APIExample: An Effective Web Search Based Usage Example Recommendation System for Java APIs

Lijie Wang, Lu Fang, Leye Wang, Ge Li, Bing Xie, Fuqing Yang
Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, P.R. China
Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, P.R. China
{wanglj07, fanglu09, wangly09, lige, xiebing, yang}@sei.pku.edu.cn

Abstract—Programmers often learn how to use an API by studying its usage examples. There are many usage examples scattered in web pages on the Internet. However, it often takes programmers much effort to find out the desired examples from a large number of web pages by web search. This paper proposes a tool named APIExample that can extract usage examples for java APIs from web pages on the Internet and recommend them to programmers. Given a java API, the tool collects its related web pages from the Internet, extracts java code snippets and their surrounding descriptive texts embedded in the pages, then assembles them into usage examples for programmers. Furthermore, in order to help programmers capture more kinds of usages of the target API by browsing fewer examples, our tool clusters and ranks the listed examples based on the target API’s usage. Besides, as a practical tool, APIExample provides multiple aspects of frequently-used information about using the target API in a concise user interface with friendly user experience. Two kinds of user-interaction style, a web search portal and an Eclipse plug-in, are now both publicly available.

Keywords—API, usage example, web search, recommendation

I. INTRODUCTION

One problem often faced by a programmer in reusing existing framework is that the programmer knows which API (e.g. a java class) should be used, but doesn’t know how to use it. This is also known as “use barriers [1]”. It happens in many cases, for instance the programmer has used the API before, but now cannot remember how to use it clearly; or the programmer learned somewhere that for a task an API should be used, but how to use the API was not presented.

In these cases, usage examples can be helpful for learning to use the API. Some approaches [4,5,6,8] which extract usage examples from project codebases were proposed. But it is often hard for them to produce easy-to-understand examples due to the inherent lack of descriptions in codebase. There are many usage examples scattered in web pages on the Internet, such as tutorials, technical blogs, and forums. These usage examples typically have rich information including example code and descriptive texts. Existing observations indicate that many programmers rely on web search (e.g. Google) to find such usage examples on the web [2, 3, 7]. However, one of the main problems faced by programmers when searching for usage examples on the web is that they often need to spend great effort in finding out relevant examples from vast data in lots of web pages due to the existence of irrelevant noises, scattering results, and data duplication. Several researches try to extract programming information from web pages and deliver them to programmers via a uniform view [2, 3, 7]. However, due to the lack of in-depth analysis on the collected data, programmers still face heavy burden of viewing lots of results to capture a full view on the target API so as to learn how to use it well.

In this paper, we propose a practical tool named APIExample which provides a full view on the usage examples of a given java API based on web search. To help programmer figure out the API’s usage examples easily from lots of web pages, the tool automatically identifies and extracts its usage examples from related web pages while eliminates irrelevant noises. The tool not only extracts code snippets but also properly extracts readable descriptive texts. This makes the usage examples more easy-to-understand and programmer can view them without disturbing of irrelevant information. To reduce programmer’s burden of viewing many duplicated results, APIExample conducts usage based clustering and ranking on the collected usage examples. With this supporting, a programmer can study more kinds of usage of the target API by browsing fewer results. In addition, based on analysis on the collected examples, APIExample can directly tell programmer which usage of the API frequently appears on the Internet, which methods of the API are frequently used, what other APIs are often used together with the API. The information can be useful to guide programmer to learn about the API better. All of these usage related information is presented to programmer in a concise user interface with friendly user experience.

With the help of APIExample, a programmer can capture a full view on the usage examples of the target API and thus learn to use the API efficiently. The tool provides two kinds of user interaction style: a web search portal and an Eclipse plug-in. By the time we finish this paper, the tool has collected 76,626 usage examples from 422,987 unique web pages for 16 famous java open source libraries (e.g. JDK, jdom, log4j, lucene) with totally 6,491 APIs. The tool is now available at: http://www.apiexample.com.

The rest of this paper is organized as followings: an overview on the tool is presented in section II; section III describes the approach behind the tool; comparison with related work and discussion is presented in section IV.

II. OVERVIEW ON APIEXAMPLE

In this section, we briefly introduce the capability of APIExample using its web search portal UI as example. Please
With APIExample, programmer can use API name (e.g. java.io.BufferedReader) as query to retrieve its usage examples on the Internet. The tool provides intelligent query suggestions simultaneously while programmer typing in search box. For instance, ‘java.io.BufferedReader’ will be suggested out even when the programmer just typed ‘buf’.

After submitting ‘java.io.BufferedReader’ as query, the API’s usage examples and other usage related information will be returned, as shown in Figure 1. The presented usage examples are all automatically extracted from web pages on the Internet. They not only contain code snippet (A) but also have proper descriptive texts (B). Important areas in the texts and codes are automatically detected and decorated with friendly highlighting to make examples more readable (C). To reduce programmer’s effort wasted on viewing duplicated results, APIExample clusters usage examples exhibiting the same usage of the target API into a category, which we call an example cluster. Typical and representative results are displayed first. The results in Figure 1 indicate that there are 12 example clusters (D) for BufferedReader. Each example cluster corresponds to a kind of usage of the API. For instance, the 40 usage examples in the first cluster (E) all invoke BufferedReader’s methods: <init> (java.io.Reader) and String readLine(). The 15 usage examples in the second cluster (F) all invoke method void close() in addition to the two methods used by the first cluster. It is obvious that, with the help of APIExample, the programmer can efficiently view many different usage examples which formerly distributed in various web pages.

APIExample also presents some other usage related information which is useful for learning about an API.

From the visual pie chart on the top right corner (G), the programmer can easily figure out the proportion of each usage of the target API exhibited by the collected usage examples. The detail of a usage appears when hovers over the corresponding pie field. By simply clicking a pie field, the programmer can quickly jump to the example cluster which exhibits the usage.

In ‘Frequently Used Methods’ section (H), APIExample lists the methods of the target API in the order of use frequency in the usage examples. For instance, it’s easy to find that methods <init> (java.io.Reader) and readLine() of BufferedReader are often used. With this information, the programmer can easily find out which methods of the API are used frequently as well as which methods are seldom used and further to adjust her learning process. If the programmer is especially interested in some methods, she can select the methods and refresh results, the tool will only present usage examples containing invocations on the specified methods.

An API typically cannot be used alone to accomplish a task. In ‘APIs Used Together’ section (I), other APIs which are often used together with the target API are displayed in the order of co-used frequency in the usage examples. The programmer can select some specific APIs and refresh search results to view usage examples which use both the target API and the selected APIs. For instance, if she wants to know about how BufferedReader is used together with java.net.Socket, she can select the corresponding item, then only the usage examples using both the two APIs will be displayed.

In addition to the above mechanisms which can be used to refine the search process, APIExample provides keywords based filtering. The programmer can view usage examples whose code or descriptive texts contain specific keywords by typing keywords in the filtering box (J).

With the usage related information in multiple aspects generated from the usage examples, the programmer can learn about how to use an API from different perspective simultaneously, so as to improve the efficiency.

III. Approach

As shown in Figure 2, the approach behind APIExample consists of four main stages: (1) web page collection, (2) usage example extraction, (3) example clustering and ranking, and (4) statistic analysis.

A. Web Page Collection

Given an API, APIExample collects its related web pages from the Internet leveraging Google through constructing query in format of “API_FQN” example java, where API_FQN is the full qualified name of the given API, terms ‘example’ and ‘java’ are used to restrict the searching scope.
The top $N$ (currently 300) web pages in the results list are downloaded as the target API’s related web pages.

B. Usage Example Extraction

This stage consists of three steps: 1) web page segmentation; 2) code snippet identification; 3) descriptive texts extraction.

1) Web Page Segmentation

This step splits the text content of a web page into page segments according to HTML document structure. We consider several specific tags as segment symbols, including P,DIV, TABLE,TR,PRE,CODE,OL,UL,LI and Heading Tags (H1 ~H6) because they are typically used as container of content segments. The text content embedded in these tags is considered as a segment. Other tags are erased while their text content is reserved to their parent tags.

2) Code Snippet Identification

For each page segment, we judge whether it is a code snippet with two steps: a) using heuristics to preliminarily determine whether the segment “looks like” a code snippet; b) using an adaptable parser to parse a possible code snippet to make the final decision and, meanwhile, to extract a code snippet’s fine-grained structural information.

a) Preliminary Code Snippet Identification

In this step, we use heuristics gotten from observations to preliminarily determine whether a page segment is a code snippet or not. We use typical surface characteristics of java source code to conduct the judgment, including a line ended with semicolon; containing brace, containing parenthesis, containing java reserved keywords public/protected/private/class/static/new/interface/import. A text line is determined as a code line if it exhibits at least two surface characteristics. If the proportion of code lines in a page segment is higher than a given threshold (currently 0.5), the segment is judged as a candidate code snippet.

b) Judge Code Snippet by Parsing

We developed an Adaptable Parser (AP) which can incrementally parse a candidate code snippet effectively. The parser makes use of Eclipse Abstract Syntax Tree Parser (EASTP) to parse code snippet. However, EASTP will fail when there are some distortions in the code snippet, such as prefixed line number, wrapped texts surrounding the code, embedded illegal characters (e.g. ‘…’). Such cases should be handled because they are quite common for the code in web pages. APIExample’s AP iteratively attempts to parse a code snippet. If a parsing error happened, AP will conduct some adjustment on the code snippet, such as delete the prefixed line number, remove the encountered illegal characters, add ‘//’ at the beginning of a non-code line to change it to a comment line, wrap place-holder (e.g. a class or method declaration place-holder) to make the code snippet a complete compile unit, to eliminate the parse error. If a candidate code snippet cannot be parsed after all adjustment attempts, it will be excluded. Only the candidates which can be parsed successfully are approved as valid code snippets.

In this step, the fine grained programming structural information (such as which APIs are used, which method are invoked in which line, which methods are declared) produced during the parsing phase are recorded for following stages.

3) Descriptive Texts Extraction

The descriptive texts interpreting a code snippet are often very close to (typically around) the code in the original web page. The text segment immediately preceding the code snippet generally has close correlation with the code snippet, and it is either an introduction to the task accomplished by the code or some descriptions like preface. The text segment immediately following the code snippet has correlation with the code snippet in some conditions, when the text generally tells something about the code snippet which is not presented in the preceding one, such as meanings of the declared methods or variables. If the following text segment has correlation with the code, it is often a complementation to the preceding one. As we observed, in this condition, the text segment often mentions some programming elements declared by the code snippet. Based on the above observations, we use the following approach to extract descriptive texts for a code snippet:

Traverse the segment list of each web page, when a code segment is encountered, we use the following strategies to extract its preceding descriptive texts: Iteratively merge segments preceding the code segment until one of the following conditions is met: 1) another code segment is encountered, 2) The length of merged descriptive texts reaches a given threshold (currently 250 terms) 3) A symbol of break in content is encountered. The following tags are considered as a symbol of break: DIV, TABLE, HR, UL, OL, H1~H6.

Regarding the descriptive texts following the code, based on the observation, we try to extract them if the text segment immediately following the code snippet mentions the programming elements declared by the code block. Then we use strategies similar to those used for preceding text extraction to extract descriptive texts after a code snippet.

C. Example Clustering and Ranking

Due to the vast results returned by web search, the number of usage examples is usually very large. Actually, many of them demonstrate similar usages of the target API. To learn about how to use an API, a programmer often needs to grasp more than one usage of the API. To reduce the effort a programmer needs to spend exploring many duplicated results,
we conduct clustering and ranking on the collected usage examples. This is important because programmers’ cognitive burden in understanding vast number of searching results during web search has been a big problem making them struggle with information.

People applying an API to accomplish different tasks generally will invoke different method sets of the API. Therefore, we take invocations on the methods of target API exposed by each example’s code snippet as usage feature. Then we leverage the feature to calculate similarity between any two usage examples. Two examples invoking the same methods set of target API will be clustered into a category.

Ranking consists of two parts: 1) inter-cluster ranking, i.e. ranking usage clusters and 2) intra-cluster ranking, i.e. ranking usage examples in a usage cluster.

1) Inter-Cluster Ranking

We rank usage clusters according to the popularity of the usage exhibited by each cluster. The more examples a cluster has, the more typical usage it exposes. Using the frequency to determine the popularity of usage is traditional in API usage specification mining research. The more frequently a usage is exposed, the more typical the usage might be.

2) Intra-Cluster Ranking

We use following strategies to rank usage examples in an example cluster to select examples with good demonstrative effect: prefer examples with short code snippets; prefer examples whose code snippets contain more comments; prefer examples using less APIs; prefer examples whose code snippets can be parsed with less adjustment attempts; prefer examples with rich descriptive texts; prefer examples extracted from tutorial web pages rather than those from forums.

D. Statistic Analysis

Based on the data produced during above stages, we conduct statistic analysis for usage related information, including the distribution of the API’s different usages, the frequency of invocations on the API’s each method, and the frequently co-used APIs. Analysis results are stored into database as well as usage examples for retrieving.

IV. COMPARISON WITH RELATED WORK AND DISCUSSION

Some approaches of extracting programming related information from web pages have been proposed in human-computer interaction area. Mica [7] automatically identifies programming elements from web search results based on predefined dictionary. Assieme [2] identifies code snippet from web pages, then use the implicit references among code, JAR files and API documents to support common programming search task. Blueprint [3] extracts examples of Adobe Flex classes from web pages and delivers results in a code-centric view. Differently from them, APIExample performs deep analysis on the collected usage examples including fine-grained structural information analyzing, example clustering, and statistics on most frequently used methods as well as co-used APIs. The information in multiple aspects is presented in a concise interface with friendly user experience supporting to help programmer grasp a full view on the API’s usage easily.

Regarding sample code extraction, there are researches retrieving API sample code from project codebase, e.g. MAPO [6], ParseWeb [4], eXoaDocs [5], and SSI [8]. However, because most existing projects are not initially designed for teaching, the code where target API is used often couples with the internal business logic tightly. This makes extracting code snippets from their codebase may produce hard-to-understand results. In addition, the lack of descriptions about the examples impacts their results. The comments embedded in source code are generally about the implemented business logic from solution domain perspective. However, the programmer learning an API often needs descriptions from problem domain perspective to help her understand the example at hand better. The same problems are also exposed by existing code search engine (such as Google Code Search Engine, Koders). Compared to sample code generated from codebase, usage examples in web pages are more readable and easy-to-understand, because many of them are produced for tutoring or sharing (e.g. tutorial, technical blogs). Code snippets in usage examples tell about the implementation from solution domain perspective while descriptive texts provide interpretation from problem domain perspective.

Although extracting usage examples from web pages is different from extracting examples from codebase, we insist on that the two approaches do not conflict with each other. In contrast, they can be combined in some way to produce novel results. We are now trying to carry out some work in this direction based on APIExample’s infrastructure.

In addition to the application scenario presented in this paper, we believe that the extracted usage examples can be used in many other potential areas, such as leveraging the rich information in the examples to detect special API usage specifications. Evaluation on the tool’s effectiveness via typical end-user study is another work to be conducted in the future.

ACKNOWLEDGEMENTS

This research was sponsored by the National Natural Science Foundation of China (No. 60803010, No. 60803011)

REFERENCES