Лабораторная работа №8: Модифицированный веб-сканер

Сканер в прошлой лабораторной работе был не особенно эффективным. В данной лабораторной работе вы расширите сканер для использования поточной обработки Java так, чтобы несколько веб-страниц можно было сканировать параллельно. Это приведет к значительному повышению производительности, так как время, которое каждый поток сканера тратит на ожидание завершения сетевых операций, может прерываться другими операциями обработки в других потоках.

Подробное введения в многопоточное программирование на Java вы можете прочитать в данном учебном руководстве. Самое главное прочитать этот подраздел.

Расширение веб-сканера

В данной лабораторной работе вы расширите и измените разработанную ранее программу:

1. Реализуйте класс с именем URLPool, который будет хранить список всех URL-адресов для поиска, а также относительный "уровень" каждого из этих URL-адресов (также известный как "глубина поиска"). Первый URL-адрес, который нужно будет найти, будет иметь глубину поиска равную 0, URL-адреса, найденные на этой странице, будут иметь глубину поиска равную 1 и т.д. Необходимо сохранить URL-адреса и их глубину поиска вместе, как экземпляры класса с именем URLDepthPair, как это было сделано в прошлой лабораторной работе. LinkedList рекомендуется использовать для хранения элементов, так как это поможет эффективно выполнить необходимые операции.

У пользователя класса URLPool должен быть способ получения пары URL-глубина из пула и удаления этой пары из списка одновременно. Должен также быть способ добавления пары URL-глубина к пулу. Обе эти операции должны быть поточно-ориентированы, так как несколько потоков будут взаимодействовать с URLPool одновременно.

У пула URL не должно быть максимального размера. Для этого нужен список необработанных URL-адресов, список уже отсканированных URL-адресов и еще одно поле, о котором будет написано ниже.

2.Чтобы выполнить веб-сканирование в нескольких потоках, необходимо создать класс CrawlerTask, который реализует интерфейс Runnable. Каждый экземпляр CrawlerTask должна иметь ссылку на один экземпляр класса URLPool, который был описан выше. (Обратите внимание на то, что все экземпляры класса CrawlerTask используют единственный пул!) Принцип работы веб-сканера заключается в следующем:

1). Получение пары URL-Depth из пула, ожидая в случае, если пара не будет сразу доступна.

2). Получение веб-страницы по URL-адресу.

3). Поиск на странице других URL-адресов. Для каждого найденного URL-адреса, необходимо добавить новую пару URL-Depth к пулу URL-адресов. Новая пара должна иметь глубину на единицу больше, чем глубина текущего URL-адреса, по которому происходит сканирование.

4). Переход к шагу 1.

Данный цикл должен продолжаться до тех пор, пока в пуле не останется пар URL-Depth.

3. Так как веб-сканер будет порождать некоторое количество потоков, измените программу так, чтобы она принимала третий параметр через командную строку, который будет определять количество порождаемых потоков веб-сканера. Функция main должна выполнять следующие задачи:

1). Обработать аргументы командной строки. Сообщить пользователю о любых ошибках ввода.

2). Создать экземпляр пула URL-адресов и поместить указанный пользователем URL-адрес в пул с глубиной 0.

3). Создать указанное пользователем количество задач (и потоков для их выполнения) для веб-сканера. Каждой задаче поискового робота нужно дать ссылку на созданный пул URL-адресов.

4). Ожидать завершения веб-сканирования.

5) Вывести получившийся список URL-адресов, которые были найдены.

4.Синхронизируйте объект пула URL-адресов во всех критических точках, так как теперь код должен быть ориентирован на многопоточность.

5. Веб-сканер не должен постоянно опрашивать пул URL-адресов в случае, если он пуст. Вместо этого пусть они ожидают в случае, когда нет доступных URL-адресов. Реализуйте это, используя метод wait() внутри «get URL» в случае, если ни один URL-адрес в настоящее время недоступен. Соответственно, метод "add URL" пула URL-адресов должен использовать функцию notify() в случае, когда новый URL-адрес добавлен к пулу.

Обратите внимание на то, что потоки веб-сканера не сами будут выполнять какие-либо из этих операций синхронизации/ожидания/уведомления. По той же причине, что и пул URL-адресов скрывает детали того, как URL-адреса хранятся и извлекаются: инкапсуляция! Точно так же, как и в вашей реализации пользователи пула URL-адресов не должны знать о деталях реализации, также они не должны знать о деталях организации потоков.

Советы по проектированию

Вот некоторые советы для успешного выполнения лабораторной работы №8:

- Вы можете использовать часть кода из лабораторной работы №7 с небольшим изменением. Класс URLDepthPair изменять не нужно. Основные отличия в том, что код загрузки URL-адреса и сканирование страницы находится теперь в классе, который реализует Runnable и код будет получать и добавлять URL-адреса в экземпляре URLPool.

- Вы должны синхронизировать доступ к внутренним полям URLPool, поскольку к ним будут обращаться сразу несколько потоков. Самый простой подход заключается в использовании методов синхронизации. Не нужно синхронизировать конструктор URLPool! Подумайте о том, какие методы должны быть синхронизированы.
- Напишите методы URLPool для использования методов wait() и notify() так, чтобы потоки сканера могли ожидать появления новых URL-адресов.
- Пусть URLPool определяет, какие URL-адреса попадают в список необработанных URL-адресов, исходя из глубины каждого URL-адреса, добавляемого в пул. Если глубина URL-адреса меньше максимальной, добавьте пару в очередь ожидания. Иначе добавьте URL-адрес в список обработанных, не сканируя страницу.
- Самая сложная часть данной лабораторной работы заключается в поиске момента для выхода из программы, когда больше нет URL-адресов для сканирования. В таком случае все потоки будут в режиме ожидания нового URL-адреса в URLPool. Для этого рекомендуется, чтобы URLPool отслеживал, сколько потоков ожидает новый URL-адрес. Поэтому необходимо добавить поле типа int, которое будет увеличиваться непосредственно перед вызовом wait() и уменьшаться сразу после выхода из режима ожидания. Создав счетчик, нужно реализовать метод, который возвращает количество ожидающих потоков. Отслеживать количество потоков вы можете в функции main() и в случае, если общее количество потоков равно количеству потоков, которое вернул соответствующий метод, необходимо вызвать System.exit() для завершения работы. Проверку можно выполнять по таймеру (с интервалом 1 сек), что приведет к более эффективной работе программы.

Дополнительное задание

- Обновите пару URL-Depth для использования класса java.net.URL и произведите соответствующие изменения в веб-сканере для того, чтобы он соответствовал и относительным URL-адресам, и абсолютным.
- Выход из программы с использованием вызова System.exit() - грубая операция. Найдите способ более корректного выхода из программы.
- Реализуйте список URL-адресов, которые были просмотрены, и избегайте возврата к ним. Используйте один из классов коллекций java. Какой-то набор, который поддерживает постоянное время поиска и вставку, будет наиболее подходящим.
- Добавьте другой дополнительный параметр командной строки для того, чтобы определить, сколько времени поток веб-сканера должен ждать сервера для возврата требуемой веб-страницы.