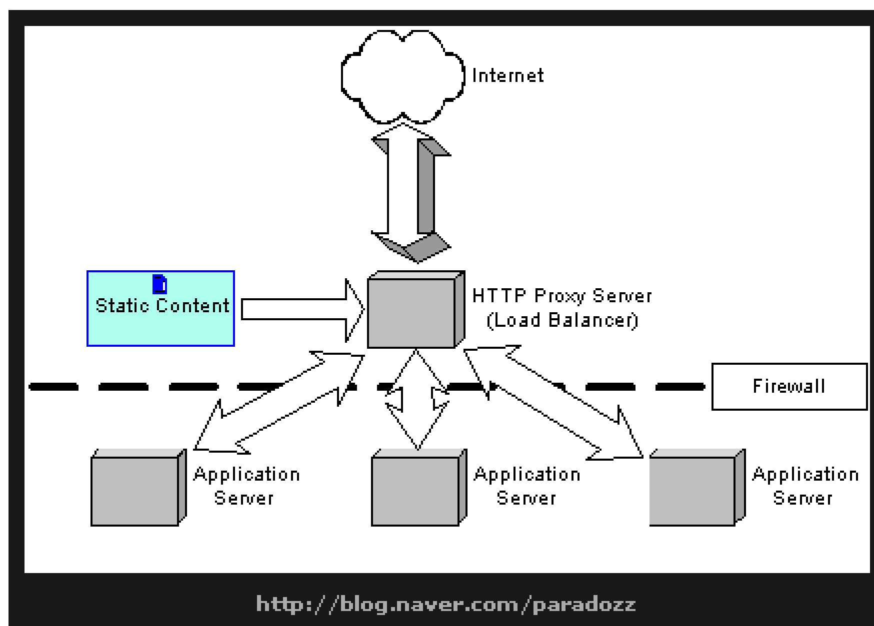


그림1. 표준 apache/tomcat의 설정

이러한 구성의 Apache는 , 각 기업의 보안에 근거해서 기업 네트워크의 외부, 혹은 내부로부터의 접속을 처리한 후, Tomcat에 대해 Dynamic-contents로 포워드 하는 기능을 담당하게 됩니다. 포워딩 된 Tomcat은 기업 내의 네트워크와 각종 데이터 소스에 접속을 하면서 Web 어플리케이션의 처리를 완수한 후 , 그 결과를 클라이언트에 돌려 보냅니다. 그 후, Apache는 정적 컨텐츠를 클라이언트에 돌려 보내는 기능을 하고 있습니다

이러한 Tomcat/Apache간의 연동은, 현재 Apache용으로 된 커넥터를 몇 개 재공하고 있습니다. 실전에서는 mod_jk 가 사용되고 있습니다. mod_jk는 Tomcat에 대한 커넥션풀을 갖추고 있어서 AJP라는 프로토콜을 이용해 mod_proxy의 2배의 성능을 발휘한다고 알려져 있습니다. 이 mod_jk를 이용하는 표준 Apache/Tomcat구성에서는, 정적 컨텐츠는 Apache상에 배치되어 일반적으로 보안에 제외되어 클라이언트로부터의 접속을 처리하게 됩니다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

이 아키텍쳐는 기업 네트워크의 내부에 있는 서버로부터, 네트워크 외부의 클라이언트에 대해 컨텐츠 전달을 제어하는 상황에서 기능면에서는 분명하게 약점이 있습니다.

구체적으로 어떠한 약점들이 존재하는지 구체적인 시스템의 예를 들어 설명하기로 하겠습니다.

2-1. 대규모 시스템에서의 문제점

한가지 대규모 시스템의 예를 들어보겠습니다, 금융기관, 각 산업기관에서의 대규모 web 어플리케이션을 생각해보면 표준으로 설정하는 Apache/Tomcat의 구성으로 기업내의 네트워크를 구성하고 있습니다. (SunApplication서버에는 동적 컨텐츠와 정적 컨텐츠와의 연계가 이와 비슷합니다.)

이 금융기관에는 , PDF로 포멧된 금융 리포트 또는 기업에 관한 조사 리포트들을 이용하는 고객에게 제공하는 수단으로 전용 web 어플리케이션을 사용하고 있습니다. 이러한 문서에 대한 접속권한은 , 고객 마다 개별적으로 설정되어 있으며,

또 각 문서는 CMS(Content Management System)으로 관리되어 내부 방화벽에서 나누어진 형식으로 내부에 여러 서버를 설치하고 있습니다.(그림2)

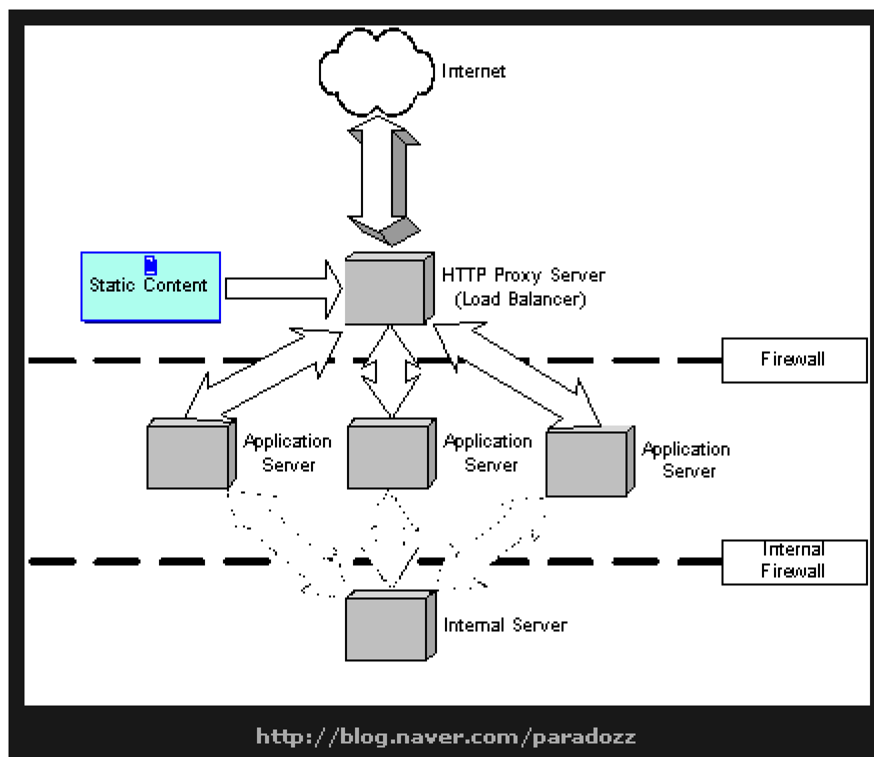


그림2. 금융권 web 어플리케이션의 일례

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

위의 그림처럼 어플리케이션의 동작은 , 어떤 A라는 고객이 web 브라우저에서 요청을 하여 특정 문서에 접속을 시도합니다, 그러면 그 요청은 유저-프로파일이나 비즈니스-룰에 따라서 처리된 후, 내부 서버로 전송이 됩니다. 그리고 요청을 받은 내부 서버가 문서를 발송하게 됩니다.

이 구성의 경우, 비즈니스 논리의 적용과 문서의 제공에 역할을 담당하는 것은 Apache입니다. 또 내부 서버는 반드시 Tomcat이 된다고는 할 수 없습니다. 그 때 Apache는 내부 서버와의 통신 시에 모듈 mod_proxy를 사용합니다. 하지만 이 방법에는 성능이 떨어지고 , 불안정하다는 데 있습니다. (그림3)

Apache에 2개의 방화벽을 통과하는 권한을 주지 않으면 안되는 중대한 문제점이 있습니다

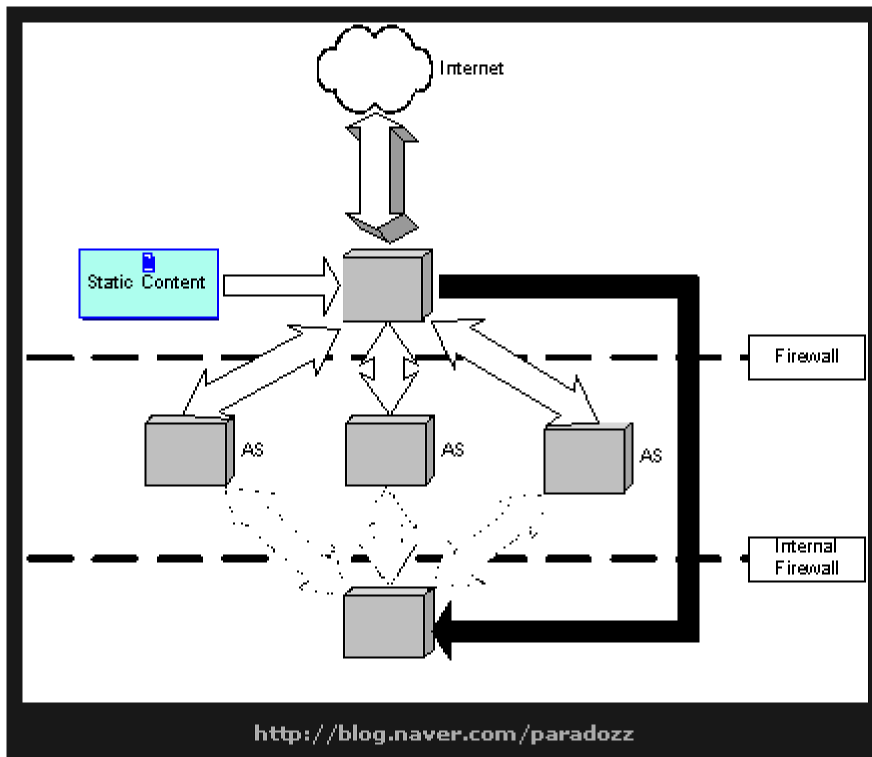


그림3. apache가 두 개의 fire-wall을 넘어 내부서버에 접근한다.

또한 보안규약으로 보호된 내부 컨텐츠에 부정 접속을 차단하기 위해서는 , Apache가 요청을 필터링 할 때, Tomcat상에 설정된 모든 비즈니스 룰을 apache상에도 설정할 필요가 있는 불편함이 있습니다.

또 그러한 비즈니스 룰의 설정을 시스템내의 다른 부분과의 통일성을 유지하여야 하며, mod_rewrite나 mod_auth라는 모듈을 Apache에 짜 넣을 수도 있겠습니다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

2-2. 소규모 시스템에서의 문제점

심플한 시스템의 예를 한번 생각해 보기로 합니다.

예를 들어서 사내 네트워크상에 이용되고 있는 web의 경우, apache을 연동하지 않고 tomcat에 갖춰지는 web서버를 이용해 tomcat만으로 정적 컨텐츠와 동적 컨텐츠의 양쪽 모두를 취급하는 케이스가 있습니다.

이 경우, 서버는 Tomcat만으로 이루어져 있어서 정적 컨텐츠와 동적 컨텐츠의 송출을 다른 서버로 이관하는 load-sharing을 할 수 없습니다. 또, 서버에 걸리는 부하를 경감하기 위해서 tomcat에 캐쉬 기능이 필요할지도 모르는 것입니다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

3. PippoProxy의 개요

이상, 대규모 시스템 / 소규모 시스템에도 표준 apache/tomcat의 구성에는 문제가 있음을 알 수 있었습니다. 그리고 , 이 문제의 해소가 바로 PippiProxy인 것입니다.

PippoProxy란 Tomcat전용의 Servlet으로 개발된 proxy-server입니다.

이것을 Tomcat에 설정하는 것으로 표준 Apache/Tomcat 연동을 대체해서 퍼포먼스가 뛰어난 서버 환경을 구성할 수 있습니다.

3-1. PippoProxy의 동작 형태

PippoProxy는 2개의 동작 모드가 준비되어 있습니다.

1개는 기존의 web어플리케이션 container전체에 영향을 주는 방법 또는 플러그인식으로의 배치가 있으며, 다른 1개는 standard-alone의 web 어플리케이션으로 동작하는 모드입니다.

전자의 모드를 선택했을 경우, 비즈니스 논리의 처리를 담당하는 클래스로부터, 필요에 따라서 PippoProxy를 이용하게 됩니다.

예를 들면, 좀 전의 금융기관의 web어플리케이션에서 특정의 PDF문서에 대한 다운로드 요청을 유저의 프로파일이나 그 외의 비즈니스 룰에 따라서 처리한 후 , pippo-proxy에 나머지를 처리하는 방법이 됩니다.

구체적으로 MVC모델에 따라서 만들어진 web 어플리케이션을 상상하여 이야기를 진행해보면, 이 web 어플리케이션의 front-end가 되는 servlet은

- **http://[domain]:[port]/[context]/servlet/***

라는 동작 형태를 지니게 됩니다.

이 때, 중앙 Servlet(또는 Struts나 SpringMVC의 중앙 servlet) 은 , 클라이언트로부터 요청을 받으면, 그 클라이언트가 필요한 권한을 가지고 있을지를 판단해 대응하는 세션이나 페이지 속성에 값을 설정한 후 , 그 요청을

http://[domain]:[port]/proxy/ 의 URL로 동작하는 pippoProxy로 포워드 합니다.

포워드된 pippo-proxy는 속성의 값을 체크한 후, 요구된 자원을 내부 서버로부터 가져와 그것을 클라이언트에 돌려보냅니다. (그림4)

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

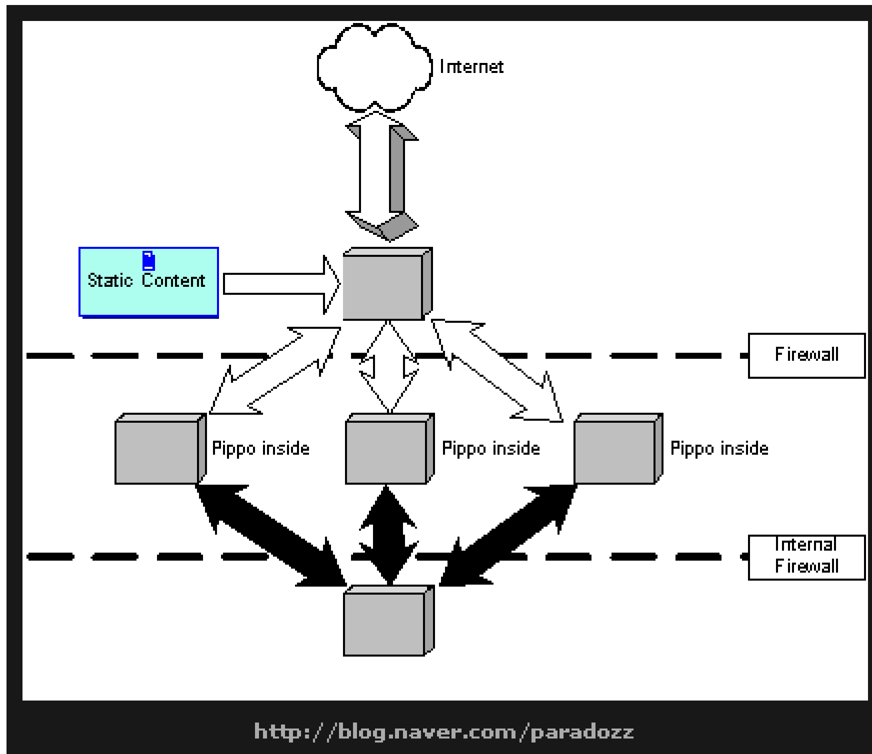


그림4. PippoProxy가 설정된 대규모의 어플리케이션

이러한 구조 때문에, 필요한 권한을 가지지 않는 부정한 유저가 security 관리된 자원을 직접 요구해도 그 요구는 통과하지 못합니다. 왜냐하면 , 그러한 유저로부터

Http Request/Session 권한 속성에는 값이 설정되어 있지 않기 때문입니다.

3-2, Chain 방식의 캐시

보안 기능외에, PippoProxy는, 내부 서버에 대한 영속성 Connection- pool 의 기능도 제공합니다. 이 기능은, PippoProxy가 한 번 내부 서버에 접속하면, 그 접속 상태를 유지하고, client가 요청할 때마다 내부 서버의 connection을 open/close 작업을 빈번히 할 필요가 없어지게 됩니다.

또, 정적 컨텐츠의 경우, 메모리 와 File System의 쌍방에, 일정한 계층 구조에 근거한 효율적인 캐시가 작용되어 새로운 퍼포먼스 향상이 도모됩니다. 또한 이 캐시의 실체는 메모리 캐시용의 1개의 Node와, File system 캐시용의 Node로 이루어져 있어 양방향적인 관계를 맺습니다. PippoProxy 캐시는, 디자인패턴의 'Chain of responsibelity'를 응용한것으로 알려져 있습니다. (그림5)

Chain of responsibelity' - http://compstat.chonbuk.ac.kr/rightway/designpatterns/chain.html

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

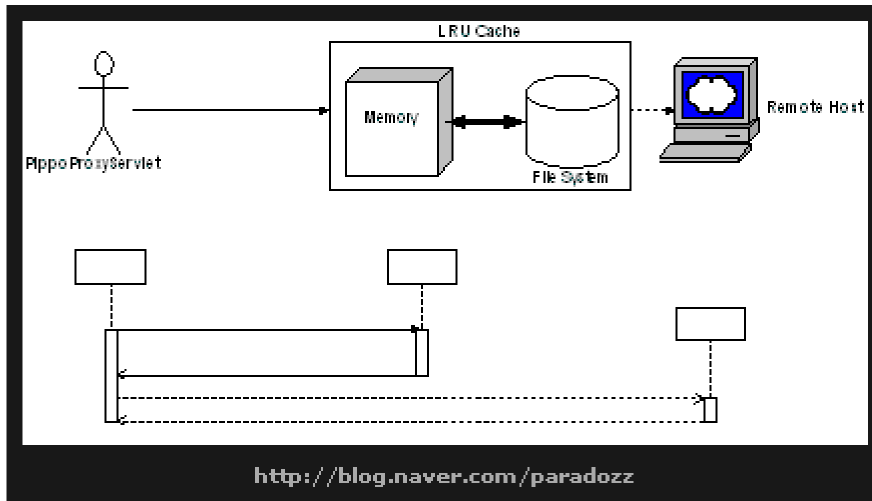


그림5. PippoProxy의 캐시 구조

위의 그림에서 Cache 구조의 관하여 설명하면 , PippoProxy의 자체로 된 Servlet은 특정한 첫번째의 자원의 Node를 요청합니다. 그 Node는 요청된 자원을 가지고 있으면 그것을 돌려 주고, 가지고 있지 않을 때는 , 그 요청을 Chain안에 2번째의 Node에 넘겨 줍니다.

2번째의 Node는 요청된 자원을 가지고 있는 경우에는 그것을 돌려 주고,

가지고 있지 않는 경우에는 그 자원을 내부 서버로 부터 요구합니다.

이상의 처리를 반복한 후, 메모리 캐시위의 모든 Node가 목적의 자원을 가지고 있지 않을 때는, File System 으로의 캐시로 요청이 전송 됩니다. 위의 그림에서 볼 수 있듯이

Cache -> File System의 상황입니다. 그리고 같은 Cache안에도 요청된 자원이 존재하지 않는 경우에는 최종적으로 Pippo-Proxy가 내부 서버에 요청을 하는 구조입니다.

덧붙여서 , Pippo-Proxy의 Memory Cache와 File System Cache는 , Cache의 최대 사이즈(size)를 MB단위로 설정할 수 있게 되어 있고, Cache가 가득 차게 되면 마지막의 Node의 경우에는 삭제해야 할 자원이 'LRU(Least-Recently Used) 알고리즘' 에 의해 근거하고 결정됩니다.

LRU(Least-Recently Used) - http://sangchin.byus.net/FlashHelp/Java_util/coll_cache.html

또한 , 이러한 Cache는 , Exclusive Cache Hierarchy 로 구성되어 있다. 즉, Momory Node와 File System의 Node의 내용은,

서로 중복되지 않게 되어 있다. PippoProxy의 상세한 내용은 PippoProxy의 Document를 참조하라.

Exclusive Cache Hierarchy - http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1291359

PippoProxy의 Document - http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1291359

영원한 우리땅, 독도
Dokdo forever belongs to Korea

3-3. PippoProxy를 설치하자.

PippoProxy의 개요를 설명했다. 이제 대망의 PippoProxy의 설치와 설정 그리고 사용법에 관하여 설명합니다. 또한, PippoProxy의 설치나 설정 등은, Ant에 의해 설치되고 거의 자동화 되어 있기 때문에, 번거로운 설정이나 설치를 병행할 필요가 없습니다.

아래는 PippoProxy의 설치 환경입니다.

Java Version : J2SE 1.41이상

Apache Ant : 1.6.2이상

Tomcat : 5.0x이상

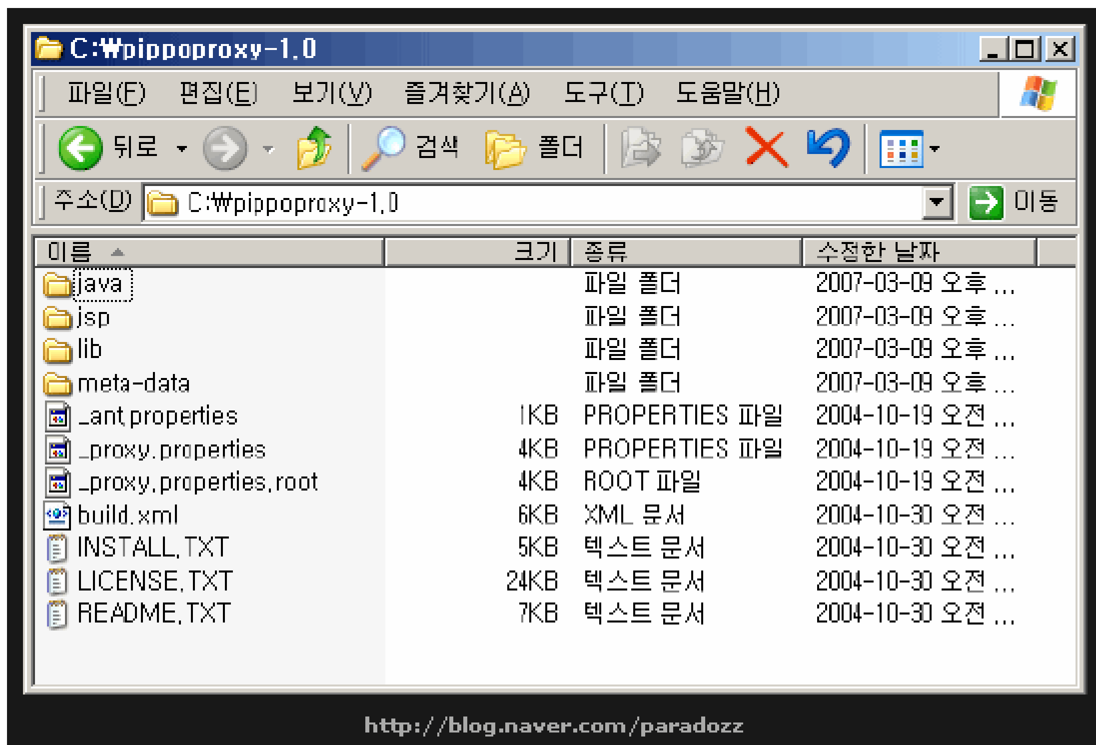
PippoProxy의 동작 모드에서는 , StandardAlone의 Web Application으로서 동작시키는 모드와, Web Application안에 전체 혹은 플러그인식으로 동작시키는 모드 2가지가 있습니다.

각각의 모드에 관하여 설명합니다. 우선 Standardalone mode의 설치/동작 방법에 대해서 설명합니다. (먼저 PippoProxy의 Web Site로부터, PippoProxy의 Achive를 다운 받습니다.)

pippo-proxy site : http://sourceforge.net/projects/pippoproxy/

C:₩pippoprocy-1.0 로 Root를 만들어 환경변수를 등록한다.

%PIPPO_PROXY_HOME%



내부의 _ant.properties파일을 열어 Tomcat의 Path를 설정한다. 환경변수에 TOMCAT_HOME을 지정한 로컬에서는 %TOMCAT_HOME%을. 경로를 직접 넣어도 된다.

영원한 우리땅, 독도
Dokdo forever belongs to Korea

%PIPPO_PROXY_HOME%₩_ant.properties 파일

```
# Ant Properties File
# Local Tomcat's webapps.
deploy_local= C:/tomcat5.5/apache-tomcat-5.5.20/webapps
# Application name.
application_name=pp
# Local temp directory to do the stuff.
outdir=build
```

커맨드 창을 열어 and build을 실행하자.

```
C:₩pippoproxy-1.0>ant deploy
```

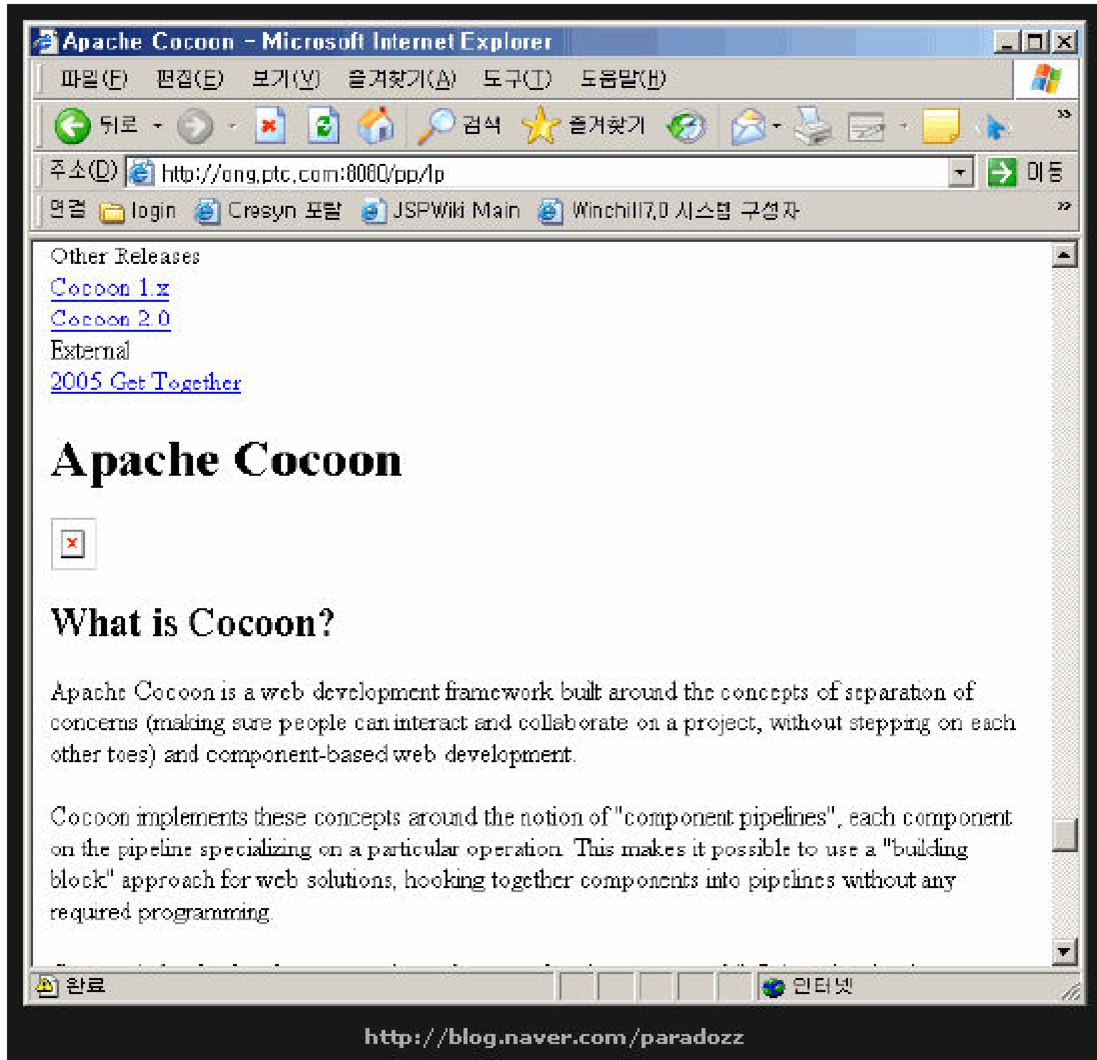
TOMCAT_HOME/webapps/pp/WEB-INF/web.xml을 보면 다음과 같다.

```
<servlet>
   <servlet-name>PippoProxyServlet</servlet-name>
   <servlet-class>org.pippo.proxy.WebCachedProxyServlet</servlet-class>
   <init-param>
      <param-name>ENABLE_SESSION_ATTR_KEY_FOR_LOGIN</param-name>
      <param-value>@ENABLE_SESSION_ATTR_KEY_FOR_LOGIN@</param-value>
   </init-param>
   ............................................
   <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
               <servlet-name>PippoProxyServlet</servlet-name>
               <url-pattern>@LOCAL_PREFIX@/*</url-pattern>
</servlet-mapping>
```

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

정상적 설치를 확인하기 위해 톰캣을 가동한 후 , http://127.0.0.1:8080/pp/lp 로 접속을 시도한다. apache Cocoon 페이지가 보이면 일단 설치는 성공입니다.



현재 설정은 단일 Application에 설정한 예이다. Tomcat container 전체적인 PippoProxy를 설정하는 방법과 플러그인 식의 설정 방법을 알아봅니다.

3-4. Tomcat 전체 또는 플러그인식의 설치

위에서 알아본 단일 Application 설정 방법 이외에 Tomcat Application영역 전체 또는 플러그인에 영향을 주는 방법이 있다. 먼저 Pippo-Proxy가 위치된 곳으로 이동한다.

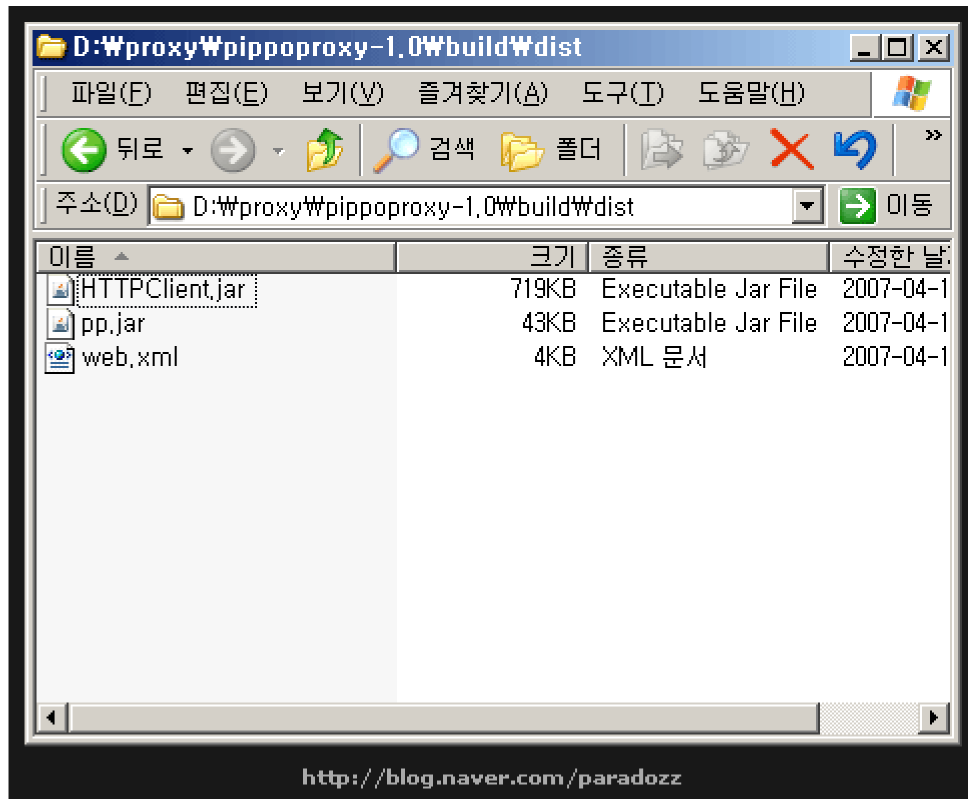
영원한 우리땅, 독도
Dokdo forever
belongs to Korea

```
cd %PIPPO_PROXY_HOME%
```

다음과 같은 ant명령을 실행한다.

```
D:₩proxy₩pippoproxy-1.0>ant jarPkg
```

위의 명령을 실행하면 다음의 디렉토리에 jar파일 두개와 web.xml이 빌드된다.



%PIPPO_PROXY_HOME%₩pippoproxy-1.0₩build₩dist₩HTTPClient.jar

%PIPPO_PROXY_HOME%₩pippoproxy-1.0₩build₩dist ₩pp.jar

%PIPPO_PROXY_HOME%₩pippoproxy-1.0₩build₩dist ₩web.xml

위의 두개의 jar파일을 %TOMCAT_HOME%/shared/lib 디렉토리에 카피를 하고,

web.xml을 열어 <servlet> 영역을 %TOMCAT_HOME%/conf/web.xml에 추가를 한다.

그리고 web.xml안에 있는 <servlet-mapping> 영역을 %TOMCAT_HOME%/conf/web.xml에 servlet-mapping부분에 추가를 한다. 위의 설정을 마치고 다시 톰켓을 시동한다.

주의 : 이전에 StandardAlone모드의 pp.war Application은 지우고 시동을 해야 servlet-name 영역이 충돌이 없다.

위의 설정에서 이제 http://localhost:8080/ 으로 접속하면 모든 Tomcat Container 영역은 PippoProxy Server의 영향을 미친다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

3-5. Default의 설정변경

다음의 PippoProxy의 설정 변경은 RemoteHost/Memory/Cache등 여러가지 설정을 변경할 수 있는 옵션이 있다. 기존의 StandardAlone의 모드에서는 다음 properties파일을 따르고

%PIPPO_PROXY_HOME%₩pippoproxy-1.0₩_proxy.properties

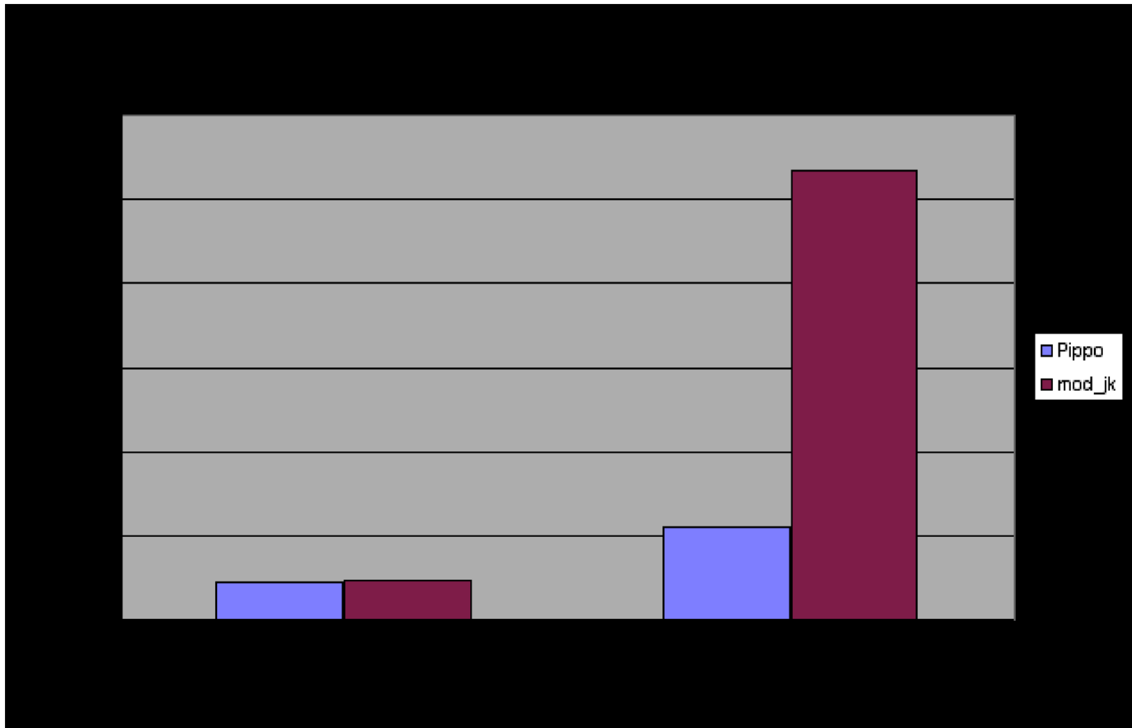
전체영역의 모드에서는 다음 properties파일을 따른다.

%PIPPO_PROXY_HOME%₩pippoproxy-1.0₩_proxy.properties.root

* **ENABLE_SESSION_ATTR_KEY_FOR_LOGIN** :프록시 실행전에 세션의 속성을 체크하는가?
* **SESSION_ATTR_KEY_FOR_LOGIN : 프록시** 실행전에 체크한 속성을 지정한다.
* **CACHE_ENABLED** :정적내용에 대한 캐시기능의 유효 / 무효를 지정
* **CACHE_TIMEOUT : 리소스가** 캐시로부터 삭제되기 까지의 시간을 지정 (밀리 (milli) 초 단위)
* **CACHE_MAX_MEMORY_SIZE** :메모리 캐시의 최대 사이즈를 지정
* **CACHE_MAX_DISK_SIZE : 파일시스템** 캐시의 최대 사이즈를 지정
* **CACHE_PATH_DIR** :리소스의 격납선을 지정
* **REMOTE_HOST** : 프록시대상의 리모트 호스트를 지정
* **REMOTE_PORT** :프록시대상의 리모트 포트를 지정
* **IS_ROOT** : Tomcat를 프록시 서버로서 동작시키는지 아닌지를 지정
* **REMOTE_PREFIX** :어플리케이션 실행시의 로컬 접두사를 지정
* **LOCAL_PREFIX** : 어플리케이션 실행시의 리모트 접두사를 지정
* **NOT_ALLOWED_HEADERS** :거부한 일련의 HTTP 헤더를 파이프 (pipe) (|) 로 지정
* **PROXY_ENABLED** : 여러의 프록시서버로 프록시 대상의 내부 서버에 하는지 아닌지를 지정
* **PROXY_HOST** :프록시 대상의 내부 서버에 접속하기 위한 프록시 호스트를 지정
* **PROXY_PORT** : 프록시 대상의 내부 서버에 접속하기 위한 프록시 포트를 지정
* **PROTOCOL** :프로토콜을 지정 (현시점에서는 HTTP만을 서포트 (support))
* **INIT_CONNECTION** : HTTP 커넥션의 초기삭을 지정
* **MAX_CONNECTION** :커넥션 ·풀 안의 HTTP 커넥션의 최대치를 지정

영원한 우리땅, 독도
Dokdo forever
belongs to Korea

4. 퍼포먼스의 측정



위의 도표는 프록시의 평균 처리 시간을 나타낸 도표이다. 그래프의 Y축은 평균 처리시간 msec 이고 , x축은 파일 사이즈이다. 위의 결과를 보면 알겠지만 일반적인 Apache/tomcat 구성에 대한 PippoProxy퍼포먼스는 파일 사이즈가 클수록 향상하고 있다.

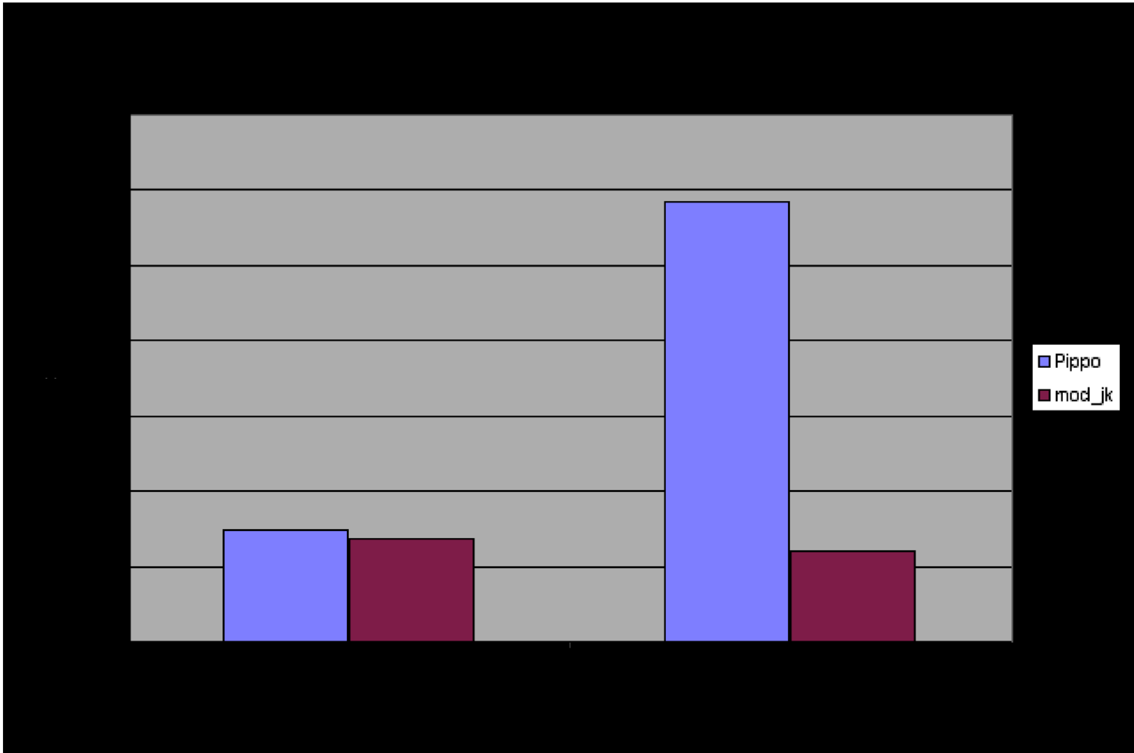
그 이유는 PippoProxy의 캐시 기능이 'Temporal Locality'라고 말한 법칙에 근거하고 동작하고 있기 때문이다.

이 법칙은, pippoProxy에 있는 리소스 접속이 1번 행해지면, 그 리소스를 캐시한 것으로 다음번의 접속에 관련된 처리 시간을 단축하고 있다.

또한 소규모 사이즈의 13이 나타내는 파일은 PippoProxy+Tomcat과 Apache/Tomcat구성의 퍼포먼스에 대부분 차이가 생기지 않는다. 이것은, 파일에의 초기 접속이 필요로 한 시간이, 리퀘스트의 처리시간의 대부분을 차지하고 있기 때문이다.

한편 대규모 사이즈의 파일 테스트는 둘 쪽의 퍼포먼스의 차이가 뚜렸하게 보이고 있다. 이 쪽의 경우, 처음의 몇번을 호출하고 둘쪽의 퍼포먼스는 그다지 차이는 보이지 않는 것처럼 보이지간 Temporal Locality의 법칙에 의하면 한번 접속 된 리소스를 캐시에 담고 , 요청을 500회 반복 할 시점에서는 PippoProxy+Tomcat쪽이 기존의 Apache/Tomcat의 구성보다도 5배가 빠른 결과로 나타난다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea



위의 도표는 검증 결과를 '시간당의 처리량의 시점에서 관찰한 것이다. 이 결과에 의하면 PippoProxy+ Tomcat의 단위 시간당 처리량은 파일 사이즈에 비례하고 증가하고 있는 것에 비하여, 표준 apache/tomcat구성의 그것은 역으로 감소하고 있는 것으로 밝혀진다.

이러한 퍼포먼스 검증 결과로 부터 , PippoProxy가 구비한 캐시기능은 중 규모로부터 대규모적 사이즈가 정적내용에 대해서 더 유연하게 된다는 것이다. 이상 일반 Apache/Tomcat구성의 대체로서 사용 가능한 프록시서버로 PippoProxy를 소개했다. PippoProxy는 보안의 제약으로 부터 Apache/Tomcat구성을 사용할 수 없는 환경 또는 중 규모-대규모적인 프로젝트에 큰 위력을 발휘할 것이다.

영원한 우리땅, 독도
Dokdo forever
belongs to Korea