Quality of Service Filter

Quality of Service Filter

The blocking nature of the standard servlet API makes it impossible to implement web applications that can guarantee some level of Quality of Service (QoS). Threads and memory are limited resources within a servlet container, yet with the standard servlet API, the only way to handle a HTTP request is with a thread allocated for the entire duration of the request. If a request is of low priority, or if other resources needed by the request are not available, then it is not possible to reuse the thread allocated to the request for high priority requests or requests that can proceed.

Jetty susports Suspendable Requests, which allows non-blocking handling of HTTP requests, so that that threads may be allocated in a managed way to provide application specific QoS. The QoSFilter is a utility servlet filter that uses Suspendable Requests to implement some QoS features.

Examples of the Problem

Waiting for Resources

Web application frequently uses JDBC Connection pool to limit the simultaneous load on the database. This protects the database from peak loads, but makes the web application vulnerable to thread starvation.

Consider a thread pool with 20 connections, being used by a web application that that typically receives 200 request per second and each request holds a JDBC connection for 50ms. Such a pool can service on average 200*20*1000/50 = 400 requests per second.

However, if the request rate raises above 400 per second, or if the database slows down (due to a large query) or becomes momentarily unavailable, the thread pool can very quickly accumulate many waiting requests. If (for example) the website is slashdotted or experiences some other temporary burst of traffic and the request rate rises from 400 to 500 requests per second, then 100 requests per second join those waiting for a JDBC connection. Typically, a web servers thread pool will contain only a few hundred threads, so a burst or slow DB need only persist for a few seconds to consume the entire web servers thread pool. This is called thread starvation.

The key issue with thread starvation, is that it effects the entire web application (and potentially the entire web server). So that even if the requests use the database are only a small proportion of the total request on the web server, all requests are blocked because all the available threads are waiting on the JDBC connection pool. This represents non graceful degradation under load and provides a very poor quality of service.

Prioritizing Resources

Consider a web application that is under extreme load. This load may be due to a popularity spike (slash dot), usage burst (christmas or close of business), or even a denial of service attack. During such periods of load, it is often desirable to not treat all requests as equals and to give priority to high value customers or administration users.

The typical behaviour of a web server under extreme load is to use all it's threads servicing requests and to build up a back log of unserviced requests. If the backlog grows deep enough, then requests will start to timeout and the users will experience failures as well as delays.

Ideally, the web application should be able to examine the requests in the backlog, and give priority to high value customers and administration users. But with the standard blocking servlet API, it is not possible to examine a

request without allocating a thread to that request for the duration of it's handling. There is no way to delay the handling of low priority requests and if the resources are to be reallocated, then the low priority requests must all be failed.

QoSFilter

The Quality of Service Filter (QoSFilter) uses Suspendable Requests to avoid thread starvation, prioritize requests and give graceful degredation under load, so that a high quality of service can be provided. The filter can be applied to specific URLs within a web application and will limit the number of active requests being handled for those URLs. Any requests in excess of the limit are suspended.

When a request completes handling the limited, URL, one of the waiting requests is resumed, so that it may be handled. Priorities may be given to each suspended request, so that high priority requests are resumed before low prority requests.

Required JARs

These JAR files must be available within WEB-INF/lib:

Jetty 6

- $JETTY_HOME/jetty-util.jar
- $JETTY_HOME/jre1.5/jetty-util5.jar - contains QoSFilter

Note that the QoSFilter requires at least Java 5, unlike the rest of Jetty-6 which only requires Java 1.4.

Jetty 7

- $JETTY_HOME/lib/jetty-util.jar
- $JETTY_HOME/lib/ext/jetty-servlet.jar - contains QoSFilter

Sample XML configuration (to be placed in a webapp's web.xml or jetty-web.xml)

This configuration will process only five requests at a time, servicing more important requests first, and queuing up the rest:

```
<filter>
  <filter-name>QoSFilter</filter-name>

<filter-class>org.mortbay.servlet.QoSFilter<
/filter-class>
  <init-param>
    <param-name>maxRequests</param-name>
    <param-value>50</param-value>
  </init-param>
</filter>
```

Mapping to URLs

You can use the `<filter-mapping>` syntax to map the QoSFilter to a servlet, either by using the servlet name, or by using a URL pattern. In this example, a URL pattern is used to apply the QoSFilter to every request within the web application context:

```
<filter-mapping>
  <filter-name>QoSFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Configuration

The following init parameters are supported:

maxRequests	the maximum number of requests to be serviced at a time.
maxPriority	the maximum valid priority that can be assigned to a request. A request with a high priority value is more important than a request with a low priority value.
waitMS	length of time, in milliseconds, to wait while trying to accept a new request. Used when the maxRequests limit is reached.

suspendMS	length of time, in milliseconds, that the request will be suspended if it is not accepted immediately. If not set, the container's default suspend period will be used.

Request Priority

The default request priorities assigned by the QoSFilter are:

- 2 - For any authenticated request
- 1 - For any request with a non-new valid session
- 0 - For all other requests

To customize the priority, subclass QoSFilter and then override the getPriority(ServletRequest request) method to return an appropriate priority for the request. Higher values have a higher priority. You can then use this subclass as your QoS filter. Here's a trivial example:

```
public class ParsePriorityQoSFilter extends
QoSFilter
{
    protected int getPriority(ServletRequest
request)
    {
        String p =
((HttpServletRequest)request).getParameter("
priority");
        if (p!=null)
            return Integer.parseInt(p);
        return 0;
    }
}
```