Strings methods analysis

Anonymous Author(s)

Аннотация

Keywords: Object-Oriented Programming

1 Introduction

A string is an ordered set of characters. The standard library of string methods varies from language to language. Then in the case of creating a new language, you need to build your own library based on some criteria. However, the number of different functionality methods for strings that can be invented is at least countable (to put it simply - infinite). Then what to base on to identify the necessary set of methods for an object-oriented language? <- RQ

This article aims to make a multivariate analysis of different parameters of string methods.

We should keep in mind that the number of string methods in different programming languages is very large, and this number can grow, as we said before, without limitations. For this reason, this article does not consider all existing methods, but only those that meet some criteria.

An important factor for any method will be the frequency of its use in the code. In this article we will describe the algorithm, according to which we will evaluate this parameter. We will also investigate the accuracy of these estimates, since there is no absolutely accurate method for estimating this parameter in a reasonable amount of time.

Another important parameter that is investigated in this article is the number of uses of one method, inside (for realization) the others. This criterion allows us to identify important dependencies of some methods on others.

An algorithm to obtain the minimum set of required string methods will be created. This algorithm will be analyzed for susceptibility to bias, since it depends on the initial input data.

Also an important parameter for methods (in this case string methods) is their running time, so this article also considers this parameter. An interesting question is: will mutable or immutable structures be considered? Since this is a complex analysis, both are considered. And also the regularities in the interrelation of mutable and nonmutable methods asymptotics are revealed.

The article then discusses the results obtained and evaluates them. The accuracy of the obtained data is also evaluated. And a conclusion is made: which set of methods we took for our language.

2 Related work

Among the other publications on this topic that exist at the time of this writing, all are of an applied nature. For this reason, they are not discussed in this article. In our article, we provide data that can be used in a free form, including for creating our own string libraries.

3 Considered methods

In order to investigate methods, we have to make a list of methods to be investigated. For this purpose, programming languages such as Java, Python, C++, Ruby were chosen. They were chosen on the basis that they are quite popular, and they implement different paradigms (e.g. strict and non-strict typing in C++ and Python respectively).

A table with the list of different methods from standard string libraries for these languages has been compiled. For the analysis will be selected methods, which are presented at least in two programming languages (out of four presented). Since for the methods presented only in one language, we can conclude that they are used in too special or rare cases.

To avoid self-repeats, a summary list of the methods to be analyzed is presented in Section 8.

4 Popularity of methods

4.1 Definition

Since popularity is a measurable parameter, let's calculate the number of uses of each string method (see below). Then we denote popularity as a percentage of the number of uses of a certain string method relative to the total number of uses of all string methods. This approach makes it possible to estimate the frequency of string methods usage exactly in the context of correlation with other string methods.

4.2 Search algorithm

The GitHub and Sourcegraph platforms were chosen to calculate the number of uses of the methods. This

platform provides the ability to search for any words or phrases in all GitHub projects, as well as if necessary to specify other parameters.

However, you should take into account that asking for the number of times a certain word occurs in the code in certain cases will not give a correct result, there are a number of reasons for that.

4.2.1 The word is a comment.

For some words, it is more common for them to appear in files as names/comments/inside strings rather than as called functions. Since the article deals with the use of string methods for object-oriented languages, and they use dot notation, it will be more efficient to make queries like .<method name>, instead of <method name>.

4.2.2 It's a another class method.

In this kind of analysis, for some methods there is a problem that it is not clear which class of objects has which method. For example, the contains method is present in string as well as in most collections.

Of course, it makes sense to examine the context in which the methods were used. However, this task is difficult because of the large amount of data to be analyzed.

It is worth noting that methods which have had this problem are popular (note here that we mean methods for all classes, not specifically string methods), this we have obtained by making queries about the number of times they occur for code bases. For this reason, this data will not be counted in this article, but a method for solving this problem is given in the Discussion section.

5 Mutual usages

For any method (in our case string methods) you can consider such a parameter as its use by other methods. It is important because it is impossible to exclude a method from the library in case many other methods of the library use it.

This parameter will be an integer numeric value that will be equal to the number of methods that can use it in their implementation (that is, it should not just be present in the implementation, but be necessary there) without affecting the asymptotics.

6 Asymptotics

6.1 Mutable

In order to define the asymptotics of the different string methods, let us define what a string is. Since we are considering mutable strings, we consider a string to be an array of characters with which we can use write and read operations. This way we can avoid copying for every mutable operation.

To understand more about the advantages of this approach: we can change any character of a string in a constant amount of time, because we won't have to copy anything. However, in order to take a substring of the string, we will have to explicitly create a new string - this is a disadvantage.

6.2 Immutable

Immutable strings impose a number of constraints on the operation. We define immutable strings as an array of characters where we are allowed to read characters in a constant time, and each mutable operation creates a new copy. We are also allowed to store a constant number of other variables, such as various flags.

However, no additional restrictions are imposed, for example, in mutable operations we can refer to past versions.

To better understand what advantages this approach gives: if we need to take a substring of a string, we can do it in a constant time, because due to immutability we can simply refer to characters inside the old string. However, for example, the operation to replace the last character of the string will take linear time from the length of the string.

7 Finding the minimum set

7.1 Necessity

Even if there are various criteria for methods, it is still necessary to determine how to get a library based on them. Therefore, we propose an algorithm that can be used for various (including not yet existing) object-oriented programming languages.

7.2 Parameter setting

It is necessary to set a criterion on the basis of which a decision will be made whether to add a method to the current set of methods.

As an example, consider the following criterion: a method cannot be implemented through three or less methods existing in the set, so that its asymptotics is not degraded.

7.3 Algorithm

Now specifically about the algorithm on the basis of which the final set of methods will be obtained.

1. Set is initially empty.
2. Functions are considered in priority order, by default the most popular methods come first. If the method meets criterion 7.2, it is added to the set.
3. The second step is repeated until there are no unselected methods left.
4. In reverse order of priority the methods in the set are considered and those that don't meet criterion 7.2 are removed.

8 Results

Below is a table with the results - metrics.

Based on these data, we have compiled our optimal library for string methods; the included methods are listed below.

todo

9 Discussion and analysis

9.1 Mutual usages presenting

When we counted the Mutual usages parameter we used a numerical value. However, it is worth considering that a representation in the form of an oriented dependency graph would give more applied information. Because based on this graph, it would be less biased to construct a minimal set of string methods.

However, it is not possible to represent the graph in this paper because the number of its edges would be too large (of the order of the square of the number of methods). And such representation would lose any comprehensible visualization.

9.2 Problems with method intersection

Earlier we mentioned the problem with popularity calculation for methods that have the same name, but are called on objects of different classes.

To solve this problem we propose to use probabilistic approach. Suppose we need to know how many times a method with name Y has been used on string. Randomly gather a sample of files where this word Y occurs. Analyze what percentage of them uses word Y as a string method. Also find out the total number of times word Y was used in the code. Multiplying the total number by the probability we will get the answer. In fact, we will apply the extrapolation method.

Note that this method does not show absolute accuracy; however, if the files to be analyzed are selected at random, the error decreases very much.

In order to estimate the error it is necessary to take several similar samples and compare how different the results are.

9.3 Unexpected advantage of immutability

todo

10 Conclusion

todo

11 Used materials

todo

	Popularity	Mutual usages	Mutable O(?)	Immutable O(?)
capitalize			1	1
char-at			1	1
compare			n	n
compare-ignore-case				
concatenate			n	n
containts			n	n
count				
ends-with			n	n
equals			n	n
find-first			n	n
find-last			n	n
format			n	n
hash			n	n
insert			n	n
is-alphabetic			n	n
is-empty			1	1
is-hexadecimal			n	n
is-int			n	n
is-lowercase			n	n
is-uppercase			n	n
is-whitespace			n	n
join			n	n
length			1	1
matches-regexp			n	n
partition			n	n
repeat			n	n
replace-all			n	n
replace-first				
split			n	n
starts-with			n	n
strip			n	n
substring			n	1
swap-case			n	n
to-float			n	n
to-int			n	n
to-lowercase			n	n
to-uppercase			n	n
trim-left			n	n
trim-right			n	n
reverse			n	1
remove				
uncapitalize				
as-bytes				
chop				
clear				

Таблица 1. String methods criteria