Идентификация изменений HTML-структур, приведенных к формату JSON
Довбенко А. В.

Довбенко Алексей Викторович / Dovbenko Alexey Victorovich – аспирант,
кафедра теоретических основ информатики,
факультет прикладной математики, информатики и механики,
Воронежский государственный университет, г. Воронеж

***Аннотация**: в работе произведен анализ идентификации изменений в структуре типа JSON, с возможностью хранить версии изменений для возможного просмотра старых данных, а также возможностью анализировать и реагировать на частоту того или иного изменения.*
***Ключевые слова**: базы данных, NoSQL, MongoDB, JSON, версионность, аналитика.*

С момента появления первых поисковых систем появилась задача обновлять данные считанных с сайтов, по умолчанию использовалась стандартная схема, поисковый робот заходил на сайт через определенный промежуток времени и обновлял данные, но со временем интернет-аудитория расширялась, сайтов становилось больше, некоторое из них стали набирать популярность, некоторое же, наоборот, имели крайне малую аудиторию и почти не обновлялись, стало понятно, что для каждого сайта интервал обновления должен быть разным. В этой статье предложена схема идентификации частоты и важности изменений, с помощью которой можно строить автоматическую систему сканирования сайтов.

Как известно, одна из основных задач парсинга сайта поисковыми системами - идентификация изменений на странице. После того как мы получили контент сайта и очистили его от ненужной шелухи (тэгов, классов, атрибутов, скриптов, стилей), контент хэшируется, пусть в данном случае алгоритмом хэширования MD5 (алгоритм не столь важен, так как в данной статье мы не рассматриваем вероятность коллизий на n-число хэшированных объектов), и присваиваем данному хэшу адрес страницы, время сканирования и полный контент страницы, т. е. структура в базе данных (для данного эксперимента использовалась MongoDB, так как данные будут не структурированы) выглядит примерно так (Рисунок 1):

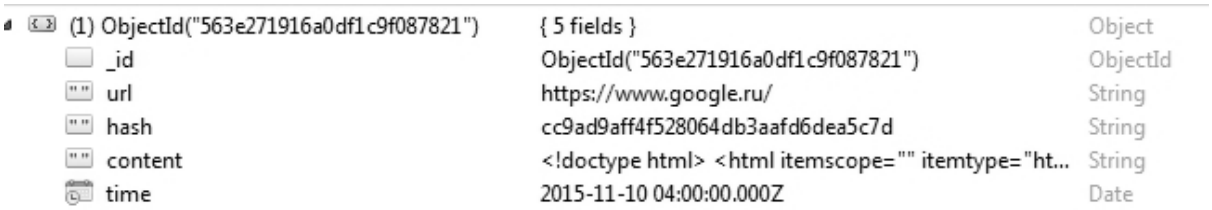

Рис. 1. Пример хранения html страницы

При последующем скане страницы google.ru будет сравниваться хэш сумма нынешнего скана с предыдущем, тем самым будет решаться - парсить страницу заново или нет. Но на каждом сайте есть статические блоки (header, footer, меню) и блоки с разной частотой обновления, как, например, на новостном сайте новость может дополняться в течении дня или же не дополняться вообще, в то время как комментарии к ней могут появляться каждую минуту на протяжении недели, также стоит учитывать, что многие сайты показывают динамическую информацию, которая зачастую не нужна поисковой машине (текущее время и дата, время загрузки страницы и т. п.). Соответственно данный метод хранения изменений не идеален, так как мы не можем быстро обнаружить изменения конкретного блока, и есть большая вероятность выполнения ненужной работы и частого сканирования сайта, где информация не меняется вообще.

Соответственно, надо хранить хэши блоков, так мы сможем довольно быстро обнаружить изменения и уже потом решать, что делать, тем самым уменьшая объем хранимых данных. Описание метода парсера, который нормализует и структурирует сайт по определенным блокам, заслуживает отдельной объемной статьи по интеллектуальной системе парсинга, поэтому тут мы пропустим этот шаг. В сущности, нам надо привести сайт к документу json по структуре типа (Рисунок 2):

Рис. 2. Шаблон для разбивки html страницы на блоки

В данной структуре блок head будет иметь свой хэш, блок body - свой, причем каждый дочерний элемент блока body также будет иметь свой хэш, соответственно сравнивание хэшей будет идти по древовидной схеме, пока не найдутся измененные блоки. Итак, испробуем это на практике. Как вы заметили, блок «head» имеет другую структуру дочерних блоков, в отличие от блока «body», сделано это потому, что реакция от изменений в блоке head и блока body кардинально отличается, поэтому их лучше рассматривать отдельно.

К нормализованной блочной структуре приходят многие поисковые системы, например, Google. Основной принцип — это использование микроразметки, которой обозначается важная часть информации для показа в превью в списке результатов поиска [1].

Возьмем стандартную архитектуру сайта в упрощенном виде (Рисунок 3):

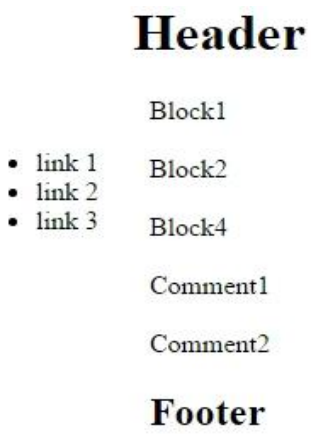

Рис. 3. Типичная структура сайта

Коллекции для такого документа будут иметь следующий вид (Рисунок 4):

Рис. 4. Основная коллекция с актуальными данными для быстрого доступа

И коллекция элементов этой страницы (Рисунок 5):

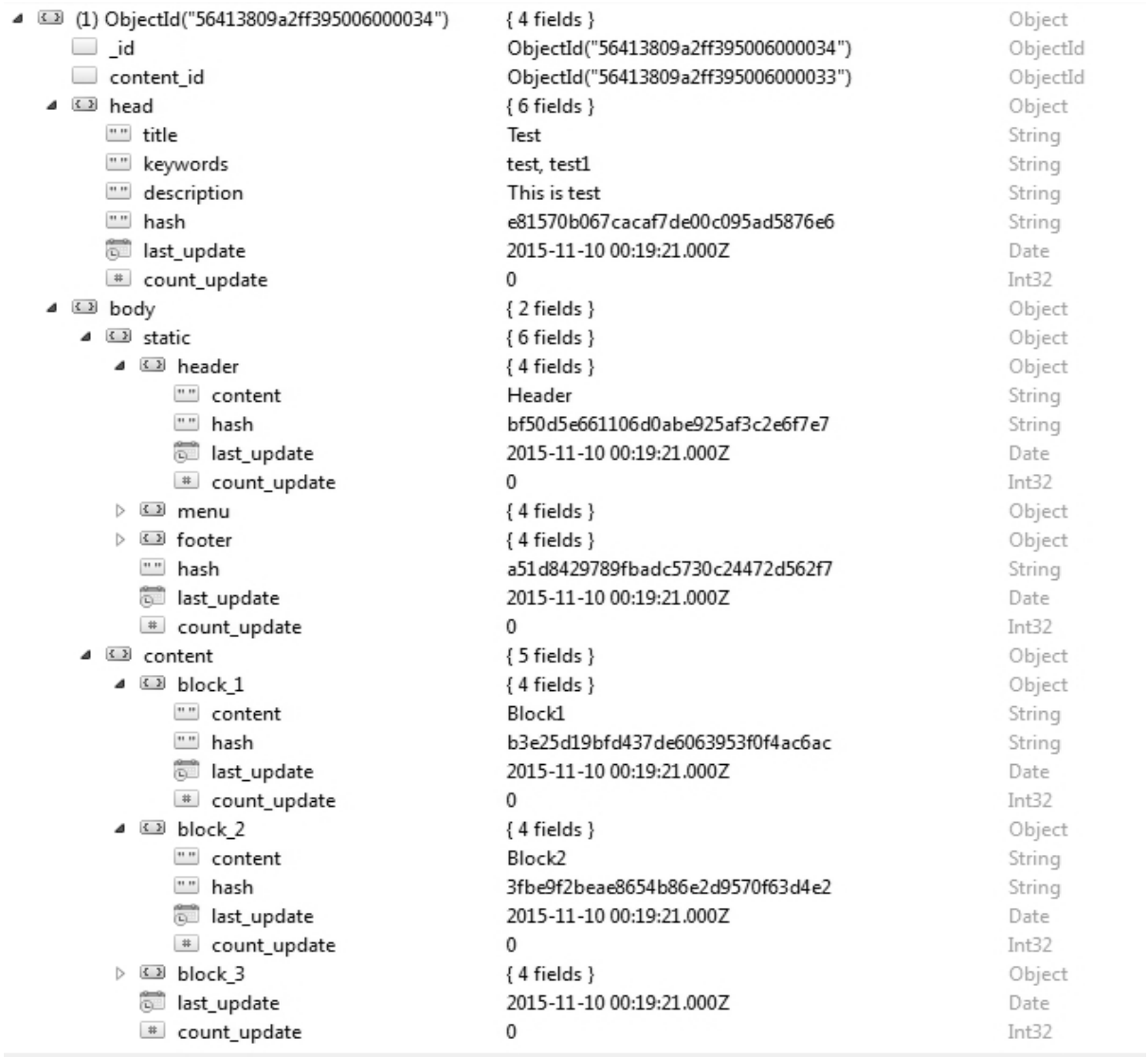

Рис. 5. Коллекция блоков html страницы, с частотой обновления блоков

Теперь рассмотрим ситуацию обновления нашего документа: меняется контент блока 3. Мы при скане страницы рекурсивно сравниваем старые блоки с новыми по хэшу, если хэш совпадает, то пропускаем блок, если нет, то обновляем данные (хэш, дата обновления и количество обновлений), в обновленном блоке также заменяем контент.

Что этот подход в итоге дает? По сути дела, это прослойка между сканом страницы и парсингом информации, которая дает определенные плюсы, а именно:

1) Упрощает работу получения информации, работать с конкретным измененным блоком намного проще и быстрей чем со всей страницей.

2) Мы имеем некоторую статистику по изменяемости сайта, благодаря которой можем обнаруживать подозрительную активность там, где её раньше не было (например, меню стало обновляться в разы чаще).

3) В зависимости от статистических данных можем по-разному реагировать на изменение определенных блоков.

4) Новая отрасль в развитии поиска сайтов дубликатов и, возможно, влияние на сам алгоритм ранжирования, если рассматривать каждый блок как отдельный источник информации со своими ключевыми словами.

Но также не будем и забывать про минусы, а именно:

1) Такая система будет эффективна только с хорошим парсером, который будет автоматически разбивать сайт на блоки, причем, чем больше уровень вложенности у блоков, тем лучше будет работать система.

2) Дополнительные затраты по памяти, но сейчас стоимость таких ресурсов относительно дешевая, плюс, мы можем отказаться от хранения самой информации, а хранить только хэш, тем самым сэкономив приличное количество места.

Так или иначе данная структура вполне имеет право на существование, причем не только в поисковых машинах, но и при работе с различными api, где преобразовывать данные изначально не надо, так как они приходят уже структурированные, соответственно можно существенно упростить работу по хранению статистики изменений сложных структур.

Хотелось бы отметить, что похожую структуру успешно используют пакетные менеджеры, такие как composer, nmp (для PHP и NodeJS соответственно), для подтягивания зависимостей того или иного пакета [2, 3].

Литература

1. Structured Data https: [Электронный ресурс]. Режим доступа: //developers.google.com/structured-data/rich-snippets/products.
2. Dependency management [Электронный ресурс]. Режим доступа: https://getcomposer.org/doc/00-intro.md#dependency-management.
3. What is npm? [Электронный ресурс]. Режим доступа: https: //docs.npmjs.com/getting-started/what-is-npm.