ATLANTIS PRESS
Advances in Social Science, Education and Humanities Research (ASSEHR), volume 130
2nd International Forum on Management, Education and Information Technology Application (IFMEITA 2017)

Extracting Data from Tutorials for Android API Recommendation

Weizhao Yuan [1, a]

[1]School of Software Engineering, Shanghai Jiao Tong University, China

[a]weizhaoy@sjtu.edu.cn

Keywords: API recommendation, code recommendation, code search

Abstract. Code recommendation is becoming more and more important for software development because increasing numbers of existing libraries are being utilized by developers and code recommendation tools offer great aids to developers in quickly finding suitable APIs to use. There have been many studies on recommending various libraries and APIs for developers according to their requirements for certain functionalities, but little effort is spent in Android API recommendation. This study aims at building a database which stores the mapping co-relationships from Android APIs and their descriptive texts which describe the functionalities that can be achieved using the APIs. The data of such mapping co-relationships are extracted from online tutorials, and the database can be further used for recommending APIs for Android development by comparing the similarity between users' requirements and the descriptive texts in the database.

Introduction

With more and more libraries developed and utilized by developers, there is a great need of code recommendation tools to help developers (especially those who are new to the libraries) to find the right APIs to use with time as short as possible.

There have been several studies which recommend libraries and APIs for developers based on their requirements (e.g., [1, 2, 3]), and some also find or generate samples to illustrate how to use the recommended APIs (e.g., [4, 5, 6, 7]). However, few studies focus on recommending APIs for Android development.

One key component to a code recommendation/search system is the database (i.e., search space) for recommendation. The quality of the database has great influence on the performance of the recommendation system. Therefore, building a proper database is a vital step for constructing an effective recommendation system. One way to build the database is to find the descriptive texts for APIs which describe the usage of the API or the functionality that can be achieved using the API. Database built in this way can be further used for code recommendation by comparing the similarity between users' requirements (i.e., queries) with the descriptive texts in the database, and recommending APIs whose descriptions are most similar to the queries.

Various kinds of data sources have been utilized by researchers for code recommendation, such as large code repositories like *Github.com*, questions and answers from *StackOverflow.com*, tutorials etc. This study uses data from Android official tutorial to construct the recommendation database, which contains considerable mapping pairs of APIs and corresponding descriptions.

The rest of the paper is organized as follows: Section 2 presents some background knowledge; Section 3 presents our approach in detail; Section 4 shows some results; Section 5 concludes our paper with future work.

Background

This section describes some background knowledge of our study.

Android tutorial. In our work, the original data are the tutorial from official website of Android. The *Android tutorial* in this paper refers to the "*training*" subsection in the Android official website for developers (https://developer.android.com/training/index.html). This tutorial covers different use cases and functionality requirements about most aspects of Android development; it is divided into

several large parts, and each large part further divided into some detailed parts, with at last each page only covering implementations of a very specific functionality (e.g. Sending Simple Data to Other Apps). Most pages will, according to the topic of the page, recommend readers what API to use to achieve a certain goal, and in most of pages there are snippets along with textual descriptions. This feature shows great practicability for extracting the co-relationships between APIs and descriptions.

Android APIs. As Android framework is an established framework, it offers a great deal of Android APIs for Android developers to utilize to implement different kinds of functionalities and develop applications of different purposes. In our work, Android APIs are limited to public classes which can be instantiated, public methods which can be invoked and public fields (often constants) of classes which can be used by developers, in the sense that these three kinds of APIs are the mostly used ones in Android development.

Approach

Filtering and Pre-processing HTML pages. We download all pages from the tutorial website, and after that, we first filter irrelevant pages which don't contain any useful information for recommendation. As we focus on pages that introduce specific functionalities and relevant Android APIs to use, we discard index pages which only serve as navigation pages and some other pages that focus more on application design than detailed implementation.

With filtered tutorial pages as raw data, we next do some pre-processing work to extract information we are interested in and save it with our predefined format. We observe each page and analyze its structure, and find that all pages have similar structure, which enables us to process them with a batch job. We first parse each HTML file to get its DOM (Document Object Model) tree. We then remove all irrelevant elements such as script elements (<*script*> tag), saving the remaining contents in an XML file with predefined format. After pre-processing, the generated XML files will have a simplified structure, with all heading elements (e.g., <*h1*>, <*h2*>), all elements with texts (e.g., <*p*>, <*li*>, <*a*>) and all sample code snippet elements (<*pre*>).

Extracting Mapping Co-relationships between APIs and Descriptions. In our approach, the mapping co-relationships between APIs and descriptions are extracted from two parts of the tutorial data, one is the textual elements (like <*p*> element) in the HTML files, and the other is the sample code snippets (<*pre*> element).

1) Extracting from textual elements.

According to our observation, there are many sentences in the tutorial texts which explicitly tell reader to use certain APIs for certain functionality (e.g., "Use the *Camera.takePicture()* method to take a picture once the preview is started."). Therefore, we try to find such kind of sentences which mention certain Android APIs and their usages, and use the sentences as the descriptions for the APIs.

To find such sentences, we first need to recognize Android APIs in the sentences. We discover that in the tutorial pages, when an Android API are mentioned in texts, it always appears in an <*a*> element, with the "*href*" attribute linking to the detailed page of the API in the Android SDK Reference (https://developer.android.com/reference/…). Therefore, to find Android APIs in the texts, we simply find all <*a*> elements in the texts and select those whose "*href*" attribute is a link to the Android SDK Reference page.

2) Extracting from sample code snippets.

In the tutorial pages, besides textual information, there are also many code snippets as examples to show reader how to use APIs programmatically. Fig.1 shows an example of code snippet in the tutorial.

```
1 private void stopPreviewAndFreeCamera() {
2
3    if (mCamera != null) {
4       // Call stopPreview() to stop updating the preview surface.
5       mCamera.stopPreview();
6       // Important: Call release() to release the camera for use by other
7       // applications. Applications should release the camera immediately
8       // during onPause() and re-open() it during onResume().
9       mCamera.release();
10
11        mCamera = null;
12    }
13 }
```

Figure .1 An example of code snippet in the tutorial

Most of the snippets have comments before some codes to explain the purpose of the code, for example, in Fig.1, the comment in line 4 describes why method ***stopPreview()*** should be invoked in the next line (line 5) and the comments from line 6 to line 8 explain the purpose and rationale for calling ***release()*** method in line 9. Following the intuition that each comment usually describes the purpose of the following line of code, we try to extract such comments and the APIs that are used in the following codes.

To extract mappings of APIs and corresponding comments from the snippets, we utilize a third-party library, eclipse JDT (Java Development Tools) [8], to parse the snippets in the tutorial. With the help of JDT, we can parse the snippet and build the Abstract Syntax Tree (AST) of the source code. We use a customized AST Visitor to traverse every node in the tree. As is mentioned in Section 2, we only focus on public classes, public methods and public fields as our target APIs, which correspond to three different kinds of AST nodes in JDT. To find public classes, we need to find instantiations of such classes, which are represented as AST nodes called ***ClassInstanceCreation***, for example, the instantiation of class ***Intent*** in line 2 of Fig.2 will be recognized as a ***ClassInstanceCreation*** by JDT parser. To find public methods, we search the AST for the ***MethodInvocation*** nodes which represent invocations of methods. As for public fields, given that public fields in Android development are often constants which are usually capitalized with separated words concatenated with underscore (e.g., ***Intent.ACTION_PICK*** in line 2 of Fig.2), we try to find tokens that conform to such patterns. Moreover, to make sure to find fields that are more likely to be used to implement functionalities, we only focus on fields that are used when invoking methods or creating new instances (i.e., must be a child node of ***MethodInvocation*** or ***ClassInstanceCreation***).

To get comments from the snippet, we also use the JDT library. We can get a list of all comments in the snippet with the help of JDT, and with some calculation of position and offset in the source code, we can get the line numbers for both comments and the three kinds of AST nodes mentioned above. Therefore, we are able to map each line of comment (like line 4 of Fig.1) or a set of continuous lines of comments (like line 6 to line 8 of Fig.1) with the API node in the line next to the comments, and extract the co-relationship pairs out of them.

```
1 private void pickContact() {
2    Intent pickContactIntent = new Intent(Intent.ACTION_PICK, Uri.parse("content://contacts"));
3    pickContactIntent.setType(Phone.CONTENT_TYPE);
4    startActivityForResult(pickContactIntent, PICK_CONTACT_REQUEST);
5 }
```

Figure .2 Another example of code snippet in the tutorial

Apart from snippets with comments, there are still some snippets that don't have comments in them, such as the snippet in Fig.2, but such snippets are often relatively short. For these kind of snippets, we find some external texts to describe the APIs in these short snippets. We observe that, before such short snippets, there are always some texts introducing what the snippet is used for. For example, in the tutorial page where Fig.2 is selected from, there is a short description which reads "here's how to start an activity that allows the user to pick a contact:" that explains the purpose of the snippet. Therefore, we use this short text as the description for all APIs in the snippet.

Results

We apply our approach on 230 HTML pages from Android tutorial. With our approach, we have managed to extract 1119 co-relationship pairs between APIs and descriptions from the code snippets in the tutorial, and 2281 co-relationship pairs from textual information in the tutorial. After detecting and removing duplicated pairs, we can get a total of 2750 co-relationship pairs from the Android tutorial.

Conclusion & Future Work

In this work, we propose an approach to extract co-relationship pairs between APIs and descriptions to build a database for API recommendation for Android development. We use official tutorial for Android developers as our data source, and use both textual information and sample code snippets for data extraction. We apply our approach on 230 HTML pages from Android tutorial, and get a total of 2750 co-relationship pairs between APIs and their descriptions.

Our approach still has many limitations such as relatively small data source and straightforward description selection. In future work, we prepare to test our approach with larger scale of data source, and find API descriptions more accurately with semantic parsing.

References

[1] Gu, X., Zhang, H., Zhang, D., & Kim, S. (2016). Deep API learning. *ACM Sigsoft International Symposium on Foundations of Software Engineering*(pp.631-642). ACM.

[2] Rahman, M. M., Roy, C. K., & Lo, D. (2016). RACK: Automatic API Recommendation Using Crowdsourced Knowledge. *IEEE, International Conference on Software Analysis, Evolution, and Reengineering* (pp.349-359). IEEE.

[3] Thung, F., Shaowei, W., Lo, D., & Lawall, J. (2013). *Automatic recommendation of API methods from feature requests. 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE) Proceedings: 11-15 November 2013, Silicon Valley, CA.*

[4] Buse, R. P. L., & Weimer, W. (2012). Synthesizing API usage examples. *International Conference on Software Engineering* (Vol.8543, pp.782-792). IEEE Computer Society.

[5] Moritz, E., Linares-Vasquez, M., Poshyvanyk, D., Grechanik, M., Mcmillan, C., & Gethers, M. (2014). ExPort: Detecting and visualizing API usages in large source code repositories. *Ieee/acm, International Conference on Automated Software Engineering* (Vol.19, pp.646-651). IEEE.

[6] Nguyen, H. A., Dyer, R., Nguyen, T. N., & Rajan, H. (2014). Mining preconditions of APIs in large-scale code corpus. *ACM Sigsoft International Symposium on Foundations of Software Engineering*(pp.166-177). ACM.

[7] Wang, J., Tong, X., Lin, S. S., Guo, B., Shum, H. Y., & Lin, Z. (2012). Modeling and rendering of heterogeneous translucent materials using the diffusion equation. *US 8243071 B2*.

[8] https://www.eclipse.org/jdt/