Université de Lille

FACULTÉ DES SCIENCES ET TECHNOLOGIES
Université de Lille

Projet GL - Partie 2

Réalisé par :
- Madjid DERMEL

Sommaire:

Méthodologie et Réflexions

Au cœur de mon projet de Génie Logiciel (GL) dédié à l'amélioration de WebMagic, un framework Java essentiel au crawling et au scraping de données web, j'ai adopté une approche systématique pour renforcer sa structure et sa facilité d'extension.

La démarche que j'ai suivie s'est appuyée sur l'utilisation rigoureuse d'un dépôt GitHub. Chaque amélioration apportée au projet a été soigneusement implémentée à travers des commits spécifiques, reflétant une modification ciblée et réfléchie. Cette pratique de commits atomiques et descriptifs m'a permis non seulement d'organiser mon travail de manière logique mais également de faciliter la compréhension et la révision par les utilisateurs du projet.

Pour chaque commit, j'ai veillé à rédiger des commentaires explicites, soulignant l'objectif et la justification des modifications introduites. Cette étape cruciale a contribué à illustrer mon processus de réflexion et les raisons sous-jacentes de chaque amélioration, renforçant ainsi la clarté du projet.

Le projet GL, divisé en deux parties, a débuté par une évaluation approfondie de la qualité logicielle de WebMagic. Cette première phase a permis d'identifier les domaines clés nécessitant des améliorations, tels que la réduction de la complexité du code, l'augmentation de la couverture des tests, et l'optimisation de la structure interne pour une meilleure flexibilité et maintenabilité. La seconde partie du projet s'est concentrée sur la mise en œuvre pratique de ces améliorations.

Les améliorations apportées

Petites modifications

Suppression du code déprécié

J'ai supprimé le code déprécié pour alléger et moderniser la base de code, encourager l'utilisation des méthodes actuelles plus robustes et mieux soutenues, et préparer le terrain pour de futures améliorations sans être entravé par des éléments obsolètes.

Suppression d'un paramètre non utilisé

J'ai supprimé le paramètre Site non utilisé dans la méthode convertHttpClientContext de la classe HttpUriRequestConverter. Cette modification rend le code plus propre et plus précis en éliminant un argument inutile, améliorant ainsi la clarté de la signature de la méthode et évitant toute confusion sur les données nécessaires pour la conversion.

Suppression de printStackTrace dans destroyEach de Spider

J'ai retiré printStackTrace suite à une recommandation de SonarQube, pour éviter la fuite d'informations sensibles en production. Cette action améliore la sécurité en empêchant l'exposition de détails internes du système et assure une meilleure gestion des erreurs.

Définition d'une constante pour un message d'erreur répété

Le message d'erreur "illegal encoding " est répété plusieurs fois. Pour résoudre cela, j'ai défini ce message comme une constante et je l'ai réutilisé quand c'est nécessaire.

Spécification de type avec opérateur diamant

L'utilisation de l'opérateur diamant "<>" est recommandée pour éviter la redondance dans les spécifications de type lors de l'instanciation d'objets génériques.

Moyennes modifications

Ajout d'une méthode utilitaire pour valider les URL

J'ai introduit la méthode isValidUrl pour centraliser et simplifier la logique de validation des URL à plusieurs endroits dans la classe Page. Cette approche offre plusieurs avantages :

- Réduction de la duplication de code : Avant cette modification, la vérification des conditions (StringUtils.isNotBlank(url) && !url.equals("#") && !url.startsWith("javascript:")) était répétée à chaque fois qu'une URL devait être validée avant d'être ajoutée aux requêtes cibles. En extrayant cette logique dans une méthode dédiée, on évite la duplication du code, ce qui rend le code plus propre et plus facile à maintenir.
- Amélioration de la lisibilité : En nommant la méthode isValidUrl, on rend le code appelant plus lisible. Il devient immédiatement évident que cette méthode vérifie si une URL est valide selon les critères définis, sans avoir besoin de plonger dans les détails des conditions utilisées pour cette vérification.
- Facilité de mise à jour : Si les critères de validation des URL doivent être modifiés à l'avenir (par exemple, en ajoutant de nouvelles conditions ou en modifiant les conditions existantes), ces changements peuvent être effectués en un seul endroit. Cela réduit le risque d'erreurs et simplifie la mise à jour du code.
- Réutilisabilité : La méthode isValidUrl peut être réutilisée à plusieurs endroits dans la classe Page ou même dans d'autres parties du framework, si la logique de validation des URL doit être appliquée ailleurs.

Initialisation des composants et refactorisation de la méthode initComponent

J'ai choisi de refactoriser l'initialisation en méthodes distinctes parce que cela rend le code plus clair et plus facile à maintenir. En isolant chaque partie de l'initialisation dans sa propre méthode, j'améliore la modularité du code. Cela signifie que si je dois modifier la façon dont le downloader est initialisé, par exemple, je peux le faire dans un seul endroit sans avoir à parcourir toute la méthode initComponent.
Cette approche rend également le code plus réutilisable. Si une autre partie de la classe Spider a besoin d'initialiser ou de réinitialiser le downloader ou les pipelines, je peux simplement appeler ces méthodes spécifiques sans dupliquer le code.
En outre, séparer l'initialisation en méthodes distinctes facilite les tests unitaires. Je peux tester chaque partie de l'initialisation indépendamment, en m'assurant que chaque composant est initialisé comme prévu.
En bref, ces modifications contribuent à une meilleure organisation du code, le rendant plus facile à lire, à maintenir et à tester.

Refactorisation de la méthode run

- Séparation de la logique en méthodes plus petites
  J'ai décidé de décomposer la grande boucle while en plusieurs méthodes plus petites pour clarifier le processus étape par étape. Cela facilite la compréhension du flux de travail principal et la gestion des différentes phases du cycle de vie d'une requête.

- Gestion centralisée des interruptions
  Au lieu de vérifier les interruptions du thread à plusieurs endroits, je propose de centraliser cette vérification pour simplifier la gestion des cas où le thread est interrompu.

- Amélioration de la gestion des erreurs
  Actuellement, l'erreur est capturée et logée dans la même méthode run. En extrayant cette logique dans une méthode dédiée, cela rendrait le code plus propre et faciliterait potentiellement la réutilisation de la logique de gestion des erreurs.

Correction des erreurs dans les tests

J'ai corrigé des erreurs dans les tests et la structure du projet en ajustant les importations et les références après avoir modifié la structure de certains packages et simplifier certaines méthodes. Ces changements visent à corriger les tests pour vérifier le comportement attendu des composants.

Correction et amélioration de SpiderTest

J'ai rationalisé et amélioré les tests dans SpiderTest en éliminant les redondances et en consolidant la configuration initiale des tests à l'aide des annotations @before. Cette approche garantit que chaque test commence avec une instance fraîche de Spider, améliorant ainsi l'isolation entre les tests. J'ai également remplacé les impressions dans la console par des assertions pour valider activement le comportement attendu, rendant les tests plus précis et plus informatifs quant à leur réussite ou échec. De plus, j'ai simplifié la logique du test testWaitAndNotify pour éliminer les boucles inutiles et concentrer le test sur des scénarios plus spécifiques. Ces modifications visent à rendre les tests plus maintenables, plus fiables et plus faciles à comprendre.

Amélioration des tests de la classe Site

J'ai enrichi la suite de tests pour la classe Site avec de nouveaux cas de test qui examinent comment elle gère différentes configurations, telles que l'ajout d'en-têtes HTTP, la définition d'un User-Agent, le réglage des timeouts, et la spécification des codes de statut HTTP acceptables. Ces tests assurent que les méthodes de configuration de Site fonctionnent comme prévu, offrant une validation importante pour éviter les régressions lors des futures modifications du code.

Grandes modifications

Suppression des packages thread et model

Dans ce commit, j'ai déplacé deux classes, CountableThreadPool et HttpRequestBody, depuis leurs packages spécifiques (us.codecraft.webmagic.thread et us.codecraft.webmagic.model) vers le package racine us.codecraft.webmagic. Cette modification simplifie la structure du projet en réduisant le nombre de packages contenant un petit nombre de classes, ce qui peut faciliter la navigation et la compréhension du code pour les développeurs travaillant sur le projet.

Suppression d'une classe static pas utilisée

J'ai retiré la classe static ContentType de HttpRequestBody car les constantes qu'elle définissait n'étaient utilisées nulle part ailleurs dans le projet. Cette simplification rend le code plus direct et évite la surcharge inutile d'une structure de classe supplémentaire, améliorant ainsi la lisibilité et la maintenabilité du code.

Intégration du design pattern Observer avec SpiderListener

J'ai intégré le pattern Observer via SpiderListener pour découpler la logique d'événements de Spider, rendant le système plus flexible et extensible. Cela permet d'ajouter facilement des comportements personnalisés en réponse aux succès ou échecs du traitement des requêtes.

Conclusion

En menant ce projet, j'ai découvert comment des outils comme SonarQube peuvent aider à vérifier la qualité d'un code. J'ai aussi réalisé qu'il est complexe de garder un code propre et qu'il est crucial de suivre les principes du génie logiciel pour bien avancer dans un projet informatique. En cours de route, j'ai noté que les solutions envisagées au début ne correspondaient pas toujours à celles appliquées plus tard. Cela m'a montré qu'avec l'expérience et une meilleure compréhension du projet, on finit par trouver les meilleures corrections à apporter, soulignant ainsi l'importance de l'apprentissage continu en développement logiciel.