Performance is een aspect van J2EE dat doorgaans weinig aandacht krijgt. Vaak wordt pas op het moment dat een systeem niet voldoende performance biedt ernaar gekeken en dan blijkt dat velen de benodigde kennis en vaardigheden missen om de performance te verhogen. Java Magazine sprak daarom met Kirk Pepperdine, CTO van javaperformancetuning.com, redacteur van TheServerSide maar bovenal Java performance goeroe.

Kirk Pepperdine over J2EE-performance

'De code reflecteert slechts wat je wil dat er gebeurt, niet wat er gebeurt'

Pepperdine was een paar dagen in Nederland omdat hij een workshop gaf, georganiseerd door Xebia. Java Magazine vroeg hem of het niet vreemd was dat er niet meer kennis was van performance, aangezien J2EE er juist is voor het maken van grote business-applicaties waarbij performance van groot belang zou moeten zijn?

Pepperdine: 'Het J2EE-framework is een heel interessant framework, het is heel succesvol en het is er om de complexiteit van grote gedistribueerde applicaties te verbergen. Dat was niet echt de intentie van J2EE aan het begin maar dat was het effectief werd, iets dat de developers van complexiteit afschermt. Als je kijkt naar de inspanning die het kostte om sommige van die grote systemen te bouwen voordat deze technologie op het toneel verscheen, had je een team van *rocket scientists* nodig om deze systemen te bouwen en te implementeren met enig succes. J2EE legt de lat lager voor de eisen aan de vaardigheden die een persoon nodig heeft om succesvol te zijn om applicaties te deployen. In dat opzicht is het heel succesvol geweest. Maar in het verlagen van de drempel zorgt het er ook voor dat de mensen die deze applicaties deployen ernaar neigen zich niet bewust te zijn van de meer subtiele issues die rond het ontwikkelen van deze grote applicaties spelen, vooral vanuit het performance perspectief. Hun job is het schrijven van business logica, ze begrijpen business, ze krijgen de requirements, ze begrijpen het framework en plaatsen de code in het framework en voilà ze hebben wat ze wilden en het is geweldig. Voor mij is het niet zo verwonderlijk dat ontwikkelaars op dat niveau zich niet bewust zijn van sommige van deze issues. Het maakt het niet minder belangrijk wat ze doen, maar je kunt je nu eenmaal niet bewust zijn van alles, en ze hebben gefocust op de dingen waarop ze dat zouden hebben moeten doen: het opleveren van functionaliteit die de business ondersteunt. Voor wat betreft developers die meer engineer-gericht zijn: misschien gaat dat terug op opleiding, wanneer ze opgeleid worden tot ontwikkelaars gaat het alleen maar om het schrijven van code om deze grote applicaties te schrijven. Je zou kunnen zeggen: systemen, maar ik vermijd dat woord opzettelijk omdat een systeem vooral vanuit een performanceperspectief iets anders is dan dat waar we van geleerd hebben ernaar te kijken op school. Dat wordt voortgezet tot in het oneindige want wanneer we erop uit gaan om applicaties te ontwikkelen kijken we nog steeds niet naar systemen maar ontwikkelen we nog steeds code. Het verschil tussen software development en performance tuning is dat je nu naar systemen moet gaan kijken en ontwikkelaars zijn niet gewend te kijken naar die dingen als systemen. Systemen zijn dynamisch, ze werken, er gebeuren allerlei dingen mee. De code reflecteert slechts wat je wilt dat er gebeurt, niet wat er gebeurt.'

Dus het is niet alleen de code op zichzelf die de performance bepaalt?

Pepperdine: 'Mensen! Zij vormen een heel belangrijk

deel van de systemen. Code is ook wel belangrijk, want die vertaalt wat de mensen willen doen in iets wat de machine kan begrijpen, maar de mensen sturen het gehele proces. Ik noem het mensen, het kunnen mensen zijn, maar ook externe systemen en een hele reeks van andere dingen, maar effectief maakt alles dat wil dat er werk gedaan wordt deel uit van het dynamische systeem. Net zoals de hardware, alle externe invloeden, al die dingen spelen een rol in de manier waarop het systeem zal gaan functioneren, dat gaat veel verder dan code.'

Een stukje code dat het heel goed doet op kleine schaal kan kleine rampen veroorzaken in grote applicaties. Zijn dat het soort fouten die u meestal tegenkomt, of zijn dat heel andere dingen?

Pepperdine: 'Ik denk dat het meer gaat om de dynamiek van het systeem. Laten we kijken naar die kleine 'codeerfout'. Was het echt een fout, of was het alleen een beslissing die iemand nam omdat ze dachten dat het onbelangrijk was, omdat ze keken naar de applicatie vanuit een zeer statisch gezichtspunt? In dat licht kunnen een aantal heel redelijke beslissingen heel grote vergissingen zijn, want je mist de dynamische component om te zeggen: OK, doet dit stukje code er echt toe in het complete systeem.

'Vrijwel ieder patroon dat je je kunt voorstellen heeft het potentieel een antipatroon te zijn'

Het gaat ook verder dan dat. Het gaat ook om het geven van de juiste aanwijzingen aan programmeurs. Hebben we ze de requirements meegegeven zodat ze een betere beslissing kunnen nemen? Het gaat ook over testen. Kunnen we die applicatie ook testen, laten onze requirements zien of het ook echt aan onze performance-eisen zal voldoen. Hebben we überhaupt wel gespecificeerde performance-eisen? Deze specifieke cursus gaat over al die dingen, over het proces. Ik probeer mensen ervan bewust te maken waaraan ze zouden moeten denken wanneer ze over performance denken. Je wil niet de manier waarop ze dingen doen radicaal veranderen, het is meer een kleine incrementele verandering. Ik specificeer een business functie requirement en ik ga met die functionaliteit ook informatie geven die me zal helpen te beslissen wat de performance-karakteristieken zouden moeten zijn in het uiteindelijke systeem. Die informatie is van vitaal belang om ontwikkelaars te doen begrijpen of ze aandacht moeten besteden aan een bepaald deel van de code of dat ze verder kunnen doen wat ze goeddunkt.'

De praktijk van het hergebruiken van code die je op internet vindt, is die juist goed of slecht voor de performance?

Pepperdine: 'O, wij doen het voortdurend, het is een geweldige praktijk en het is in feite iets dat heel goed is voor performance.'

Waarom? Omdat die code heeft bewezen te werken?

Pepperdine: 'Het ligt eraan hoe je ernaar kijkt. We gaan steeds meer naar een architectuur op componentniveau. Het punt is dat wanneer we een component gebruiken die niet zo geweldig is voor deze situatie, dat we door de architectuur met *loosely coupled* componenten in staat zouden moeten zijn deze componenten heel gemakkelijk te vervangen. JMS is een goed voorbeeld. Als je een product hebt en het werkt niet voor jou, als je de JMS interfaces regels volgt (component interfaces, de jms-interfaces) kun je het gewoon vervangen zonder iets aan de code te veranderen. Dat is een voorbeeld, JDBC-drivers zijn een ander voorbeeld van goed gedefinieerde componenten die je kunt vervangen. Appservers zijn er bijna, mensen graven er nog in en gaan tegen de API's in om heel specifiek gedrag te krijgen van een applicatie (wat goed of slecht kan zijn, dat is een andere discussie) maakt het veel gemakkelijker om componenten te schijven die we uit de een applicatieserver halen en in een andere applicatieserver plaatsen per configuratie. Deze dingen zijn dus goed voor de performance, het betekent dat een aantal van de moeilijkste keuzes van te voren moeten maken, we kunnen ze dichter bij de back-end maken wanneer we meer informatie hebben over wat de dynamiek van het systeem is.'

U spreekt regelmatig over anti-patterns voor performance. Kunt u er een paar noemen?

Pepperdine: 'Er zijn dingen die we doen die in het algemeen niet goed zijn voor performance. Dat is een bron voor *anti patterns*, anti-patronen. Er zijn ook patronen die we fout gebruiken, zij worden tot antipatronen. Wanneer ik praat over patronen of antipatronen richt ik me meestal vooral op de performance-aspecten van een patroon, niet op de architecturale belang of betekenis. Het idee is dat je mensen de architectuur laat definiëren, ze een patroon kiezen, en waar we over praten zijn de consequenties op het gebied van performance van het kiezen van een patroon in plaats van een ander. Het patroon dat we op het moment het liefst afkraken is DTO. Het is een heel handig patroon. Het heeft vele voordelen wanneer het op de juiste manier gebruikt wordt. Wanneer het echter ver genoeg uit de context haalt waarvoor het ontworpen is, wordt het een performance antipatroon en eerlijk gezegd ook een design antipatroon. een design antipatroon dat performance gevolgen heeft.

Vanuit design perspectief krijg je objecten parallel aan je business objecten, wat in het algemeen betekent dat je significant meer code hebt en je hebt meer onderhoud, want wanneer dingen veranderen moet je je bezig houden met veranderingen op een aantal plaatsen. Het is een performance antipatroon omdat het extra niveaus van marshalling toevoegt in je communicatielaag wanneer je die dingen moet pushen van de ene plaats naar de andere. Misschien kost het niet zo veel om er een of twee zo te doen, maar wanneer je over grote systemen praat begin je problemen met je netwerkresources te krijgen en je latency te vergroten en zo meer.

Vrijwel ieder patroon dat je je kunt voorstellen heeft het potentieel een antipatroon te zijn. Wanneer je het uit de context haalt dan wordt het ineens verkeerd gebruikt, en dan kan het positieve of negatieve invloeden hebben op de algemene performance van je applicatie. Wat lost DTO op in zijn pure vorm? Het is draaideur-antipatroon, want dat is wat een performance antipatroon is. Buiten zijn context zie je dat het geen goede keus is.'

En' shot in the dark'?

Pepperdine: 'Shot in the dark is een antipatroon. Enige relativering is wel op zijn plaats, want ik heb het zo genoemd en daarmee wordt het nog niet gedragen door de gehele Java community. Maar goed, de industrie doet dat soort dingen ook dus waarom niet?

Dat gezegd hebbende, een 'shot in the dark' zien we in feite wanneer een ontwikkelaar met een performance-probleem geconfronteerd wordt, en hij niet zo goed weet wat te doen. Hij heeft vele verwarrende informatie die hem bestormt, en neemt een willekeurig stuk informatie dat hem bereikt en hij begint veranderingen aan te brengen in de applicatiecode. Het komt voort uit een hoop stress. Managers zien je graag voor je beeldscherm zitten en code schrijven, want daar betalen we ontwikkelaars ook voor. Misschien heeft hij alle informatie die hij nodig heeft om de juiste aannames te maken, maar negeert de signalen. Misschien mist hij ook informatie, maar hij mist in ieder geval dat wat nodig is voor de juiste aanpak van het probleem. Hij heeft geen maatregelen genomen om het probleem beter te zien, zodat hij precies weet wat het is en dus ook wat hij eraan moet doen. Het grootste deel van de cursus gaat over de vraag: hoe neem ik de juiste maatregelen. Hoe stel ik de juiste vraag zodat ik weet welke maatregelen te nemen.'

Wat moet je doen wanneer je niet veel tijd hebt en je wordt geconfronteerd met een performanceprobleem?

Pepperdine: 'Bel een expert. Als je weinig tijd hebt en je hebt de kennis en vaardigheden niet in huis, ben je niet in de positie om die vaardigheden snel genoeg te verwerven. Het ligt er natuurlijk aan hoe kritiek de situatie is. Ik ben applicaties tegengekomen die op het punt stonden om te vallen. In die situatie heb je tijd niet om iets te doen en je bent ook niet in de stemming om nieuwe dingen te leren. Je geest staat er niet voor open, je moet iets doen. Daarom is het waarschijnlijk beter om er iemand bij te halen die je vertrouwt dat hij in staat is het probleem op te lossen. Het eerste gevoel van veel bedrijven is de vendor te bellen. Ik wil niet zeggen dat de vendors geen mensen hebben die dit kunnen. Er zijn zeer capabele mensen in dienst van vendors, maar onder tijdsdruk is het mogelijk dat je niet één van die goede mensen krijgt. Als je zo iemand inhuurt, doe dan net alsof het een sollicitatie betreft. Vraag naar referenties, stel de juiste vragen zodat je erachter komt of deze persoon je echt zal kunnen helpen. Je hebt namelijk niet de tijd om tot de ontdekking te komen dat je niet de juiste persoon gevonden hebt.'

'Code vertaalt wat men wil doen in iets wat de machine kan begrijpen, maar mensen sturen het gehele proces'

Hoe staat u tegenover het vergroten van de capaciteit van de hardware als oplossing?

Pepperdine: 'Ja, het werkt geweldig, zo lang je niet beperkt wordt door de theorie die zegt dat hardware niet helpt. Er zijn situaties waarin het toevoegen van

Advertentie
Pragm. Ontwikkelen

Jeroen Borgers van Xebia volgde de *Speeding Up Java Applications*-cursus met Kirk Pepperdine. Java Magazine vroeg hem naar zijn bevindingen.

Wat vond je van de cursus en hoe Kirk het gaf?
Borgers: 'Ik vond de cursus erg goed. Je kunt duidelijk merken dat Kirk boven de stof staat. Hij brengt een rijkdom aan ervaringen en praktijkverhalen met zich mee. Hij kent veel mensen uit de Java-wereld, kent veel verhalen en weet gevestigde opinies te ontkrachten. Zo dacht ik altijd dat James Gosling de Java-taal heeft ontwikkeld, toen nog onder de naam Oak. Niets is echter minder waar. Kirk wist mij te vertellen dat dit niet Gosling, maar Bill Joy was. Joy werkte eind jaren 70 al aan het idee voor de nieuwe taal! Kirk is redacteur van Java Developers Journal geweest en nu redacteur van The ServerSide. In die rol spreek je ook veel mensen en verzamel je veel kennis. Daarnaast is zijn voornaamste werk het oplossen van performance problemen bij bedrijven, waar ook ter wereld.
Verder heeft Kirk uitgelegd waarom het gevestigde idee dat object pooling goed is voor performance, niet klopt. Ook helpt hij Sun om diens performancetips te verbeteren, aangezien Kirk aantoonde dat juist bij toepassing van het *tegenovergestelde* van veel tips, de performance verbeterde.
Naast dat Kirk de theorie duidelijk overdraagt, besteedt hij veel aandacht aan het toepassen van de theorie. De cursus bestaat voor de helft van de tijd uit opdrachten. Dat is nuttig, want door het zelf te doen en ervaren leer je als deelnemer het meest. Hij geeft de deelnemers dan een voorsprong en na enige tijd gaat hij zelf de opgave voordoen op het scherm. Je ziet dan dus hoe de expert het probleem aanpakt en dat is heel leerzaam.
Je leert in de cursus om problemen te herkennen en met gebruik van eenvoudige tooling aan te tonen waar de bottlenecks optreden. Je krijgt ook handvaten aangereikt om de bottlenecks vervolgens aan te pakken en op te lossen. Ook leer je hoe de performance-problemen zijn te voorkomen.

Welk onderdeel vond je het meest leerzaam?
Borgers: 'Nou, zelf kende ik niet de details van garbage collection, de structuur van de heap en hoe je garbage collection kunt tunen om betere performance te halen. Nu weet ik dat wel. Zo hebben we inzicht gekregen in Mark en Sweep garbage collection door dit te simuleren. Daarnaast zagen we dat de performance van een programma met ruim een factor twee verbeterde door het kiezen van de juiste garbage collection JVM instelling. Daarmee krijg je dus op een heel eenvoudige wijze een twee keer zo snelle toepassing. Dat vind ik indrukwekkend. Ook hebben we de verschillen tussen de IBM heap en de Sun heap gezien en hoe je met tooling het garbage collection gedrag inzichtelijk kunt maken. Verder waren ook de aanpak, hoe te load testen, hoe te ontwerpen en de best practices erg leerzaam.'

Hoe ben je ertoe gekomen om deze cursus te organiseren?
Borgers: 'Zelf werk ik meerdere jaren aan de performance van Java-applicaties, als ontwikkelaar, architect, tester en in audits. Ik vind het een uitdagend en boeiend vakgebied. Het viel me op dat er helemaal geen cursussen op dit gebied gegeven worden in Nederland. Aangezien we al diverse cursussen geven, was het voor mij logisch om op het gebied van Java performance een cursus te gaan ontwikkelen. We hebben hierover toen met wat collega's gebrainstormd en kwamen tot de conclusie dat het veel werk zou zijn om dit zelf te doen. Daarna kwam ik op Internet de cursus van Kirk tegen en die kwam goed overeen met onze ideeën. Het contact was vervolgens snel gelegd. Uiteraard moesten we toen nog uitwerken hoe we dit gingen aanpakken. Dat is allemaal gelukt en het heeft uiteindelijk tot deze cursus met Kirk geleid. Het was leuk om te zien dat de deelnemers er zo tevreden over waren.'

Meer informatie op http://blog.xebia.com/2006/09/06.

hardware helpt. Maar er zijn er ook waarin het niet helpt en het de situatie mogelijk nog slechter maakt. We hadden laatst bij een klant te maken met een programma dat op een hoger niveau heuristische zaken deed met order management, in feite datamining. Het duurde vier uur om het te laten draaien en ze hadden twee dozen dus men ging ervan uit dat het met acht dozen een uur zou duren. Maar toen die dozen aangeschaft waren bleek het tien uur te duren. Ze begrepen niet waar de bottlenecks in hun systemen zaten en ze begrepen niet wat het effect zou zijn van het uitoefenen van meer druk op die bottlenecks. Dat was dus een geval waarin het toevoegen van hardware niet hielp en er zijn meer van dat soort gevallen dan de meeste mensen denken. Maar in het algemeen, als je architectuur netjes is, en er zijn geen horizontale of verticale schaalbaarheidsissues, dan is hardware soms de beste oplossing voor een probleem. Het is ook een kwestie van kosten. Als je maanden nodig hebt om te tunen en je kunt het oplossen met een extra server, dan is dat een goede oplossing. Je moet alleen wel uitzoeken of die extra server het probleem ook zal oplossen.'

Tekst en fotografie: Dré de Man.

Advertentie
Info Support