트랜잭션 추적을 위한 END-POINT 설정 V0.0.3

본 문서는 Http서버가 아닌 일반 통신 데몬이나 Http 서버 내부의 백그라운드 스레드에 대한 성능 추적을 위한 모니터링 방법을 설명한다.

트랜잭션 End-Point란

트랜잭션의 시작 메소드를 의미한다. HTTP 트랜잭션의 경우에는 HttpServlet.service() 혹은 Filter.doFilter()가 트랜잰션의 시작점이고 이곳을 트랜잭션 엔드포인트라고 부른다.

Non-HTTP 추적 (or 프로파일 미확보 HTTP) 추적

트랜잭션 end-point 로 지정된 메소드가 시작해서 종료될때까지의 성능을 트랜잭션 성능이라고 한다. Non-HTTP 트랜잭션을 추적하기 위해서는 end-point 를 지정해야 한다.
거의 대부분의 HTTP 트랜잭션의 경우, 사전에 확인된 정보를 통해 end-point가 지정되어 있으나, 트랜잭션이 정상 프로파일링 되지 않는다면 end-point를 지정하도록 한다.

기본원칙

1. 트랜잭션이 시작되는 명확한 end-point를 설정한다. (**hook_httpservlet_classes**)

2. end-point가 불명확한 트랜잭션은 가급적 모든 모듈이 호출할 가능성이 높은 모듈(클래스 + 메소드)을 특정하여 back stack 을 확보한다.

    1) 메소드 레벨의 프로파일 추적 설정을 설정한다. (**hook_method_patterns**)
    2) 트랜잭션의 스택을 역추적 하기 위한 백스택을 확보한다.
    (**trace_auto_transaction_enabled**, **trace_auto_transaction_backstack_enabled**)
    3) 어플리케이션 서버를 재기동 하거나, 클래스를 재정의 한다. (**class redefine**)

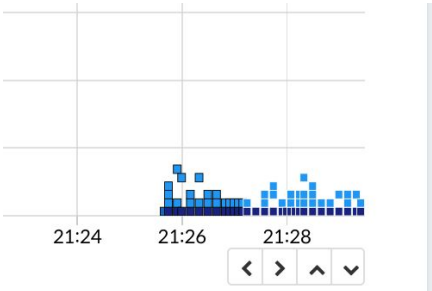
메소드 프로파일 설정

일단 메소드 프로파일을 설정해야한다. 하지만 해당 클래스들이 잠재적인 트랜잭션 end point 보다 선행 호출되면 적절한 프인트를 찾기 어려움으로 가급적 확실하게 트랜잭션 중간지점에 호출되는 클래스가 좋다.
그래서 이런것을 만족하는 클래스로는 JDBC드라이버를 생각할 수있다.

hook_method_patterns*=*jdbc.*.*
    * {full pakcage class명, * 사용가능}.{method 명, * 사용가능}
    * 가급적 모든 호출 경로에 노출된 만한 클래스를 Application Servers > More > Loaded Classes에서 확인하여 지정한다.
hook_method_access_public_enabled*=*true *(default 설정임)*
hook_method_access_private_enabled=false
hook_method_access_protected_enabled=true
hook_method_access_none_enabled=true
trace_auto_transaction_enabled*=*true

trace_auto_transaction_backstack_enabled=***true***

위와 같이 설정하고 재기동하면 트랜잭션이 추작되는 것을 볼수 있다.
* 재기동이 어려울 경우, Application Servers > More > Loaded Classes 에서 R (**Redefine**) 하여 클래스를 리로딩 한다.



Hit	Error
160388	0

End Time	Elapsed	
21:25:40	266	jdbc.FakePreparedStatement
21:25:39	99	jdbc.FakePreparedStatement
21:25:39	36	jdbc.FakeConnection.prepare
21:25:39	1	jdbc.FakeResultSet.next()Z
21:25:39	1	jdbc.Control.rsget()I
21:25:39	1	jdbc.Control.rsget()I
21:25:39	1	jdbc.FakeResultSet.next()Z
21:25:39	1	jdbc.FakeResultSet.next()Z
21:25:39	1	jdbc.FakeResultSet.next()Z
21:25:39	1	jdbc.Control.rsget()I
21:25:39	1	jdbc.Control.rsget()I
21:25:39	1	jdbc.Control.rsget()I
21:25:39	1	jdbc.FakeResultSet.next()Z

트랜잭션들을 조회해 보면 모든 jdbc.*로 시작하는 클래스의 메소드를 이 트랜잭션으로 나타남을 알 수있다.

트랜잭션 프로파일을 조회하면
<TRANSACTION BACKSTACK>라는 메세지 스탭을 확인할 수있다.

jdbc.FakePreparedStatement.executeQuery(FakePreparedStatement.java),
com.virtual.dao.SelectDAO.execute2(SelectDAO.java:29),
com.virtual.web.SimulaNonHttp.execute(SimulaNonHttp.java:147),
com.virtual.web.SimulaNonHttp.process(SimulaNonHttp.java:76),
com.virtual.web.SimulaNonHttp.run(SimulaNonHttp.java:100)

스택내용을 보면 어떤 메소드로 부터 출발하고 있는지 얼추 추정할 수 있다.
본예제 에서는
com.virtual.web.SimulaNonHttp.execute(SimulaNonHttp.java:147),
com.virtual.web.SimulaNonHttp.process(SimulaNonHttp.java:76),
com.virtual.web.SimulaNonHttp.run(SimulaNonHttp.java:100)

위 3개의 메소드 중에 하나를 트랜잭션 시작점으로 판단하면 된다.
이정도 상황에서는 역컴파일을 수행해서 적절한 트랜잭션 end point를 결정해야한다.

로직을 간단히 보면 SimulaNonHttp.run내에서 while()가 돌면서 SimulaNonHttp.process()을 호출하고 SimulaNonHttp.execute()가 수행된다. process()가 적당하다는 것을 알 수있다. (이 부분은 어쩔 수없이 소스를 보고 판단해야 한다.)

end point의 가장 중요한 기;준은 종료되어야 한다는 것이다. 정상적인 상황에서 지연되지 않고 곧바로 종료되어야 성능적인 판단을 할 수 있다.

트랜잭션 END POINT 지정하기

Non-HTTP 트랜잭션의 시작 지점을 아래와 같이 설정한다. (클래스+메소드명, 컴마 구분자 사용가능, *사용가능)

hook_service_patterns=com.virtual.web.SimulaNonHttp.process

HTTP 트랜잭션의 시작 지점은 아래와 같이 설정한다. (클래스명, 컴마 구분자 사용가능, *사용불가)

hook_httpservlet_classes=com.javax.Servlet

그리고 재기동하면 process() 메소드가 새로운 트랜잭션의 end point가 된다.

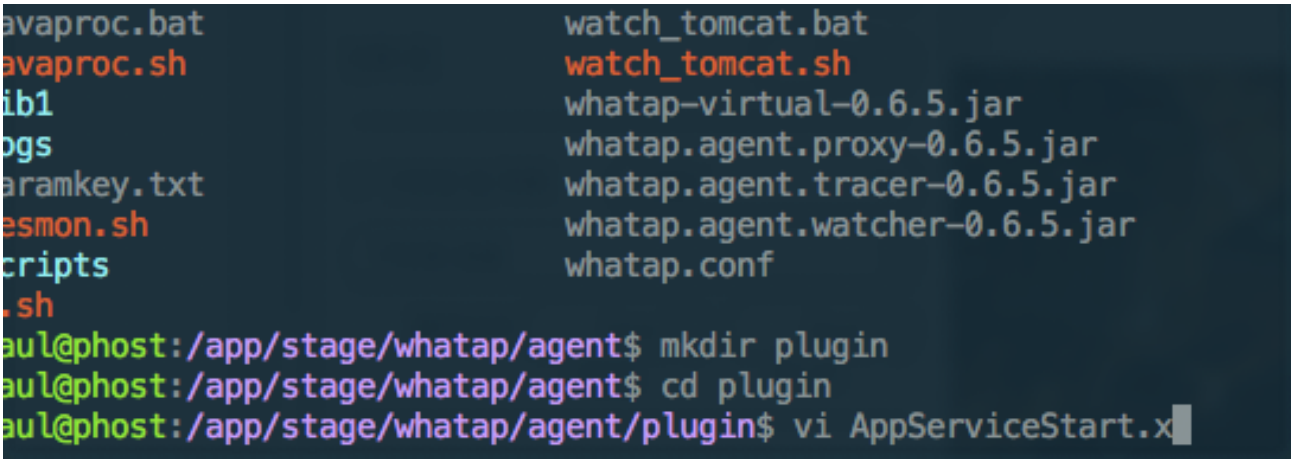
트랜잭션 이름 추적하기

보통의 경우에는 메소드 명칭으로 충분히 트랜잭션을 구분할 수 있다. 그래서 와탭은 메소드의 첫번째 String 파라미터를 트랜잭션 이름으로 사용한다.그런데 문자열 파라미터가 없는 경우에는 커스터마이징을 해야하는데 이때 와탭의 플러그인을 사용한다.

주의) 플러그인 사용은 전문가영역이기 때문에 충분히 이해한 경우에만 사용하길 권장한다.

에이전트 플러그인 사용하기

WHATAP_HOME 디렉토리 아래에 plugin디렉토리를 만든다.
그리고 vi를 통해 AppServiceStart.x 파일을 만든다.



그리고는 println("test");라고 타이핑하고 저장하면 화면에 "test"라는 문자열이 출력되는 것을 알 수있다.
그러면 파라미터에서 정보를 추출해 본다.

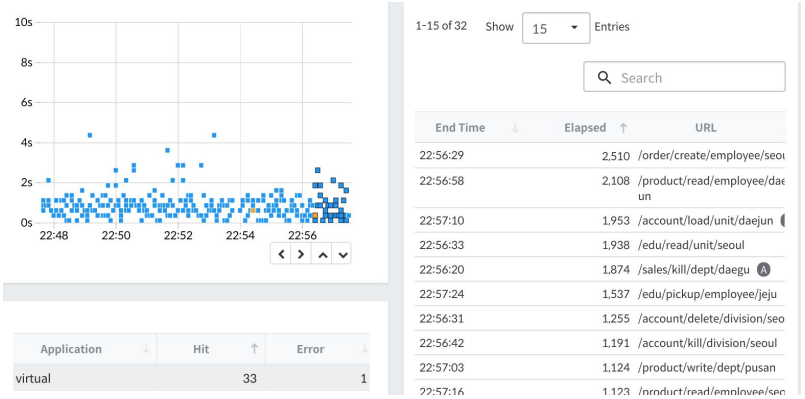
본예제에는 파라미터에 HashMap에 전달되고 거기에는 url파라미터가 전달되고 있다.

```
Object url =((java.util.HashMap)$point.getArgs()[0]).get("url");
$ctx.service((String)url);
```

```
//println("url="+url);
```

이렇게 플러그인을 만들면 트랜잭션 이름에 url값이 사용된다.

마무리



백그라운드 쓰레드나 통신 데몬의 성능을 추적하기 위해서는 가상의 트랜잭션을 지정한다.
그렇게 지정된 트랜잭션을 통해서 처리 성능을 튜닝한다.
그런데 만약 이렇게 만들어진 가상의 트랜잭션을 구분하는 명칭을 부여하려면 에이전트 플러그인을 활용한다.