# 11 个简练的 Java 性能调优技巧

**想要让你的项目一直高性能运作吗？以下有一些技巧你可以拿去消除缓存瓶颈，还有一些其他的性能调优建议。**

**大多数开发者认为性能优化是一个复杂的话题，它需要大量的工作经验和相关知识理论。好吧，这也不完全错。优化一个应用做到性能最优化可能不是件容易的任务，但是这并不意味着你没有相关的知识就什么也做不了。这里有一些易于遵循的建议和最佳实践可以帮助你创建一个性能良好的应用程序。**

**这些建议的大部分都是针对 Java 语言的。但是也有一些是跟语言无关的，你可以运用到任意的应用和程序中。在我们学习特定的 Java 编程性能调优之前，先来探讨一些通用的技巧。**

##### 1.在明确必要之前别急着优化

这可能是最重要的性能优化技巧之一。你应该遵循常见的最佳实践做法并在案例中高效地应用它。但是这并不意味在证明必要之前，你应该更换任何标准库或构建复杂的优化。

多数情况下，过早地优化会占用大量的时间，而且会使代码变得难以理解和阅读。更糟糕的是，这些优化通常并没带来任何好处，因为你花了大量的时间在优化应用中的非关键部分。

那么，要怎么证明东西需要优化呢？

首先，你需要定义你的代码速度得多快。例如，为所有 API 调用指定最大响应时间，或者指定在特定时间范围内要导入的记录数量。在做完这些后，你需要确定你应用中哪些部分太慢需要改进。当完成这些后，你就可以来看看第二个技巧提示。

##### 2.使用分析器找到真正的瓶颈

在完成第一部分的优化建议以鉴别出你应用中需要提升的部分后，要从哪里入手呢?

你可以有两种途径来解决这个问题：

查看你的代码，从看起来可疑的或者你觉得可能会导致出现问题的地方入手。

或者使用分析器获取代码每个部分的行为(执行过程)和性能的详细信息。

希望我不需要解释为什么应该始终遵循第二种途径/方法的原因。

很显然，基于分析器的方式可以让你更好地理解代码的性能影响，并允许你去专注于更关键的部分(代码)。即使你曾经使用过分析器，你一定记得你曾经多么惊讶于一下就找到了代码的哪些部分产生了性能问题。我第一次的猜测不止一次地导致我走错了方向。

##### 3. 为整个应用程序创建一个性能测试套件

这是另一个通用的可以帮助你避免在将性能改进部署到产品中之后经常会发生的许多意外问题的技巧。你应该总是定义一个性能测试套件来测试整个应用程序，并在性能改进之前和之后运行它。

这些额外的测试运行将帮助你识别你的改动所引起的功能和性能上的副作用，并确保不会导致弊大于利的更新。如果你处理的是被应用程序的多个不同部分使用的组件，如数据库或缓存，那这一点尤为重要。

#### 4. 优先关注最大瓶颈

在创建了测试套件并使用分析器分析你的应用程序之后，你可以列出一系列需要解决以提高性能的问题列表。这很好，但这并没有回答你需要从哪里开始的问题。你可以专注于速成方案，或从最重要的问题开始。

速成方案一开始可能会很有吸引力，因为你可以很快显示第一个成果。但有时，可能有必要说服其他团队成员或管理层认为性能分析是值得的。

一般来说，我建议从顶层开始，首先开始处理最重要的性能问题。这将为你提供最大的性能改进，而且你可能仅需要解决这些问题中的一小部分就能满足你的性能要求。

常见的通用调优技巧到此结束。接下来让我们仔细看看一些特定于 Java 的技巧。

##### 5.使用 StringBuilder 以编程方式连接字符串

在 Java 中有很多不同的选项来连接字符串。例如，你可以使用简单的 + 或 + = ，以及老的 [StringBuffer](https://docs.oracle.com/javase/9/docs/api/java/lang/StringBuilder.html" \t "_blank) 或 StringBuilder 。

那么，你应该选择哪种方法呢？

答案取决于连接字符串的代码。如果你是以编程方式将新内容添加到字符串中，例如在 for 循环中，则应使用 StringBuilder 。它很易于使用，并提供比 StringBuffer 更好的性能。但请记住，与 StringBuffer 相比， StringBuilder 不是线程安全的，可能并不适用于所有情况。

你只需要实例化一个新的 StringBuilder 并调用 append 方法来向 String 中添加一个新的部分。在你添加完了所有的部分后，你可以调用 toString() 方法来检索已连接的字符串。  
  
下面的代码片段展示了一个简单的例子。在每次迭代期间，该循环将 i 转换为一个 String ，并将其与空格一起添加到 *StringBuilder sb*中。所以，最后，这段代码在日志文件中写入 “This is a test0 1 2 3 4 5 6 7 8 9” 。

StringBuilder sb = new StringBuilder(“This is a test”);for (int i=0; i<10; i++) {

    sb.append(i);

    sb.append(” “);

}

log.info(sb.toString());

正如你在代码片段中看到的。我们可以为字符串的第一个元素提供到构造函数中。这会创建一个 *StringBuilder*，其中包含了你所提供的字符串以及 16 个额外字符的容量。当你向 *StringBuilder*中添加更多字符时，你的 JVM 将动态的增加 *StringBuilder*的大小。

如果你已经知道字符串将包含多少个字符，则可以将该数字提供给不同的构造方法以实例化具有指定容量的 StringBuilder 。这进一步提高了效率，因为它不需要动态扩展其容量。

##### 6. 使用 + 连接一个语句中的字符串

当你使用 Java 实现你的第一个应用程序时，可能有人告诉过你不要使用 + 来连接字符串。如果你是在应用程序逻辑内连接字符串的话，这是对的。字符串是不可变的，每个字符串的连接结果都被存储在一个新的字符串对象中。这需要额外的存储空间，并可能使你的应用程序运行缓慢，特别是当你在一个循环内连接多个字符串的情况下。

在这些情况下，你应该遵循技巧 5 中的内容，并使用 *StringBuilder 。*

但如果你只是将字符串分成多行来改善代码的可读性，这并不适用。

Query q = em.createQuery(“SELECT a.id, a.firstName, a.lastName ”

+ “FROM Author a ”

+ “WHERE a.id = :id”);

在这些情景下，你应该使用简单的 + 来连接字符串。你的 Java 编译器会优化它，并在编译时完成连接。因此，在运行时，你的代码将只使用一个字符串，并不需要任何连接操作。

##### 7. 尽可能使用基本类型

避免任何开销并提高应用程序性能的另一种简便快速的方法是使用基本类型而不是其包装类。所以，最好使用 int 而不是 Integer ，是 double 而不是 Double 。这将使得你的 [JVM](https://stackify.com/jvm-metrics/) 将[值存储在堆栈而不是堆中](https://www.javaworld.com/article/2150208/java-language/a-case-for-keeping-primitives-in-java.html" \t "_blank)，以减少内存消耗，并更有效地处理它。

##### 8. 尽量避免大整数和小数

由于我们已经在讨论数据类型，所以我们也应该快速浏览[大整数](https://docs.oracle.com/javase/9/docs/api/java/math/BigInteger.html)和[小数](https://docs.oracle.com/javase/9/docs/api/java/math/BigDecimal.html)。尤其是后者因其精确性而受欢迎。但这是有代价的。  
  
大整数和小数比一个简单的 *long* 型或 *double* 型需要更多的内存，并会显著减慢所有的运算。所以，如果你需要额外的精度，或者如果你的数字超出一个较长的范围，最好要三思。这可能是你需要更改并解决性能问题的唯一方法，尤其是在实现数学算法时。

##### 9. 优先检查当前日志级别

这个建议应该是显而易见的，但不幸的是，很多人在写代码的时候都会忽略它。 在创建调试消息之前，应该总是优先检查当前日志级别。 否则，你可能会创建一个附加你[日志消息](https://stackify.com/log-management/)的字符串，而该字符串之后将被忽略。

这里有两个你不应该这样做的反面例子。

// don’t do this

log.debug(“User [” + userName + “] called method X with [” + i + “]”);

// or this

log.debug(String.format(“User [%s] called method X with [%d]”, userName, i));

在这两个示例中，你都将执行创建日志消息所有必需的步骤，而不知道日志框架是否将使用日志消息。 因此在创建调试消息之前，最好先检查当前的日志级别。

// do thisif (log.isDebugEnabled()) {

    log.debug(“User [” + userName + “] called method X with [” + i + “]”);

}

##### 10. 使用 Apache Commons StringUtils.Replace 而不是 String.replace

一般来说，String.replace 方法可以正常工作，并且效率很高，尤其是在你使用 Java 9 的情况下。但是，如果你的应用程序需要大量的替换操作，并且没有更新到最新的 Java 版本，那么检查更快和更有效的替代品依然是有必要的。

有一种候选方案是 [Apache Commons Lang](https://commons.apache.org/proper/commons-lang/) 的 *[StringUtils.replace](https://commons.apache.org/proper/commons-lang/javadocs/api-release/org/apache/commons/lang3/StringUtils.html" \l "replace-java.lang.String-java.lang.String-java.lang.String-" \t "_blank)*方法。正如 Lukas Eder 在他最近的一篇[博客文章](https://blog.jooq.org/2017/10/11/benchmarking-jdk-string-replace-vs-apache-commons-stringutils-replace/" \t "_blank)中所描述的，它远远胜过了 Java 8 的 String.replace 方法。

而且它只需要很小的改动。你只需要将 Apache Commons Lang 项目的 [Maven](https://stackify.com/gradle-vs-maven/) 依赖项添加到你的应用程序的 pom.xml 中，并将 String.replacemethod 的所有调用替换为 StringUtils.replace 方法。

// replace this

test.replace(“test”, “simple test”);

// with this

StringUtils.replace(test, “test”, “simple test”);

##### 11.昂贵的缓存资源，如数据库连接

缓存是避免重复执行昂贵或常用代码片段的流行解决方案。总的思路很简单：重复使用这些资源比创建一个新的资源更划算。

一个典型的例子是缓存池中的数据库连接。新连接的创建需要时间，如果你重用现有连接，则可以避免这种情况。

你也可以在 Java 语言源码中找到其他的例子。例如，在 Integer 类中的 *valueOf*  方法缓存了介于 -128 到 127 之间的值。你可能会说创建一个新的 Integer 并不是太昂贵，但是由于它经常被使用，因此缓存最常用的值也可以提供性能优势。

但是，当你考虑使用缓存时，请记住缓存实现也会产生开销。你需要花费额外的内存来储存可重复使用的资源，因此你可能需要管理你的缓存以使资源可访问，并删除过期的资源。

所以，在开始缓存任何资源之前，请确保它们是经常使用的，以超过缓存实现的开销(代价)。

总结

正如你所看到的，有时不需要太多的工作就可以提高你的应用程序的性能。本文中的大部分建议只需要稍作努力就可以将它们应用于你的代码中。

但还是那句话，最重要的还是那些与是什么编程语言无关的技巧：

在你知道其必要性之前不要进行优化

使用分析器（profiler）来查找真正的瓶颈

优先处理最大的瓶颈

**Want to keep your programs running performantly? Here are some steps you can take to eliminate bottlenecks, tips for caching, and other performance tuning suggestions.**

**Most developers expect that performance optimization is a complicated topic that requires a lot of experience and knowledge. Okay, that’s not entirely wrong. Optimizing an application to get the best performance possible isn’t an easy task. But that doesn’t mean that you can’t do anything if you haven’t acquired that knowledge. There are several easy to follow recommendations and best practices which help you to create a well-performing application.**

**Most of these recommendations are Java-specific. But there are also several language-independent ones, which you can apply to all applications and programming languages. Let’s talk about some of these generic ones before we get to the Java-specific performance tuning tips.**

##### 1. Don't Optimize Before You Know It's Necessary

That might be one of the most important performance tuning tips. You should follow common best practices and try to implement your use cases efficiently. But that doesn’t mean that you should replace any standard libraries or build complex optimizations before you proved that it’s necessary.

In most cases, premature optimization takes up a lot of time and makes the code hard to read and maintain. And to make it even worse, these optimizations most often don’t provide any benefits because you’re spending a lot of time optimizing non-critical parts of your application.

So, how do you prove that you need to optimize something?

First of all, you need to define how fast your application code has to be, e.g., by specifying a maximum response time for all API calls or the number of records that you want to import within a specified time frame. After you’ve done that, you can measure which parts of your application are too slow and need to be improved. And when you’ve done that, you should take a look at the second tip.

##### 2. Use a Profiler to Find the Real Bottleneck

After you followed the first recommendation and identified the parts of your application you need to improve, ask yourself where to start?

You can approach this question in two ways:

You can take a look at your code and start with the part that looks suspicious or where you feel that it might create problems.

Or you use a profiler and get detailed information about the behavior and performance of each part of your code.

I hope I don’t need to explain why you should always follow the second approach.

It should be obvious that the profiler-based method gives you a better understanding of the performance implications of your code and allows you to focus on the most critical parts. And if you ever used a profiler, you will remember a few situations in which you were surprised by which parts of your code created the performance issues. More than once my first guess would have led me in the wrong direction.

##### 3. Create a Performance Test Suite for the Whole Application

This is another general tip that helps you avoid a lot of unexpected problems that often occur after you have deployed your performance improvement to production. You should always define a performance test suite that tests the whole application, and run it before and after you worked on a performance improvement.

These additional test runs will help you to identify the functional and performance side effects of your change and make sure that you don’t ship an update that caused more harm than good. That is especially important if you work on components that are used by several different parts of your application, like databases or caches.

##### 4. Work on the Biggest Bottleneck First

And after you have created your test suite and analyzed your application with a profiler, you have a list of issues you want to address to improve the performance. That’s good, but it still doesn’t answer the question where you should start. You could focus on the quick wins, or start with the most significant issue.

It might be tempting to start with the quick wins because you will be able to show first results soon. Sometimes, that might be necessary to convince other team members or your management that the performance analysis was worth the effort.

But in general, I recommend starting at the top and begin work on the most significant performance problem first. That will provide you with the biggest performance improvement, and you might not need to fix more than a few of these issues to fulfill your performance requirements.

Enough about general performance tuning tips. Let’s take a closer look at some Java-specific ones.

##### 5. Use StringBuilder to Concatenate Strings Programmatically

There are lots of different options to concatenate Strings in Java. You can, for example, use a simple + or +=, the good old StringBuffer or a[StringBuilder](https://docs.oracle.com/javase/9/docs/api/java/lang/StringBuilder.html).

So, which approach should you prefer?

The answer depends on the code that concatenates the String. If you’re programmatically adding new content to your String, e.g., in a for-loop, you should use the StringBuilder. It’s easy to use and provides better performance than StringBuffer. But please keep in mind, that the StringBuilder, in contrast to StringBuffer, is not thread-safe and might not be a good fit for all use cases.

You just need to instantiate a new StringBuilder and call the append method to add a new part to the String. And when you’ve added all parts, you can call the toString() method to retrieve the concatenated String.

The following code snippet shows a simple example. During each iteration, this loop converts iinto a String and adds it together with a space to the StringBuilder sb. So, in the end, this code writes “This is a test0 1 2 3 4 5 6 7 8 9” to the log file.

StringBuilder sb = new StringBuilder(“This is a test”);for (int i=0; i<10; i++) {

    sb.append(i);

    sb.append(” “);

}

log.info(sb.toString());

As you can see in the code snippet, you can provide the first element of your String to the constructor method. That will create a new StringBuilder containing the provided String and a capacity for 16 additional characters. When you add more characters to the StringBuilder, your JVM will dynamically increase the size of the StringBuilder.

If you already know how many characters your String will contain, you can provide that number to different constructor method to instantiate a StringBuilder with the defined capacity. That improves its efficiency even further because it doesn’t need to dynamical extend its capacity.

##### 6. Use + to Concatenate Strings in in One Statement

When you implemented your first application in Java, someone probably told you that you shouldn’t concatenate Strings with +. And that’s correct if you’re concatenating Strings in your application logic. Strings are immutable, and the result of each String concatenation is stored in a new String object. That requires additional memory and slows down your application, especially if you’re concatenating multiple Strings within a loop.

In these cases, you should follow tip number 5 and use a StringBuilder.

But that’s not the case if you’re just breaking a String into multiple lines to improve the readability of your code.

Query q = em.createQuery(“SELECT a.id, a.firstName, a.lastName ”

+ “FROM Author a ”

+ “WHERE a.id = :id”);

In these situations, you should concatenate your Strings with a simple +. Your Java compiler will optimize this and perform the concatenation at compile time. So, at runtime, your code will just use 1 String, and no concatenation will be required

##### 7. Use Primitives Where Possible

Another quick and easy way to avoid any overhead and improve the performance of your application is to use primitive types instead of their wrapper classes. So, it’s better to use an intinstead of an Integer, or a double instead of a Double. That allows your [JVM](https://stackify.com/jvm-metrics/) to[store the value in the stack instead of the heap](https://www.javaworld.com/article/2150208/java-language/a-case-for-keeping-primitives-in-java.html) to reduce memory consumption and overall handle it more efficiently.

##### 8. Try to Avoid BigInteger and BigDecimal

As we’re already talking about data types, we should also take a quick look at[BigInteger](https://docs.oracle.com/javase/9/docs/api/java/math/BigInteger.html)and[BigDecimal](https://docs.oracle.com/javase/9/docs/api/java/math/BigDecimal.html). Especially the latter one is popular because of its precision. But that comes at a price.

BigInteger and BigDecimal require much more memory than a simple long or double and slow down all calculations dramatically. So, better think twice if you need the additional precision, or if your numbers will exceed the range of a long. This might be the only thing you need to change to fix your performance problems, especially if you’re implementing a mathematical algorithm.

9. Check the Current Log Level First

This recommendation should be obvious, but unfortunately, you can find lots of code that ignores it. Before you create a debug message, you should always check the current log level first. Otherwise, you might create a String with your[log message](https://stackify.com/log-management/) that will be ignored afterward.

Here are two examples of how you should NOT do it.

// don’t do thislog.debug(“User [” + userName + “] called method X with [” + i + “]”);// or thislog.debug(String.format(“User [%s] called method X with [%d]”, userName, i));

In both cases, you will perform all required steps to create the log message without knowing if your logging framework will use the log message. It’s better to check the current log level first before you create the debug message.

// do thisif (log.isDebugEnabled()) {

    log.debug(“User [” + userName + “] called method X with [” + i + “]”);

}

##### 10. Use Apache Commons StringUtils.Replace Instead of String.replace

In general, the String.replace method works fine and is pretty efficient, especially if you’re using Java 9. But if your application requires a lot of replace operations and you haven’t updated to the newest Java version, it still makes sense to check for faster and more efficient alternatives.

One candidate is[Apache Commons Lang’s](https://commons.apache.org/proper/commons-lang/)[StringUtils.replace](https://commons.apache.org/proper/commons-lang/javadocs/api-release/org/apache/commons/lang3/StringUtils.html#replace-java.lang.String-java.lang.String-java.lang.String-) method. As Lukas Eder described in[one of his recent blog posts](https://blog.jooq.org/2017/10/11/benchmarking-jdk-string-replace-vs-apache-commons-stringutils-replace/), it dramatically outperforms Java 8’s String.replace method.

And it just requires a minimal change. You need to add a [Maven](https://stackify.com/gradle-vs-maven/) dependency for the Apache’s Commons Lang project to your application pom.xml, and replace all calls of the String.replacemethod with the StringUtils.replace method.

// replace thistest.replace(“test”, “simple test”);// with thisStringUtils.replace(test, “test”, “simple test”);

##### 11. Cache Expensive Resources, Like Database Connections

Caching is a popular solution to avoid the repeated execution of expensive or frequently used code snippets. The general idea is simple: Reusing such resources is cheaper than creating a new one over and over again.

A typical example is caching database connections in a pool. The creation of a new connection takes time, which you can avoid if you reuse an existing connection.

You can also find other examples in the Java language itself. The valueOf method of the Integerclass, for example, caches the values between -128 and 127. You might say that the creation of a new Integer isn’t too expensive, but it’s used so often that the caching of the most used values provides a performance benefit.

But when you think about caching, please keep in mind that your caching implementation also creates an overhead. You need to spend additional memory to store the reusable resources, and you might need to manage your cache to make the resources accessible or to remove outdated ones.

So, before you start caching any resources, make sure that you use them often enough to outweigh the overhead of your cache implementation.

Summary

As you’ve seen, it sometimes doesn’t require a lot of work to improve the performance of your application. Most of the recommendations in this post just need a small additional effort to apply them to your code.

But as usual, the most important recommendations are language-independent:

Don’t optimize before you know it’s necessary

Use a profiler to find the real bottleneck

Work on the biggest bottleneck first