Clustering Issues

Clustering Issues

The Servlet API was designed with the intention that there would be only a modest amount of server-side state, and that the stored values would be individual numbers and strings, and thus, immutable.

However, many web applications do not use the HttpSession this way, instead storing large, mutable objects in the session. This is not a problem for single servers, but in a cluster, anything stored in the session must be serialized to a bytestream and distributed to other servers within the cluster, and restored there.

Most application servers perform that serialization and distribution whenever HttpSession.setAttribute() is called. This creates a data consistency problem for mutable objects, because if you read a mutable session object, change its state, but *don't* invoke setAttribute(), the changes will be isolated to just a single server in the cluster.

Tapestry attempts to solve this: any session-persisted object that is read during a request will be re-stored back into the HttpSession at the end of the request. This ensures that changed internal state of those mutable objects is properly replicated around the cluster.

But while this solution solves the data consistency problem, it does so at the expense of performance, since all of those calls to setAttribute() result in extra session data being replicated needlessly if the internal state of the mutable object hasn't changed.

Tapestry has solutions to this, too:

@ImmutableSessionPersistedObject Annotation

Tapestry knows that Java's String, Number and Boolean classes are immutable. Immutable objects do not require a re-store into the session.

You can mark your own session objects as immutable (and thus not requiring session replication) using the ImmutableSessionPersistedObject annotation.

OptimizedSessionPersistedObject Interface

The OptimizedSessionPersistedObject interface allows an object to control this behavior. An object with this interface can track when its mutable state changes. Typically, you should extend from the BaseOptimizedSessionPersistedObject base class.

SessionPersistedObjectAnalyzer Service

The SessionPersistedObjectAnalyzer service is ultimately responsible for determining whether a session persisted object is dirty or not (dirty meaning in need of a restore into the session). This is an extensible service where new strategies, for new classes, can be introduced.