ARTIGO: 11388
Veloster Benchmark
Introdução

Nesse artigo vamos apresentar os resultados dos testes de desempenho realizados com o Veloster Framework, comparando suas operações de CRUD com operações nativas, onde podemos ter uma ideia do desempenho do Veloster se comparado ao seu não uso.

Antes de mais nada, disponibilizamos o projeto do maven usado para os testes: veloster-benchmark.rar

Considerações

Quando falamos sobre framework de persistência sempre abrimos uma boa discussão, onde alguns defendem com "unhas e dentes" seus benefícios, enquanto outros defendem de maneira consistente seu não uso. Particularmente, nessas discussões, concordo com vários pontos de ambos os lados, sendo que na maioria dos casos não podemos resolver todos os problemas com um único framework. Por isso, acho que sempre devemos procurar o melhor framework, aquele que nos dá mais recursos e que também não nos deixe de mãos atadas no caso de precisarmos fazer alguma coisa sem ele, com isso chegamos aos frameworks não intrusivos, que além de terem essa concepção dependem de uma boa abstração no seu uso.

O Veloster, apesar de ser considerado intrusivo por não usar as anotações padrão definidas pela especificação do Java, pode ter sua utilização abstraída em certas camadas do sistema, permitindo seu abandono em qualquer fase do projeto (desde que você substitua essa camada) sem grandes complicações. Outro ponto a ser observado, é que normalmente aplicações que se destinam a dispositivos móveis não são gigantescas, e como em uma aplicação web, dificilmente você sentira a necessidade da mudança.

O maior benefício do Veloster é sua facilidade de uso e configuração. Em poucos minutos, você configura suas classes de persistência com validações, cria um DAO sem fazer SQL usando a api de criteria e cria interfaces repositório, que usam o conceito de métodos dinâmicos. Tudo isso em apenas alguns minutos, e quando você precisar dar manutenção nesse código terá a certeza de sua facilidade.

Muitos acham que não vale a pena usar um framework de persistência em dispositivos móveis, mas se compararmos seu custo beneficio em produtividade e organização de código com a construção manual de SQL's ou persistência em arquivos de texto, vemos que é um bom preço a se pagar. Abaixo veremos como saíram nossos testes de desempenho:

Benchmark

O dispositivo usado para os testes foi um Motorola Defy MB525, CPU ARMv7 Processor rev 2 com 477MB de memória e Android versão 2.3.4. A versão do Veloster usada foi:

veloster-api:1.7
veloster-droid:1.4

Os testes realizados foram os seguintes:

- Criação de uma classe modelo chamada Person com alguns atributos
- Criação de 100 objetos do tipo Person com alguns valores em seus atributos
- Monitoramento do tempo de inserção (insert) usando o Veloster (com e sem validação)
- Monitoramento do tempo de inserção (insert) usando a api nativa de SQL do Android
- Monitoramento do tempo de alteração (update) usando o Veloster (com e sem validação)
- Monitoramento do tempo de alteração (update) usando a api nativa de SQL do Android
- Monitoramento do tempo seleção (select) usando o Veloster (com e sem validação)
- Monitoramento do tempo de seleção (select) usando a api nativa de SQL do Android

Após executarmos os testes três vezes e tivemos os seguintes resultados:

Teste 01

Teste	Tempo Veloster	Tempo Nativo
Insert	6829 ms	5929 ms
Insert com validação	7025 ms	-
Update	8252 ms	5945 ms
Update com validação	8280 ms	-
Select	171 ms	140 ms

Teste 02

Teste	Tempo Veloster	Tempo Nativo
Insert	6336 ms	8060 ms
Insert com validação	6626 ms	-
Update	8228 ms	6098 ms
Update com validação	8038 ms	-
Select	178 ms	107 ms

Teste 03

Teste	Tempo Veloster	Tempo Nativo
Insert	6515 ms	5894 ms
Insert com validação	6998 ms	-
Update	7744 ms	6665 ms
Update com validação	7550 ms	-
Select	253 ms	137 ms

Teste 01

Droid Unti Result

VelosterPersonTestCase

Method: testPersonMergeWithValidation
Message: Benchmark Time: 8280 ms
Status: test passed ms
Time: 17991 ms

Method: testPersonMergeWithoutValidation
Message: Benchmark Time: 8252 ms
Status: test passed ms
Time: 15431 ms

Method: testPersonSaveWithValidation
Message: Benchmark Time: 7025 ms
Status: test passed ms
Time: 7120 ms

Method: testPersonSaveWithoutValidation
Message: Benchmark Time: 6829 ms
Status: test passed ms
Time: 6921 ms

Method: testPersonSelect
Message: Benchmark Time: 171 ms
Status: test passed ms
Time: 6747 ms

NativeQueryPersonTestCase

Method: testPersonMerge
Message: Benchmark Time: 5945 ms
Status: test passed ms
Time: 12462 ms

Method: testPersonSave
Message: Benchmark Time: 5929 ms
Status: test passed ms
Time: 6037 ms

Method: testPersonSelect
Message: Benchmark Time: 140 ms
Status: test passed ms
Time: 7999 ms

Teste 02

Droid Unti Result

VelosterPersonTestCase

Method: testPersonMergeWithValidation
Message: Benchmark Time: 8038 ms
Status: test passed ms
Time: 14929 ms

Method: testPersonMergeWithoutValidation
Message: Benchmark Time: 8228 ms
Status: test passed ms
Time: 15012 ms

Method: testPersonSaveWithValidation
Message: Benchmark Time: 6626 ms
Status: test passed ms
Time: 6719 ms

Method: testPersonSaveWithoutValidation
Message: Benchmark Time: 6336 ms
Status: test passed ms
Time: 6413 ms

Method: testPersonSelect
Message: Benchmark Time: 178 ms
Status: test passed ms
Time: 7411 ms

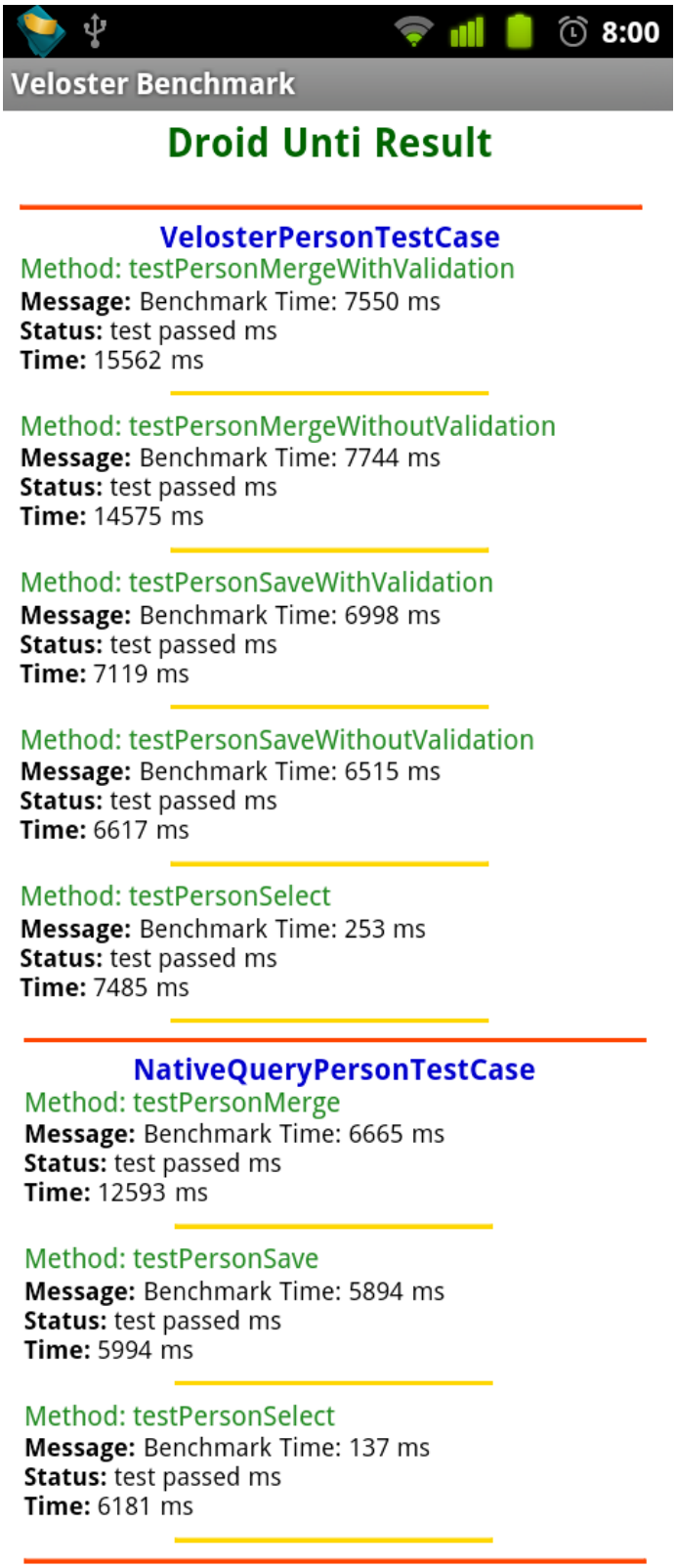
NativeQueryPersonTestCase

Method: testPersonMerge
Message: Benchmark Time: 6098 ms
Status: test passed ms
Time: 12991 ms

Method: testPersonSave
Message: Benchmark Time: 8060 ms
Status: test passed ms
Time: 8173 ms

Method: testPersonSelect
Message: Benchmark Time: 107 ms
Status: test passed ms
Time: 5831 ms

Teste 03



Conclusão

Com a realização dos testes, nós da Mobile Mind, apenas confirmamos que o preço a pagar pela utilização do Veloster Framework em nossos projetos é baixo, se comparado com a produtividade e fácil manutenção que ele oferece. Mas cada projeto é um caso, e necessita de uma analise previa comparando seus prós e seus contras.

.