Tapestry5HowToWorkQueue

for occasional tasks that need a separate thread Tapestry5HowToRunTaskInThread is a nice example.

But if you want to create a work queue, then the requirement for PerThreadManager.cleanup() changes.

In this case you are strongly recommended to call PerThreadManager.cleanup() after each task so that every task can have clean Threaded services (hibernate session for example).

If you start a long running thread and use hibernate session, after some time you will face "strange" problems with entities. Unlike code in pages and components that gets a fresh hibernate session for each request, you will soon end up with stale data.

One fine implementation already exists in java.concurrent package ThreadPoolExecutor

Only small ammount of code is needed to make it work well within tapestry

```
package tapestryutil.services;

import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

import org.apache.tapestry5.ioc.services.PerthreadManager;
import org.apache.tapestry5.ioc.services.RegistryShutdownHub;
import org.apache.tapestry5.ioc.services.RegistryShutdownListener;
import org.slf4j.Logger;

/**WorkQueue implementation that is in-line with tapestry practices regarding threads.
 * It is important when using tapestry services to call PerthreadManager.cleanup(); after
 * a task if the same thread that executed the task will be reused again.
 *
 * @see {@link ThreadPoolExecutor}*/
public class WorkQueue extends ThreadPoolExecutor implements RegistryShutdownListener{

    protected final PerthreadManager _perthreadManager;
    private final Logger _log;

    public WorkQueue(PerthreadManager perthreadManager, Logger log, RegistryShutdownHub hub){
        super(1, 1, 0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());
        _perthreadManager = perthreadManager;
        _log = log;
        hub.addRegistryShutdownListener(this);
    }

    @Override
    protected void afterExecute(Runnable r, Throwable t) {
        super.afterExecute(r, t);
        _perthreadManager.cleanup();
    }

    public void registryDidShutdown(){
        int activeCount = getActiveCount();
        if(activeCount > 0) _log.warn(String.format("Shutting down worker and waiting for %d tasks to finish",
activeCount));
        shutdown();
    }

}
```

This is just a basic example from which you can work further. The limit for threads running is 1 but you can change this, or set the limits later on.

If you want to have less:

```
 new Runnable(){...}
```

arround your code you, and you want to create a service that handles some data (but with a queue).... do this:

```
package tapestryutil.services;

import org.apache.tapestry5.ioc.services.PerthreadManager;
import org.apache.tapestry5.ioc.services.RegistryShutdownHub;
import org.slf4j.Logger;

public abstract class DataWorkerQueue<T> extends WorkQueue{


    public DataWorkerQueue(PerthreadManager perthreadManager, Logger log, RegistryShutdownHub hub){
        super(perthreadManager, log, hub);
    }

    public void queueData(final T data){
        execute(new Runnable(){
            public void run() {
                handleData(data);
            }
        });
    }

    protected abstract void handleData(T data);

}
```

Now all you need to make a WorkerQueue for some operation on some data is to extend this class and implement handleData()

If you don't mind Runnables .. you can use it like this

```
package tapestryutil.services;

import org.apache.tapestry5.ioc.services.PerthreadManager;
import org.apache.tapestry5.ioc.services.RegistryShutdownHub;
import org.slf4j.Logger;

public class MailQueue extends WorkQueue{

    public MailQueue(PerthreadManager perthreadManager, Logger log, RegistryShutdownHub hub) {
        super(perthreadManager, log, hub);
    }

    public void queueEmail(final String address, final String subject){
        execute(new Runnable(){
            public void run() {
                sendEmail(address, subject);
            }
        });
    }

    private void sendEmail(String address, String subject){
        //send email
    }

}
```