Проект RADOMS: программные компоненты серверной части — импорт метаданных из сервиса elibrary, группы пользователей и операции над ними

Грибков К.В., студент кафедры информатки СПбГУ, стажёр-исследователь СПИИРАН, gribkov.kirill.v@gmail.com
Суворова А.В., лаборатория теоретических и междисциплинарных проблем информатики, СПИИРАН, suvalv@mail.ru
Тулупьев А.Л., кафедра информатики СПбГУ, лаборатория теоретических и междисциплинарных проблем информатики, СПИИРАН, alexander.tulupyev@gmail.com [1]

Аннотация

Разработан импорт списка публикаций на серверной части из внешнего сервиса elibrary.ru. Разработана модель группы пользователей и набор обработчиков событий для работы с ними.

Введение

Российские научные и научно-педагогические работники, исследователи и иные творческие деятели — всем им нужно предъявлять сведения о результатах работы в самых разных видах, структурах, срезах для рейтингов, конкурсов и т.д. Структурирование списка при большом количестве интеллектуальных результатов становится сложной задачей. Также, стоит заметить, что для разных ситуаций требуется представить список о результатах интеллектуальной деятельности в разных форматах, таких как *.xml*, *.doc* и другие.

Целью настоящей работы в рамках проекта по созданию распределенной информационной системы для управления сведениями о результатах интеллектуальной деятельности является реализация импорта метаданных публикаций из сервиса elibrary[2], а также создание модели группы пользователей и реализация API для работы с данной моделью.

[1] Работы выполнялись в рамках проекта по государственному заданию СПИИРАН № 0073-2014-0002.

Используемые технологии

Среди множества технологий и языков программирования для написания серверной части, был выбран язык программирования JavaScript и фреймворк node.js[6]. Этот фреймворк имеет удобный сервис по установке пакетов — npm. Указанный сервис удобен тем, что он интегрирован во многие среды разработки, например, в JetBrains WebStrom[8], и не требует дополнительной настройки.

В качестве СУБД используется MongoDB[4]. Это документо-ориентированная СУБД, не требующая описания схемы таблиц, классифицируется как NoSQL база данных. Вместо традиционной реляционной структуры базы данных MongoDB использует JSON-подобные документы с динамическими схемами, из-за этого интеграция в определенных видах приложениях происходит проще и быстрее. Для работы с этой СУБД в node.js используется пакет mongoose[5].

Для парсинга html страниц используется пакет cheerio[1]. Данный пакет имеет очень удобный синтаксис, он основан на зарекомендовавшем себя синтаксисе фронтенд фреемворка jQuery[3]. Также cheerio работает с простой, согласованной DOM моделью, в результате чего парсинг, манипуляция и рендеринг HTML кода выполняется быстро.

Импорт метаданных публикаций из сервиса elibrary

В сервисе реализовано несколько способов ввода списка публикаций: ввод поштучно с заполнением формы вручную и импорт с помощью таблиц csv, xls. Добавление большого количества публикаций с помощью данных методов оказывается достаточно затруднительным. Поэтому необходимо было разработать новый метод импорта публикаций, который бы работал, буквально, в несколько кликов. Т.к. большое количество целевой аудитории нашего проекта пользуются сервисом elibrary, то решением данной задачи стала разработка импорта публикаций из сервиса elibrary.

Elibrary не имеет внешнего API, поэтому выгрузку метаданных публикаций из этого сервиса необходимо провести “вручную”. Т.е. необходимо скачать нужные страницы профиля, распарсить их, выделить нужные данные и загрузить в систему. Общая стратегия импорта метаданных публикаций выглядит следующим образом:

1. На первом шаге происходит проверка введенной пользователем

ссылки. Проверяется корректность ссылки и если ссылка корректна, возвращается id пользователя в системе elibrary.

2. На следующем шаге происходит закачка первой страницы списка публикаций. После закачки первая страница разбирается и устанавливается точное количество страниц списка публикаций, также подготавливается объект, в котором будут содержаться все полученные данные.

3. После этого происходит закачка и разбор остальных страниц списка публикаций в параллельном режиме.

4. После первоначального парсинга, у каждой публикации есть три поля — это название публикации, список авторов и все остальные данные по публикации.

5. Все три поля представляют из себя обычную строку, поэтому следующим шагом необходимо из строки списка авторов получить список отдельных авторов. Также, по возможности, разбирается третье поле с остальными данными по публикации. В большинстве случаев, удачно можно получить год публикации, страницы и номер издания, где находится публикация. Также, довольно часто, разбирается название журнала или сборника публикаций.

6. Далее все полученные данные возвращаются на клиентскую часть, для дальнейшей работы с ними.

Рассмотрим несколько наиболее значимых пунктов стратегии подробно.

Получение общих данных публикаций с первой страницы

Для получения DOM-объекта первой страницы списка, используется функция *downloadElibraryPageByAuthoridAndPagenum* — данная функция представляет из себя простой запрос, который скачивает страницу, разбирает её на DOM-дерево, с помощью пакета cheerio, и передает готовый объект в функцию обратного вызова. Далее из данного объекта необходимо достать данные публикаций, делается это с помощью методов пакета cheerio:

Листинг 1: Выборка нужных строк из страницы

```
var tr_it = [];
var tr_list = $('tr[id^=arw]');
tr_list.each(function () {
  tr_it.push(objFromTr($(this)));
});
```

Каждая публикация из списка находится в своем строковом теге с идентификатором начинающимся с подстроки arw. Чтобы выбрать такое подмножество из DOM-объекта, необходимо воспользоваться соответствующим css-селектором: $tr[id\hat{} = arw]$.

Далее необходимо разобрать каждую строку с публикацией на отдельные фрагменты данных: название, авторов и описание.

Листинг 2: Парсинг общих данных

```
function objFromTr(tr) {
  var td = tr.find('td[align=left]');
  var name = td.find('a b').text();
  var authors = td.find('font i').text();
  var desc = td.find('font').last().text();
  var authorsAP = authorsParser(authors);
  var prDesc = descParser(desc);
  return {name: name, authors: authors, desc: desc,
      authorsList: authorsAP, possibleDesc: prDesc}
}
```

В самом начале из строкового тега tr выбирается столбцовый тег td с выравниванием по левому краю, именно там находятся все данные по публикации. Название публикации находится с помощью css-селектора "$a\ b$", что обозначает тег b внутри тега a. Авторы — с помощью css-селектора "$font\ i$" — тег i внутри тега font. Описание находится в самом последнем теге $font$, его выборка осуществляется с помощью css-селектора "$font$" и метода $last()$, который выдает только последний найденный элемент.

Подробный разбор объекта публикации

На данном этапе все описание конкретной публикации представляет из себя единую строку, где информация разделена точкой. Поэтому на этом шаге происходит парсинг этой информации на более подробные части. Чтобы выделить из этой строки конкретные метаданные публикации используется свойства этих метаданных. Т.е. например год публикации всегда представлен в виде четырехразрядного числа:

Листинг 3: Парсинг года

```
if (tempDesc.length == 4 && tempDesc.search(/[1-3]\d\d\d/)
    != -1) {
  prDesc.year = parseInt(tempDesc);
}
```

А, например, страницы на которых находится публикация всегда начинаются с буквы “С”:

Листинг 4: Парсинг страниц

```
else if (tempDesc[0] == C'') {
  var fromToNum = parseFromToNum(dotSplit[i+1]);
  prDesc.firstPage = fromToNum.from;
  prDesc.lastPage = fromToNum.to;
}
```

Т.к. страницы могут быть представлены в виде диапазона, используется отдельный метод для парсинга диапазона страниц $parseFromToNum$.

Группы пользователей

Одной из поставленных задач проекта является реализация формирования групповых списков результатов интеллектуальной деятельности. Один из примеров такого списка — годовой отчет некоторой лаборатории. Решить данную задачу можно без создания дополнительных структур. Например, в случае лаборатории, можно было создать новый обычный профиль пользователя в сервисе и добавить туда публикации всех сотрудников. Но такой подход является очень неудобным, такому профилю необходимо постоянная модерация, если у какого-либо сотрудника появилась новая публикация, её необходимо добавить не только в личный профиль, но и также в профиль лаборатории. Поэтому для решения данной задачи было принято создать новую модель в сервисе — модель группы. Данная модель позволяет объединить несколько пользователей в одну структуру и взаимодействовать со всеми публикациями всех пользователей в этой структуре. Такой подход позволяет удобно управлять групповыми списками публикаций. Если пользователь добавляет некоторую публикацию в свой личный профиль, данная публикация автоматически будет отображаться и в списке публикаций группы.

Модель группы пользователей

Как говорилось выше в качестве базы данных в проекте используется документно-ориентированная MongoDB. В node.js для работы с этой базой используется пакет mongoose. Данные в MongoDB хранятся в виде json-подобных документах, по этому для описания модели данных используется т.н. схема.

Модель группы содержит все данные о группе: название, описание, уникальная ссылка и другие, также в модели находится список пользователей группы с ролями в ней:

Листинг 5: Схема модели группы

```
var Group = new Schema({
  title: {type: String, unique: true, required: true},
  description: {type: String, required: true},
  link: {type: String, unique: true, required: true},
  users: [{
    id: {type: ObjectId},
    role: {type: Number, default: -1, min: -1, max: 4}
  }]
});
```

Отличительной особенностью данной схемы является массив данных в качестве значения поля users. Т.е. данное поле в объекте группы является списком, в него можно добавлять, удалять объекты и взаимодействовать с ними.

Структура API групп

При постановки задачи, было описано, что должно делать API для групп:

1. Создание, удаление, изменение группы
2. Добавление и удаление пользователей из группы, а также изменение привилегий пользователей внутри группы
3. Возможность отправить запрос на вступление в группу посредством приглашения
4. Выдача списка публикаций всех участников группы

В процессе разработки API, была придумана следующая структура API для групп:

- API *groups*:
  - *POST* — создание группы. При создании группы, пользователь который создал группу автоматически становится её администратором.
  - *PUT* — обновление информации о группе. Т.е. обновление названия, описания группы и ссылки на группу. Обновлять группу может только администратор.
  - *DELETE* — удаление группы. Удалить группу может только создатель группы.
- API *groupUsers*:
  - *GET* — получение списка пользователей группы.
  - *POST* — добавление пользователя в группу. По сути этот обработчик высылает приглашение в группу.
  - *PUT* — изменение роли пользователя в группе.
  - *DELETE* — удаление пользователя из группы.
- API *groupsPublications*:
  - *GET* — выдача публикаций всех пользователей в группе.

Рассмотрим одни из наиболее важных методов подробно.

API groups. Обработка метода POST

Создание группы происходит в несколько этапов:

1. Клиент передает на сервер название, описание и уникальную ссылку группы (см. пример в листинге 6).
2. Сервер проверяет полученные данные и создает новый объект *Group* с полученными данными. При ошибке, сохранение прерывается и на клиентскую часть отправляется соответствующее сообщение.
3. Далее объект *Group* сохраняется в базе данных.

Листинг 6: Пример запроса на создание группы

```
{
  name: "Title",
  description: "Some description",
  link: "someLink"
}
```

Листинг 7: Создание группы

```
var group = new Group({
  title: req.body.title,
  description: req.body.description,
  link: req.body.link,
  users: [{id: req.user._id, role: 4, commit: req.user.
      commit}]
});
group.save(function (err) {
  // ...
})
```

Рассмотрим листинг 7 подробно. Весь процесс происходит в два шага: создание экземпляра группы и его сохранение. Поле *users* заполнено списком с одним единственным объектом пользователя, этому объекту присваивается *id* пользователя который создает группу и высшая роль в группе. Проверка корректности имени и описания группы производится встроенными средствами mongoose: на указанные поля налагаются правила целостности, указанные в листинге 5, в частности, проверяется, что данные поля являются строками и не пусты. Проверка корректности поля *link* осуществляется посредствам метода, который был написан при создании модели группы. Этот метод вызывается всякий раз, когда вызывается метод *save* у объекта группы.

API invite

Данное API нужно для отправки запроса на вступление в группу. Данный запрос отправляется в тот момент, когда клиент переходит по ссылке “radoms.ru/invite?link=someLink”. Именно для такой уникальной ссылки и нужно поле *link* в объекте группы.

Листинг 8: Приглашение в группу

```
Group.update({
  link: req.query.link,
  "users.id": {$ne: req.user._id}
}, {
  $push: {
```

```
      users: {
        id: req.user._id,
        commit: req.user.commit
      }
    }
}, function (err, result) {/*...*/})
```

Как видно из листинга выше, для отправки запроса в группу используется метод update. Запросом на вступление в модели группы считается пользователь с ролью равной -1. Соответственно цель данного обработчика — добавить в группу с определенным *link* пользователя с ролью -1. Добавление элемента в список осуществляется с помощью оператора $*push*. Также важно, чтобы пользователь с таким не состоял в данной группе на момент добавления, поэтому в операторе поиска указывается, что в объекте группы не должно быть пользователя с *id* равным *id* пользователя который отправил запрос. Отрицание выполняется с помощью ключа $*ne*. Если же пользователь попытается зайти в группу в которой он состоит или уже послал запрос на вступление, то ему выводится соответствующая ошибка.

Заключение

В рамках настоящей работы были успешно решены поставленные задачи, связанные с реализацией импорта метаданных публикаций из сервиса elibrary, созданием модели группы пользователей и реализацией API для данной модели в приложении RADOMS.

Все частные цели, а, в следствии, и общая цель достигнуты. Сервис доступен по адресу http://radoms.ru/[7] и находится в стадии тестирования.

Литература

[1] Cheerio // 2017 // https://cheerio.js.org/

[2] ELIBRARY.RU - НАУЧНАЯ ЭЛЕКТРОННАЯ БИБЛИОТЕКА // 2017 // https://elibrary.ru/defaultx.asp

[3] jQuery // 2017 // https://jquery.com/

[4] MongoDB for GIANT Ideas | MongoDB // 2017 // https://www.mongodb.com/

[5] Mongoose ODM v4.11.0 // 2017 // http://mongoosejs.com/

[6] Node.js // 2017 // https://nodejs.org/en/

[7] RADOMS: R&D Outcomes Management System // 2017 // http://radoms.ru/

[8] WebStorm: The Smartest JavaScript IDE // 2017 // https://www.jetbrains.com/webstorm/