

Apache / Tomcat Performance & Tuning

톰캣 성능 분석과 Tuning

아젠다

톰캣 아키텍처

- JavaEE Spec

- 클래스로더

- 디플로이

튜닝

- 톰캣 튜닝 옵션

 - 커넥터

 - 스레드풀

 - DBConnectionPool

 - 로깅

- JVM 튜닝 옵션

M/W 성능 분석과 Tuning

아젠다

모니터링

VirtualVM

MBean

Pinpoint

트러블슈팅

쓰레드

메모리

More...

로거

컬렉션

톰캣 네이티브



아파치 소프트웨어 재단에서 개발하는 **JAVA** 기반의 웹서버이자
JavaEE 서블릿 컨테이너 지원하는 웹어플리케이션 서버(WAS : Web Application Server)이다.

톰캣은 Java Application이다.

<https://apache.googlesource.com/tomcat/>

1) startup.sh

```
exec "$PRGDIR"/"$EXECUTABLE" start "$@"
```

catalina.sh를 호출 할 때 'start' 라는 argument를 전달해서 catalina.sh가 시작 하도록 한다. Tomcat을 시작 하기 위한 유틸리티 스크립트이다.

2) catalina.sh

```
eval exec "W"$RUNJAVAW"" "W"$LOGGING_CONFIGW"" $LOGGING_MANAGER $JAVA_OPTS $CATALINA_OPTS W
-Djava.endorsed.dirs="W"$JAVA_ENDORSED_DIRSW"" -classpath "W"$CLASSPATHW"" W
.....
-Djava.io.tmpdir="W"$CATALINA_TMPDIRW"" W
org.apache.catalina.startup.Bootstrap "$@" start
```

java 명령어로 클래스를 실행하는 명령어이다. Target Class는 org.apache.catalina.startup.Bootstrap 클래스이고, VM Option으로 다양한 정보를 파라미터로 넘겨준다.

3) Bootstrap.java

```
public static void main(String args[]) {
    if (daemon == null) {
        // Don't set daemon until init() has completed
        Bootstrap bootstrap = new Bootstrap();
        try {
            bootstrap.init();
        } catch (Throwable t) {
            handleThrowable(t);
            t.printStackTrace();
            return;
        }
        daemon = bootstrap;
    }
    ...
    daemon.load(args);
    daemon.start();
}
```

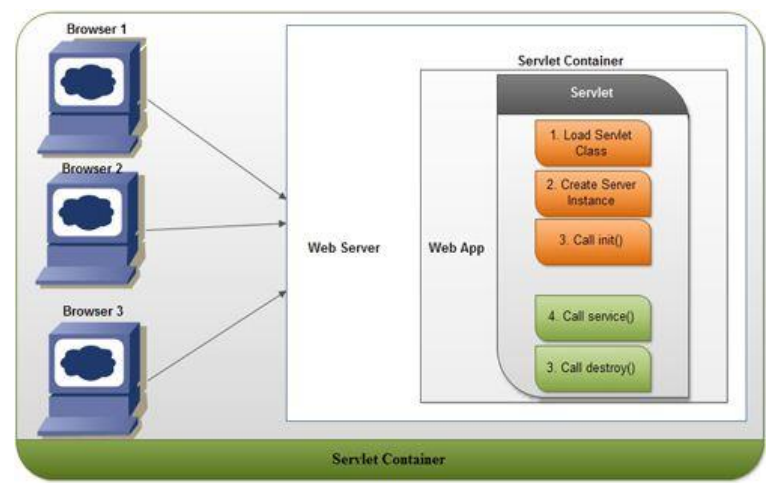
- JVM (Java) 디렉토리의 lib, jre/ext/lib 안에 있는 Class(JAR) 로딩
- 내가 만든 웹 애플리케이션의 /WEB-INF/classes 폴더 Class 로딩
- 내가 만든 웹 애플리케이션의 /WEB-INF/lib 폴더의 Class(JAR) 로딩
- 시스템 클래스 로딩 (톰캣 실행시 파라미터로 넘어온 CLASSPATH 위치들에 있는 Class(JAR) 들)
- 공통 클래스 로딩 (CATALINA_HOME/lib , CATALINA_BASE/lib)
- java.io.tmpdir (임시디렉토리) 존재 여부 체크
- [URL_PKG_PREFIXES](#), [INITIAL_CONTEXT_FACTORY](#) 를 Key로 하는 프로퍼티를 셋팅
- server.xml을 파싱하기 위해 Digester를 생성합니다. server.xml이 제대로된 규칙을 가진 XML 인지 검사하는 클래스.
- server.xml을 파싱 하고, 파싱된 XML정보를 기반으로 Object Mapping Tool로 객체를 생성.
- 모든 컴포넌트를 초기화 하면서 JMX agent에 등록
- webapps 디렉토리 및 conf/Catalina/localhost/*.xml 파일의 정보를 기반으로 context를 생성하고 배포 작업을 한다(war 압축 풀기).
- init(), load(), start() 단계를 거치면서 Tomcat은 특정포트(기본값 : 8080) 로 ServerSocket을 Bind 하고 있는 상태
- 브라우저에서 포트를 통해 커넥션을 요청하면 ThreadPoolExecutor가 생성해놓은 스레드 중 하나가 시작, 즉 Tomcat은 1개의 요청당 1개의 Thread를 사용한다.
- Connector를 통해서 HTTP 요청을 받으면 데이터를 정형화 하고, Valve를 통과 시킨뒤 실제 맵핑된 서블렛(Servlet)으로 전달.

톰캣은 자바
톰캣은 서블릿 컨테이너

톰캣은 Servlet Container이다.

Java에서 서블릿 컨테이너(웹 컨테이너라고도 함)는 동적 웹 페이지를 생성한다. 따라서 서블릿 컨테이너는 자바 서블릿과 상호작용하는 웹 서버의 필수적인 부분이다. 서블릿 컨테이너는 클라이언트 브라우저와 서블릿 간에 통신이다.

서블릿의 수명주기를 관리하는 서블릿 컨테이너. 서블릿 컨테이너를 메모리에 로드하여 서블릿 방법을 초기화 및 호출하고 이를 파기한다.



- 클라이언트 브라우저는 페이지의 웹 서버 또는 HTTP 서버에 액세스한다.
- 웹 서버는 요청을 서블릿 컨테이너로 리디렉션한다(서블릿은 서블릿 컨테이너 내부에서 실행되는 HTTP 리스너).
- 서블릿 컨테이너는 요청을 적절한 서블릿으로 리디렉션한다.
- 서블릿은 컨테이너에 없는 경우 동적으로 회수되어 컨테이너의 주소 공간으로 로드한다.
- 초기화를 위해 서블릿을 처음 로드할 때 서블릿 컨테이너는 Servlet init() Method을 한 번 호출한다.
- 서블릿 컨테이너는 servlet service() Method을 호출하여 HTTP 요청을 처리한다. 즉, 요청에서 데이터를 읽고 응답을 HTML로 전달한다.
- 서블릿은 컨테이너의 주소 공간에 남아 있고 다른 HTTP 요청을 처리할 수 있다.
- 웹 서블릿은 데이터(HTML 페이지, 그림 ...)를 생성하여 동적으로 생성된 결과를 클라이언트에게 전달한다.

톰캣은 자바
톰캣은 서블릿 컨테이너

톰캣은 Servlet Container이다.

JAVA	서블릿	JSP	SPEC	톰캣	웹로직	WildFly/JBoss	제우스
J2EE 8	4.0	2.3	JSR 369	9.0.X	-	14.0.0	-
J2EE 7	3.1	2.3	JSR 340	8.5.X	12cR2	13.0.0 ~ 8.0.0	8
J2EE 6 / SE 6	3.0	2.2	JSR 315	7.0.X	12cR1	7.1.0 ~ 6.0.0	7
J2EE 5 / SE 5	2.5	2.1	JSR 154	6.0.X	11gR1 10gR3 10.0	5.0.X	6
J2EE 1.4 / SE 1.3	2.4	2.0	JSR 154	5.5.X	9.0	4.3.0 ~ 3.2.4	5
J2EE 1.3 / SE 1.2	2.3	1.2	JSR 54	4.1.X	8.1 7.0 6.1	3.2.3	-

Tomcat 설치

- Tomcat 엔진은 <https://tomcat.apache.org>에서 설치하고자 하는 Tomcat 엔진 버전을 다운로드 받아 설치파일의 파일을 압축을 푸는 것으로 설치를 수행한다.
- Tomcat Engine 8.x 설치 시 아래의 URL을 통해 zip, tar.gz 파일을 다운로드 받을 수 있다.

다운로드 URL: <https://tomcat.apache.org/download-80.cgi>

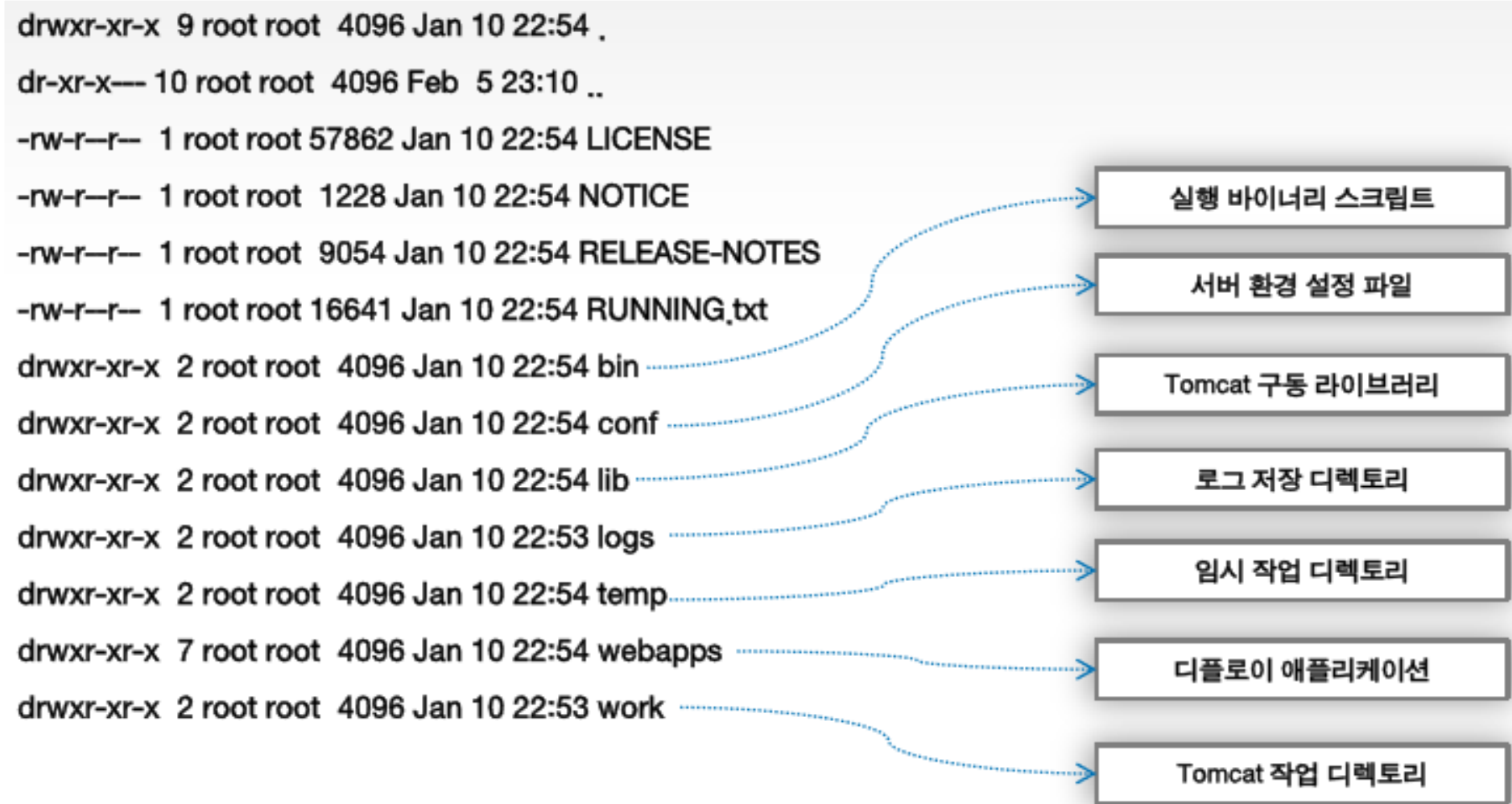
- 설치 방법은 서버에 업로드를 수행한 후 압축을 풀면 설치가 완료.

** 일반적으로 엔진과 애플리케이션 위치로 구성한다.

```
[wasuser@centos10 /tomcat/was]$ unzip apache-tomcat-8.5.23.zip
....생략
[wasuser@centos10 /tomcat/was]$ls -al
drwxr-xr-x 9 wasuser wasuser 149 Sep 28 2018 apache-tomcat-8.5.23
```


톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치

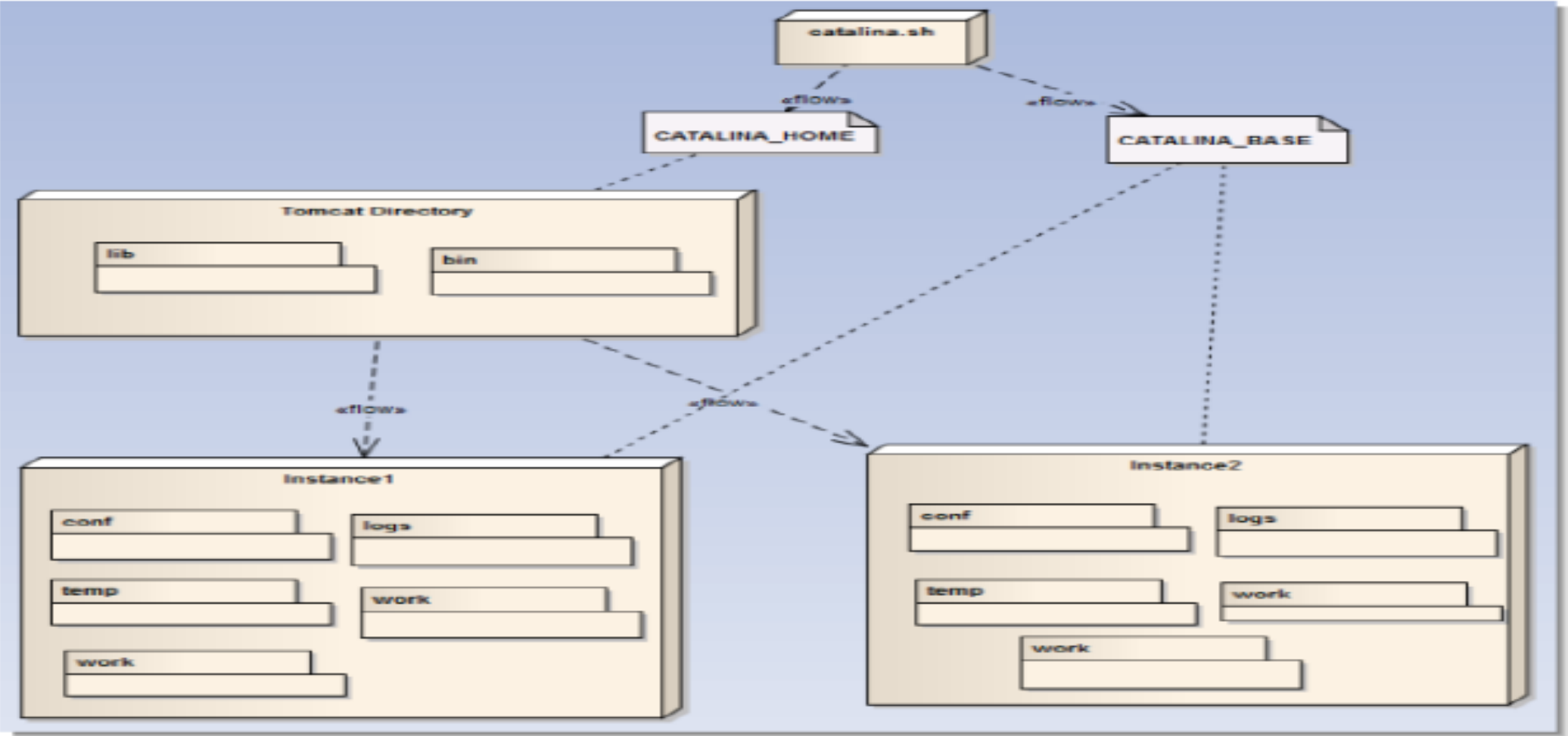
Tomcat 엔진 디렉토리 구조



Tomcat 인스턴스

Tomcat Engine은 CATALINA_HOME을 기준으로 "lib", "bin" 디렉토리만 사용합니다.

- 실제 서비스를 담당하는 인스턴스들은 "conf", "logs", "temp", "webapps", "work" 디렉토리를 사용합니다.
- 이와 같은 디렉토리를 분리함으로써 "멀티 인스턴스" 환경을 구성합니다.



Tomcat 환경변수

Tomcat 내의 실제 인스턴스의 속성을 정의 한 환경 변수입니다

- CATALINA_HOME : Tomcat Engine 홈 디렉토리
- CATALINA_BASE : 멀티 인스턴스의 홈 디렉토리
- SERVER_NAME : 멀티 인스턴스의 서버명
- PORT_OFFSET : 멀티 인스턴스에서 사용하는 계산되는 port 범위
- COMP_USER : 인스턴스를 조작하기 위한 사용자 계정
- LOG_BASE : 멀티 인스턴스의 로그 디렉토리 지정

\$CATALINA_HOME/bin/startup.sh 실행하면 실제로 catalina.sh이 실행된다.

catalina.sh 은 CATALINA_HOME 환경 변수를 통해서 "Tomcat Engine" 디렉토리를 통해서 기동한다.

Tomcat 인스턴스 서비스 포트 변경

각 인스턴스 별로 server.xml을 수정합니다.

변경 이유는 동일한 포트를 사용하면서 서버를 기동하려 할 때, Port 충돌로 인해 서버 기동이 되지 않습니다. 인스턴스에서 지정한 port offset(env.sh) 과 default port 를 더하기 연산을 하여 인스턴스 내에 offset연산으로 인해 포트 충돌에 대한 문제점을 해결할 수 있습니다.

Shutdown Port

```
<Server port="${shutdown.port}" shutdown="SHUTDOWN">
```

http port

```
<Connector port="${http.port}" protocol="HTTP/1.1" connectionTimeout="20000"  
    redirectPort="${ssl.port}" />
```

ajp port

```
<Connector port="${ajp.port}" protocol="AJP/1.3" redirectPort="${ssl.port}"/>
```

Tomcat 기본 튜닝

/etc/sysctl.conf

항목	설명	기본값	권장값
net.ipv4.neigh.default.unres_qlen	Increase TCP	6	100
net.ipv4.tcp_keepalive_time	Drop Keep-alive time	7200	30
net.ipv4.tcp_fin_timeout	Drop it so lack of FIN times out quicker	60	10
net.core.netdev_max_backlog	Increase number of incoming connections backlog	1000	2500
net.ipv4.tcp_retries1	How many times to retry killing an alive TCP connection	3	2
net.ipv4.tcp_retries2	How many times to retry killing an alive TCP connection	15	3
net.ipv4.ip_local_port_ran	Increase Local port range	32768 60999	1024 65000

Tomcat 기본 튜닝

/etc/sysctl.conf

항목	설명	기본값	권장값
net.core.rmem_max	Max TCP receive Buffer	25165824	16777216
net.core.rmem_default	Default TCP receive Buffer	8388608	16777216
net.core.wmem_max	Max TCP Send Buffer	2516582	16777216
net.core.wmem_default	Default TCP send Buffer	212992	16777216
net.ipv4.tcp_window_scaling	Enable really big(>65kb) TCP window scaling	0	1
net.ipv4.tcp_timestamps	Turn off timestamp	1	0
net.ipv4.tcp_sack	Turn off tcpsack	1	0
net.ipv4.tcp_orphan_retries	유저 파일 핸들에 할당되지 않은 연결에 몇 번 재시도할지	7	0
vm.swappiness	Swap 사용량 결정	10	1

Tomcat 기본 튜닝

/etc/security/limits.conf(ulimit -a)

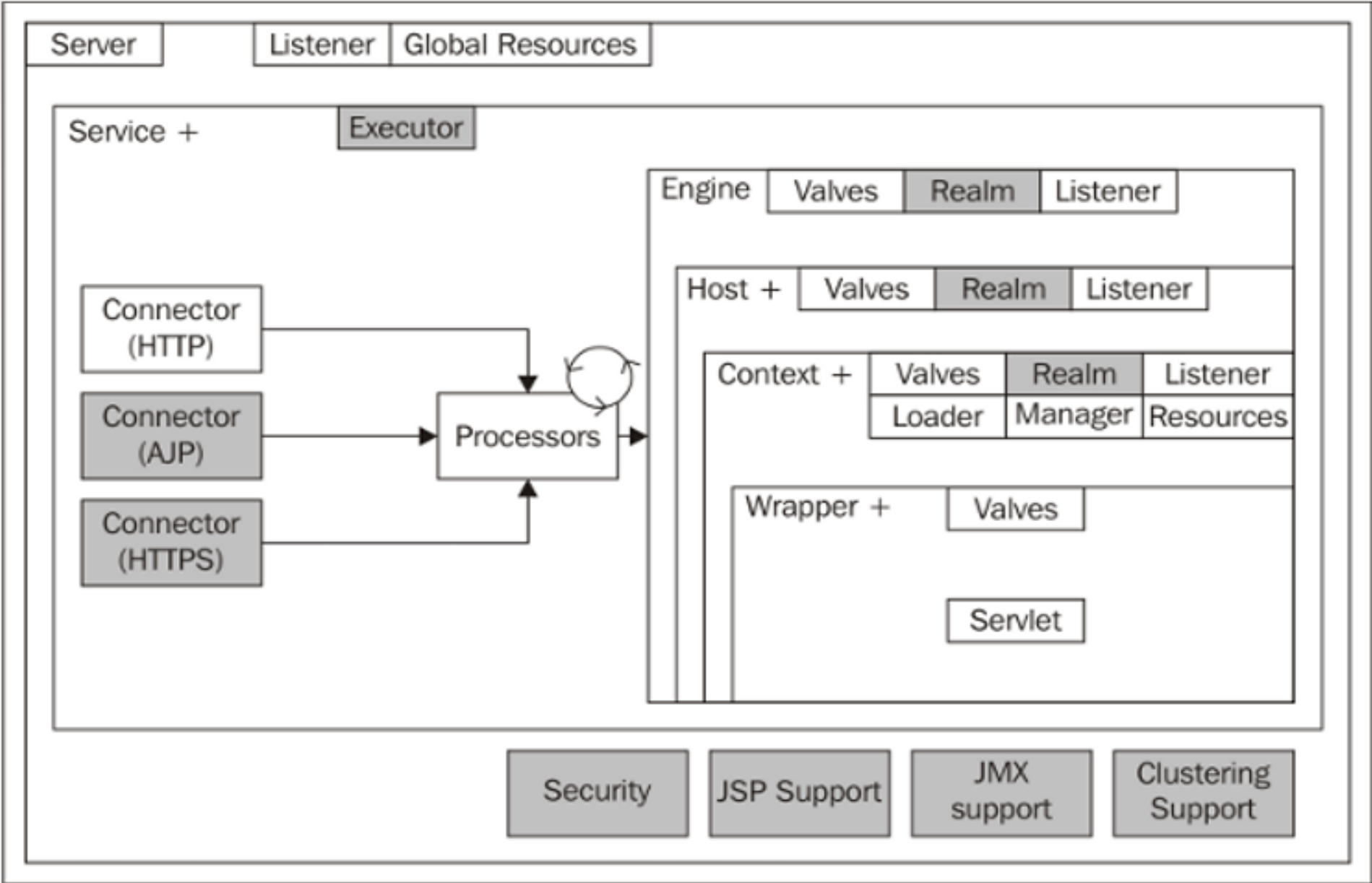
asmanager	soft	nofile	8192
asmanager	hard	nofile	65535
asmanager	soft	nproc	8192
asmanager	hard	nproc	65535

공통 JVM 옵션 설정

항목	설명
client/server	-client : JVM hotspot 의 client 모드로 사용, client 모드는 Application 시작 시간과 메모리 사용량을 최소로 수행
	-server : 컴파일 등 최적화를 수행하여 최적의 성능으로 수행
Xms	초기 메모리 크기(1024M)
Xmx	최대 메모리 크기(1024M)
XX:+PrintGCDetails	Garbage Collection 시 조금 더 상세한 항목의 로그 출력을 위한 설정
XX:+UseG1GC	- G1 GC를 사용하기 위한 설정
XX:+HeapDumpOnOutOfMemoryError	JVM 에서 OutOfMemory 에러가 발생할 경우 Heap Memory Dump 를 생성할지 여부를 결정
XX:HeapDumpPath	Heap Memory 생성시 생성 위치
Xloggc	GC 로그를 별도의 파일로 저장하기 위한 옵션 Memory 사용 추이를 확인할 수 있음.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처

Tomcat Architecture



Tomcat Architecture

- Server
 - ✓ 서버의 개념으로 Tomcat 컨테이너 자체를 나타냄
- Service
 - ✓ 서버 내에 존재하는 컴포넌트에 대한 중재자 역할을 수행하며, 엔진에 하나 혹은 그 이상의 커넥터 사용
- Connector
 - ✓ 클라이언트와의 프로토콜 통신을 처리하며, HTTP 및 AJP 커넥터를 가짐
- Engine
 - ✓ 특정 서비스를 위한 요청 처리 파이프라인을 나타냄
 - ✓ 하나의 서비스는 여러 개의 커넥터를 가질 수 있으며, 엔진은 이러한 커넥터들로부터 모든 요청을 수신하고 처리함
- Host
 - ✓ 특정 네트워크의 이름을 Tomcat 서버에 할당
 - ✓ 하나의 엔진의 여러 개의 호스트를 가질 수 있으며, 호스트의 요소는 도메인등과 같이 네트워크 별칭을 지원
- Context
 - ✓ 웹 애플리케이션을 나타내며, 하나의 Host는 여러 개의 Context를 가질 수 있음

Tomcat을 시작하면 Tomcat이 실행되는 Java Virtual Machine(JVM) 인스턴스에 전체 Tomcat 서버를 나타내는 **Singleton** Server 최상위 수준 요소가 포함된다. 서버는 일반적으로 수신 요청을 단일 Catalina 서블릿 엔진으로 전달하는 하나 이상의 Connector(HTTP, HTTPS, AJP)를 결합하는 구조 요소 Service를 하나만 포함한다.

Engine은 Tomcat 내의 핵심 요청 처리 코드를 나타내며 그 안에 있는 여러 virtual host의 정의를 지원한다. virtual host는 하나의 실행 중인 Tomcat engine을 통해 하나의 시스템에서 호스팅되는 여러 개의 개별 도메인으로 보이도록 한다.

Virtual Host는 차례로 배포되는 Context라고 하는 여러 웹 애플리케이션을 지원할 수 있다. 컨텍스트는 단일 압축 WAR(Web Application Archive) 파일 또는 압축하지 않은 디렉토리로 서블릿 규격에 의해 지정한 웹 응용프로그램 형식을 사용하여 표현한다. 또한 컨텍스트를 서블릿 규격에 정의된 web.xml 파일을 사용하여 구성한다.

Context는 차례로 여러 개의 서블릿을 포함하며, 각 서블릿은 Wrapper 컴포넌트로 포장된다.

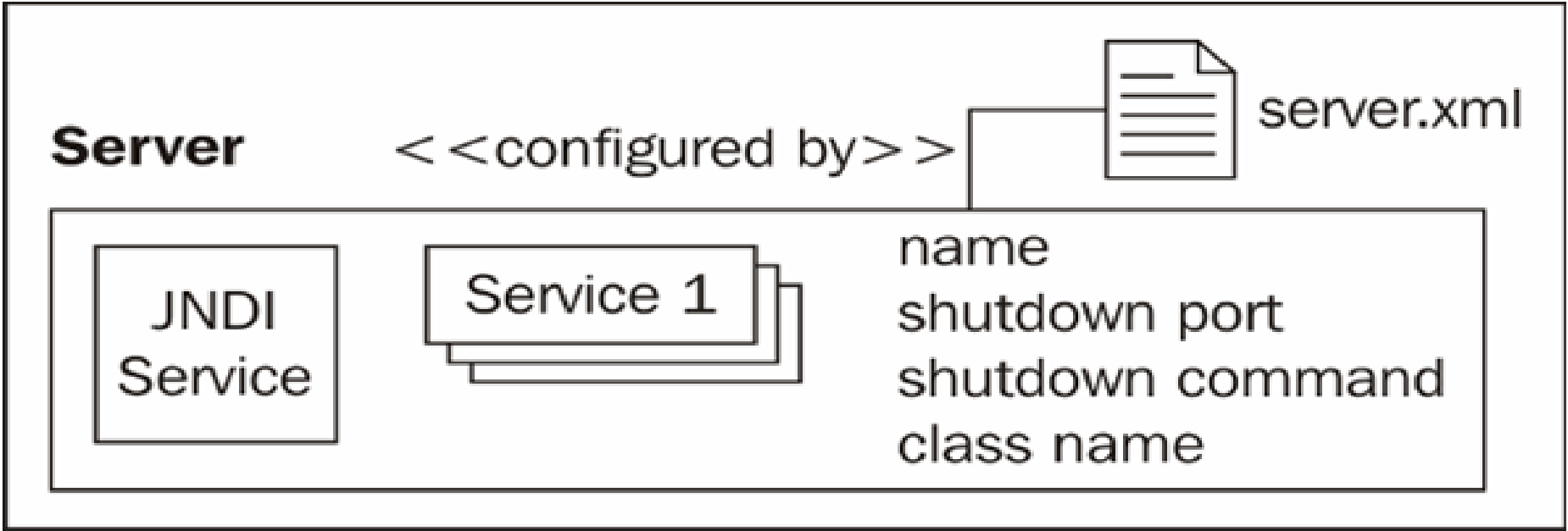
실행 중인 특정 Tomcat 인스턴스에 존재할 Server, Service, Connector, Engine, Host 및 Context는 server.xml 파일을 사용하여 구성한다.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서버

Tomcat Server

Server = Catalina Servlet Container = Tomcat Instance
JVM 안에 Singleton 으로 존재, Server내에 포함되는 Service들 Life Cycle 관리 담당

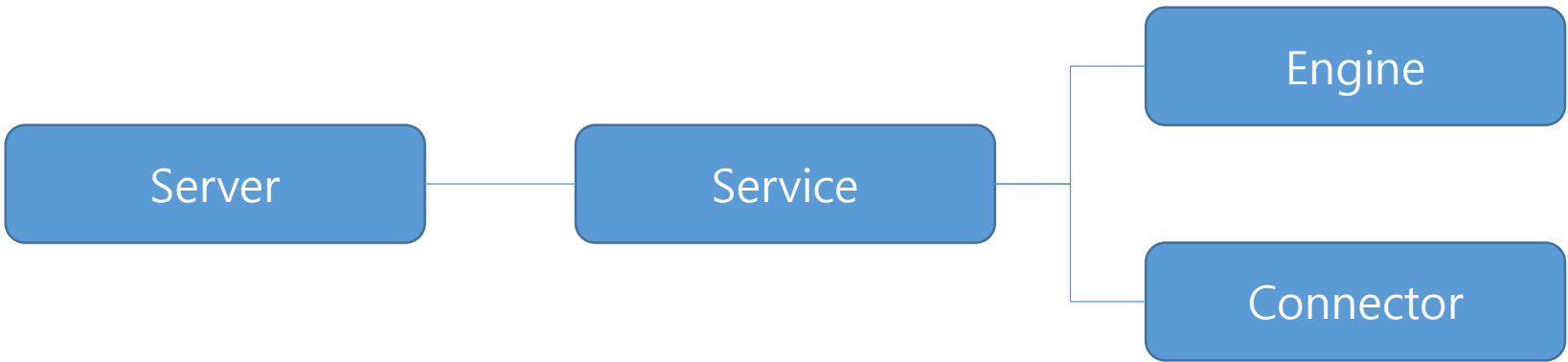
Attribute	Description
className	사용할 구현의 Java 클래스명. 이 클래스는 org.apache.catalina.Server interface를 실행해야 한다. ← 클래스 이름이 지정되지 않은 경우 Default로 사용.
address	서버가 종료 명령을 대기하는 TCP/IP 주소. 주소가 지정되지 않은 경우 localhost가 사용.
port	서버가 종료 명령을 대기하는 TCP/IP 포트 번호. 종료 포트를 사용하지 않으려면 -1로 설정.
shutdown	Tomcat을 종료하기 위해 지정된 포트 번호에 TCP/IP 연결로 수신되어야 하는 명령 문자열.



Tomcat Service

서버 내부에서 하나 이상의 커넥터를 하나의 엔진에 연결하는 중간 설정이다. Tomcat의 기본 구성은 HTTP 와 AJP 커넥터를 Catalina 엔진에 연결하는 서비스를 포함한다. 따라서 커넥터와 엔진은 서비스 요소의 하위 요소다.

Attribute	Description
className	사용할 구현의 Java 클래스 이름. 이 클래스는 org.apache.catalina.Service 를 실행한다.서비스 인터페이스. 클래스 이름이 지정되지 않은 경우 표준 구현 사용
name	표준 Catalina 구성 요소를 사용하는 경우 로그 메시지에 포함되는 이 서비스의 표시 이름. 특정 서버와 관련된 각 서비스의 이름은 고유해야 한다. <Service name="Catalina">



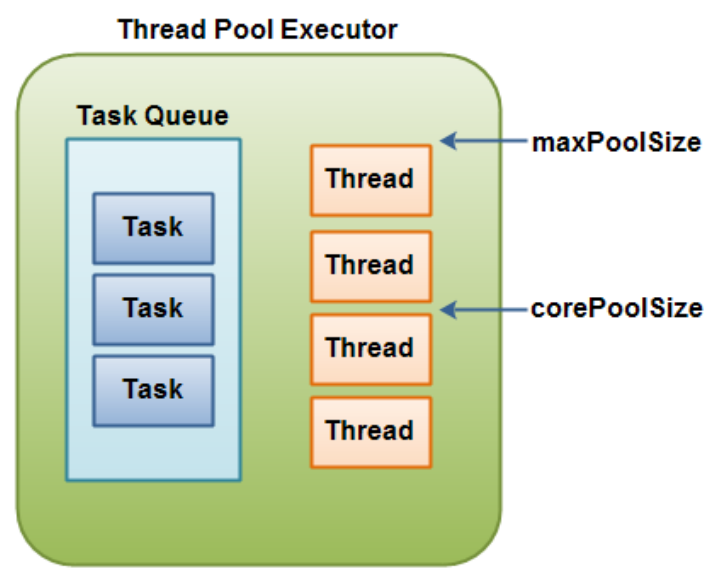
Tomcat Connector

병렬 처리

병렬 작업 처리가 많아 지면 스레드 개수가 증가되고 그에 따른 스레드 생성과 스케줄링으로 인해 CPU가 바빠 메모리 사용량이 늘어납니다. 따라서 어플리케이션의 성능이 저하한다. 갑작스런 병렬 작업의 극대화로 인한 스레드 증폭을 막으려면 스레드 풀(Thread Pool)을 사용해야 한다. 스레드 풀은 작업 처리에 사용되는 스레드를 제한된 개수만큼 정해 놓고 작업 큐(Queue)에 들어오는 작업들을 하나씩 스레드가 맡아 처리합니다.

작업 처리가 끝난 스레드는 다시 작업 큐에서 새로운 작업을 가져와 처리합니다.

ThreadPoolExecutor



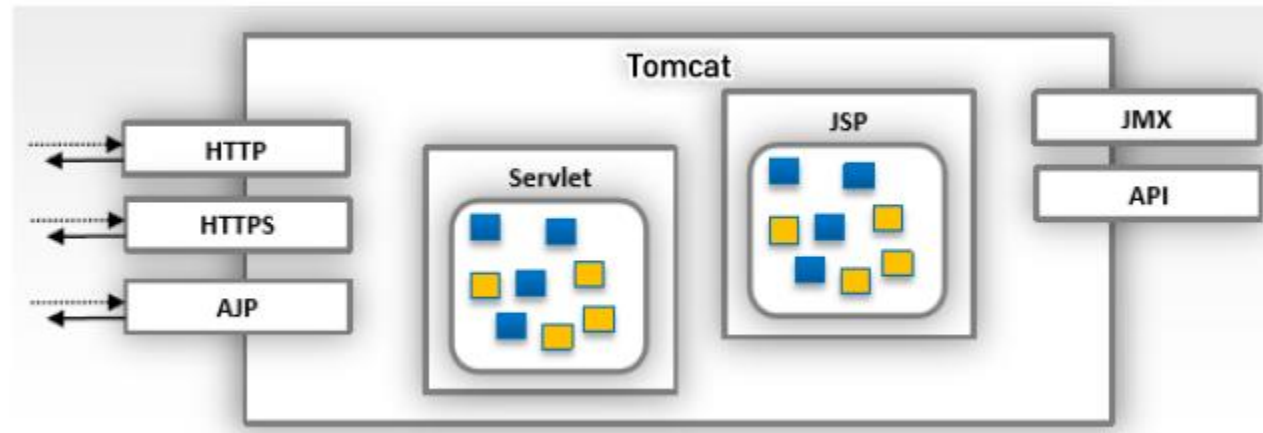
- Thread Pool을 정교히 조작하여 사용할 수 있다. (max thread 수, cache 사용여부 등)
- Thread 개수를 한정할 수 있으므로 메모리 사용량이 적어지고 이를 통해 성능 향상이 되는 것 같다.
- 풀 안에 들어갈 thread 개수가 max 값을 넘었을 때 생성된 thread에 대해 RejectedExecutionHandler의 rejectedExecution() 메소드 안에서 처리할 수 있다.
- 이 외에도 특정 태스크가 종료되기를 기다리거나, 태스크가 끝날 때마다 그 결과를 차례로 가져올 수 있는 등 Executor 안의 다양한 메소드를 이용하여 다양한 작업을 할 수 있다.

SynchronousQueue
LinedBlockingQueue
ArrayBlockingQueue

각 태스크가 상호의 실행에 영향을 주지 않는다.

Tomcat Connector

Tomcat Connectors (3-Main Connectors)



- HTTP
 - ✓ Application 서버로의 direct request를 받음
 - ✓ Default: enabled, 8080 port
- HTTPS
 - ✓ HTTPS를 직접 입력으로 받음, SSL keystore 필요
 - ✓ Default: disabled, 8443 port
- AJP
 - ✓ Apache Web Server를 통해 mod_jk를 통해 입력 받는 커넥터
 - ✓ Default: enabled, 8009 port

Tomcat Connector

- Tomcat Connector configuration

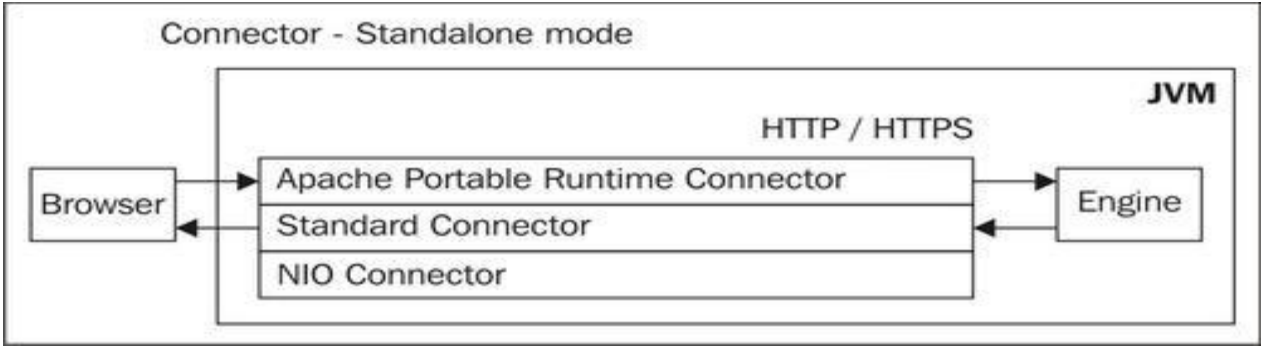
```
<Connector enableLookups="false" //이 설정은 remote ip에 대해서 DNS lookup 을 하여 성능에 영향을 준다.  
    protocol="AJP/1.3" // AJP handler를 사용하기 위해서는 반드시 AJP/1.3 이어야 합니다.  
    URIEncoding="UTF-8" // URI Encoding Type 을 지정 합니다(GET Method 한글 깨짐).  
    connectionTimeout="5000" // 설정하지 않을 경우 기본 60초로 설정 됨. 5초로 connectionTimeout 지정을 합니다.  
    maxSpareThreads="75" // idle 상태로 유지 할 max thread pool size, 설정하지 않을 경우 50으로 설정  
    maxThreads="500" // 동시 요청에 의해 Connector 가 생성할 수 있는 최대 request size 입니다.  
    minSpareThreads="25" // tomcat 이 실행 될때 생성되는 thread size 입니다.  
    port="8100" // httpd 와 tomcat 연동 시 사용할 port  
</>
```

- Connector Protocol
 - HTTP/HTTPS (HTTP/1.1)
 - AJP (AJP/1.3)
- Connector 처리 방식
 - Java Blocking I/O (BIO) : All I/O operations are blocking in processor thread (Read, Write, etc.) → Simple, stable, mature
 - Java Non-blocking I/O (NIO) : Allows huge number of parallel requests
 - Java NIO.2 (NIO2) : like the NIO connector but uses the NIO2 framework
 - Native / Apache Portable Runtime (APR) : Native code (C), Use JNI

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터

Tomcat Connector

Tomcat Connector – HTTP Protocol



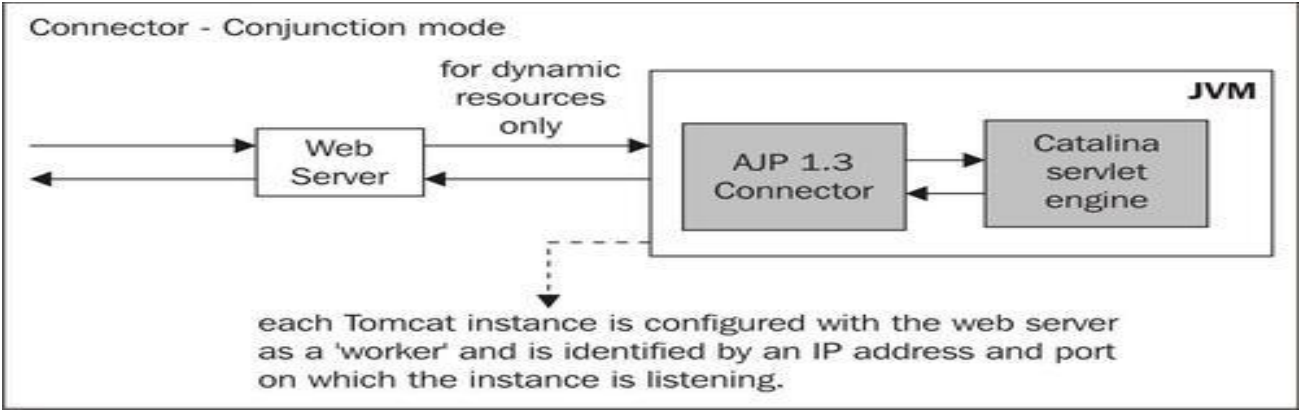
```
<Service name="Catalina">  
  <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />  
  ...  
</Service>
```

항목	설명	기본값
protocol	HTTP, AJP 등 Connector의 Protocol을 설정하는 항목으로 구현체를 직접 설정도 가능함 -BIO(java connector) : org.apache.coyote. http11.Http11Protocol -NIO(java connector) : org.apache.coyote. http11.Http11NioProtocol -NIO2(java connector) : org.apache.coyote. http11.Http11Nio2Protocol -APR(native connector) : org.apache.coyote. http11.Http11AprProtocol	HTTP/1.1

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터

Tomcat Connector

Tomcat Connector – AJP Protocol



```
<Service name="Catalina">  
  <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />  
  ...  
</Service>
```

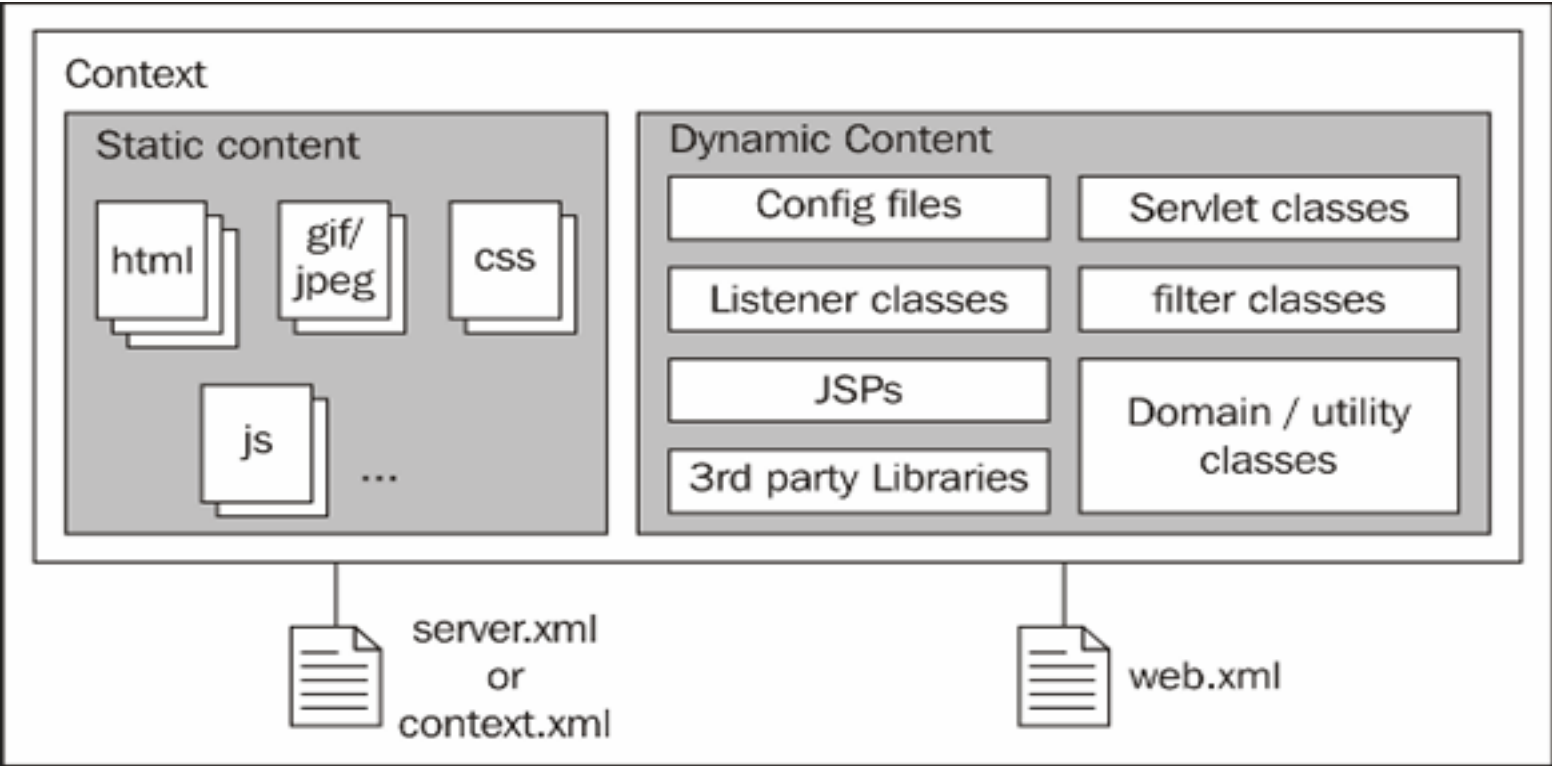
항목	설명	기본값
protocol	HTTP, AJP 등 Connector의 Protocol을 설정하는 항목으로 구현체를 직접 설정도 가능함 -BIO : org.apache.coyote ajp.AjpProtocol -NIO : org.apache.coyote ajp.AjpNioProtocol -NIO2 : org.apache.coyote ajp.AjpNio2Protocol -APR : org.apache.coyote ajp.AjpAprProtocol	AJP/1.3

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트

Tomcat Context

Web Application., Spec 상 ServletContext와 매핑. WAS/EAR 파일이나, Exploded된 디렉토리
<Host>안에 <Context> 설정이 없을 경우 기본적으로 webapps 하위의 ROOT 디렉토리(or ROOT.war)을 기본경로로 설정한다.

```
<Context path="" docBase="/test" debug="0" reloadable="true" crossContext="true" swallowOutput="true">
<!--
path : 요청 URI의 시작부분이 컨텍스트 경로와 같을때 해당 웹어플리케이션이 그 요청을 처리. http://localhost/컨텍스트의 path/\*\*\*\*.jsp
docBase : 파일들이 존재하는 디렉토리 절대경로나 host의 appBase의 상대경로를 입력
reloadable : 클래스들의 변경시 자동으로 재적재
crossContext : ServletContext.getContext()를 통해 현재 Host내의 다른 Context에 대한 Request Dispatcher를 얻을 수 있다.
swallowOutput : 이 값이 true이면 System.out 또는 System.err의 텍스트는 모두 Logger로 재지정. Logger에서 지정한 파일에 저장, Console에는 안보임.
-->
```



Tomcat Context

Application Deploy

Tomcat 인스턴스는 \$CATALINA_BASE/webapps 또는 다른 경로일 경우 server.xml에 context 를 지정하여 deploy 설정이 가능합니다.

Web Archive

- Catalina.base/webapps 디렉토리에 .war 파일을 복사하여 디플로이

Exploded

- Catalina.base/webapps 디렉토리에 exploded 형태의 디렉토리 구조를 사용

```
<Host name="localhost" appBase="webapps" unpackWARs="false" autoDeploy="false">  
  <Context path="" docBase="/opt/was/apps/test.war" reloadable="false"/>  
</Host>
```

항목	설명
path	컨텍스트 경로에 대한 설정
docBase	애플리케이션 document 경로 설정
reloadable	클래스 변경 시 마다 자동 reload

Tomcat Listener

<Server>의 하위 컴포넌트. 특정 이벤트를 리스닝하고 이벤트 발생 시 응답한다.

<!-- Tomcat이 시작할 때 Tomcat, Java 및 운영 체제 정보를 기록한다. -->

<Listener className="org.apache.catalina.startup.VersionLoggerListener" />

<!-- APR Lifecycle Listener는 APR/native library의 존재 여부를 확인하고, 존재하는 경우 라이브러리를 로드한다.

<Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="off" />

<!-- JRE 메모리 누수 처리 -->

<Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />

<!-- 글로벌 리소스를 지원하고 데이터베이스와 같은 리소스에 액세스하는 데 JNDI를 사용할 수 있도록 한다. -->

<Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />

<!-- 컨텍스트가 중지될 때 Executor pools의 스레드 갱신을 트리거하여 스레드 로컬 관련 메모리 누출을 방지한다-->

<Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

<!-- root 사용자로 start 하지 못하게 함 -->

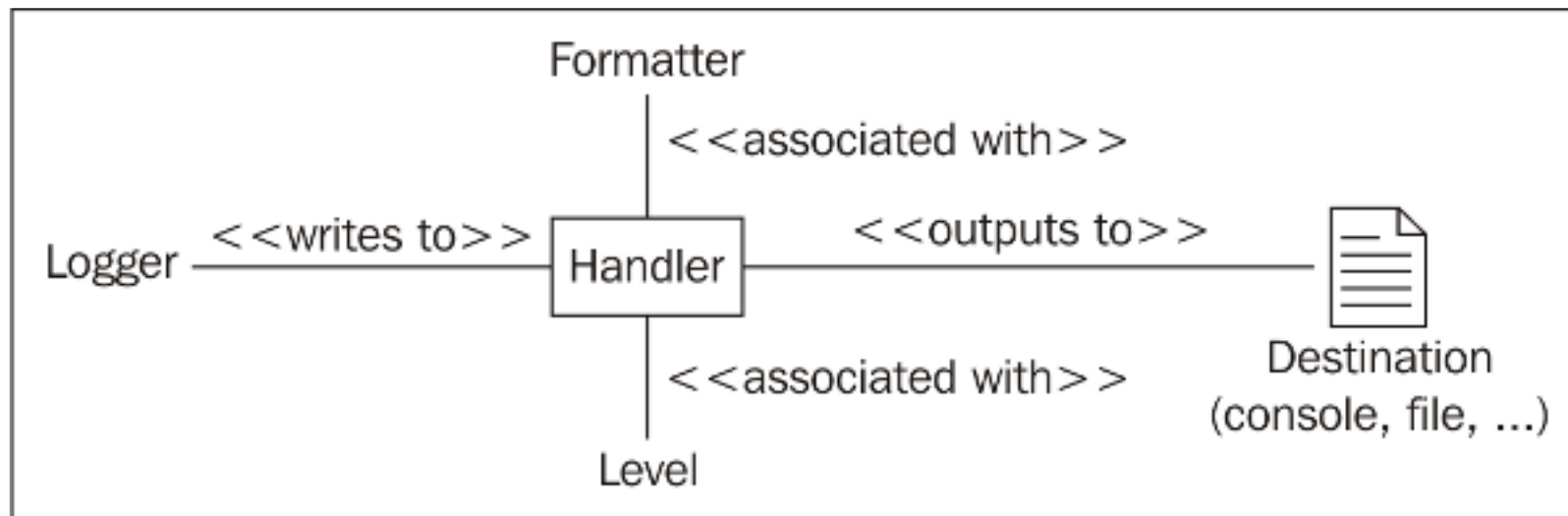
<Listener className="org.apache.catalina.security.SecurityListener" checkedOsUsers="root" />

Tomcat Logging

- **JULI Logging Framework 사용**

Apache Commons Logging 기반으로 구현 → java.util.logging 사용 (default)
extra 패키지를 통해 Log4j 등 Logging Framework 변경 가능

- **JULI Logging Architecture**



Tomcat Logging

- **logging.properties**

Java 1.4 부터 java.util.logging이라는 java Logging Framework이 제공한다. 별도 설치가 필요없다는 장점이 있는 반면에 JVM 레벨의 설정만 가능하다는 단점이 있다. 즉, 웹 애플리케이션 단위로는 로그를 남길 수 없다.

Tomcat 은 이러한 Java Logging 의 단점을 해결하기 위하여 Java Logging 을 확장한 JULI 라는 이름의 로그 관리를 사용하고 있다. 때때로 JULI 는 Java Logging의 Tomcat 버전으로 표현되기도 하며, Java Logging의 설정 방법과 기본적으로 동일하다. JULI 는 \${INSTANCE_DIR}/bin/tomcat-juli.jar 파일로 존재하고 있다.

JULI 는 Java Logging 설정 뿐 아니라 classloader 당 설정이 가능하다. 따라서 다음과 같이 logging.properties를 위치시킬 수 있다.

- **ClassLoaderLogManager**

java.util.logging.LogManager를 확장

- \$CATALINA_BASE/conf/logging.properties 파일 Loading 하여, Tomcat에 적용
- ClassLoader의 path에서 logging.properties를 찾아, ClassLoader별 Log 구성을 적용
 - > 웹어플리케이션 별 log Configuration 구현

```
# -----  
# LOGGING_MANAGER (Optional) Override Tomcat's logging manager  
# Example (all one line)  
# LOGGING_MANAGER="-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"  
# -----  
if [ -z "$LOGGING_MANAGER" ]; then  
    LOGGING_MANAGER="-Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager"  
fi
```

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그

Tomcat Logging

- logging.properties

```
handlers = 1catalina.org.apache.juli.AsyncFileHandler, 2localhost.org.apache.juli.AsyncFileHandler, java.util.logging.ConsoleHandler

.handlers = 1catalina.org.apache.juli.AsyncFileHandler, java.util.logging.ConsoleHandler

#####
# Handler specific properties.
# Describes specific configuration info for Handlers.
#####

1catalina.org.apache.juli.AsyncFileHandler.level = FINE
1catalina.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
1catalina.org.apache.juli.AsyncFileHandler.prefix = catalina.

2localhost.org.apache.juli.AsyncFileHandler.level = FINE
2localhost.org.apache.juli.AsyncFileHandler.directory = ${catalina.base}/logs
2localhost.org.apache.juli.AsyncFileHandler.prefix = localhost.

#####
# Facility specific properties.
# Provides extra control for each logger.
#####
java.util.logging.ConsoleHandler.level = FINE
java.util.logging.ConsoleHandler.formatter = org.apache.juli.OneLineFormatter

org.apache.catalina.core.ContainerBase.[Catalina].[localhost].level = INFO
org.apache.catalina.core.ContainerBase.[Catalina].[localhost].handlers = 2localhost.org.apache.juli.AsyncFileHandler
```

Tomcat Logging

- Tomcat Logging 종류

java.util.logging, javax.servlet.ServletContext.log, Console, Access Logging 이 제공됨

구분	설명	파일명
Java Logging API	Tomcat에서는 Application당 로깅 설정을 제어하기 위해 java.util.logging 기반의 구현체 JULI를 사용 logging.properties 파일의 설정에 따름	ex) localhost.yyyy-MM-dd.log
Servlets logging API	javax.servlet.ServletContext.log(...)를 통해 메시지를 출력할 때 사용됨 logging.properties 파일의 설정 중 org.apache.catalina.core.ContainerBase.[\${engine}].[\${host}].[\${context}] 의 설정에 따름	ex) manager.yyyy-MM-dd.log
Console	Tomcat운영 시 출력되는 STDERR / STDOUT	ex) catalina.out
Access logging	AccessLogValve 또는 ExtendedAccessLogValve에서 생성하는 로그 Access Log Valve의 설정에 따름	ex) access_log.yyyy-MM-dd

Why is JDBC Connection Pool?

웹 어플리케이션에서 DB 서버에 접근을 시작하고 데이터를 가져오기까지의 단계에서 가장 비용이 드는 부분이 어디일까?

```
String query = "SELECT ... where id = ?";

try {
    //1. DB 서버 접속을 위해 JDBC 드라이버를 로드한다
    Class.forName("com.mysql.jdbc.Driver");

    //2. DB 접속 정보와 DriverManager.getConnection() Method를 통해 DB Connection 객체를 얻는다.
    Connection conn = null;
    conn = DriverManager.getConnection("jdbc:mysql://localhost/dev", "test", "test");

    //3. Connection 객체로 부터 쿼리를 수행하기 위한 PreparedStatement 객체를 받는다
    PreparedStatement ps = con.prepareStatement(query);
    ps.setString(1, id);

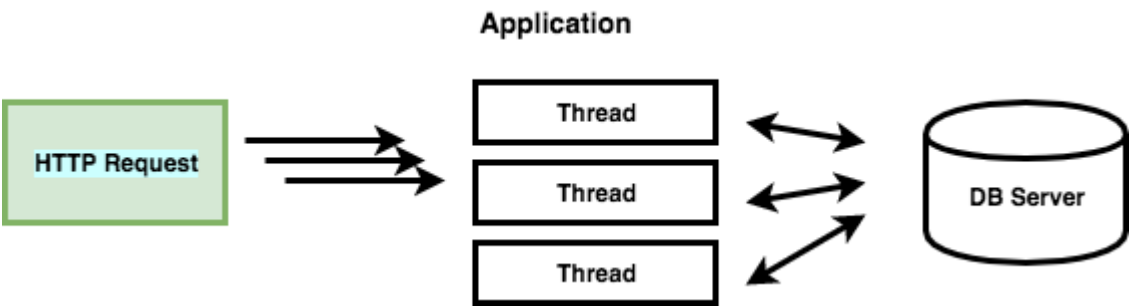
    //4. executeQuery를 수행하여 그 결과로 ResultSet 객체를 받아서 데이터를 처리한다.
    ResultSet rs = get.executeQuery();

} catch (Exception e) { }

} finally {
    //5. 처리가 완료되면 처리에 사용된 리소스들을 close하여 반환한다.
    rs.close();
    ps.close();
    conn.close()
}
```


Why is JDBC Connection Pool?

가장 느린 부분은 웹 서버에서 물리적으로 DB서버에 연결하고 Connection 객체를 정리하는 부분이다.




웹 애플리케이션은 HTTP 요청에 따라 할당한 Thread가 처리하는 데, 대부분의 비즈니스 로직은 DB 서버로부터 데이터를 얻게 된다. 만약 위와 같이 모든 요청에 대해 DB접속을 위한 Driver를 로드하고 Connection 객체를 생성하여 연결한다면 물리적으로 DB 서버에 지속적으로 접근해야 될 것이다. 이러한 상황에서 DB Connection 객체를 생성하는 부분에 대한 비용과 대기 시간을 줄이고, 네트워크 연결에 대한 부담을 줄일수 있는 방법이 있는데 바로, Database Connection Pool를 이용하면 이러한 문제를 해결할 수가 있다.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool

Tomcat Connection Pool

- Tomcat은 DataBase Connection Pool로 **Commons DBCP**와 **Tomcat JDBC Pool**의 2개지 Pool을 제공



Apache Commons
http://commons.apache.org/

Apache Commons DBCP™

Last Published: 06 August 2015 | Version: 2.1.1

ApacheCon Apache Commons

COMMONS DBCP

- Overview
- Configuration
 - Javadoc (1.2.2 release)
 - Javadoc (1.3 release)
 - Javadoc (1.4 release)
 - Javadoc (2.0.1 release)
 - Javadoc (2.1 release)
 - Javadoc (2.1.1 release)
- Developers Guide
- Examples
- Downloads
- Wiki
- DEVELOPMENT
 - History
 - Building
 - Mailing lists
 - Issue Tracking
 - Team
 - Source Repository

The DBCP Component

Many Apache projects support interaction with a relational database. Creating a new connection for each user can be time consuming (often requiring multiple seconds of clock time), in order to perform a database transaction that might take milliseconds. Opening a connection per user can be unfeasible in a publicly-hosted Internet application where the number of simultaneous users can be very large. Accordingly, developers often wish to share a "pool" of open connections between all of the application's current users. The number of users actually performing a request at any given time is usually a very small percentage of the total number of active users, and during request processing is the only time that a database connection is required. The application itself logs into the DBMS, and handles any user account issues internally.

There are several Database Connection Pools already available, both within Apache products and elsewhere. This Commons package provides an opportunity to coordinate the efforts required to create and maintain an efficient, feature-rich package under the ASF license.

The `commons-dbcp2` package relies on code in the `commons-pool2` package to provide the underlying object pool mechanisms that it utilizes.

DBCP now comes in three different versions to support different versions of JDBC. Here is how it works:

- DBCP 2 compiles and runs under Java 7 only (JDBC 4.1)
- DBCP 1.4 compiles and runs under Java 6 only (JDBC 4)
- DBCP 1.3 compiles and runs under Java 1.4-5 only (JDBC 3)


DBCP 2 binaries should be used by applications running under Java 7.

DBCP 1.4 binaries should be used by applications running under Java 6.

DBCP 1.3 should be used when running under Java 1.4-5.

Commons DBCP

(http://commons.apache.org/proper/commons-dbcp/)



Apache Tomcat 8
Version 8.0.28, Oct 7 2015

The Tomcat JDBC Connection Pool

Version 8.0.28, Oct 7 2015

The Tomcat JDBC Connection Pool

Table of Contents

- Introduction
- How to use
 - Additional features
 - Inside the Apache Tomcat Container
 - Standalone
 - JMX
- Attributes
 - JNDI Factory and Type
 - System Properties
 - Common Attributes
 - Tomcat JDBC Enhanced Attributes
- Advanced usage
 - JDBC interceptors
 - Configuring JDBC interceptors
 - org.apache.tomcat.jdbc.pool.jdbcinterceptor
 - org.apache.tomcat.jdbc.pool.interceptor.ConnectionState
 - org.apache.tomcat.jdbc.pool.interceptor.StatementFinalizer
 - org.apache.tomcat.jdbc.pool.interceptor.StatementCache
 - org.apache.tomcat.jdbc.pool.interceptor.StatementDecoratorInterceptor
 - org.apache.tomcat.jdbc.pool.interceptor.QueryTimeoutInterceptor
 - org.apache.tomcat.jdbc.pool.interceptor.SlowQueryReport
 - org.apache.tomcat.jdbc.pool.interceptor.SlowQueryReportJmx
 - org.apache.tomcat.jdbc.pool.interceptor.ResetAbandonedTimer

Links

- Docs Home
- FAQ
- User Comments

User Guide

- 1) Introduction
- 2) Setup
- 3) First webapp
- 4) Deployer
- 5) Manager
- 6) Realms and AAA
- 7) Security Manager
- 8) JNDI Resources
- 9) JDBC DataSources
- 10) Classloading
- 11) JSPs
- 12) SSL/TLS
- 13) SSL
- 14) CGI
- 15) Proxy Support
- 16) MBean Descriptor
- 17) Default Servlet
- 18) Clustering
- 19) Load Balancer
- 20) Connectors
- 21) Monitoring and Management
- 22) Logging

The Tomcat JDBC Connection Pool

(https://tomcat.apache.org/tomcat-8.0-doc/jdbc-pool.html)

** Tomcat 7.0.18 – extra package 포함(Tomcat 7.0.19 – 정식 Release)

Tomcat JDBC Connection Pool

- Tomcat은 DataBase Connection Pool로 Commons DBCP와 Tomcat JDBC Pool의 2개지 Pool을 제공
 - **Commons DBCP** : **org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory (default)** -
org.apache.commons -> org.apache.tomcat. 패키지 변경
 - **Tomcat JDBC** : **org.apache.tomcat.jdbc.pool.DataSourceFactory**
- DBCP는 Apache Commons 프로젝트가 제공하는 DataBase Connection Pool로 오픈 소스 DataBase Connection Pool를 가장 많이 사용하고 있다.
- Tomcat JDBC는 Apache Tomcat 프로젝트가 독립적으로 개발한 DataBase Connection Pool 이다.

```
<Resource type="javax.sql.DataSource"  
  name="jdbc/TestDB"  
  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"  
  driverClassName="com.mysql.jdbc.Driver"  
  url="jdbc:mysql://localhost:3306/mysql"  
  username="mysql_user"  
  password="mypassword123" />
```

JDBC Connection Pool 기본 설정

- 공통 DB 접속 설정

구분	설명	비고
username	DB 연결을 위해 JDBC 드라이버에 전달되는 사용자 이름	
Password	DB 연결을 위해 JDBC 드라이버에 전달되는 사용자 패스워드	
url	DB 연결을 위해 JDBC 드라이버에 전달되는 연결 URL	
driverClassName	JDBC 드라이버의 FQCN (Fully Qualified Class Name)	
connectionProperties	DB 연결을 위해 JDBC 드라이버에 전달되는 연결 Properties	

항목	설명	기본값
defaultAutoCommit	생성된 Connection의 자동 Commit Mode	JDBC Dirver 기본값 - Oracle : true - MySQL/MariaDB : true
defaultReadOnly	생성된 Connection의 ReadOnly	JDBC Dirver 기본값
defaultTransactionIsolation	생성된 Connection의 TransactionIsolation	JDBC Dirver 기본값
defaultCatalog	생성된 Connection의 defaultCatalog	JDBC Dirver 기본값

****Commons DBCP** : defaultAutoCommit 의 기본값은 true

JDBC Connection Pool 기본 설정

- 공통 DB 접속 설정

구분	설명	비고
initialSize	Pool 기동 시 생성되는 Connection의 초기 사이즈	10
maxActive	Pool에서 동시에 유지할수 있는 최대 연결 수	100
maxIdle	Pool에서 유지하는 최대 Connection 수	100
minIdle	Pool에서 유지하는 최소 Connection 수	initialSize
maxWait	사용 가능한 Connection이 존재하지 않는 경우 대기하는 최대시간	30,000 (30초)
maxAge	Connection을 보존하는 시간. Connection 반환 시에 maxAge 이상 경과했으면, 해당 Connection을 Close 시킴	0 (msc)
fairQueue	getConnection 호출이 Connection이 FIFO방식으로 공정하게 제공하려면 true로 설정. org.apache.tomcat.jdbc.pool.FairBlockingQueue 로 구현 되어 있으며, 스레드가 도착한 순서대로 Connection을 획득하는 것이 보장됨	true

** fairQueue : 비동기 기반 개발에서 의미

JDBC Connection Pool 기본 설정

- Connection Validation 관련 설정

구분	설명	비고
validationQuery	Connection 유효성 확인을 위한 SQL Query	MySQL/MariaDB : SELECT 1 Oracle : SELECT 1 FROM DUAL
testOnBorrow	Pool에서 고낸 Connection을 전달하기 전에 유효성을 검증 검증에 실패했을 경우, 해당 연결을 삭제하고, 새롭게 Pool에서 Connection을 꺼내려고 시도	False, true 설정 시 validationQuery 필요
testOnReturn	Pool에 Connection을 반환하기 전에 Connection의 유효성을 검증	False, true 설정 시 validationQuery 필요
testWileIdle	Idle 상태의 Connection 에 대한 유효성을 검증. 검증에 실패할 경우 해당 Connection을 파 기함 이 기능은 Evictor 를 이용하기 때문에 Evictor를 활성화 해야됨.	False, true 설정 시 validationQuery 필요
initSQL	Connection을 만들 때 실행되는 SQL Query	
validationClassName	Validation Class명을 지정. 이 속성을 지정 시 validationQuery에 의한 검증 대신, Validator Class에 의한 연결의 검증 이 이루어짐 (org.apache.tomcat.jdbc.pool.Validator 인터페이스 구현 필요)	
validationInterval	Validation의 실행 간격 지정. 마지막 검증으로부터 이 값 이상 경과하지 않은 경우 연결 검증을 거치지 않음	30000 (30초)
logValidationErrors	Validation중 오류 발생시, 로그 출력할지에 대한 설정	False

Idle 상태의 Connection에 관한 파라미터(Eviction)

- PoolCleaner 관련 설정 (Evictor)

항목	설명	기본값
timeBetweenEvictionRunsMilles	Idle 상태의 Connection 을 check 하는 간격,	5,000 (5초)
minEvictableIdleTimeMillis	Idle Connection의 생존 기간	60,000(60초)

* Commons DBCP의 속성 중 numTestsPerEvictionRun 속성 삭제

Connection Leak 검출 설정 (removeAbandoned)

- PoolCleaner 관련 설정 (removeAbandoned)

항목	설명	기본값
removeAbandoned	Connection 누수 감지 설정 사용 여부	false
removeAbandonedTimeout	Connection 누수로 판단하기까지의 시간	60 초
logAbandoned	Connection 누수를 감지했을 때, Connection을 Close하지 않은 어플리케이션의 Stuck Thread를 로그로 출력 여부의 설정	false
suspectTimeout	Connection 누수의 의심에 대한 판단 할 때 까지의 시간 suspectTimeout 시간을 초과한 Connection에 대해 경고 로그 및 JMX 에 통지를 수행	0
abandonWhenPercentageFull	누수된 Connection 을 파기할 비율을 설정 0~100 사이로 설정하며, 이 설정을 초과한 누수 Connection 을 파기, removeAbandonedTimeout가 초과한 Connection은 모두 파기.	0

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool

Tomcat JDBC Connetcion pool 설정 예제

- DataSource Resource 설정

```
<Resource name="${jdbc.resource.name}"
  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
  auth="Container"
  type="javax.sql.DataSource"
  username="${jdbc.username}"
  password="${jdbc.password}"
  driverClassName="${jdbc.driverClassName}"
  url="${jdbc.url}"
  testWhileIdle="true"
  testOnBorrow="true"
  testOnReturn="false"
  validationQuery="SELECT 1"
  validationInterval="30000"
  timeBetweenEvictionRunsMillis="5000"
  maxActive="100"
  minIdle="10"
  maxWait="10000"
  initialSize="10"
  removeAbandonedTimeout="60"
  removeAbandoned="true"
  logAbandoned="true"
  minEvictableIdleTimeMillis="30000"
  jmxEnabled="true"
  jdbcInterceptors="ConnectionState;StatementFinalizer;SlowQueryReportJmx(threshold=10000)"/>
```


Tomcat Data Source 설정

- Global Resource Level
 - DataSource Pool Global level 공유
 - Global Resource(server.xml) + Resouce Link(application-context.xml)
 - 해당 Application 에서만 사용 가능
 - Global Resource(server.xml) + Resouce Link(Global context.xml)
 - 모든 Application에서 사용 가능
- Context Level
 - DataSource Pool 해당 Context 별 생성됨
 - Global Context (context.xml)
 - 모든 Application에서 사용 가능
 - Application Context (application-context.xml)
 - 해당 Application에서만 사용 가능

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool

Global Resource Level(+ Resouce Link

- Global Resource Level (+ Resouce Link)
- Server : \${CATALINA_HOME}/conf/server.xml

```
<Server port="${port.shutdown}" shutdown="ARGO_INSTANCE_SHUTDOWN">
  <GlobalNamingResources>
    <Resource name="jdbc/tak"
      driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://10.0.1.88:3306/oom"
      username="root"
      password="uCO5FkZ3NfgDo/LWB0l+0A=="
      .....
    />
  </GlobalNamingResources>
</Server>
```

- Context(1) : \${CATALINA_HOME}/conf/context.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<Context>
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <ResourceLink global="jdbc/tak" name="jdbc/tak" type="javax.sql.DataSource"/>
</Context>
```

- Context(2) : \${CATALINA_HOME}/conf/Catalina/localhost/\${CONTEXT_NAME}.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <Context docBase="/src/argo/application/sample-simpleweb" path="/" reloadable="false">
    <ResourceLink global="jdbc/argo" name="jdbc/argo" type="javax.sql.DataSource"/>
  </Context>
```

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool

Context Level

- Context Level
- Context(1) : \${CATALINA_HOME}/conf/context.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Context>

    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <Resource name="jdbc/tak"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://10.0.1.88:3306/oom"
        username="root"
        password="uCO5FkZ3NfgDo/LWB0l+0A=="
        .....
    />

</Context>
```

- Context(2) : \${CATALINA_HOME}/conf/Catalina/localhost/\${CONTEXT_NAME}.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Resource name="jdbc/tak"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://10.0.1.88:3306/oom"
    username="root"
    password="uCO5FkZ3NfgDo/LWB0l+0A=="
    .....
/>
</Context>
```

Tomcat JDBC 로 변경해야 하는 이유.

- Commons DBCP 는 Single Thread를 사용한다. 그래서 Thread가 안전할 수 있도록 전체 풀 (pool)에 대해서 lock을 건다. 멀티 코어/CPU 환경에서 성능이 떨어진다. (느리다) → (not DBCP 2.x)
- Commons DBCP 는 복잡하다. Commons DBCP 는 관리해야 할 클래스가 60여개, Tomcat 은 8개의 핵심클래스만 사용한다.
- Tomcat JDBC는 Common DBCP 에서 크게 변경 없이 전환이 가능하며, 추가적으로 JMX 지원 확장 등으로 외부에서 Connection Validation, JdbcInterceptor를 통한 DB처리의 독립적인 처리 추가 기능 등 다양한 기능이 추가되어 있음
- 소스 코드도 Common DBCP에 비해, 상당히 간단하며, 릴리즈도 Tomcat 과 동시에 되기 때문에, 버그 수정 등 유지 보수에 대한 안정성/신뢰성을 제공함.

※ Commons DBCP vs Tomcat JDBC :

<http://vigilbose.blogspot.kr/2009/03/apache-commons-dbcip-and-tomcat-jdbc.html>

<http://people.apache.org/~fhanik/tomcat/jdbc-pool.html>

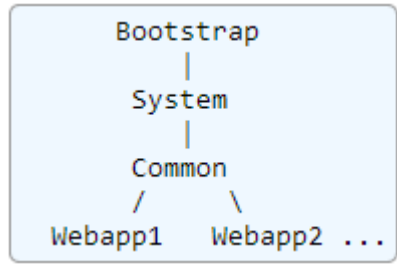
Java ClassLoader.

자바 바이트코드를 읽어 클래스 객체를 생성하는 역할을 담당한다. 자바는 동적으로 클래스를 읽는다. 런타임 시에 모든 코드가 JVM에 링크, 로딩된다. 이러한 동적인 클래스 로딩을 담당하는 것이 Class Loader이며, 자바는 기본적으로 java.lang.ClassLoader라는 Class Loader를 제공한다.

역으로 동적으로 클래스를 로딩한다는 것은 JVM은 클래스에 대한 정보(메소드, 필드, 상속관계)를 가지고 있지 않다는 것을 의미한다. 클래스 로딩 방식에는 load-time dynamic loading과 run-time dynamic loading이 있다. load-time dynamic loading은 클래스를 로딩하는 과정에서 다시 필요한 클래스를 로딩한다(new 연산자를 쓰기 위해). run-time dynamic loading은 클래스 내 코드를 실행하는 과정에서 필요한 클래스를 로딩한다(Class.forName).

클래스로더가 복수 개 존재할 때 공통으로 사용되는 클래스나 리소스들을 모두 각자 로딩한다면 해당 자원들을 중복되게 로딩함으로써 JVM의 permanent generation를 낭비하게 된다. 하지만 부모 클래스로더에서 해당 자원들을 먼저 찾는 방식으로 동작할 경우 복수 개 클래스로더의 부모가 모두 같을 경우 부모에 로드된 자원을 함께 사용하게 되므로 자원의 낭비를 막을 수 있다. 이러한 방식의 클래스로딩 메커니즘을 **parent-first / child-last 방식**이라고 명명하도록 하겠다.

톰캣에서는 parent-first / child-last 방식의 클래스로딩 메커니즘이 아닌 parent-last / child-first 방식의 클래스로딩 메커니즘을 사용한다.



Bootstrap
System
/WEB-INF/classes
/WEB-INF/lib/*.jar
Common

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 설치
톰캣 로그
톰캣 DB Pool
톰캣 클래스로더

Java ClassLoader.

Tomcat Classloader 순서

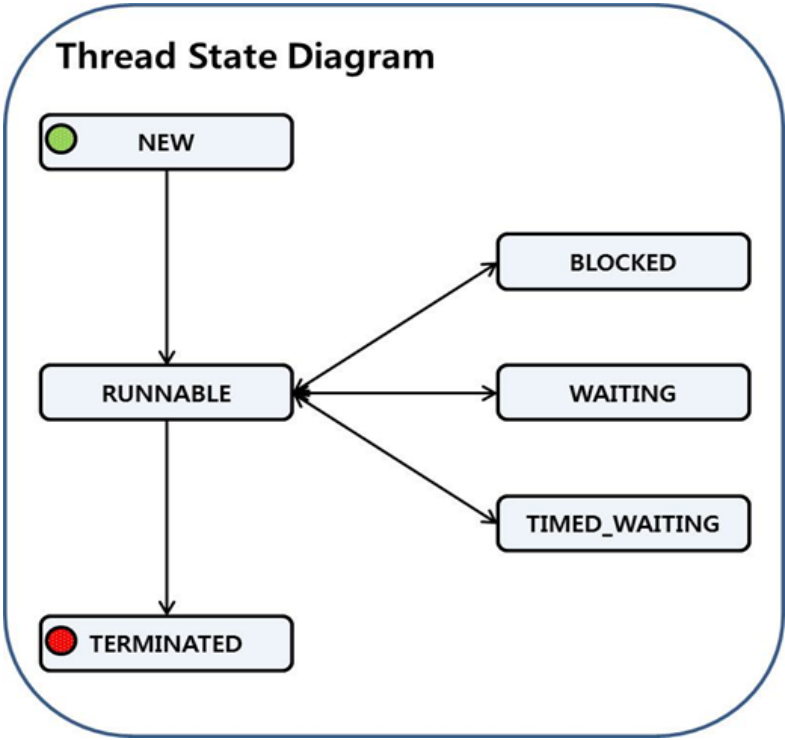
1. Bootstrap
2. WEB-INF/classes
3. WEB-INF/lib
4. System
5. \$CATALINA_BASE/lib 하위의 unpacked 클래스
6. \$CATALINA_BASE/lib 하위의 jar
7. \$CATALINA_HOME/lib 하위의 unpacked 클래스
8. \$CATALINA_HOME/lib 하위의 jar

System Class Loader에 의해 로딩되는 클래스는 다음과 같다.

- \$CATALINA_HOME/bin/bootstrap.jar
- \$CATALINA_HOME/bin/tomcat-juli.jar (만약 \$CATALINA_BASE/bin/tomcat-juli.jar가 존재한다면 이것을 로딩함)
- \$CATALINA_HOME/bin/commons-daemon.jar (존재 시)

스레드 덤프

- JVM상에서 실행되는 스레드(Thread)들의 현재 상태를 덤프로 추출하는 행위
- 통상 스레드의 상태 변화를 확인하려면 5초 이상 간격으로 5~10회 정도 스레드 덤프를 생성
- 스레드 상태



[스레드 상태 다이어그램]

상태	설명
NEW	스레드가 생성되었으나 아직 실행되지 않음
RUNNABLE	현재 CPU를 점유하며 실행 중, OS의 자원분배로 WAITING상태가 될 수도 있음
BLOCKED	Monitor를 소유하기 위해 다른 스레드가 잠금 해제하기를 기다림
WAITING	wait(), join(), park() 메서드 등을 이용해 대기하는 상태
TIMED_WAITING	sleep(), wait(), join(), park() 메서드 등을 이용해 대기하는 상태, 메서드 파라미터로 최대 대기 시간을 명시할 수 있어 시간에 따라 WAITING 상태가 해제될 수 있음
TERMINATED	작동을 끝마친 상태

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프

스레드 덤프 생성

- jstack

\$ jstack {pid}

- kill

\$ kill -3 {pid} 또는 kill -quit {pid}

```
[user@localhost ~] jstack 11655
```

```
2018-12-02 23:44:20
```

```
Full thread dump OpenJDK 64-Bit Server VM (25.152-b19 mixed mode):
```

```
"Attach Listener" #4623 daemon prio=9 os_prio=31 tid=0x00007f9f12151000 nid=0xf803 waiting on condition [0x0000000000000000]
```

```
    java.lang.Thread.State: RUNNABLE
```

```
.....
```

[jstack 예시]

** pid 확인하는 방법

- 1) ps

\$ ps -ef | grep java

- 1) jps

\$ jps -v

```
[user@localhost ~] jps -v
```

```
11655 -Xms128m -Xmx750m -XX:ReservedCodeCacheSize=240m -XX:+UseCompressedOops -Dfile.encoding=UTF-8 -XX:+UseConcMarkSweepGC -XX:SoftRefLRUPolicyMSPerMB=50 -ea -Dsun.io.useCanonCaches=false -Djava.net.preferIPv4Stack=true -Djdk.http.auth.tunneling.disabledSchemes="" -XX:+HeapDumpOnOutOfMemoryError -XX:-OmitStackTraceInFastThrow -Xverify:none -XX:ErrorFile=/Users/user/java_error_in_idea_%p.log -XX:HeapDumpPath=/Users/user/java_error_in_idea.hprof -Djb.vmOptionsFile=/Applications/IntelliJ IDEA.app/Contents/bin/idea.vmoptions -Didea.java.redist=jdk-bundled -Didea.home.path=/Applications/IntelliJ IDEA.app/Contents -Didea.executable=idea -Didea.paths.selector=IntelliJIDEA2018.2
```

[jps 예시]

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프

스레드 덤프 정보

1

2

3

4

"ApplicationImpl pooled thread 4040" #4622 daemon prio=4 os_prio=31 tid=0x00007f9f17561800 nid=0x2897b

5 waiting on condition [0x0000700003db2000] 6

java.lang.Thread.State: TIMED_WAITING (parking)
at sun.misc.Unsafe.park(Native Method)
- parking to wait for <0x000000007a0caf918> (a java.util.concurrent.SynchronousQueue\$TransferStack)
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:215)
at java.util.concurrent.SynchronousQueue\$TransferStack.awaitFulfill(SynchronousQueue.java:460)
at java.util.concurrent.SynchronousQueue\$TransferStack.transfer(SynchronousQueue.java:362)
at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:941)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1066)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1127)
at java.util.concurrent.ThreadPoolExecutor\$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)

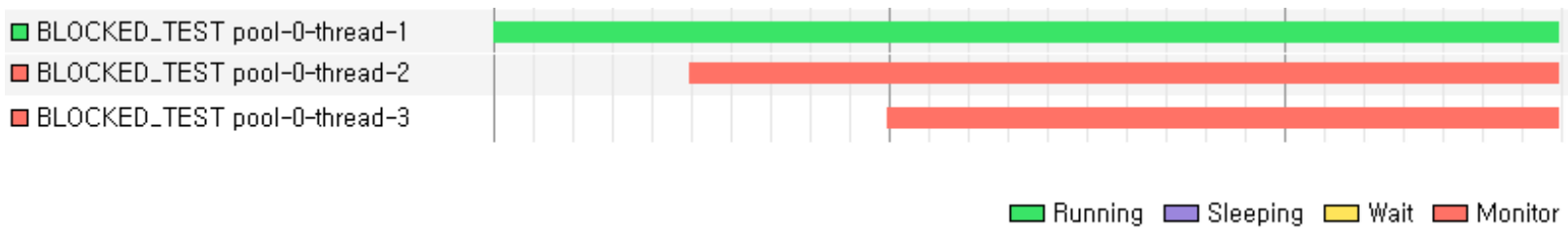
7

- 1. 스레드 이름
- 2. 스레드 우선순위
- 3. 스레드 ID
- 4. Native 스레드 ID
- 5. 스레드 상태
- 6. 스레드 스택 주소 범위(from-to)
- 7. 스레드 콜스택 정보

스레드 덤프 패턴

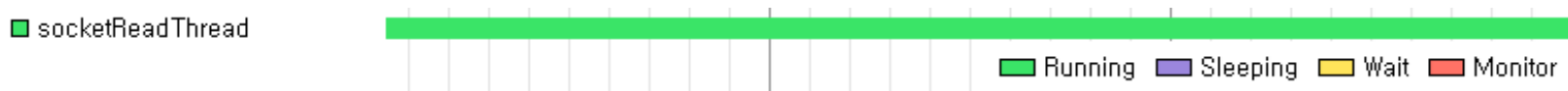
- 락을 획득하지 못한 상태

다음 그래프는 pool-0-thread1이 lock을 소유하고 있어 pool-0-thread2, 3 스레드가 lock을 획득하기 위해 대기하고 있는 상태이다. 이러한 상태를 BLOCKED라고 하며, 지속되는 경우 특정 스레드가 리소스에 대한 lock release를 제대로 하는지에 대해 확인을 해야한다.



- 원격 서버로부터 응답을 받기 위해 계속 대기하고 있는 상태

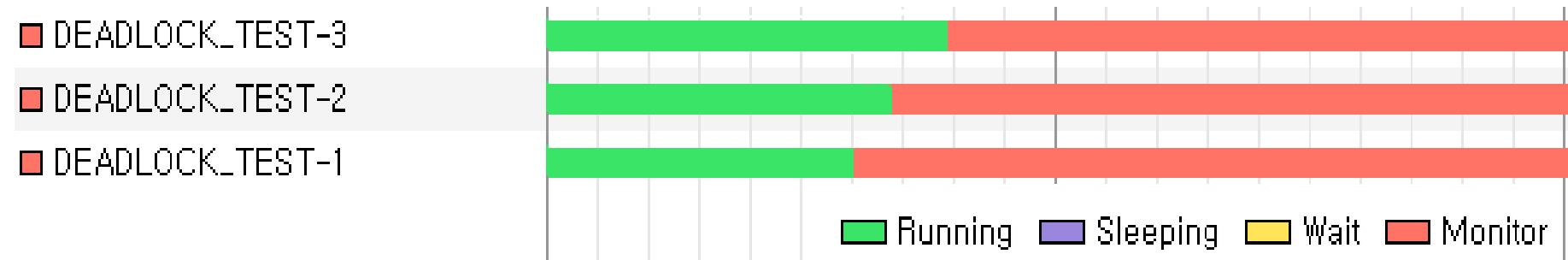
아래 그래프는 원격 서버로 요청을 보내고, 응답을 받기 위해 계속 스레드가 Runnable 상태를 유지하고 있는 그래프이다. 언뜻보면 정상처럼 보이겠지만 계속 해당 상태가 유지된다면 클라이언트에서 해당 원격 서버로의 접속을 확인해보거나 원격 서버에서의 응답이 정상적인지를 확인해야 한다.



스레드 덤프 패턴

- 데드락 상태인 경우

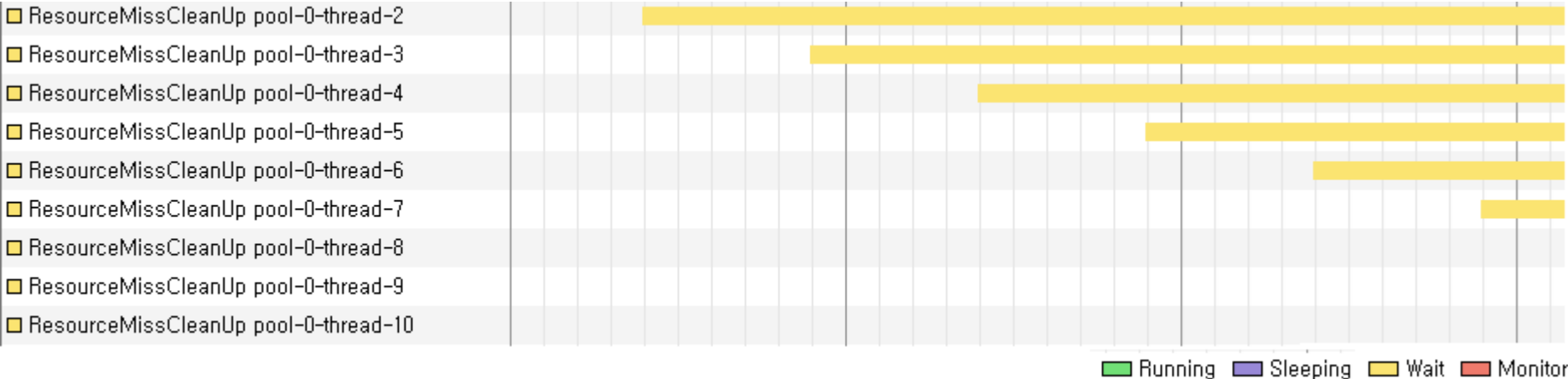
각 스레드가 서로 사용해야 할 자원을 locking하고 있는 상태를 말한다.
다음 그래프에서는 DEADLOCK_TEST-1이 리소스 A에 대한 lock을 획득한 채 DEADLOCK_TEST-2가 lock을 획득한 리소스 B의 lock을 획득하려고 한다. DEADLOCK_TEST-2는 리소스 B의 lock을 획득하고, DEADLOCK_TEST-3이 lock을 획득한 리소스 C의 lock을 획득하려고 하며, 마지막으로 DEADLOCK_TEST-3은 리소스 C의 lock을 획득한 채 DEADLOCK_TEST-1이 lock을 획득한 리소스 A의 lock을 획득하려고 하는 상황을 나타냈다.



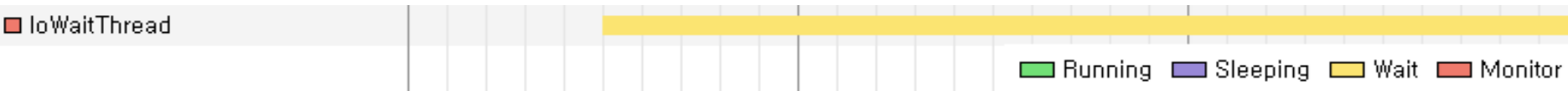
스레드 덤프 패턴

- Waiting

스레드가 계속 waiting 상태에 있는 것으로 다양한 원인이 있을 수 있다. 아래 그림 중 첫 번째는 불필요한 스레드가 정리되지 못한 채 남아 있는 것으로 태스크가 끝난 스레드는 시스템 리소스를 낭비를 막기 위해 정리될 수 있도록 한다.



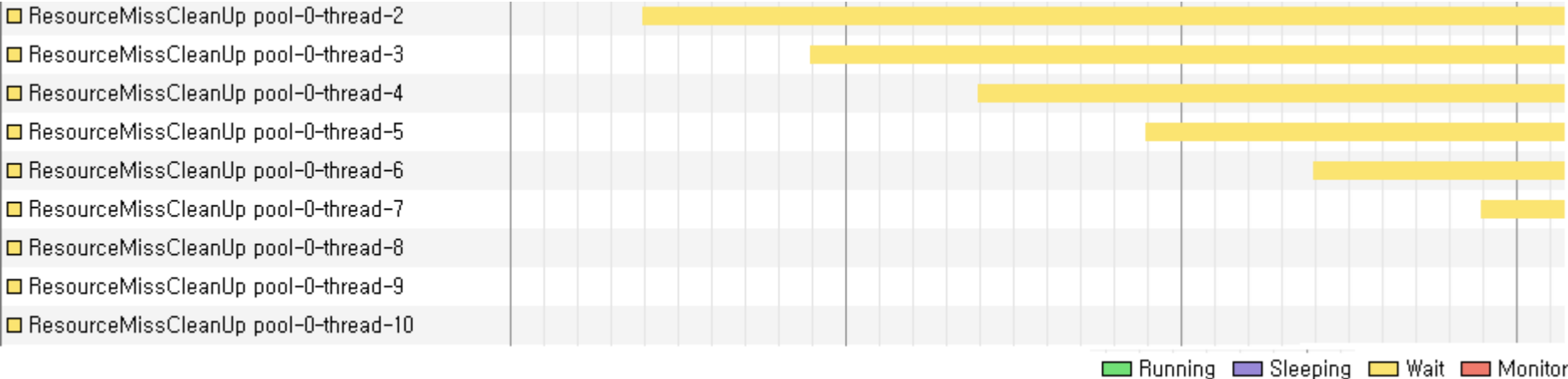
두 번째 그래프는 내/외부의 queue를 사용할 때, LinkedBlockingQueue 객체에서 메시지 수신을 위해 계속 대기하는 상태이다. LinkedBlockingQueue가 메시지를 얻지 못한다면 상태는 바뀌지 않게 된다.



스레드 덤프 패턴

- Waiting

스레드가 계속 waiting 상태에 있는 것으로 다양한 원인이 있을 수 있다. 아래 그림 중 첫 번째는 불필요한 스레드가 정리되지 못한 채 남아 있는 것으로 태스크가 끝난 스레드는 시스템 리소스를 낭비를 막기 위해 정리될 수 있도록 한다.



두 번째 그래프는 내/외부의 queue를 사용할 때, LinkedBlockingQueue 객체에서 메시지 수신을 위해 계속 대기하는 상태이다. LinkedBlockingQueue가 메시지를 얻지 못한다면 상태는 바뀌지 않게 된다.



가비지 컬렉션

메모리 영역(Heap)에서 사용중인 객체와 그렇지 않은 객체를 확인 후 사용하지 않는 객체를 삭제하는 프로세스를 말한다. 사용 중인 객체나 참조된 객체는 프로그램 일부가 해당 객체에 대한 포인터를 유지한다는 것을 의미한다. 반대로 사용되지 않거나 참조되지 않은 오브젝트는 그렇지 않기 때문에 이러한 객체가 사용하는 메모리를 다시 확보한다.

C언어 등에서는 메모리 할당/해제는 수동 프로세스였지만 Java에서의 메모리 할당/해제는 가비지 컬렉터에 의해 자동으로 수행된다.

Java는 프로그램 코드에서 메모리를 명시적으로 지정하여 해제하지 않는다. 가끔 명시적으로 해제하려고 해당 객체를 null로 지정하거나 System.gc() 메서드를 호출하는 개발자가 있다. null로 지정하는 것은 큰 문제가 안 되지만, System.gc() 메서드를 호출하는 것은 시스템의 성능에 매우 큰 영향을 끼치므로 System.gc() 메서드는 절대로 사용하면 안 된다.

stop-the-world

GC을 실행하기 위해 JVM이 애플리케이션 실행을 멈추는 것이다. stop-the-world가 발생하면 GC를 실행하는 스레드를 제외한 나머지 스레드는 모두 작업을 멈춘다. GC 작업을 완료한 이후에야 중단했던 작업을 다시 시작한다.

어떤 GC 알고리즘을 사용하더라도 stop-the-world는 발생한다.

대개의 경우 GC 튜닝이란 이 stop-the-world 시간을 줄이는 것이다.

시리얼 콜렉터 (Serial Collector)

병렬 콜렉터 (Parallel Collector)

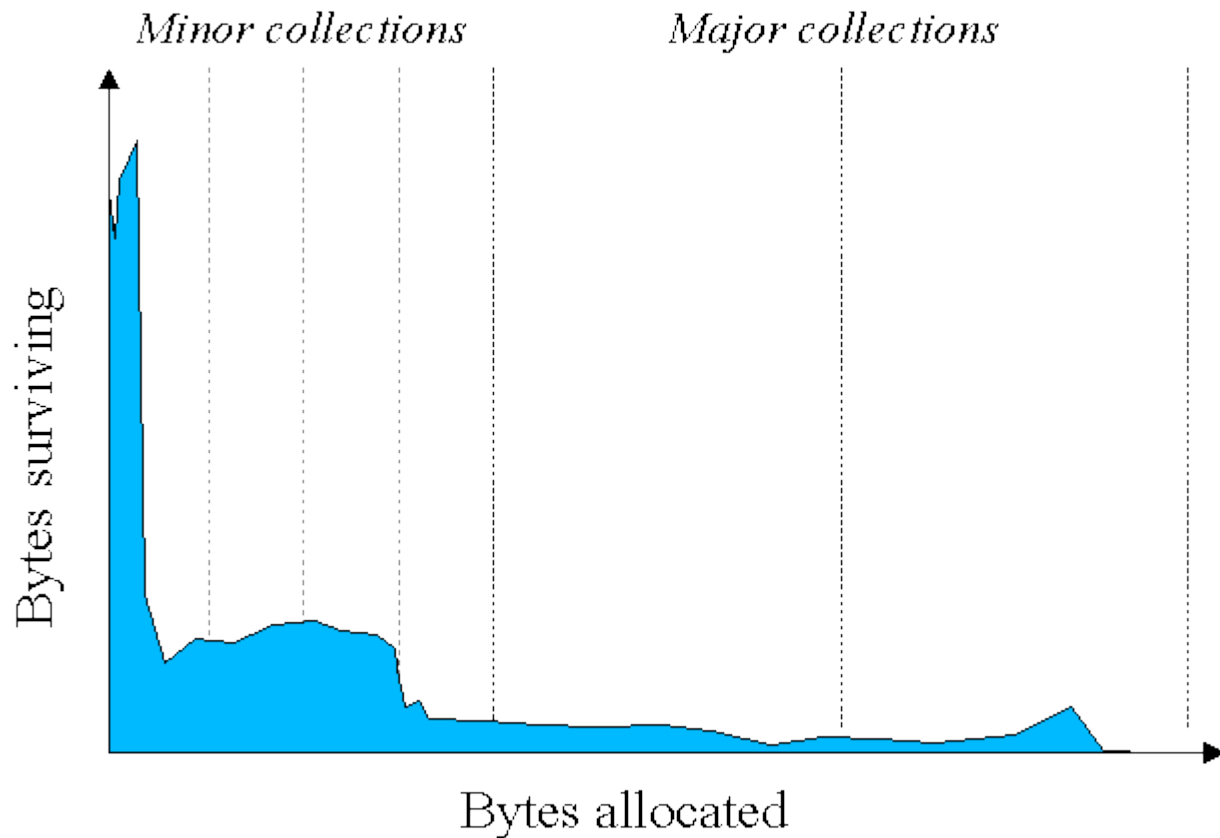
병렬 압축 콜렉터 (Parallel Compacting Collector)

CMS 콜렉터 (CMS Collector)

G1 콜렉터 (Garbage First Collector)

통켓은 자바
통켓은 서블릿 컨테이너
통켓 설치
통켓 아키텍처
통켓 서비스
통켓 커넥터
통켓 컨텍스트
통켓 리스너
통켓 로그
통켓 DB Pool
자바 클래스로더
자바 스레드덤프
자바 가비지 컬렉션

Young과 Old



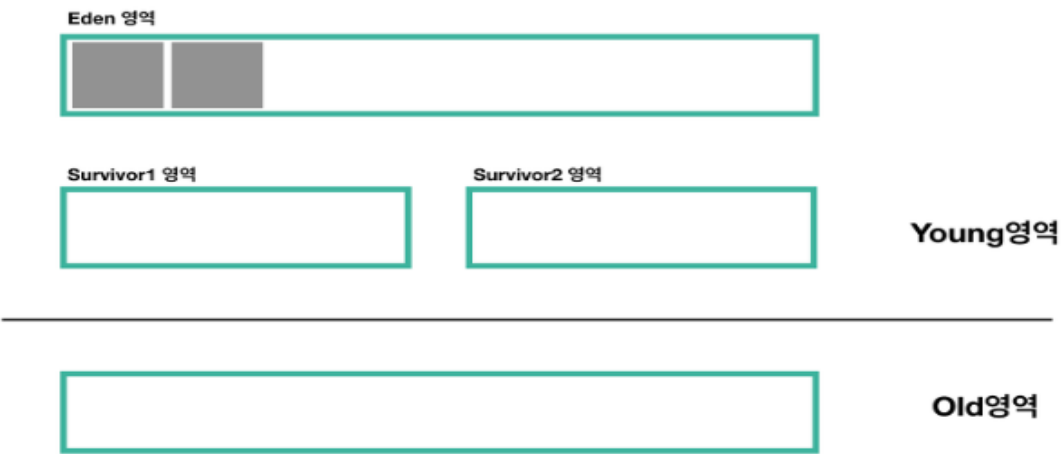
Y축은 할당된 바이트 수를 나타내고
X 액세스는 시간에 따라 할당된 바
이트 수를 보여준다.

보시다시피, 시간이 지남에 따라 할
당된 개체 수가 점점 더 적어진다.
사실 대부분의 물체는 그래프 왼쪽
에 있는 높은 값에 의해 보여지듯이
매우 짧은 수명을 가지고 있다.

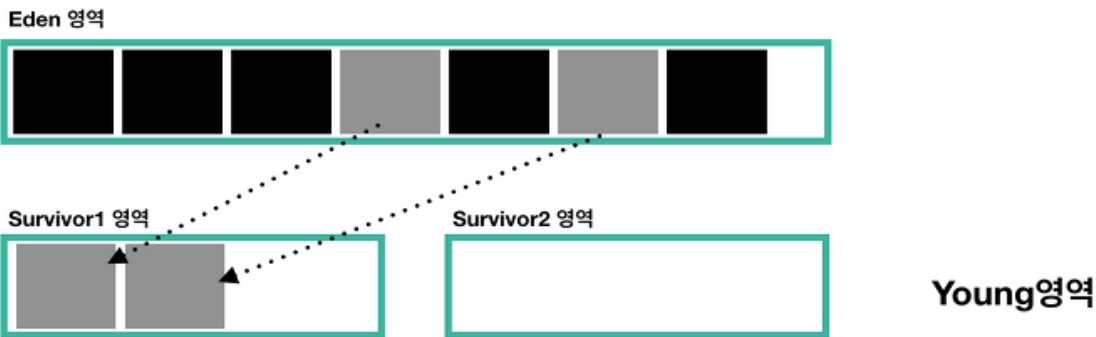
- 대부분의 객체는 금방 참조가 끊긴다.(접근 불가능 상태)
- 오래된 객체에서 젊은(생성된지 얼마 안된)객체로의 참조는 드물다.

- 톰캣은 자바
- 톰캣은 서블릿 컨테이너
- 톰캣 설치
- 톰캣 아키텍처
- 톰캣 서비스
- 톰캣 커넥터
- 톰캣 컨텍스트
- 톰캣 리스너
- 톰캣 로그
- 톰캣 DB Pool
- 자바 클래스로더
- 자바 스레드덤프
- 자바 가비지 컬렉션

가비지 컬렉션 절차



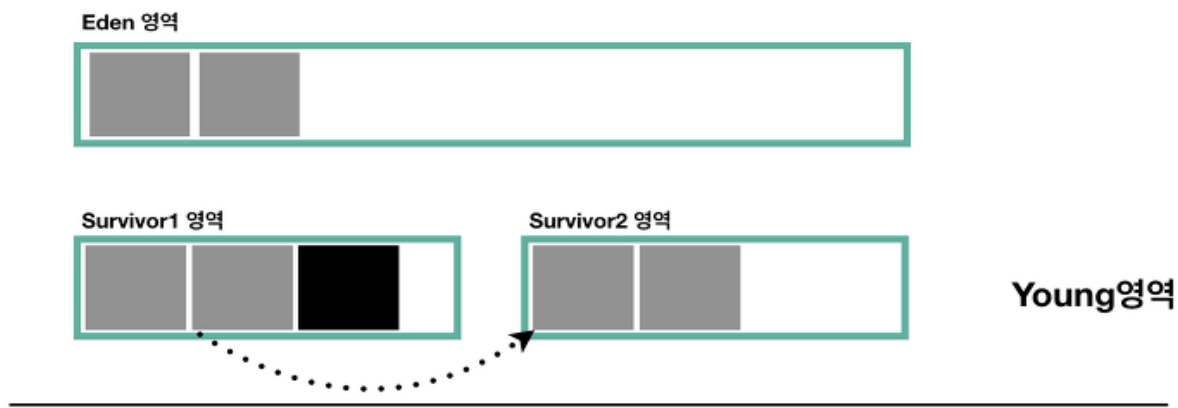
객체가 new 키워드로 생성될 때, Eden 영역에 생성한다.



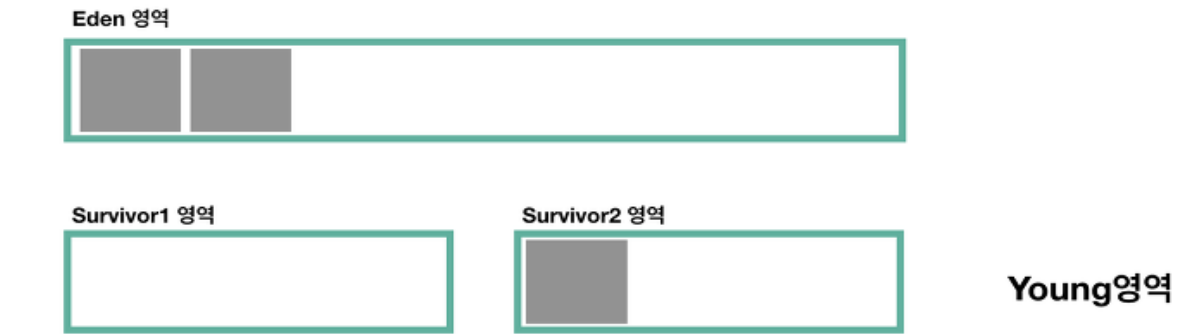
그리고 Eden 영역이 꽉 차게 되면 Eden 영역에서 GC가 한번 일어나게 되며(참조가 끊긴 객체는 지워진다.) 여기서 살아남은 객체들은 Survivor영역 중 하나로 이동한다.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프
자바 가비지 컬렉션

가비지 컬렉션 절차



Eden 영역에서 GC가 일어나는 일이 반복되다가 하나의 Survivor영역이 가득 차게되면, 그 Survivor영역에서 GC가 일어난다. 그리고 여기서 살아남은 객체들은 나머지 Survivor영역으로 이동한다. 이때, GC가 일어난 Survivor영역은 비어있어야 한다.

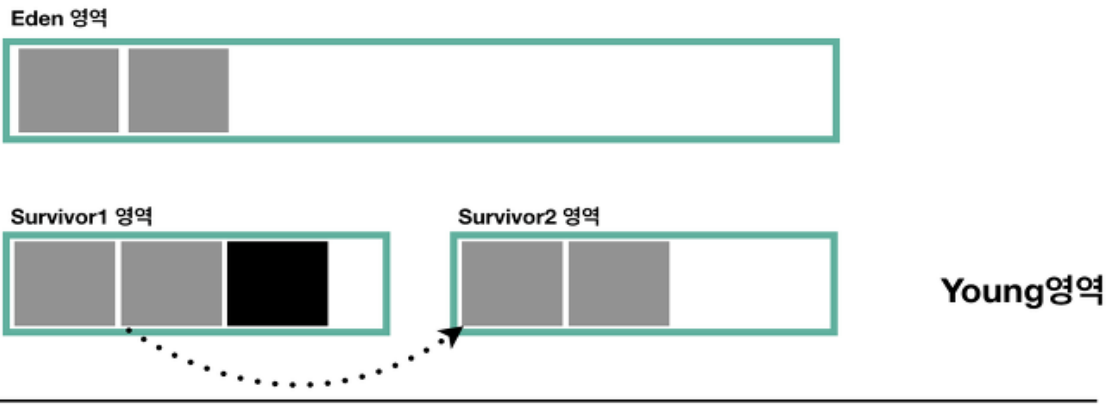


Survivor1 영역과 2영역을 왔다 갔다하는 과정을 반복하다가 계속 살아있는 객체는 Old 영역으로 이동한다. 만약 Eden영역의 살아있는 객체가 Survivor영역보다 더 큰 경우, 바로 Old영역으로 옮겨진다.



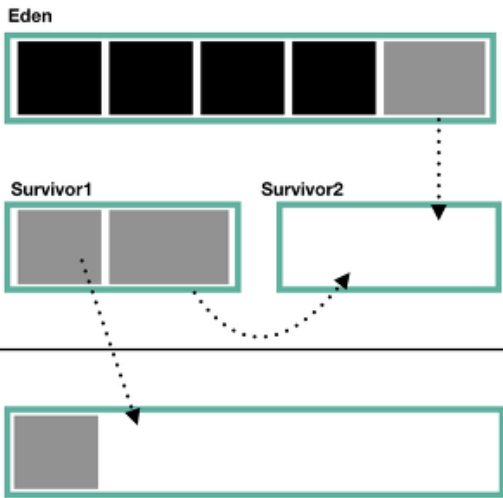
- 톰캣은 자바
- 톰캣은 서블릿 컨테이너
- 톰캣 설치
- 톰캣 아키텍처
- 톰캣 서비스
- 톰캣 커넥터
- 톰캣 컨텍스트
- 톰캣 리스너
- 톰캣 로그
- 톰캣 DB Pool
- 자바 클래스로더
- 자바 스레드덤프
- 자바 가비지 컬렉션

가비지 컬렉션 절차

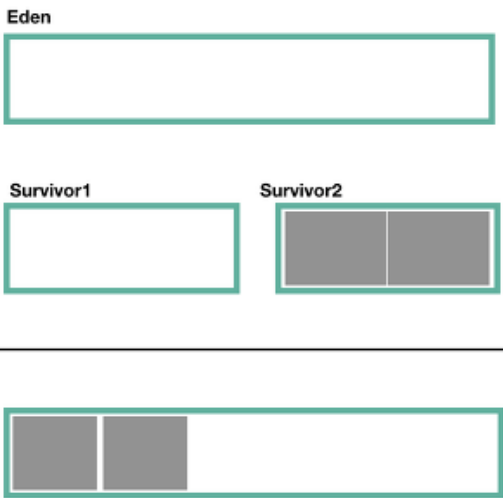


Eden 영역에서 GC가 일어나는 일이 반복되다가 하나의 Survivor영역이 가득 차게되면, 그 Survivor영역에서 GC가 일어난다. 그리고 여기서 살아남은 객체들은 나머지 Survivor영역으로 이동한다. 이때, GC가 일어난 Survivor영역은 비어있어야 한다.

Garbage Collection 전



Garbage Collection 후



Survivor1 영역과 2영역을 왔다 갔다하는 과정을 반복하다가 계속 살아있는 객체는 Old 영역으로 이동한다. 만약 Eden영역의 살아있는 객체가 Survivor영역보다 더 큰 경우, 바로 Old영역으로 옮겨진다.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프
자바 가비지 컬렉션

시리얼 콜렉터(Serial GC : -XX:+UseSerialGC)

수행전



수행후



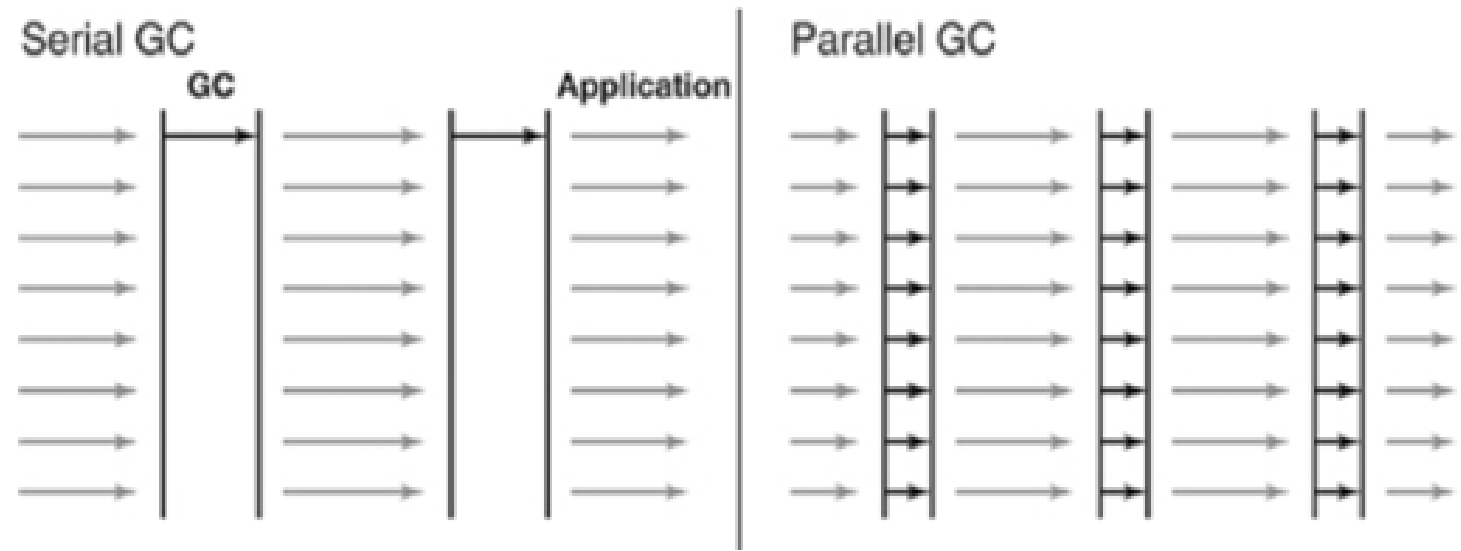
Minor GC에서 살아 남아 Old 영역으로 이동한 객체는 Mark-sweep-compact 알고리즘을 따른다.

이 알고리즘의 첫 단계는 Old 영역에 살아 있는 객체를 식별(Mark)하는 것이다. 그 다음에는 힙(heap)의 앞 부분부터 확인하여 살아 있는 것만 남긴다(Sweep). 마지막 단계에서는 각 객체들이 연속되게 쌓이도록 힙의 가장 앞 부분부터 채워서 객체가 존재하는 부분과 객체가 없는 부분으로 나눈다(Compaction).

이 방식은 굉장히 느리다. 그렇지만 그만큼 시스템 자원을 덜 사용한다. 속도와 별 상관이 없고, 시스템 자원이 부족할 때 사용하는 GC 방식이다.

- 톰캣은 자바
- 톰캣은 서블릿 컨테이너
- 톰캣 설치
- 톰캣 아키텍처
- 톰캣 서비스
- 톰캣 커넥터
- 톰캣 컨텍스트
- 톰캣 리스너
- 톰캣 로그
- 톰캣 DB Pool
- 자바 클래스로더
- 자바 스레드덤프
- 자바 가비지 컬렉션

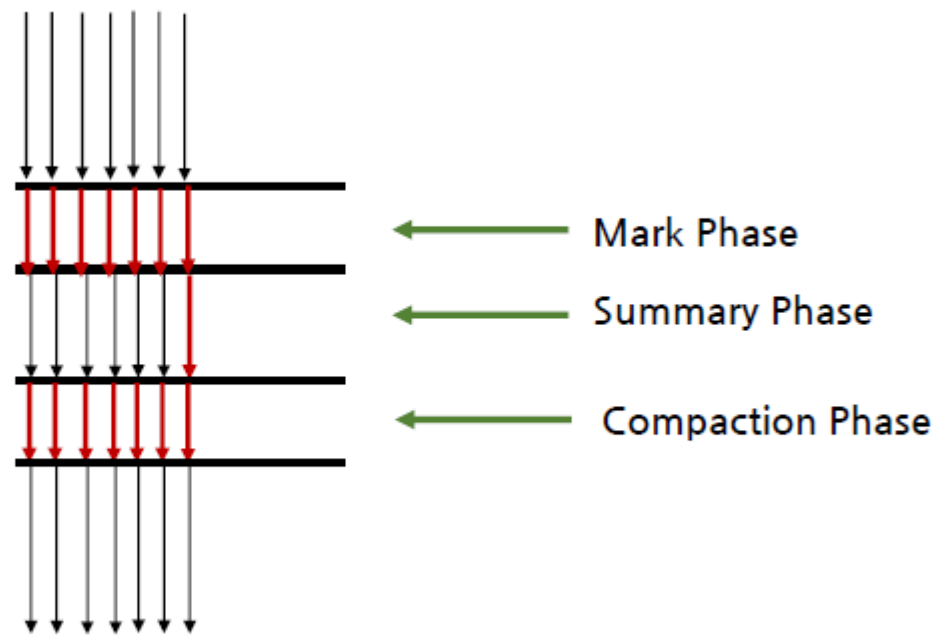
병렬 콜렉터(Parallel GC : -XX:+UseParallelGC)



병렬 콜렉터 방식의 경우 시리얼 콜렉터 방식과 GC 알고리즘은 같다.(Young영역과 Old영역 모두 같다.) 하지만 단일 스레드가 GC를 수행하는 시리얼 콜렉터 방식과 달리, 병렬 콜렉터 방식은 GC를 처리하는 스레드가 여러개이다. 그렇기 때문에 Serial GC보다 빠른게 객체를 처리할 수 있다. 당연히 시스템 자원이 넉넉할 때 유리한 방식이다.

- 통캣은 자바
- 통캣은 서블릿 컨테이너
- 통캣 설치
- 통캣 아키텍처
- 통캣 서비스
- 통캣 커넥터
- 통캣 컨텍스트
- 통캣 리스너
- 통캣 로그
- 통캣 DB Pool
- 자바 클래스로더
- 자바 스레드덤프
- 자바 가비지 컬렉션

병렬 압축 콜렉터(Parallel Old GC : -XX:+UseParallelOldGC)

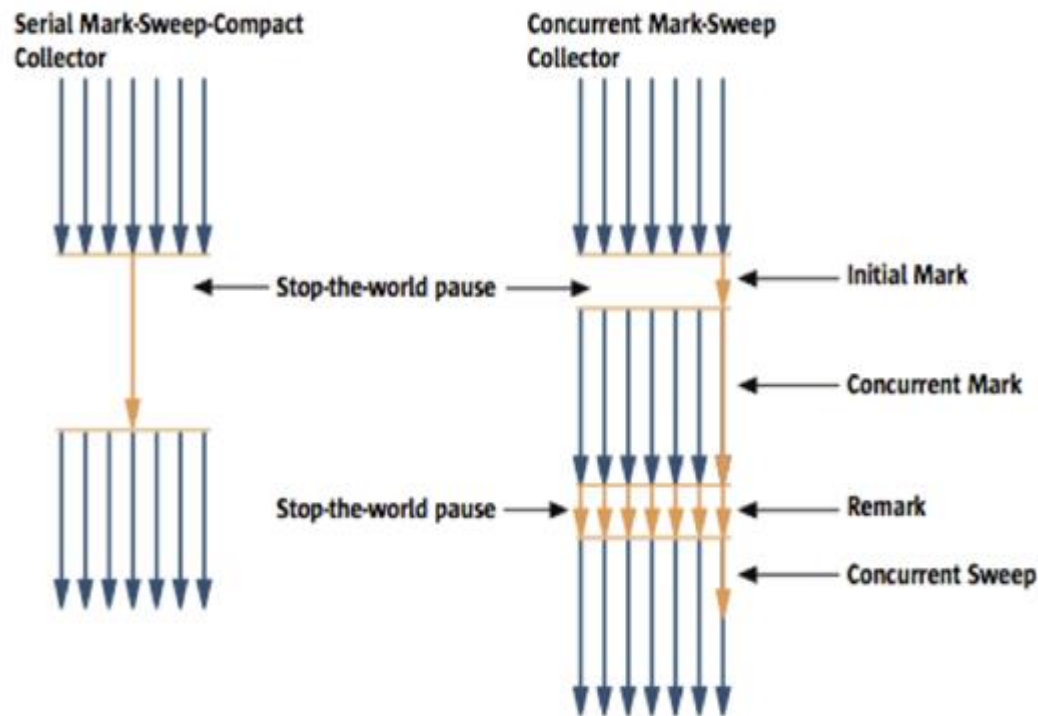


Parallel Old GC는 JDK5 update 6부터 제공한 GC 방식이다. Parallel GC와 비교하여 Old 영역의 GC 알고리즘만 다르다. 이 방식은 Mark-Summary-Compaction 단계를 거친다.

Sweep은 단일 스레드가 Old 영역 전체를 훑어 살아있는 객체만 찾아내는 방식이지만, Summary는 여러 스레드가 Old 영역을 분리하여 훑는다. 또한 효율을 위해 앞선 GC에서 Compaction된 영역을 별도로 훑는다.

통켓은 자바
통켓은 서블릿 컨테이너
통켓 설치
통켓 아키텍처
통켓 서비스
통켓 커넥터
통켓 컨텍스트
통켓 리스너
통켓 로그
통켓 DB Pool
자바 클래스로더
자바 스레드덱
자바 가비지 컬렉션

CMS 콜렉터(Concurrent Mark-Sweep GC : --XX:+UseConcMarkSweepGC)



Initial Mark

Single Thread만이 사용되는 Serial Phase. Application에서 직접 Reference되는 Live Object만 빠르게 구별해냄. Heap은 Suspend상태가되는데 root set에서 직접 Reference되는 Object만 대상으로 하기 때문에 Suspend시간은 최소화됨.

Concurrent Mark Phase

Single Thread만이 사용되는 Serial Phase. GC Thread외의 Working Thread은 Application을 수행. Initial Mark Phase에서 선별한 Live Object를 대상으로 reference 하고 있는 Object를 추적. Live 여부 구별.

Remark Phase

모든 Thread들이 GC에 동원. Application의 수행은 잠시 중단. Remark라는 말 그대로 Mark 작업을 다시 수행한다는 의미로 Concurrent 작업으로 이해. 발생한 변경사항을 반영하기 위한 작업으로 각 Thread들이 이미 Marking된 Object를 다시 추적, live 여부 확정.

이 작업은 Initial Mark보다 작업량이 많기 때문에 가용한 모든 resource를 투입하여 수행.

Concurrent Sweep Phase

Remark 작업을 통해 최종 Live로 판명한 Object를 제외한 Dead Object를 지움. Dead Object를 지워 그 공간을 다시 사용할 수 있도록 하는 작업수행.

CMS 콜렉터 방식은 stop-the-world 시간이 매우 짧다는 장점이 있다. 하지만 그만큼 시스템 자원(메모리, CPU)를 더 많이 사용하고, Compaction 단계가 없어 Old영역의 크기가 충분하지 않거나 크기에 비해 조각난 메모리가 많을 경우 오히려 stop-the-world 시간이 늘어날 수 있다.

통캣은 자바
통캣은 서블릿 컨테이너
통캣 설치
통캣 아키텍처
통캣 서비스
통캣 커넥터
통캣 컨텍스트
통캣 리스너
통캣 로그
통캣 DB Pool
자바 클래스로더
자바 스레드덱프
자바 가비지 컬렉션

G1 콜렉터(Garbage First GC : --XX:+UseG1GC)

H: humongous		U: Unused		
Y: Young Generation		O: Old Generation		
H	O	U	Y	O
H	O	Y	Y	O
H	U	Y	O	Y
H	Y	Y	Y	U
O	Y	O	Y	O

- CMS GC를 대체하기 위해서 만들어짐.
- 물리적인 Generation의 구분을 없애고 전체 Heap을 1Mb ~ 32Mb 단위의 Region으로 재편
- Garbage로 딱 차있는 Region부터GC시작
- Concurrent + Parallel + Compacting의 조합

기존 GC는 모두 Young영역과 Old영역이 정해져서 나뉘어있었지만, G1 콜렉터 방식에서는 바둑판 모양의 영역이 각각 Eden, Survivor, Old 영역의 역할을 동적으로 바꿔가며 GC가 일어난다.

Java 7 u60 이상 버전에서 G1 GC 사용할 것. 그 이하버전에 버그가 좀 많다고 함.

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덱프
자바 가비지 컬렉션
톰캣 모니터링

모니터링 항목

대항목	중항목	세부항목	설명
OS	System Resource Usage	CPU 사용량	CPU 전체 사용량
		Memory 사용량	Memory 전체 사용량
서버 상태	Process Resource Usage	Process CPU 과부하 사용	프로세스 CPU 사용률
		Disk Usage	설치 디스크의 사용량
	Server Status	Server Status	Server Life Cycle
실행 상태	Execute Status	Thread status	실행 스레드 및 idle thread status
	Garbage Collection	Full GC Duration	Full GC 처리 상태
	Heap Memory	Heap Memory Init	힙 메모리 초기값
		Heap Memory Max	힙 메모리 최대값 설정
		Perm Area Max Size	MaxPermSize 의 설정
JDBC 커넥션풀	Configuration	JDBC url	jdbc connection meta data URL
		JDBC driver name	jdbc driver class name
	Monitorming	Connections	사용 커넥션 개수
로그 분석		Critical Message	부팅 Critical 에러
		Too many open files	파일 오픈 개수 초과
		OutOfMemoryError	힙메모리 부족

MBean

JMX(Java Management eXtensions) API에서 Managed Bean (줄여서 MBean 이라 하는)이란 Dependency Injection에 의해 만들어진 일종의 JavaBean이다. MBean은 어플리케이션이나 JavaEE의 서비스(트랜잭션 모니터, JDBC 드라이버 등) 내에서 동작 중인 하나의 리소스를 나타낸다. 이것은 어플리케이션의 configuration(설정 정보)를 얻기/설정하거나, 통계 정보(성능, 리소스 사용, 문제점 등)를 수집하거나, 이벤트를 전달하기 위해 사용될 수 있다.

톰캣 JMX 라이브러리 다운

<http://repo2.maven.org/maven2/org/apache/tomcat/tomcat-catalina-jmx-remote/8.5.6/tomcat-catalina-jmx-remote-8.5.6.jar>

톰캣(Tomcat)시작 스크립트에 다음과 같이 JMX 파라미터 추가

```
export JMX_OPTS=" -Dcom.sun.management.jmxremote ₩  
                -Dcom.sun.management.jmxremote.authenticate=false ₩  
                -Djava.rmi.server.hostname=192.168.96.6 ₩  
                -Dcom.sun.management.jmxremote.ssl=false "
```

```
export CATALINA_OPTS=" ${JMX_OPTS} ${CATALINA_OPTS}"
```

server.xml 파일에서 Server 섹션에 JMX 설정

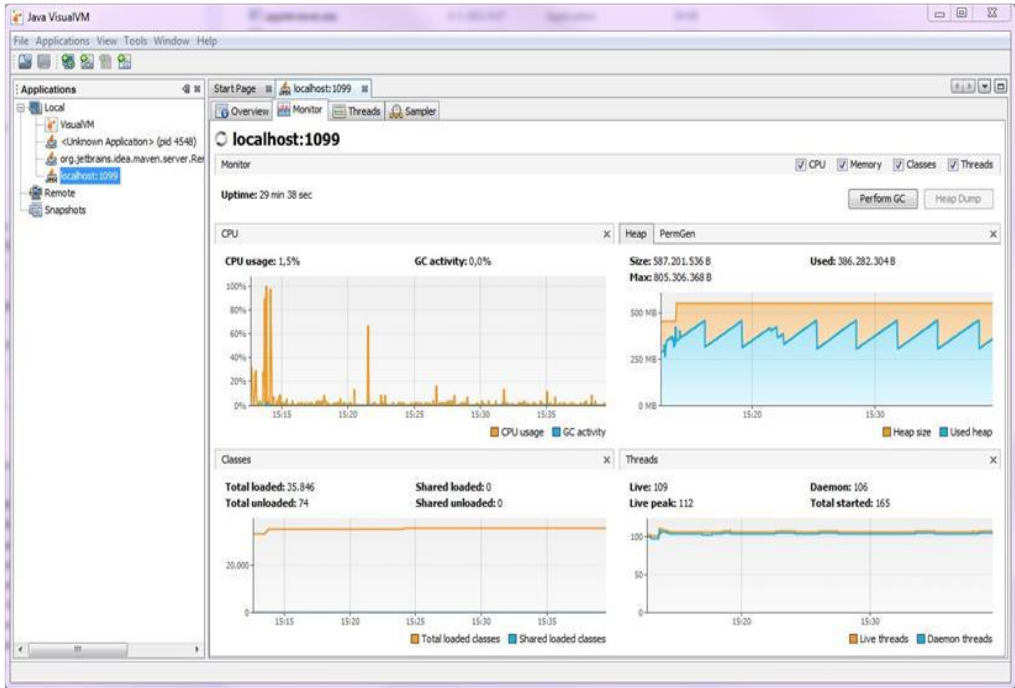
```
<!-- Enable JMX -->  
<Listener className="org.apache.catalina.mbeans.JmxRemoteLifecycleListener"  
          rmiRegistryPortPlatform="9180" rmiServerPortPlatform="9181"/>
```

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프
자바 가비지 컬렉션
톰캣 모니터링

VirtualVM

- JVM 전반에 대해 모니터링을 제공하는 툴
- JVM에 포함 (\$JAVA_HOME/bin/jvisualvm) ****JDK 버전에 존재**
 - ✓ Display local and remote Java applications
 - ✓ Display Application Configuration and runtime environment
 - ✓ Monitor application threads, application performance and memory consumption
 - ✓ Take and display thread dumps
 - ✓ Take and browse heap dumps

plugins

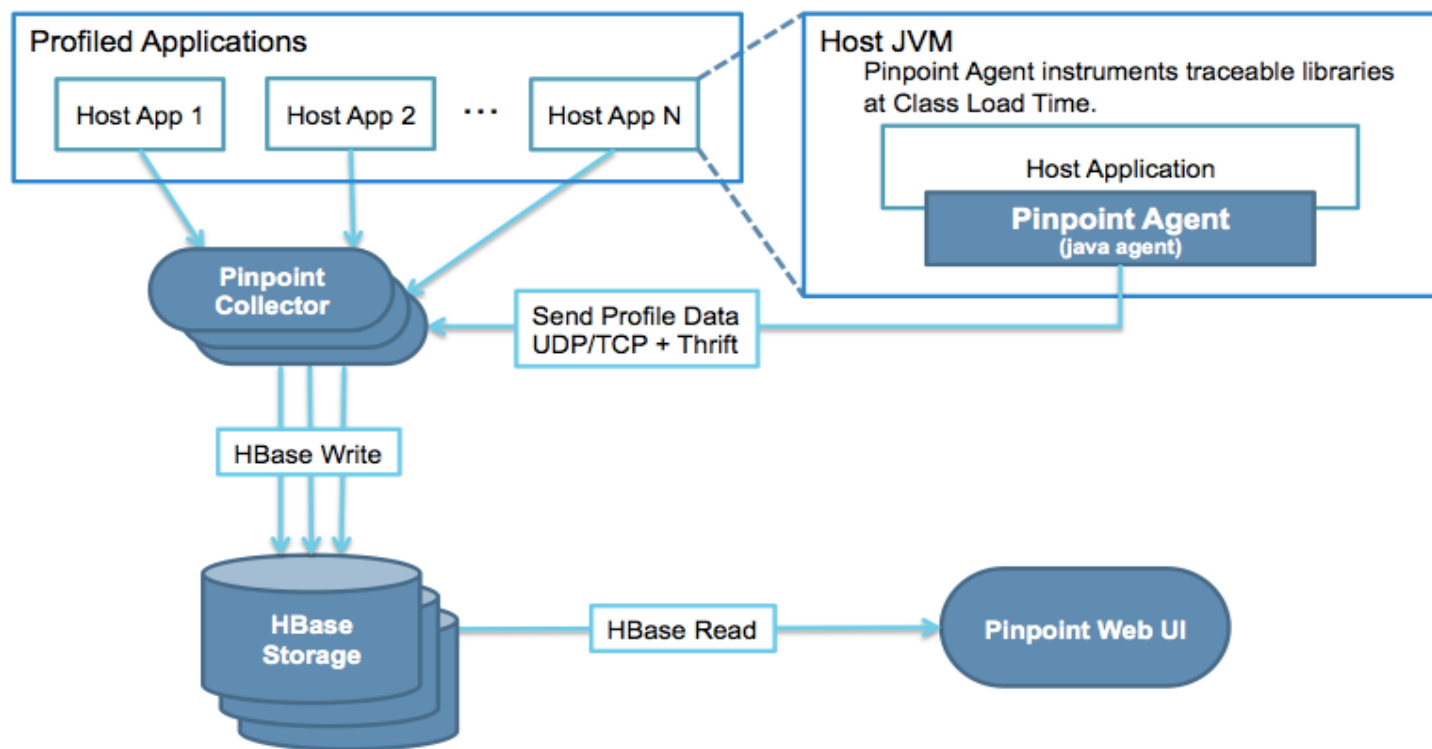


Check for Newest			
Install	Name	Category	Source
<input type="checkbox"/>	VisualVM-Glassfish	Application Se...	
<input type="checkbox"/>	VisualVM-Extensions	Platform	
<input type="checkbox"/>	Startup Profiler	Profiling	
<input type="checkbox"/>	BTrace Workbench	Profiling	
<input type="checkbox"/>	VisualVM-Security	Security	
<input checked="" type="checkbox"/>	Visual GC	Tools	
<input type="checkbox"/>	VisualVM-BufferMonitor	Tools	
<input type="checkbox"/>	Threads Inspector	Tools	
<input type="checkbox"/>	VisualVM-JConsole	Tools	
<input type="checkbox"/>	VisualVM-MBeans	Tools	
<input type="checkbox"/>	KillApplication	Tools	
<input type="checkbox"/>	Tracer-Jvmstat Probes	Tracer	
<input type="checkbox"/>	Tracer-Monitor Probes	Tracer	
<input type="checkbox"/>	Tracer-Swing Probes	Tracer	
<input type="checkbox"/>	Tracer-IO Probes	Tracer	
<input type="checkbox"/>	Tracer-Collections Probes	Tracer	
<input type="checkbox"/>	Tracer-JVM Probes	Tracer	
<input type="checkbox"/>	OQL Syntax Support	UI	

톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드 덤프
자바 가비지 컬렉션
톰캣 모니터링

Pinpoint

- 대규모 분산 시스템의 성능을 분석하고 문제를 진단, 처리하는 java 플랫폼
- 네이버에서 2012년 7월부터 개발하여 2014년 말에 배포를 시작



톰캣은 자바
톰캣은 서블릿 컨테이너
톰캣 설치
톰캣 아키텍처
톰캣 서비스
톰캣 커넥터
톰캣 컨텍스트
톰캣 리스너
톰캣 로그
톰캣 DB Pool
자바 클래스로더
자바 스레드덤프
자바 가비지 컬렉션
톰캣 모니터링

Pinpoint

지원모듈

- JDK 6+
- Tomcat 6/7/8/9, Jetty 8/9, JBoss EAP 6/7, Resin 4, Websphere 6/7/8, Vertx 3.3/3.4/3.5, Weblogic 10/11g/12c
- Spring, Spring Boot (Embedded Tomcat, Jetty)
- Apache HTTP Client 3.x/4.x, JDK HttpURLConnection, GoogleHttpClient, OkHttpClient, NingAsyncHttpClient
- Thrift Client, Thrift Service, DUBBO PROVIDER, DUBBO CONSUMER
- ActiveMQ, RabbitMQ
- MySQL, Oracle, MSSQL(jtds), CUBRID, POSTGRESQL, MARIA
- Arcus, Memcached, Redis, CASSANDRA
- iBATIS, MyBatis
- DBCP, DBCP2, HIKARICP
- gson, Jackson, Json Lib
- log4j, Logback

참고 URL

<https://medium.com/chequer/tomcat-spring-bootstrapping-sequence-1%ED%8E%B8-tomcat-4402102c0585>

<http://ecomputernotes.com/servlet/intro/servlet-container>

로그 <http://dbheart.tistory.com/28>

<https://tomcat.apache.org/tomcat-8.0-doc/config>

<http://pshcode.tistory.com/109>

<https://www.theserverside.com/tip/Two-most-commonly-misconfigured-Tomcat-performance-settings>

http://www.gnujava.com/board/article_view.jsp?article_no=832&board_no=6&table_cd=EPAR02&table_no=02

<https://sarc.io/index.php/tomcat/155-tomcat-root-listener>

<https://sarc.io/index.php/tomcat/230-tomcat-7-x-8-x-default-configuration>

<http://www.kwangsiklee.com/2018/01/상용에서-쓸-수-있는-톰캣8-실행환경-정리/>

<https://www.c2b2.co.uk/middleware-blog/tomcat-performance-monitoring-and-tuning.php>

<https://howtodoinjava.com/tomcat/tomcats-architecture-and-server-xml-configuration-tutorial/>

<http://jjeong.tistory.com/693>

https://www.slideshare.net/jieunsys/ss-56543446?from_action=save

https://www.packtpub.com/mapt/book/networking_and_servers/9781847197283/3

<https://hello-nanam.tistory.com/55>

<https://www.holaxprogramming.com/2013/01/10/devops-how-to-manage-dbc/>

<http://forgiveall.tistory.com/447>

<https://sarc.io/index.php/tomcat/900-apache-tomcat-java-logging-juli>

<http://2dubbing.tistory.com/12>

<http://dimdim.tistory.com/entry/DBCP의-일반적인-설정값-가이드>

<https://dzone.com/articles/how-analyze-java-thread-dumps>

<https://d2.naver.com/helloworld/10963>

<http://iyoong.github.io/jekyll/update/2015/04/19/threaddump.html>

https://www.slideshare.net/ssuserb77a05/garbage-collection-20150213-44717452?from_action=save

<http://kwonnam.pe.kr/wiki/java/g1gc>

<https://www.holaxprogramming.com/2017/10/09/java-jvm-performance/>