

Inhaltsverzeichnis

Vorwort	13
A Java-Memory-Modell	15
1 Einführung in das Java-Memory-Modell	19
1.1 Atomare Zugriffe	21
1.2 Sequential Consistency	22
1.3 Sichtbarkeitsregeln für volatile-Variablen	23
1.4 Sichtbarkeitsregeln für Synchronisation	24
1.5 Zusammenfassung	26
2 Das Java-Memory-Modell im Überblick	27
2.1 Das Java-Memory-Modell	28
2.2 Sichtbarkeitsregeln im JMM	30
2.2.1 Atomicity	30
2.2.2 Ordering	31
2.2.3 Visibility	31
2.3 Zusammenfassung	34
3 Die Kosten der Synchronisation	35
3.1 Performanceeinbußen durch Synchronisation	36
3.2 Skalierbarkeitseinbußen durch Synchronisation	37
3.2.1 Amdahl's Law	38
3.2.2 Vollständig parallel?	39

Inhaltsverzeichnis

3.3	Sequenzialisierung bekämpfen	42
3.4	Zusammenfassung	45
4	Details zu volatile-Variablen	47
4.1	volatile als Alternative zur Synchronisation	47
4.2	Missverständnis	50
4.3	Kosten von volatile	52
4.4	Speichereffekte von volatile auf andere Variablen	53
4.5	volatile-Referenzvariablen	56
4.6	Zusammenfassung	60
5	volatile und das Double-Check-Idiom	61
5.1	Ausgangssituation	61
5.2	Das Double-Check-Idiom	62
5.3	Optimierung à la Bloch	65
5.4	Single-Check-Idiome	68
5.5	Zusammenfassung	72
6	Regeln für die Verwendung von volatile	73
6.1	Effekte von Synchronisation	73
6.2	Regeln für den Einsatz von volatile	75
6.2.1	Unabhängigkeit vom gegenwärtigen Wert	75
6.2.2	Unabhängigkeit von anderen Variablen	76
6.3	Beispiele für den erfolgreichen Einsatz von volatile	77
6.3.1	Einmalige Ereignisse	77
6.3.2	Mehrfachveröffentlichung von Informationen	80
6.3.3	Kombination von Lock und volatile	81
6.3.4	Ziel des Einsatzes von volatile	82
6.4	Zusammenfassung	84

7	Die Initialisation-Safety-Garantie	85
7.1	Das Racy-Single-Check-Idiom	85
7.1.1	Racy-Single-Check mit einem primitiven Typ (außer long und double)	85
7.1.2	Racy-Single-Check mit einem Referenztyp	86
7.2	Anforderungen an unveränderliche Typen	88
7.3	Speichereffekte im Zusammenhang mit final-Feldern	89
7.3.1	Sichtbarkeitsprobleme im Detail	91
7.3.2	Sichtbarkeitsgarantien für abhängige Objekte	94
7.4	Unterschiede zu volatile	98
7.5	final Variablen vs. final Felder	98
7.6	Zusammenfassung	100
8	Über die Gefahren allzu aggressiver Optimierungen	101
8.1	Racy-Single-Check und unveränderlichen Typen	101
8.2	Genereller Verzicht auf Synchronisation/ volatile bei unveränderlichen Typen?	103
8.3	Race Conditions bei der Konstruktion von Objekten	106
8.4	Zusammenfassung	111
9	Atomic Scalars	113
9.1	Ein Thread-sicherer Zähler mit Synchronisation	114
9.2	CAS – Compare and Swap	115
9.3	Ein Thread-sicherer Zähler unter Verwendung von atomaren Variablen	116
9.4	Zusammenfassung	118

Inhaltsverzeichnis

10	Atomare Referenzvariablen	119
10.1	Fallstudie zu atomaren Referenzen	119
10.1.1	Sichtbarkeitsprobleme	121
10.1.2	Race Condition	121
10.1.3	Optimierung mit atomaren Referenzen	123
10.2	Atomare Referenzen – Übersicht	127
10.3	Praxisrelevanz	129
10.4	Zusammenfassung	130
11	CopyOnWriteArrayList	133
11.1	Was ist eine CopyOnWriteArrayList?	133
11.2	Wie funktioniert eine CopyOnWriteArrayList?	134
11.3	Wie funktioniert der Iterator von CopyOnWriteArrayList	138
11.4	Eine andere Anwendung desselben Prinzips	140
11.5	Zusammenfassung	143
	Verweise zu Teil A	145
B	Memory Management und Garbage Collection	147
12	Generational Garbage Collection	149
12.1	Generational Garbage Collection	150
12.2	Die Heap-Aufteilung in einer Sun JVM	153
12.2.1	Perm-Space	154
12.2.2	Young Generation	154
12.2.3	Old Generation	155
12.3	Vor- und Nachteil der Generational Garbage Collection	156
12.4	Zusammenfassung	158

Inhaltsverzeichnis

13 Young Generation Garbage Collection	159
13.1 Mark and Sweep	159
13.2 Mark and Copy	162
13.2.1 Die Mark-and-Copy-Variante in der Sun JVM	164
13.2.2 Vor- und Nachteile von Mark and Copy auf der Young Generation	166
13.2.3 Intergenerational References	167
13.2.4 Parallele Garbage Collection	170
13.3 Parallele Allokation und TLABs	173
13.4 Zusammenfassung	174
14 Old Generation Garbage Collection – Teil 1	175
14.1 Wozu braucht man auf der Old Generation einen anderen Algorithmus?	176
14.2 Serial Mark and Compact	178
14.3 Alternative Algorithmen für die Old Generation	182
14.4 Parallel Mark and Compact	184
14.5 Zusammenfassung	187
15 Old Generation Garbage Collection – Teil 2	189
15.1 Concurrent Mark and Sweep	189
15.2 Parallele Threads in CMS	193
15.3 Synchronisationsbedarf und Interaktion mit der Young Generation Garbage Collection	194
15.4 Rückfall in die serielle Garbage Collection	196
15.5 Vor- und Nachteil des CMS-Collectors	197
15.6 Zusammenfassung	198

Inhaltsverzeichnis

16	Garbage Collection Tuning – die Ziele	199
16.1	Garbage-Collection-Tuning-Ziele	199
16.1.1	Durchsatzziel	200
16.1.2	Pausenziel	201
16.1.3	Relevanz der Garbage-Collection-Tuning-Ziele	202
16.1.4	Durchsatz	203
16.1.5	Pausen	204
16.1.6	Speicherverbrauch	205
16.2	Multi-Core/Multi-Prozessor-Architektur und Garbage Collection	207
16.3	Zusammenfassung	209
17	Garbage Collection Tuning – die Details	211
17.1	Auswahl des Garbage Collectors	211
17.2	Anpassen der Collectors an die Objektpopulation der Applikation	214
17.2.1	Objektpopulation der Applikation und Generationenauslastung	214
17.2.2	Anpassen der Collectors	220
17.2.3	Hoher Durchsatz	221
17.2.4	Konstant kurze Pausen	222
17.2.5	Selbstanpassung der Collectors	225
17.3	Zusammenfassung	226

Inhaltsverzeichnis

18 Garbage-First (G1) Garbage Collector – Teil 1	229
18.1 Motivation	229
18.2 Funktionale Übersicht	231
18.2.1 Aufteilung in Regionen	231
18.2.2 Objektallokation	232
18.2.3 Objektevakuierung	233
18.3 Zusammenfassung	236
18.4 Danksagung	236
19 Garbage-First (G1) Garbage Collector – Teil 2	237
19.1 Details der G1-Verwaltung	238
19.1.1 Remembered Sets	238
19.1.2 Concurrent Marking	240
19.2 Die Garbage-Collection-Pause	242
19.2.1 Bestimmung des Starts einer Collection-Pause	245
19.3 G1 Collection Lifecycle	246
19.4 Zusammenfassung	248
Verweise zu Teil B	249