CachingIsNewStrategyFactory

February 23, 2015

Contents

1 Cache for IsNewStrategyFactory instances

The IsNewStrategyFactory is responsible of creating IsNewStrategy instances, for a given Class. The resulting instances are suitable of being cached:

- They are immutable and stateless.
- They get instantiated via reflection, which is an expensive operation.

1.1 Simple proposal: CachingIsNewStrategyFactory

The most straight-forward approach is by using a Delegate pattern: an envelop that adds a cache, and only performs the actual instantiation upon a cache miss.

We start by defining a new class CachingIsNewStrategyFactory, which implements IsNewStrategyFactory, to cache resolved {@link IsNewStrategy} instances per type to avoid re-resolving them on each an every request.

The new class must behave exactly as a regular IsNewStrategyFactory, so it has to implement IsNewStrategyFactory's method definitions. Any client using instances of this class shouldn't notice any difference.

Luckily for us, we only have to implement a method called getIsNewStrategy(Class<?>). Our purpose is simple:

1. Check if the strategy instance is already cached.

2. If it's cached, return the cached instance.

3. Otherwise, find out how to perform the expected instantiation, cache the outcome, and return it.

Obviously, we need a cache. Even though there're many caching solutions, initially we want to start simple, with a Map<Class<?>, IsNewStrategy>. However, we expect concurrent access to the cache, so let's use ConcurrentHashMap instead of the usual HashMap.

But we need also to wrap an actual IsNewFactoryStrategyFactory, and we want this class to be immutable (from the outside world, even though the cache itself is stateful).

1.1.1 getIsNewStrategy(Class<?> type)

The idea is, as described above, to gradually fill the cache with IsNewStrategy instances, so after a while all required strategy instances are already instantiated, effectively bypassing the reflection logic. Keep in mind that if the cached object gets stale, with this simple implementation we won't refresh it.

First, we check whether the cache contains a cached instance associated to the type. If it doesn't, we use our wrapped factory to perform the actual operation.

We assume that, in this Map-based implementation, the cost of calling Map.put() is not significant, and therefore we can cache the resulting instance regardless of if it was already in the cache or not.

Finally, we just return the new strategy instance.