Задание #8. Улучшенный поисковый робот

Сделанный вами в прошлом задании поисковый робот не очень эффективен. На этой неделе вы расширите его возможности применив потоки Java. Робот сможет обрабатывать несколько страниц параллельно. Это должно существенно улучшить производительность программы потому, что всякий раз когда какой-либо поток робота будет ждать завершения загрузки страницы по сети, другие потоки смогут продолжить свои операции по обработке данных на своих страницах.

Подробное введение в многозадачное программирование на языке Java (*очень* сложный вопрос!), вы можете прочитать в этом учебном руководстве. Наиболее важный раздел этот .

Улучшение поискового робота

Вам нужно добавить новые возможности к коду написанному в прошлом задании:

1. Сделайте класс с именем `URLPool`, который сохраняет список всех найденных URLs, вместе с "уровнем" каждого из этих URL (мы называем это также "глубина поиска"). Первый URL с которого вы начали поиск имеет глубину поиска 0. Все URL найденные на этой странице имеют глубину 1, и так далее. Сохраняйте URL и их глубину вместе как экземпляр класса `URLDepthPair` также как вы это делали в прошлом задании.
   Для хранения элементов рекомендуется использовать `LinkedList`, так как он очень эффективно выполняет требуемые операции.
   Нужно чтобы пользователь класса `URLPool` мог извлекать пару URL-глубина из пула, и одновременно удалять ее из списка. Так же нужно обеспечить возможность добавления пары URL-глубина к пулу. Обе эти операции должны быть рассчитаны на вызов из нескольких потоков так как несколько потоков могут одновременно работать с `URLPool`.
   В отличие от примера очереди FIFO, который рассматривался в лекциях, пул URL не должен иметь ограничений размера. В него нудно добавить список URL ожидающих обработки, список просмотренных URL и еще одно поле, которое мы рассмотрим ниже.

2. Для того чтобы поиск можно было выполнять в нескольких потоках, создайте класс `CrawlerTask`, который реализует интерфейс `Runnable`. Каждый экземпляр `CrawlerTask` должен иметь ссылку на *единственный* экземпляр класса `URLPool` который описан выше. (Обратите внимание, что *все* экземпляры `CrawlerTask` используют один общий пул ссылок!) Поток поискового робота должен работать так:
   1. Извлекает пару URL-глубина из пула (ждет если в данный момент ссылок нет).
   2. Загружает веб страницу на которую ссылается URL.
   3. Ищет на странице другие URL. Для каждого найденного на странице поток робота должен добавить новую пару URL-глубина в пул ссылок. Новая пара должна иметь глубину на единицу большую, чем URL из которого эта ссылка извлечена на шаге 1.
   4. Возвращается к шагу 1!

   Эти операции следует продолжать до тех пор, пока в пуле ссылок не останется ни одной пары (URL, глубина).

3. Так как ваш робот запускает некоторое количество потоков, добавьте в программу третий параметр командной строки, задающий число потоков робота. Ваша функция main должна работать примерно так:

   1. Обрабатывает аргументы командной строки. Сообщает пользователю об ошибках, если они есть.
   2. Создает экземпляр класса пула URL, и помещает указанный пользователем URL с глубиной 0 в пул.

3. Создает указанное пользователем число задач робота (и потоков для их запуска). Каждая задача робота должна иметь ссылку на пул URL, который только что создан.
4. Ожидает пока робот не закончит работу! (Об этом ниже.)
5. Печатает в результате список найденных URL.

4. *Синхронизируйте* доступ к объекту пула URL во всех важных местах, потому что доступ к этому объекту из потоков должен быть безопасным.

5. Робот извлекая очередную ссылку не должен непрерывно опрашивать пул URL если он пуст. Он должен жать появления URL более эффективно. Метод пула URL "получить URL" должен сам `ждать используя функцию wait()`, если в пуле в момент вызова нет ссылок. Соответственно в метод пула ссылок "добавить URL" надо добавить вызов `notify()`, когда новая ссылка добавлена в пул.
   Заметим, что потоки робота сами по себе не выполняют операции synchronize / wait / notify. Это следует сделать по той же причине, по которой пул URL прячет внутри себя детали реализации добавления и извлечения ссылок: **инкапсуляция**! Так же как вы хотите избавить пользователей от деталей реализации пула URL, вы избавляете их от деталей синхронизации потоков.

Советы по проектированию

Вот некоторые полезные советы по выполнению задания 8!

- Возможно вы перенесете с небольшими изменениями значительные фрагменты кода из прошлого задания в новое. Класс `URLDepthPair` вероятно вовсе не нужно изменять. Большая часть кода обрабатывающего страницу также может быть перенесена без изменений. Основное отличие от предыдущего задания в том, что код загрузки и обработки URL теперь должны быть помещен в класс реализующий `Runnable`, и теперь код должен извлекать и добавлять ссылки в экземпляр класса `URLPool`.
- Надо синхронизировать доступ к внутренним полям `URLPool`, так как к ним будут обращаться из нескольких потоков. Как обсуждалось на лекции, наиболее простой подход, это использование синхронизированных методов (`synchronized`) вместо вставки блоков `synchronized` в методах класса. Не нужно синхронизировать конструктор `URLPool`! Подумайте о том, какие методы нужно синхронизировать.
- Напишите методы `URLPool`, которые используют `wait()` и `notify()` так, чтобы робот мог эффективно ждать пока ссылки появятся в пуле.
- Пусть `URLPool` контролирует добавление ссылок в список на обработку, проверяя глубину каждой добавляемой в пул ссылки. Если глубина URL меньше максимальной, пара URL-глубина добавляется в список ссылок для обработки. Если нет, просто добавьте URL в список "просмотренных" ссылок.
- Наименее понятная часть работы, это определение условия завершения исполнения программы, в случае, если не осталось ссылок для просмотра (или достигнута максимальная глубина просмотра). Когда это происходит, все потоки `CrawlerTask` переходят в состояние ожидания (`wait()`) появления новой ссылки в `URLPool`. Рекомендуется в `URLPool` следить за количеством потоков ожидающих новую ссылку. Для этого следует добавить поле типа `int`, которое инкрементируется перед вызовом `wait()`, и декрементируется сразу после того как `wait()` возвращает значение.
  Когда у вас есть счетчик числа ждущих потоков, можно добавить (синхронизированный!) метод возвращающий число таких потоков. Когда этот счетчик станет равен общему числу потоков робота, время завершать работу программы. Этот счетчик можно проверять в функции `main()`, а вызывать `System.exit()` для завершения работы. Это конечно еще один пример ужасного поллинга, но его стоимость можно свести на нет, переводя поток main в спящее состояние на короткое время между этими проверками. Значение между 0.1 секунды и 1 секундой вероятно наиболее подходящее значения периода опроса.

Дополнительные баллы

- Сделайте так, чтобы в паре URL-глубина использовался класс `java.net.URL`, и добавьте роботу возможность обрабатывать *относительные* URL так же как и абсолютные. Для этого нужно немного изменить код, но это не слишком сложно.
- Завершение программы с помощью `System.exit()` не лучший способ сделать это. Найдите лучший способ выхода из программы после того, как поиск закончен.
- Сделайте общий список обработанных ссылок, и сделайте так, чтобы не допускалась повторная обработка ссылок. Используйте один из видов коллекций Java, который упрощает эту задачу. Лучше всего использовать некоторые виды множеств, которые имеют постоянное время поиска и вставки.

- Добавьте параметр командной строки, который задает время ожидания ответа от сервера во время загрузки веб страницы. Используйте этот параметр как порог для *начального* ожидания загрузки страницы, а затем добавьте другой параметр `maxPatience` который задает время обработки страницы, после которого обработка прерывается и происходит переход к следующей странице.
- Итак, сделайте ваш конкурент системе поиска Google ;-)