Summary of Common Java Basics Interview Questions (Part 2)

Guide Java Java Basics About 9962 words About 33 minutes

Object-oriented basics

⭐The difference between object-oriented and process-oriented

Procedural-Oriented Programming (POP) and Object-Oriented Programming (OOP) are two common programming paradigms. The main difference between the two lies in the different ways they solve problems:

- **Procedure-oriented programming (POP)** : Procedure-oriented programming breaks down the problem-solving process into individual methods and solves the problem by executing each method.
- **Object-oriented programming (OOP)** : Object-oriented programming first abstracts objects and then uses objects to execute methods to solve problems.

Compared with POP, programs developed using OOP generally have the following advantages:

- **Easy to maintain** : OOP programs are usually easier to maintain due to good str and encapsulation.
- **Easy to reuse** : Through inheritance and polymorphism, OOP design makes the code more reusable and facilitates the expansion of functions.

- **Easy to expand** : Modular design makes system expansion easier and more flexible.

POP programming is generally simpler and more direct, suitable for handling some simpler tasks.

The performance difference between POP and OOP depends mainly on their operating mechanisms, not just the programming paradigm itself. Therefore, simply comparing the performance of the two is a common misunderstanding (related issue: Process-oriented: Is process-oriented performance higher than object-oriented??).

Performance is not the only consideration when choosing a programming paradigm. Code maintainability, scalability, and development efficiency are equally important.

Modern programming languages basically support multiple programming paradigms and can be used for both procedural programming and object-oriented programming.

The following is an example of finding the area and circumference of a circle, which briefly shows two different solutions: object-oriented and process-oriented.

Object-oriented :

```java
public class Circle {
    //
    private double radius;

    //
    public Circle(double radius) {
        this.radius = radius;
    }

```

```
    //
    public double getArea() {
        return Math.PI * radius * radius;
    }

    //
    public double getPerimeter() {
        return 2 * Math.PI * radius;
    }

    public static void main(String[] args) {
        //             3
        Circle circle = new Circle(3.0);

        //
        System.out.println("          " + circle.getArea());
        System.out.println("          " + circle.getPerimeter());
    }
}
```

We define a `Circle` class to represent a circle, which contains the radius property of the circle and methods for calculating the area and circumference.

Process-oriented :

```java
public class Main {
    public static void main(String[] args) {
        //
        double radius = 3.0;

        //
        double area = Math.PI * radius * radius;
        double perimeter = 2 * Math.PI * radius;

        //
        System.out.println("          " + area);
        System.out.println("          " + perimeter);
    }
}
```

We directly define the radius of the circle and use this radius to directly calculate the area and circumference of the circle.

What operator is used to create an object? What is the difference between an object entity and an object reference?

The new operator creates an object instance (the object instance is in the heap memory), and the object reference points to the object instance (the object reference is stored in the stack memory).

- An object reference can point to 0 or 1 objects (a string can have no balloon tied to it or one balloon tied to it);
- An object can have n references pointing to it (a balloon can be tied with n strings).

⭐The difference between object equality and reference equality

- Object equality generally compares whether the contents stored in memory are equal.
- Reference equality generally compares whether the memory addresses they point to are equal.

Here is an example:

```java
String str1 = "hello";
String str2 = new String("hello");
String str3 = "hello";
//      ==
System.out.println(str1 == str2);
System.out.println(str1 == str3);
//      equals
System.out.println(str1.equals(str2));
System.out.println(str1.equals(str3));
```

Output:

```plain
false
true
true
true
```

From the output of the above code, we can see that:

- `str1` and `str2` are not equal, but `str1` and `str3` are equal. This is because `==` the operator compares string references for equality.
- `str1` The contents of , `str2` , `str3` and are all equal. This is because `equals` the method compares the contents of the strings. Even if the object references of these strings are different, as long as their contents are equal, they are considered equal.

If a class does not declare a constructor, will the program execute correctly?

The constructor is a special method whose main function is to complete the initialization of the object.

Even if a class doesn't declare a constructor, it can still be executed! This is because a class will have a default constructor with no parameters even if it doesn't declare a constructor. If we add a constructor to the class ourselves (regardless of whether it has parameters or not), Java will not add a default constructor with no parameters.

We use constructors all the time without realizing it, which is why we add parentheses after creating an object (because we are calling the no-argument constructor). If we overload a constructor with parameters, remember to also write out the no-argument constructor (whether we use it or not), because this can help us avoid pitfalls when creating objects.

What are the characteristics of the constructor? Can it be overridden?

The construction method has the following characteristics:

- **The name is the same as the class name** : The name of the constructor must be exactly the same as the class name.
- **No return value** : Constructors have no return type and cannot be `void` declared using .
- **Automatic execution** : When an object of a class is generated, the constructor is automatically executed without explicit call.

Constructors **cannot be overridden** , but **they can be overloaded** . Therefore, a class can have multiple constructors with different parameter lists to provide different obj initialization methods.

⭐Three major characteristics of object-oriented

Encapsulation

Encapsulation means hiding an object's state information (i.e., properties) within the object, preventing external objects from directly accessing the object's internal information. However, methods that can be accessed from the outside world can be provided to manipulate the properties. This is like how we can't see the internal components (i.e., properties) of an air conditioner hanging on the wall, but can control it using a remote control (methods). If the properties are not to be accessed by the outside world, there is no need to provide methods for them to access. However, if a class does not provide methods for external access, then the class is meaningless. This is like how if there is no air conditioner remote control, then we cannot control the air conditioning, and the air conditioning itself is meaningless (of course, there are many other methods, but this is just an example).

```java
public class Student {
    private int id;//id
    private String name;//name

    //    id
    public int getId() {
        return id;
    }

    //    id
    public void setId(int id) {
        this.id = id;
    }

    //    name
    public String getName() {
        return name;
    }
```

```
    //    name
    public void setName(String name) {
        this.name = name;
    }
}
```

inherit

3

1.
2.
3.

:

- /
- “ ”
- ⭐

-
-
-
- Java
- `public static final` `private` , `protected` , `public`
-
  - Java 8 `public abstract` Java 8 `default` `static` Java 9 `private`
  -

Java 8 `default` `static` `private`

Java 8 `default`

```java
public interface MyInterface {
    default void defaultMethod() {
        System.out.println("This is a default method.");
    }
}
```

Java 8 `static` `MyInterface.staticMethod()` `static`

```java
public interface MyInterface {
    static void staticMethod() {
        System.out.println("This is a static method in the interface.");
    }
}
```

Java 9 private private

```java
public interface MyInterface {
    // default
    default void defaultMethod() {
        commonMethod();
    }

    // static
    static void staticMethod() {
        commonMethod();
    }

    //  static  default
    private static void commonMethod() {
        System.out.println("This is a private method used internally.");
    }

    //  default
    private void instanceCommonMethod() {
        System.out.println("This is a private instance method used internally.");
    }
}
```

•

- Cloneable clone() clone() Object clone()

```java
public class Address implements Cloneable{
    private String name;
    //            Getter&Setter
    @Override
    public Address clone() {
        try {
            return (Address) super.clone();
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }
}

public class Person implements Cloneable {
    private Address address;
    //            Getter&Setter
    @Override
    public Person clone() {
        try {
            Person person = (Person) super.clone();
            return person;
        } catch (CloneNotSupportedException e) {
            throw new AssertionError();
        }
    }
}
```

```java
Person person1 = new Person(new Address("    "));
Person person1Copy = person1.clone();
// true
System.out.println(person1.getAddress() ==
person1Copy.getAddress());
```

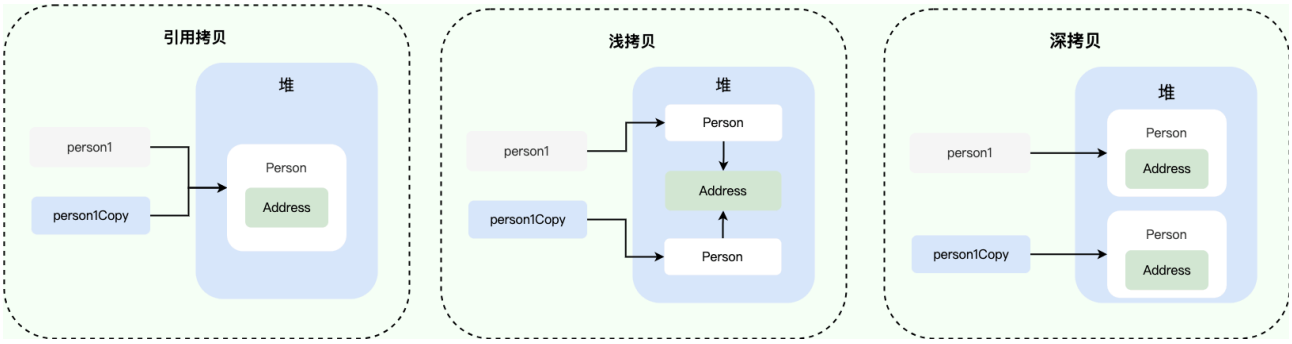
person1 person1
Address

Person clone() Person
Address

```java
@Override
public Person clone() {
    try {
        Person person = (Person) super.clone();
        person.setAddress(person.getAddress().clone());
        return person;
    } catch (CloneNotSupportedException e) {
        throw new AssertionError();
    }
}
```

```java
Person person1 = new Person(new Address("    "));
Person person1Copy = person1.clone();
// false
System.out.println(person1.getAddress() ==
person1Copy.getAddress());
```

person1 person1 Address



⭐Object

Object

Object 11

```java
/**
 * native Class final

 */
public final native Class<?> getClass()
/**
 * native JDK
HashMap
 */
public native int hashCode()
/**
 * 2 String

 */
public boolean equals(Object obj)
/**
 * native
 */
protected native Object clone() throws CloneNotSupportedException
/**
 * 16 Object

 */
public String toString()
/**
 * native (
```

```
            )
 */
public final native void notify()
/**
 * native                    notify

 */
public final native void notifyAll()
/**
 * native                                  sleep
wait              timeout
 */
public final native void wait(long timeout) throws
InterruptedException
/**
 *     nanos                                          0-999999
                     nanos
 */
public final void wait(long timeout, int nanos) throws
InterruptedException
/**
 *        2  wait
 */
public final void wait() throws InterruptedException
/**
 *
 */
protected void finalize() throws Throwable { }
```

== equals()

==

- ==
- ==

> Java ==

equals()

equals() Object Object equals()

Object equals()

```java
public boolean equals(Object obj) {
     return (this == obj);
}
```

equals()

- **equals()** equals() “==” Object equals()
- **equals()** equals() true( )

IDEA IDE == equals()

```java
String a = new String("ab"); // a
String b = new String("ab"); // b ,
String aa = "ab"; //
String bb = "ab"; //
System.out.println(aa == bb);// true
System.out.println(a == b);// false
System.out.println(a.equals(b));// true
System.out.println(42 == 42.0);// true
```

String equals Object equals String equals

String

String

String equals()

```java
public boolean equals(Object anObject) {
    if (this == anObject) {
        return true;
    }
    if (anObject instanceof String) {
        String anotherString = (String)anObject;
        int n = value.length;
        if (n == anotherString.value.length) {
```

```
            char v1[] = value;
            char v2[] = anotherString.value;
            int i = 0;
            while (n-- != 0) {
                if (v1[i] != v2[i])
                    return false;
                i++;
            }
            return true;
        }
    }
    return false;
}
```

hashCode()

hashCode() int

hashCode() JDK Object Java hashCode() Object hashCode() C C++

> ⚠️ **Oracle OpenJDK8** " Marsaglia's xor-shift ", " " " ", JDK/VM **Oracle OpenJDK8** ( ), VM : -XX:hashCode=4 :
>
> - https://hg.openjdk.org/jdk8u/jdk8u/hotspot/file/87ee5ee27509/src/share/vm/runtime/globals.hpp 1127
> - https://hg.openjdk.org/jdk8u/jdk8u/hotspot/file/87ee5ee27509/src/share/vm/runtime/synchronizer.cpp 537

```
public native int hashCode();
```

(key-value) “ ” “ ”

hashCode

“ HashSet ” hashCode

Java Head First Java :

> HashSet HashSet hashCode
> hashCode
> hashCode HashSet hashCode
> equals() hashCode
> HashSet
> equals

hashCode() equals()

JDK

HashMap HashSet hashCode()
HashSet

HashSet HashSet
hashCode equals()
hashCode

hashCode()

hashCode

hashCode

hashCode()

hashCode)

- hashCode
- hashCode equals() true
- hashCode

hashCode() equals()

equals() hashCode()

hashCode equals hashCode

equals() hashCode() equals hashCode

equals() hashCode() HashMap

- equals hashCode
- hashCode

hashCode() equals() Java hashCode() equals()

String

⭐String StringBuffer StringBuilder

String

StringBuilder StringBuffer AbstractStringBuilder AbstractStringBuilder final private AbstractStringBuilder append

```java
abstract class AbstractStringBuilder implements Appendable, CharSequence {
    char[] value;
    public AbstractStringBuilder append(String str) {
        if (str == null)
            return appendNull();
```

```
        int len = str.length();
        ensureCapacityInternal(count + len);
        str.getChars(0, len, value, count);
        count += len;
        return this;
    }
    //...
}
```

String
AbstractStringBuilder StringBuilder StringBuffer
expandCapacity append insert indexOf
StringBuffer
StringBuilder

String String
String StringBuffer StringBuffer
StringBuilder
StringBuffer 10%~15%

- : String
- : StringBuilder
- : StringBuffer

⭐String ?

String final ~~String~~

```java
public final class String implements java.io.Serializable,
Comparable<String>, CharSequence {
    private final char value[];
  //...
}
```

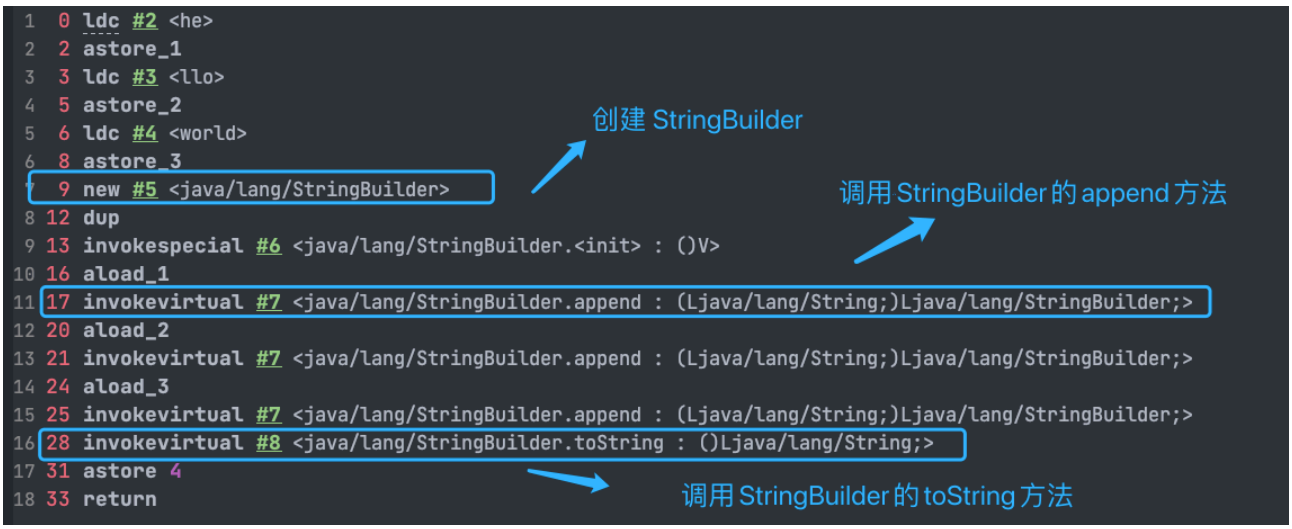
> 🐛 final
>
> final String final String
>
> 1. final String /
>
> 2. String final String
>
> String - issue 675 Java 9 String StringBuilder StringBuffer byte
>
> ```java
> public final class String implements java.io.Serializable,Comparable<String>, CharSequence {
>  // @Stable “ ”
>  @Stable
>  private final byte[] value;
> }
>
> abstract class AbstractStringBuilder implements Appendable, CharSequence {
>  byte[] value;
>
> }
> ```
>
> **Java 9 String char[] byte[] ?**
>
> String Latin-1 UTF-16
>
> Latin-1 Latin-1 Latin-1
>
> byte (8) char 2 16 byte char
>
> JDK Latin-1
>
> **Motivation**
>
> The current implementation of the `String` class stores characters in a `char` array, using two bytes (sixteen bits) for each character. Data gathered from many different applications indicates that strings are a major component of heap usage and, moreover that most `String` objects contain only Latin-1 characters. Such characters require only one byte of storage, hence half of the space in the internal `char` arrays of such `String` objects is going unused.

> Latin-1 `byte` `char`
>
> https://openjdk.java.net/jeps/254

⭐ “+” StringBuilder?

Java “+” “+=” String
Java

```java
String str1 = "he";
String str2 = "llo";
String str3 = "world";
String str4 = str1 + str2 + str3;
```



“+” `StringBuilder` `append()` `toString()` `String`

“+” **`StringBuilder`** **`StringBuilder`**

```java
String[] arr = {"he", "llo", "world"};
String s = "";
for (int i = 0; i < arr.length; i++) {
    s += arr[i];
}
System.out.println(s);
```

StringBuilder

StringBuilder

```
21 26 arraylength
22 27 istore 4
23 29 iconst_0
24 30 istore 5
25 32 iload 5
26 34 iload 4
27 36 if_icmpge 71 (+35)
28 39 aload_3
29 40 iload 5
30 42 aaload
31 43 astore 6
32 45 new #7 <java/lang/StringBuilder>
33 48 dup
34 49 invokespecial #8 <java/lang/StringBuilder.<init> : ()V>
35 52 aload_2
36 53 invokevirtual #9 <java/lang/StringBuilder.append : (Ljava/lang/String;)Ljava/lang/StringBuilder;>
37 56 aload 6
38 58 invokevirtual #9 <java/lang/StringBuilder.append : (Ljava/lang/String;)Ljava/lang/StringBuilder;>
39 61 invokevirtual #10 <java/lang/StringBuilder.toString : ()Ljava/lang/String;>
40 64 astore_2
41 65 iinc 5 by 1
42 68 goto 32 (-36)
43 71 getstatic #11 <java/lang/System.out : Ljava/io/PrintStream;>
44 74 aload_2
45 75 invokevirtual #12 <java/io/PrintStream.println : (Ljava/lang/String;)V>
46 78 return
```

循环内部创建 StringBuilder 对象

StringBuilder

```java
String[] arr = {"he", "llo", "world"};
StringBuilder s = new StringBuilder();
for (String value : arr) {
    s.append(value);
}
System.out.println(s);
```

```
20 new #6 <java/lang/StringBuilder>
23 dup
24 invokespecial #7 <java/lang/StringBuilder.<init> : ()V>
27 astore_2
28 aload_1
29 astore_3
30 aload_3
31 arraylength
32 istore 4
34 iconst_0
35 istore 5
37 iload 5
39 iload 4
41 if_icmpge 63 (+22)
44 aload_3
45 iload 5
47 aaload
48 astore 6
50 aload_2
51 aload 6
53 invokevirtual #8 <java/lang/StringBuilder.append : (Ljava/lang/String;)Ljava/lang/StringBuilder;>
56 pop
57 iinc 5 by 1
60 goto 37 (-23)
```

IDEA IDEA

JDK 9 “+” `makeConcatWithConstants()`

`a+b+c`

append StringBuilder

JDK9 JEP 280

StringBuilder issue#2442

String#equals() Object#equals()

`String` `equals` String

`Object` `equals`

⭐

JVM String

```java
//                    ”ab“
//          ”ab“      aa
String aa = "ab";
//                    ”ab“      bb
String bb = "ab";
```

```
System.out.println(aa==bb); // true
```

Java

⭐String s1 = new String("abc");

1 2

1. "abc" 2 ldc new String() "abc"
2. "abc" 1 new String() "abc"

1 “abc” "abc" "abc"

JDK 1.8

```java
String s1 = new String("abc");
```

```java
//                          String
// #2                          java/lang/String
//
java/lang/String
0 new #2 <java/lang/String>
//          String
//

3 dup
// JVM                          "abc"
//                  "abc"
//                  "abc"     JVM

//
4 ldc #3 <abc>
//                              "abc"            String
```

```
//     String                    "abc"

6 invokespecial #4 <java/lang/String.<init> :
(Ljava/lang/String;)V>
//         String
9 astore_1
//
10 return
```

ldc (load constant)

ldc

1. ldc

2. ldc

3. JVM

2 “abc” 1 “abc”

JDK 1.8

```java
//                “abc”
String s1 = "abc";
//                1          “abc”
String s2 = new String("abc");
```

```java
0 ldc #2 <abc>
2 astore_1
3 new #3 <java/lang/String>
6 dup
7 ldc #2 <abc>
9 invokespecial #4 <java/lang/String.<init> :
(Ljava/lang/String;)V>
12 astore_2
13 return
```

7 ldc
“abc” 0 ldc
“abc” 7 ldc
“abc”

String#intern ?

String.intern() native ( )

1. intern() String intern()
2. intern() intern()

- intern()
- intern()

JDK 1.8 :

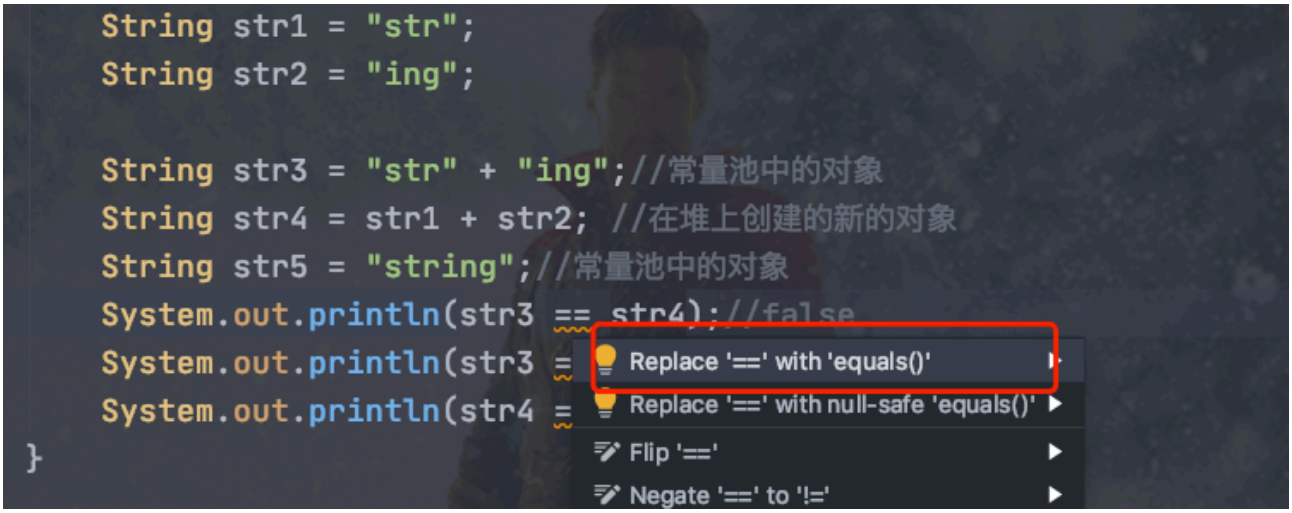
```java
// s1 "Java"
String s1 = "Java";
// s2 "Java" s1
String s2 = s1.intern();
// "Java" s3
String s3 = new String("Java");
// s4 "Java" s1
String s4 = s3.intern();
// s1 s2
System.out.println(s1 == s2); // true
// s3 s4
System.out.println(s3 == s4); // false
// s1 s4
System.out.println(s1 == s4); // true
```

String “+”

final JDK1.8

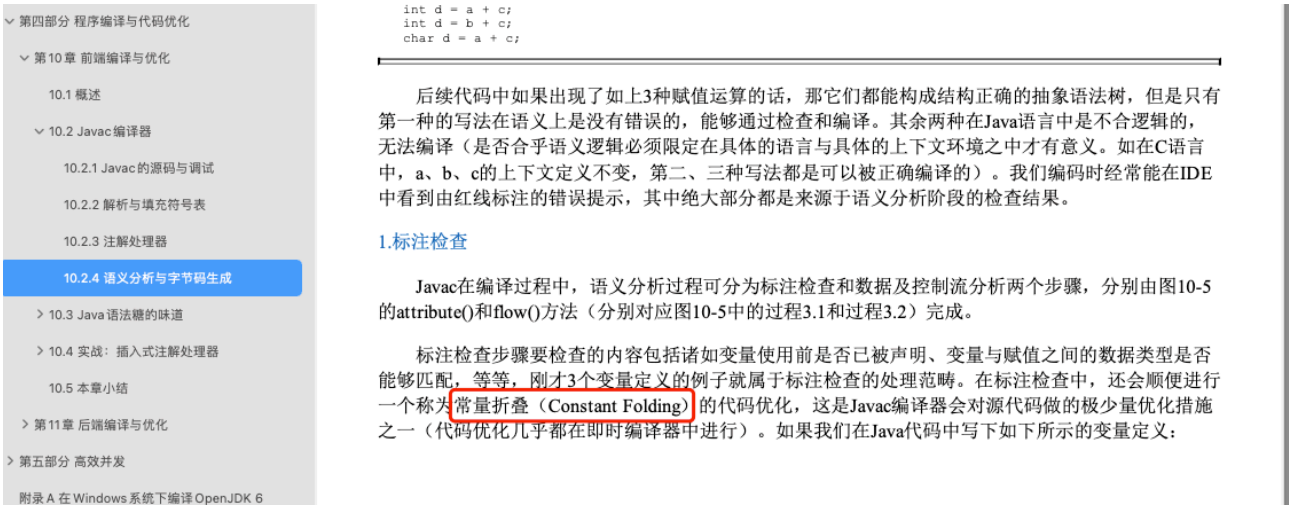
```java
String str1 = "str";
String str2 = "ing";
String str3 = "str" + "ing";
String str4 = str1 + str2;
String str5 = "string";
System.out.println(str3 == str4);//false
System.out.println(str3 == str5);//true
System.out.println(str4 == str5);//false
```

> String equals() String equals Object equals String equals == IDEA equals()



jvm

Javac **(Constant Folding)** Java



Javac ( ) `String str3 = "str" + "ing";` `String str3 = "string";`

- (`byte` `boolean` `short` `char` `int` `float` `long` `double`)
- `final`
- “+” `<<` `>>` `>>>`

“+” `StringBuilder` `append()` `toString()` `String`

```java
String str4 = new StringBuilder().append(str1).append(str2).toString();
```

`StringBuilder` `StringBuffer` `final`

```java
final String str1 = "str";
final String str2 = "ing";
//
String c = "str" + "ing";//
String d = str1 + str2; //
System.out.println(c == d);// true
```

`final` `String`

Sample code (`str2` its value can only be determined at runtime):

```java
final String str1 = "str";
final String str2 = getStr();
String c = "str" + "ing";//
String d = str1 + str2; //
System.out.println(c == d);// false
public static String getStr() {
      return "ing";
}
```

refer to

- In-depth understanding of String#intern: https://tech.meituan.com/2014/03/06/in-depth-understanding-string-intern.html
- Java String source code interpretation: http://keaper.cn/2020/09/08/java-string-mian-mian-guan/
- R Da (RednaxelaFX)'s answer about constant folding: https://www.zhihu.com/question/55976094/answer/147302764

Recently Updated2025/7/27 10:31

Contributors: guide , Itswag , Tianci.Xiong , Zail , liyanan , WT-AHA , zxf19930626 , Seeking Truth, Pursuing Honesty, Cultivating Harmony, and Managing Peace , Guide , Erzbir , Mr.Hope , cxrwy , Well Honey , Leisiya , Dongcp , ddouddle , sweetning0809 , qksuki , Barsit