

Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>. Saisissez la référence ENI
de l'ouvrage **EIECR** dans la zone de recherche et validez.
Cliquez sur le titre du livre puis sur le lien de téléchargement

Préface

Chapitre 1

Introduction

1. Avant-propos 13
2. Les risques de la sur-optimisation 14
3. Le principe du profilage 16
4. Approche pédagogique adoptée par l'auteur 18
5. Les éléments à télécharger 21

Chapitre 2

Principes du profilage

1. Une activité strictement régulée 23
2. Stabilité de la plate-forme 28
 - 2.1 Pourquoi cette règle ? 28
 - 2.2 Comment appliquer cette règle 29
 - 2.3 Extraire les cas particuliers 30
 - 2.4 Expérience vécue 31
 - 2.5 Lien avec la maintenabilité 32
3. Neutralité de l'environnement 34
 - 3.1 Le principe 34
 - 3.2 Virtualisation 35
 - 3.3 Déport d'affichage 36
 - 3.4 Effets de caches externes 37
 - 3.5 Anti-virus 38
 - 3.6 Services divers 39

2 _____ Écrire du code .NET performant

Profilage et bonnes pratiques

3.7	Comparaison avec la nano-technologie	40
4.	Objectifs fixés avant l'analyse	41
5.	Améliorations mesurables	43
5.1	Pourquoi cette règle ?	43
5.2	Comment appliquer cette règle	44
5.3	Expérience vécue	44
6.	Granularité descendante	45
6.1	Commençons par une parabole	45
6.2	Un exemple	46
6.3	Avertissement	46
6.4	À qui la faute ?	51

Chapitre 3

Profilage d'une application .NET

1.	Gestion de la mémoire par .NET	53
1.1	Principe de base	53
1.2	Gestion de mémoire automatisée et performances	54
1.3	Le cas particulier du temps réel	54
1.3.1	Lever un malentendu	54
1.3.2	Non-déterminisme des ramasse-miettes	56
1.4	Affectation de la mémoire	59
1.5	Comment .NET nous aide	64
1.6	Types valeurs et références	65
1.6.1	Fonctionnement d'une pile	66
1.6.2	Retour sur le code IL	71
1.6.3	Différence entre valeur et référence	74
1.6.4	Cas particulier des structures	79
1.6.5	Lien à la performance	80
1.7	Calcul de la taille mémoire à affecter	81
1.7.1	Cas du codé 100 % géré	81
1.7.2	Cas des objets sujets à interopérabilité	82

1.8	Collecte de la mémoire	84
1.8.1	Critères de collecte de la mémoire	84
1.8.2	Mécanisme de compactage	86
1.8.3	Mémoire fixée et fragmentation	87
1.8.4	Déclenchement et exécution du ramasse-miettes.	89
1.8.5	Notion de génération	90
1.8.6	Impact sur le codage pour la performance	91
1.8.7	Choix du ramasse-miettes	95
1.9	Boxing, unboxing et performances associées	96
1.9.1	Quel est le problème ?	96
1.9.2	Le boxing / unboxing et les performances	100
1.9.3	Le remède	102
1.10	Gestion de mémoire et finalisation	105
1.10.1	Laissons faire .NET !	105
1.10.2	Relâcher des ressources externes lors du passage du GC	106
1.10.3	Relâcher les ressources au plus tôt	109
1.10.4	Combiner les deux opérations	112
1.11	Une dernière remarque	116
2.	Particularité des fonctions inline	117
2.1	Mécanisme des fonctions inline	117
2.2	Problématiques de performance et fonctions inline	120
2.3	Impact sur les profileurs	123
3.	Impact de la gestion mémoire sur la performance	129
3.1	Une grande diversité dans les impacts	129
3.2	Utilisation de la mémoire virtuelle	130
3.3	Fuites mémoires	134
3.4	Tas spécial pour les gros objets et fragmentation mémoire	140
4.	Les autres ressources à surveiller	142
4.1	La mémoire n'est pas tout	142
4.2	Le CPU	143
4.3	Les entrées/sorties	146

4 — Écrire du code .NET performant

Profilage et bonnes pratiques

4.4	L'espace disque disponible	149
4.5	La bande passante	150

Chapitre 4

Application de test

1.	Critères de choix.	153
1.1	Pourquoi une vraie application ?	153
1.2	Utiliser des retours d'expérience.	154
1.3	Le choix de la transparence	154
1.4	Les limites de la transparence	156
2.	Application choisie	156
2.1	Domaine d'utilisation	156
2.2	Architecture	157
2.3	Interface	157
2.4	Description du métier	157
3.	Détail de l'application	158
3.1	Trouver l'application	158
3.2	Pourquoi PROFI ?	159
3.3	Installation de la base de données	159
3.3.1	Création manuelle.	160
3.3.2	Utilisation des scripts de génération	161
3.3.3	Restauration de la sauvegarde	162
3.3.4	Attachement des fichiers de base.	162
3.4	Installation de l'application	163
3.4.1	Ouverture de la solution	163
3.4.2	Compilation de la solution	164
3.4.3	Lancement de l'application	165
3.5	Détails des assemblages.	168
3.6	Architecture du client	169
3.7	Structure des services web.	170
3.8	Structure de la base de données.	170

4. Explication de la lourdeur de l'application. 171
5. Méthode recommandée 172

Chapitre 5
Présentation des outils

1. Choix des outils 175
2. Red Gate ANTS Performance Profiler 176
 2.1 Présentation 176
 2.2 Points forts 178
 2.3 Détail de l'utilisation pour l'analyse du serveur 179
 2.3.1 Gestion par le profileur des serveurs web. 179
 2.3.2 Utilisation d'IIS sans redirection de port 180
 2.3.3 Utilisation d'IIS avec redirection de port 182
 2.3.4 Utilisation du serveur web de développement 183
 2.3.5 Cas particulier de .NET 4 185
 2.4 Conclusion 187
3. JetBrainsDotTrace Performance 188
 3.1 Présentation 188
 3.2 Points forts 189
 3.3 Modes de profilage 190
 3.4 Conclusion 191
4. YourKit Profiler for .NET 191
 4.1 Présentation 191
 4.2 Points forts 195
 4.3 Conclusion 197
5. Equatec Profiler 197
 5.1 Présentation 197
 5.2 Points forts 200
 5.3 Conclusion 200
6. Profileurs Open Source 200

6 — Écrire du code .NET performant

Profilage et bonnes pratiques

Chapitre 6

Profilage du client

1. Cible.	203
2. Configuration du profilage	204
2.1 Scénarios de profilage	204
2.1.1 Scénario sur les personnes	204
2.1.2 Scénario sur les contrats	207
2.2 Besoin d'automatisation des interactions.	209
2.2.1 NUnitForms	210
2.2.2 UIAutomation.	211
2.2.3 Directives de compilation	212
2.3 Configuration du profileur	217
3. Analyse des résultats sur le premier scénario.	219
3.1 Chargement des composants.	219
3.1.1 Analyse des résultats.	219
3.1.2 Diagnostic.	224
3.1.3 Première tentative d'optimisation	225
3.1.4 Seconde étape d'optimisation	228
3.2 Instanciation des contrôleurs.	229
3.2.1 Analyse	229
3.2.2 Diagnostic.	230
3.2.3 Optimisation.	231
3.2.4 Résultats	232
3.2.5 Poussons l'optimisation.	232
3.2.6 Résultats	235
3.3 Construction des clients de service web.	236
3.3.1 Analyse	236
3.3.2 Diagnostic.	237
3.3.3 Décision	239
3.4 Tentative d'optimisation de l'authentification.	240

3.5	Récupération des settings du serveur	242
3.5.1	Analyse	242
3.5.2	Optimisation du code	244
3.5.3	Résultats	244
3.5.4	Libération des ressources	245
3.5.5	Occupation mémoire	247
3.6	Circulation des messages	248
3.6.1	Remarque préliminaire sur les décomptes d'appels	248
3.6.2	Analyse	251
3.7	Initialisation des clients d'appel aux services web	254
3.7.1	Analyse	254
3.7.2	Diagnostic	256
3.7.3	Correction	258
3.7.4	Résultats	262
3.7.5	Optimisation supplémentaire	263
3.8	Retour sur le premier scénario	264
4.	Analyse des résultats sur le second scénario	265
4.1	Contexte de l'analyse	265
4.2	Chargement des composants	267
4.3	Instanciation des clients de services web	268
4.4	Chargement d'un contrat	270
4.4.1	Analyse	270
4.4.2	Transformation XSL/T	272
4.4.3	Optimisation possible	277
4.4.4	Résultats de l'optimisation sur le XSL/T	279
4.4.5	Utilisation de l'expression régulière	282
4.4.6	Résultat de l'optimisation de l'expression régulière	283
4.5	Remplissage de la liste des contrats	284
4.6	Vérifications supplémentaires	285
4.6.1	Concept	285
4.6.2	Manipulation	287
4.6.3	Analyse des résultats	289

8 _____ Écrire du code .NET performant

Profilage et bonnes pratiques

Chapitre 7

Profilage du serveur

1.	Configuration de l'application web à profiler	295
1.1	Mode de débogage	295
1.2	Authentification éventuelle	297
1.3	Besoin de démarrer en mode administrateur	299
1.4	Gestion des settings sur les URL de services web	308
2.	Lancement du profilage sur le premier scénario	315
2.1	Profilage de la mémoire	315
2.2	Méthode détaillée de lancement	316
2.3	Premiers résultats	317
2.4	Méthode d'analyse	318
2.5	Récupération des settings serveur	319
2.5.1	Analyse	319
2.5.2	Appels multiples d'une méthode web inutiles	321
2.5.3	Traitement de la complexité du calcul des settings	325
2.5.4	Manipulation de chaînes inefficace	327
2.5.5	Conditions d'utilisation des StringBuilder	329
2.5.6	Mise en place d'un cache	333
2.5.7	Limitation de l'approche avec cache	338
2.5.8	Utilisation correcte de Count	340
2.5.9	Traitement sur la casse	341
2.5.10	Suppression des accents	350
2.5.11	Sortie finale de la chaîne	352
2.5.12	Bilan sur les settings	353
2.6	Lecture des données de contrat	354
2.6.1	Suppression des fichiers temporaires	356
2.6.2	Fusion du document de contrat	357
2.6.3	Extraction du contrat type	363
2.6.4	Fusion par manipulation XML	364
2.6.5	Fusion en mode texte	366
2.6.6	Utilité de la fusion initiale du contrat	368
2.6.7	Code mort	373

2.7	Lecture de données au niveau personne	376
2.8	Poids des échanges de services web	376
2.8.1	Méthode de capture	376
2.8.2	Résultats sur le scénario	379
2.8.3	Utilitaires de limitation de ressources	380
3.	Lancement du profilage sur le second scénario	382
3.1	Contexte	382
3.2	Profilage initial	383
3.3	Requête SQL complexe	384
3.3.1	Diagnostic	384
3.3.2	Le profileur du pauvre	385
3.3.3	Fonctionnalités SQL des profileurs	386
4.	Lancement des scénarios en charge	388
4.1	Concepts	388
4.2	Caveat	389
4.3	Présentation de WebLOAD	389
4.4	Enregistrement des scénarios	391
4.5	Création de la session de charge	400
4.6	Analyse des résultats WebLOAD	414
4.7	Mélanger WebLOAD et ANTS Performance Profiler	415
4.8	Correction problème sur Random	417
4.9	Analyse finale des résultats	419
4.10	Comparaison avec le serveur non optimisé	423
4.10.1	Collecte des résultats	423
4.10.2	Comparaison des sorties WebLOAD	426
4.10.3	Comparaison des sorties ANTS Performance Profiler	429
5.	Remarques supplémentaires	431
5.1	Avertissement du profileur	432
5.2	Utilisation d'autres profileurs	433
5.3	Effets de démarrage	433
5.4	Faible dans l'authentification	433

Chapitre 8

Au-delà du profilage

1. Introduction	435
2. Pistes d'amélioration restantes	436
2.1 Introduction	436
2.2 Vitesse d'affichage	437
2.3 Temps de chargement de l'application.	441
2.3.1 NGen	441
2.3.2 XmlSerializers	442
2.4 Amélioration du ressenti	444
2.4.1 Performance réelle et performance perçue	444
2.4.2 Asynchronisme des traitements.	445
2.4.3 Lazy-loading	450
2.4.4 SplashScreen	452
2.4.5 Marquer les changements d'application.	453
2.5 Quelques dernières problématiques	454
2.5.1 Gestion correcte des traces	454
2.5.2 Duplication des requêtes SQL	455
2.5.3 Éviter la sur-architecture	457
2.5.4 Ne pas oublier les fichiers temporaires.	458
2.5.5 Pagination des résultats.	460
2.5.6 Le ramasse-miettes prend du temps.	460
2.5.7 Limiter les exceptions	462
2.5.8 Fonctions Equals et GetHashCode.	463
2.5.9 AddRange	464
3. Tuning	465
3.1 Caveat.	465
3.2 La base : compiler en release	467
3.3 Le curseur de la consistance.	468
3.3.1 BASE au lieu d'ACID.	468
3.3.2 Un second exemple	469
3.3.3 Passer le code de PROFI en BASE	471

3.4	Appels HTTP "hard-codés"	475
3.5	Asynchronisme globalisé	477
3.6	Limiter la réflexion	478
3.7	Conseils pour ASP.NET	478
3.8	Utiliser des références faibles	479
3.9	Attention au tuning extrême	480
3.9.1	Limites du tuning	480
3.9.2	Struct au lieu de class	480
3.9.3	Instanciation tardive et déréférencement précoce	482
3.9.4	Byte au lieu de int dans Enum ?	482
4.	Aller plus loin en réarchitecturant	483
4.1	Problématique	483
4.2	Scalabilité	484
4.2.1	Concept	484
4.2.2	Modes de scalabilité	485
4.2.3	Parallélisation des traitements	485
4.2.4	Bonnes pratiques pour la scalabilité	486
4.2.5	Parallel linq	486
4.3	Institutionnaliser le cache	488
4.4	Penser Lean / Agile	488
4.4.1	IMDB	488
4.4.2	NoSQL	489
4.4.3	CQRS	489
4.4.4	Prévalence	490
	490
4.5	Performance des nouvelles architectures	491
4.5.1	Problématique	491
4.5.2	Scale Out	491
4.5.3	Parallélisation	492
4.5.4	Mobilité	492
4.5.5	SOA / EDA / ESB	492
4.6	Et pour aller encore plus loin	494

12 _____ Écrire du code .NET performant

Profilage et bonnes pratiques

Chapitre 9

Conclusion

1. Tout peut poser problème	497
2. Checklist	498
3. Les causes des erreurs	500
4. Coder léger	501
5. Conclusion	503
Index	505