

复旦大学计算机科学技术学院



编程方法与技术

4.1. 上次课复习

周扬帆

2021-2022第一学期

面向对象

□ 根据数据(对象)实现模块化

■ 数据的操作**封装**在数据对象内部

- 子过程（方法）**针对数据各司其职**
- 可读性更好
- 改了数据，要改子过程更方便（不难找）

■ 控制对象的数据访问权限

- public/private的变量和方法
- 私有数据不能在对象外访问 → 更鲁棒

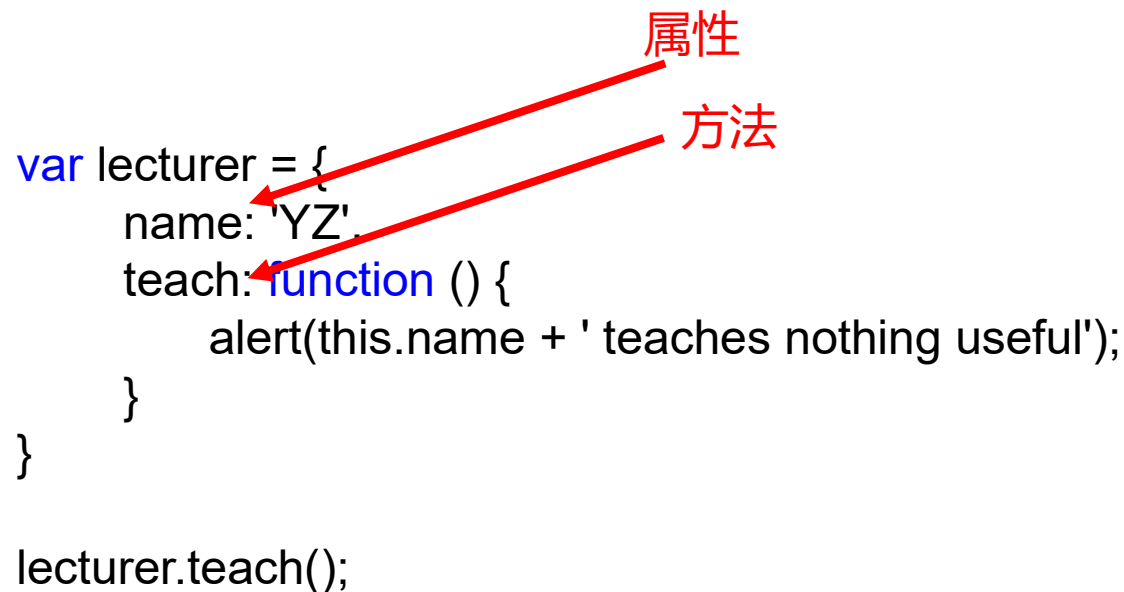
□ 程序：对象的相互调用过程

数据的保护

```
class Student {  
    private char name[] = new char[10];    //名字  
    public void setName(char [] studentName) {  
        for(int j = 0; j < name.length &&  
            j < studentName.length; j ++) {  
            //最多只取前10个  
            name[j] = studentName [j];  
        }  
    }  
    Student () {  
        //...  
    }  
};  
Student student = new Student();
```

JavaScript对象

- 对象由属性(properties)和方法(methods)两个基本元素构成



The diagram illustrates the components of a JavaScript object. Two red arrows originate from the right side of the slide. The top arrow, labeled '属性' (Property) in red, points to the 'name' property within the object literal. The bottom arrow, labeled '方法' (Method) in red, points to the 'function' keyword used to define the 'teach' method.

```
var lecturer = {  
  name: 'YZ',  
  teach: function () {  
    alert(this.name + ' teaches nothing useful');  
  }  
}  
  
lecturer.teach();
```

JavaScript对象

■ 对象的属性和方法可以动态增删

```
var lecturer = {  
  name: 'YZ',  
  teach: function () {  
    alert(this.name + ' teaches nothing useful');  
  }  
}
```

```
lecturer.teach();  
lecturer.title = 'Dr. ';  
lecturer.quit = function () {  
  alert(this.title + this.name + ' quits.');
```

```
  }  
lecturer.quit();  
  
delete lecturer.quit;  
lecturer.quit();
```

JavaScript对象的创建

- **new func(...)**就是以func为构造函数，构造了一个对象，并返回
 - 函数内部的this指向新构建的对象
- 例子

实例属性

实例方法

```
function Lecturer (name) {  
    this.name = name; //var name = name  
    alert(name + ' teaches nothing useful');  
    this.getName = function () {  
        return this.name;  
    }  
}  
var Y = new Lecturer ('Y');  
alert(Y.getName());  
  
var Z = new Lecturer ('Z');  
alert(Z.getName());
```

JavaScript闭包

- 闭包：内部函数及其定义时的上下文
 - 作用：外部作用域访问内部作用域中变量
- 用途之一：实现对象成员的访问控制

```
function generateCounter() {  
  var count = 0;  
  var innerCounter = function () {  
    ++count;  
    return count;  
  }  
  return innerCounter;  
}  
var counter = generateCounter();  
console.log(counter());
```

```
var Student = function () {  
  var name = 'default';  
  return {  
    getName: function () {  
      return name;  
    },  
    setName: function (newName) {  
      name = newName;  
    }  
  };  
};  
var student = Student();  
console.log(student.getName());
```

封装
getter/setter

Java项目的源代码组织

□ 多个源文件如何组织其层次结构

- 根据逻辑关系，按模块分目录，树状管理

```
src/.../module1/module1.1/filename1.java
                        | filename2.java
                        | ...
                        | module1.2/filename3.java
                        | filename4.java
                        | ...
                        | ...
module2/filename5.java
...
...
```

方便理解、方便分工 ...

Java项目的源代码组织

- 一个源文件可以包含多个类
- 每个文件只能包含一个**public**类
 - 整个项目可访问
 - 不指定public的类，只有包内部可访问
 - **方便JVM快速定位**需要的类

命名空间

- 类不能重名，否则JVM出错
- 保证不重名太麻烦 → 引入命名空间
 - 保证在一个命名空间内不重名
 - 不同命名空间可以重名
- package: Java命名空间
 - 给源文件指定package：第一条语句
 - `package pkg1[.pkg2[.pkg3...]]`
- 一般一个功能模块的源文件放在一个包

Java包(package)

- 指定了包名的源文件需**放在特定目录**
 - 如用package abc.de.fg指定了包名的文件
 - 放在 abc/de/fg/ 目录下
 - 方便JVM查找
- 不指定包名的文件，属于“无名包”
 - 只能放在源文件根目录下

使用别的源文件的类

□ 假如包p2的类B需要使用包p1里定义了类A

- 方法一: `p1.A a = new p1.A();`

- 方法二: **import语句**

`import pkg1[.pkg2[.pkg3...]].类名;`

```
import p1.A; //置于package语句后
...
A a = new A();
```

复旦大学计算机科学技术学院



编程方法与技术

4.2. 数据的存储

周扬帆

2021-2022第一学期

Java的数据类型

□ 基本数据类型

- short, int, long, double, float, byte, boolean, char

□ 类

- 变量
- 方法

□ 引用

Element a;

Element a = new Element();

a是创建的类的引用

```
class Element
{
    private int n;
    private Element next;
    public void setNum(int num) {
        n = num;
    }
    public void setNext(Element nextElement) {
        next = nextElement;
    }
    public int getNum() {
        return n;
    }
    public Element getNext() {
        return next;
    }
}
```

Java的内存管理

□ 方法中声明的

- 基本数据类型
- 对象的引用

□ 一般存在栈中

- 生命周期短
- 作用域外就释放
- 访问速度快

```
public int getSize() {  
    int ret = 0;  
    Element curElement = first;  
    if(curElement == null) {  
        return 0;  
    }  
    while(true) {  
        ret ++;  
        curElement = curElement.getNext();  
        if(curElement == null) {  
            return ret;  
        }  
    }  
}
```

Java的内存管理

□ 类的成员变量

- 基本数据类型
- 对象的引用

□ 一般存在堆中

- 生命周期: new开始分配, 不用了垃圾收集机制(GC)负责清理

```
public void add(int i) {  
    if(last == null){  
        first = new Element();  
        first.setNum(i);  
        last = first;  
    }  
    else  
    {  
        Element newLast = new Element();  
        newLast.setNum(i);  
        last.setNext(newLast);  
        last = newLast;  
    }  
}
```

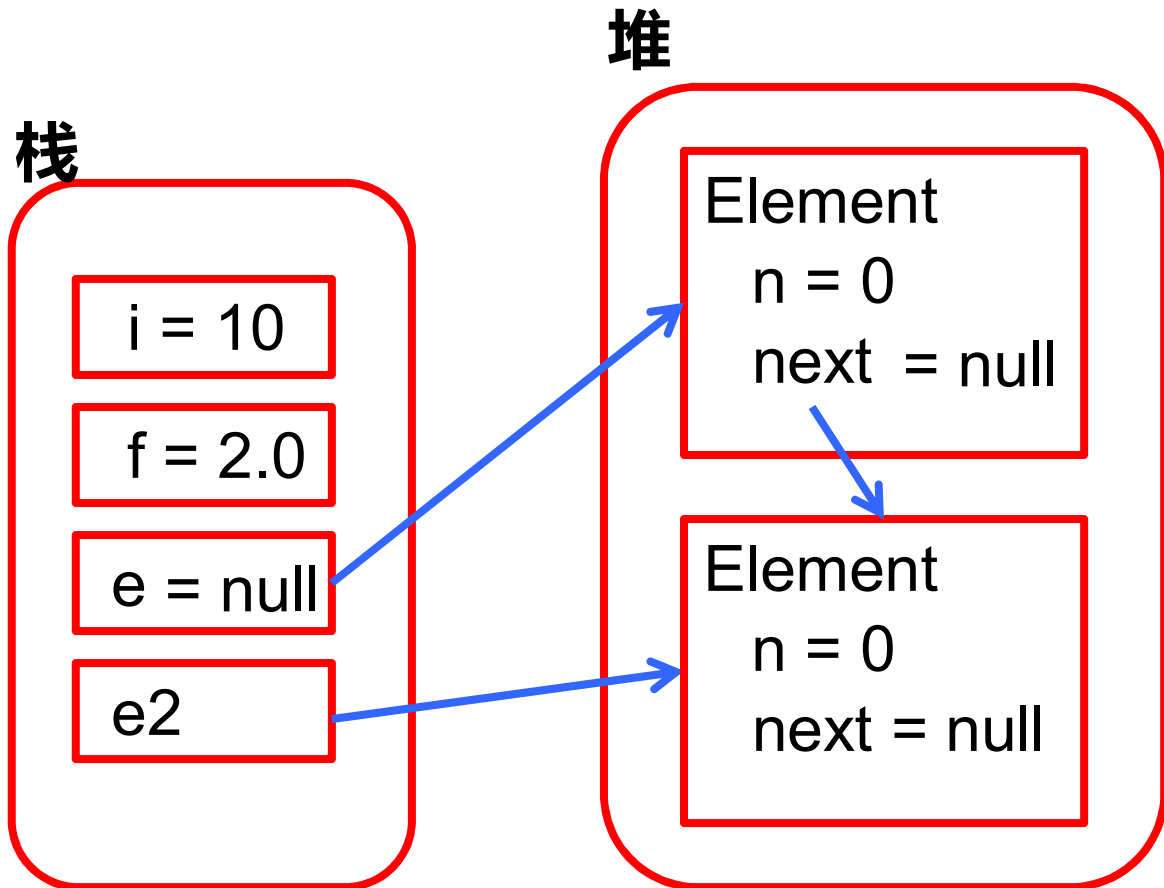
```
class Element  
{  
    private int n;  
    private Element next;  
}
```

堆/栈? 什么硬件? 为什么有所谓快慢

Java的内存管理

```
void example() {  
    int i = 10;  
    float f = 2.0;  
    Element e;  
    e = new Element();  
    Element e2 =  
        new Element();  
    e.setNext(e2);  
}
```

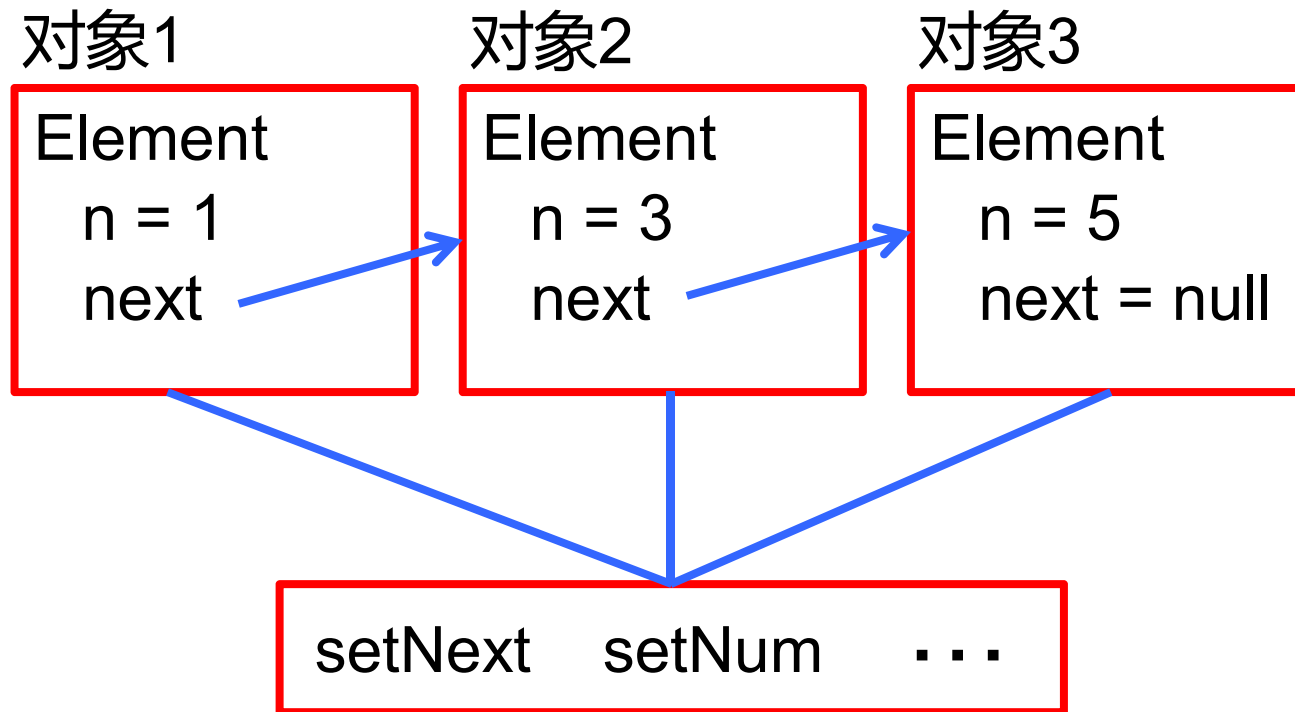
```
class Element  
{  
    private int n;  
    private Element next;  
    public void setNext(Element nextElement) {  
        next = nextElement;  
    }  
}
```



Java的内存管理

□ 类的方法的实现（代码）

- 对于类的任何对象，方法的逻辑都是一样的
- 因此，方法只需要放一份



static变量

- 类的变量一般通过类的实例化（创建对象）来分配空间

```
Element e;  
e = new Element();
```

回忆构造函数

- 有些变量是本类所有对象共用的

- 如

```
public class Minion {  
    static int color = YELLOW;  
}
```

- 希望在对象中共享一个公共变量

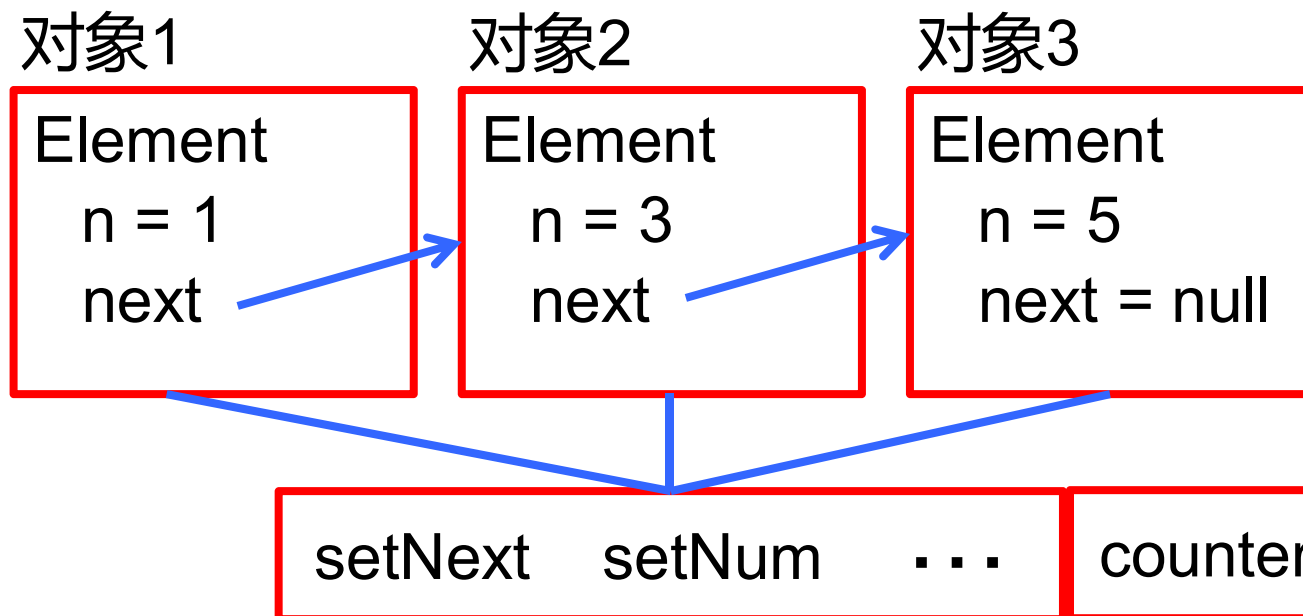


static变量

□ 类的static数据 (ugly?)

- 节约空间，只存一份
- 访问快速方便，不需创建对象
- 方便对象共享

```
class Element
{
    private static int counter = 0;
    private int n;
    private Element next;
    Element() {
        counter ++;
    }
}
```



static方法

- 类的方法一般通过类的实例化对象来调用

```
Element e;  
e = new Element();  
Element e2 = new Element();  
e.setNext(e2);
```

- 有些方法和**对象**的实例化数据没有关系

- 如

```
void printUsage() {  
    System.out.println("usage: cat filename");  
}
```

static方法

- 只处理static数据
- 因此没必要通过类的实例化的对象来调用
- static方法
 - 只可以访问类的static数据
 - 调用：类名.方法名
 - 调用方法区别于普通方法： 对象名.方法名
 - `public static void main(String args[])`

final 修饰的类的变量

□ 有些数据不希望在对象创建后被修改

- 减少代码错误：其他程序员可能错误修改了它

```
class Student {  
    private final String description; //学号  
    private final String department; //系  
    ...  
}
```

- 不希望在后续操作中被修改

□ final 变量须在分配内存空间时赋值

- （在哪里赋值？）
- 不能在除了**构造方法**之外的方法中赋值

final 修饰的引用

- final 修饰**对象的引用**，该引用只能指向一个对象，不能再修改

```
class Student {  
    private final int SID; //学号  
    private final int department; //系  
    private final GPAList GPA; //成绩单  
}
```

- GPA 指向一个 GPAList 对象

- 注意：指向不能改，但是被指向的对象的数据可以改

```
GPA.setValue("Java Language", 1.0);
```


final/static

□ final static 用于修饰常量

- **final static double** PI =
3.1415926535897932;

- 必须在对象创建前时赋值？可以在构造函数赋值吗？

 - 因为是static的，会在对象创建前分配空间

 - 因为是final的，必须在分配空间的时候赋值

Java的类成员数据：小结

- 类的成员数据有两类：基本数据类型，对象的引用
- 作用域修饰
 - public: 全局可访问
 - private: 仅自己可访问
 - 无: 包(package)内可访问
 - protected: 子类可访问(以后讲)
- final: 必须在分配内存空间的时候赋值
- static: 全局，所有类的对象共享一份，在对象创建前分配内存空间

思考

- ❑ 消化JVM关于内存管理的设计
- ❑ 理解final和static修饰的变量内存空间的分配

复旦大学计算机科学技术学院



编程方法与技术

4.3. 字符串类

周扬帆

2021-2022第一学期

字符串

□ 之前的例子

```
class Student
{
    char name[] = new char[10];
    double gpa;
    int studentID;
    void setName(char [] studentName)
    {
        for(int j = 0; j < name.length && j < studentName.length; j++)
        { //只取前10个
            name[j] = studentName[j];
        }
    }
    boolean compareSID(Student s2)
    {
        return studentID > s2.studentID;
    }
    boolean compareGPA(Student s2)
    {
        return gpa > s2.gpa;
    }
};
```

声明并创建字符数组来存字符串

很麻烦地进行赋值

字符串

- **字符串操作是非常常用普遍的操作**
 - 应实现复用：字符串常用操作应实现为库 (library)
 - **回忆面向对象的思想**
 - 数据：字符串
 - 方法：字符串的常用操作
- } 封装为字符串类
String

字符串

□ 构造方法

- 怎么构造一个String类用于存储字符串

□ 常用的操作

- 一些常用的public方法

□ Java对String的特殊处理

String类常用构造方法

```
byte [] ascii={97,98,99,100,101};  
char [] chars = {'a','b','c','d','e'};  
String a      = new String(String b); //拷贝字符串b  
              = new String(ascii);   //ascii码数组  
              = new String(ascii, 1, 2);  
                //从ascii索引为1的元素开始, 长度为2  
              = new String(chars);   //字符数组  
              = new String(chars, 1, 3);  
                //从chars索引为1的元素开始, 长度为3  
              = "abcde" ;
```


String类的数据

- `private final char [] value;`
 - 存储字符串
- String设计为构造之后，字符串**不能更改**
 - `String a = new String();` //没有意义
- 原因？
 - 可以实现字符串池，复用空间
 - 效率：拷贝的时候只需要复制引用
 - 安全：多线程安全
 - Hash值固定，不用重算
 - 便于索引查找
 - 便于内容比较

String类的常用方法

取字符

`char charAt(int index)`

取子串

`String subString(int beginIndex)`

`String subString(int beginIndex, int endIndex);`

获取字符串长度

`int length()`

String类的常用方法

字符串中定位

`int indexOf(int ch)`

`int indexOf(int ch, int fromIndex)`

`int indexOf(String str)`

`int indexOf(String str, int fromIndex)`

连接两个字符串

`String concat(String str)`

String类的常用方法

字符串大小写转换

String toUpperCase()

String toLowerCase()

判断字符串后缀内容

boolean endsWith(String suffix)

String类的常用方法

字符串比较

`boolean compareTo(String anotherString)`

`boolean compareToIgnoreCase(String str)`

判断字符串对象是否相等

`boolean equals(Object anObject)`

`boolean equalsCase(String anotherString)`

String类的常用方法

字符串到字符数组的转换

```
getChars(int srcBegin, int srcEnd,  
         char[] dst, int dstBegin)
```

字符数组到字符串之间的转化

```
String copyValueOf(char[] data)  
String copyValueOf(char[] data,  
                   int offset, int count)
```

字符串

- ❑ **String**创建完不能更改
- ❑ **Java**提供另外可以动态更改内容的字符串类
 - **StringBuffer/StringBuilder**
- ❑ **StringBuffer/StringBuilding**常用构造方法
 - StringBuffer/StringBuilder()
 - StringBuffer/StringBuilder(int length)
 - StringBuffer/StringBuilder(String str)

常用方法

添加操作

StringBuffer/StringBuilder append(boolean b)
StringBuffer/StringBuilder append(char c)
StringBuffer/StringBuilder append(char[] str)
StringBuffer/StringBuilder append(char[] str, int offset, int len)
StringBuffer/StringBuilder append(double d)
StringBuffer/StringBuilder append(float f)
StringBuffer/StringBuilder append(int I)
StringBuffer/StringBuilder append(long l)
StringBuffer/StringBuilder append(Object obj)
StringBuffer/StringBuilder append(String str)

StringBuffer类的常用方法

插入操作

StringBuffer/StringBuilder insert(int offset, Boolean b)
StringBuffer/StringBuilder insert()
StringBuffer/StringBuilder insert(int offset, char[] str)
StringBuffer/StringBuilder insert(int offset, char[] str,
int offset, int len)
StringBuffer/StringBuilder insert(int offset, double d)
StringBuffer/StringBuilder insert(int offset, float f)
StringBuffer/StringBuilder insert(int offset, int i)
StringBuffer/StringBuilder insert(int offset, long l)
StringBuffer/StringBuilder insert(int offset, Object obj)
StringBuffer/StringBuilder insert(int offset, String str)

常用方法

其他操作

取子串

String subString(int start, int end)

String subString(int start)

取字符

char charAt(int index)

常用方法

StringBuffer其他操作

删除字符

StringBuffer/StringBuilder delete(int start, int end)

StringBuffer/StringBuilder deleteCharAt(int index)

StringBuffer类到String类的转换

String toString()

内容替换

StringBuffer/StringBuilder replace(int start, int end, String str)

常用方法

其他操作

字符串反转

String reverse()

获取字符串缓冲区剩余的长度

int capacity()

获取字符串缓冲区的长度

int length()

字符串类

□ String

- 不可变
- 修改会创建新对象

□ StringBuilder

- 可变
- 线程不安全、快

□ StringBuffer

- 可变
- 线程安全、慢



加不加锁的区别 (?)

复旦大学计算机科学技术学院



编程方法与技术

4.4. String的存储

周扬帆

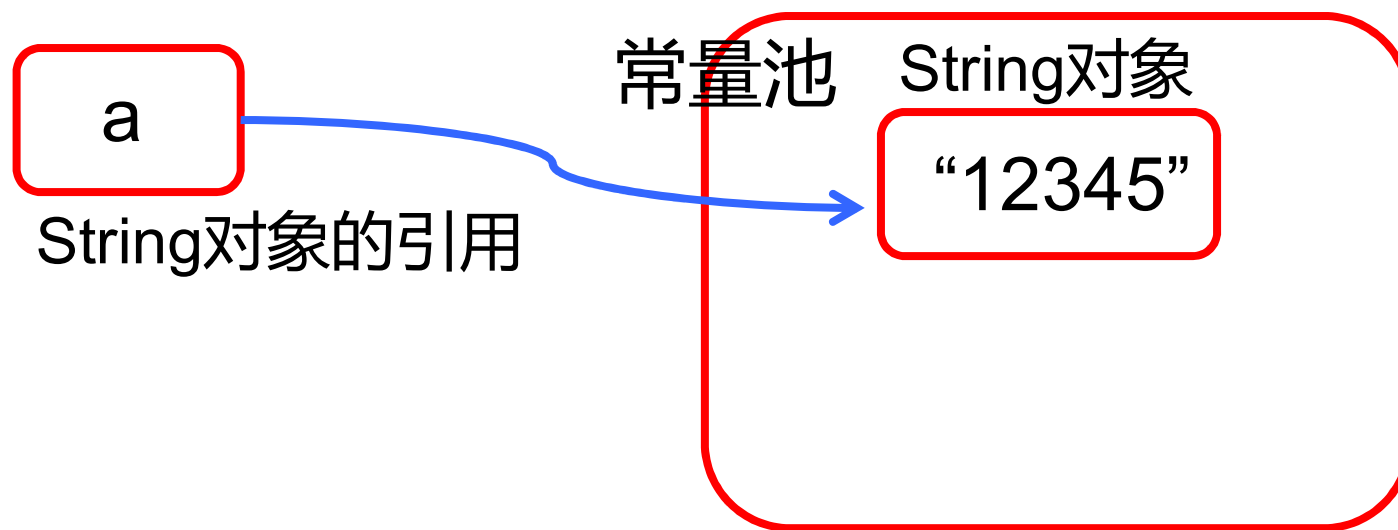
2021-2022第一学期

String对象和字符串常量

□ Java将字符串常量实现为String对象

String a = "12345";

- 编译时创建字符串
- 将引用赋值给a

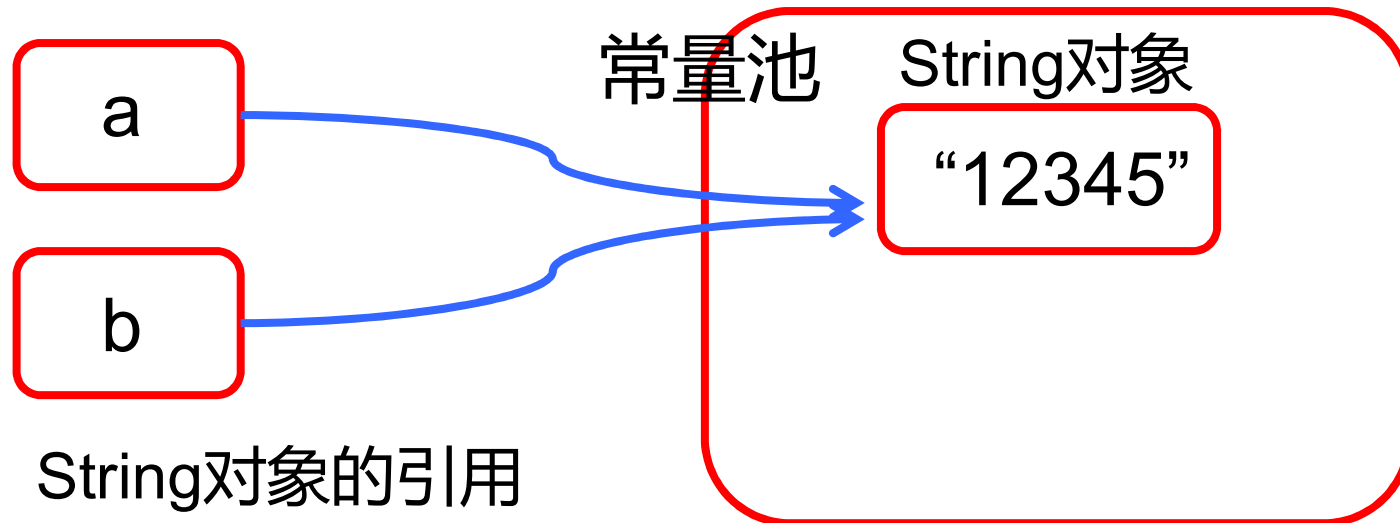


String对象

□ 引用的赋值

```
String a = "12345";  
String b = a;
```

- 创建内容为 12345的String对象，引用赋值a
- 将该引用赋值给b

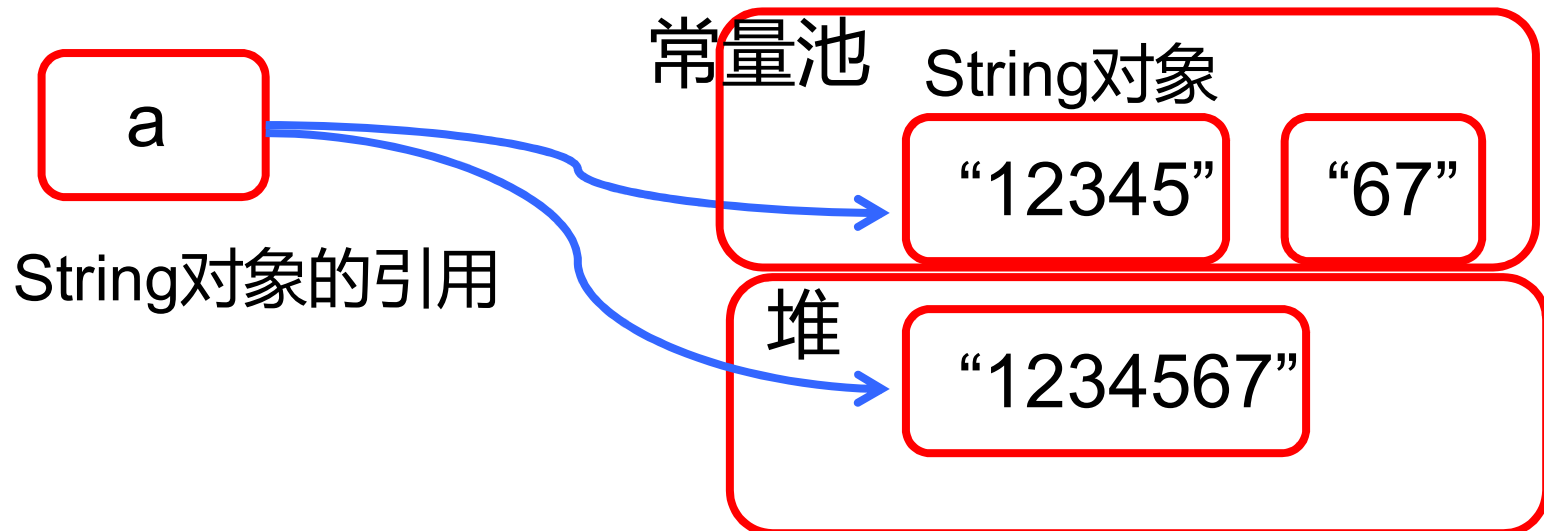


String对象

□ 字符串操作

```
String a = "12345";  
a = a.concat("67");
```

- 创建内容为 12345的String对象1, 67的对象2
- 对象1的引用赋值a
- 创建内容为 1234567的String对象, 引用赋值a

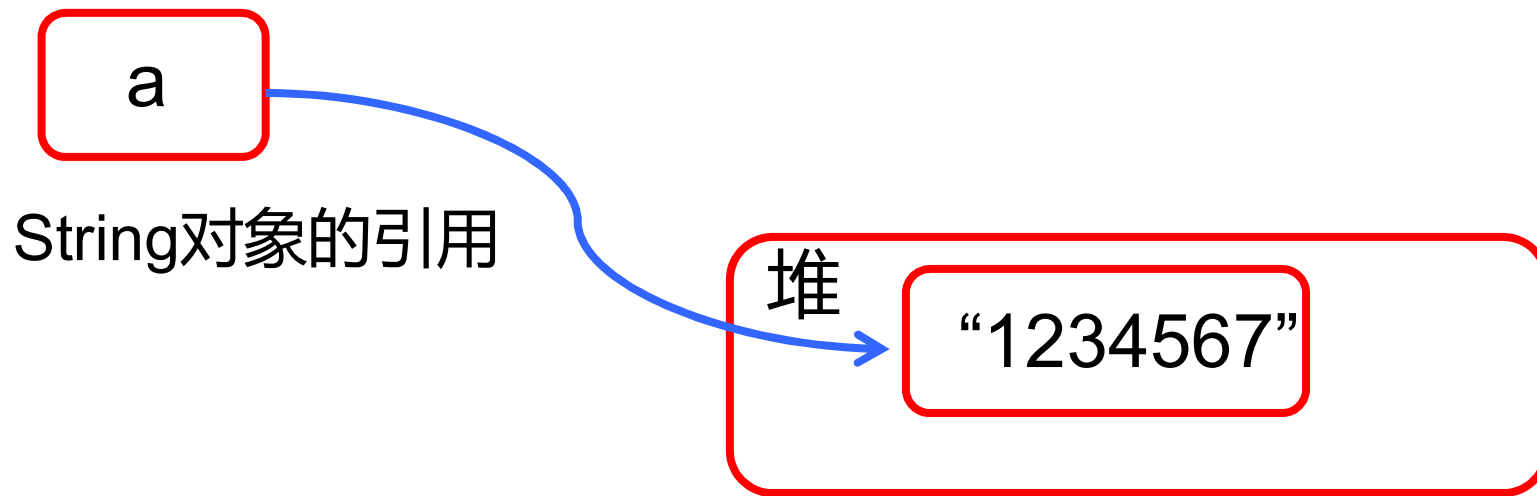


String对象

□ 字符串操作

```
String a = new String("1234567");
```

- 创建内容为 1234567的String对象，引用赋值a



String对象的+

□ Java不支持操作符重载

- 避免代码可读性差

□ 只实现了String类的 + 重载

```
String a = "12345";  
a = a + "67";
```

- 类似于concat
- 可以+各种基本变量，甚至各种对象(?)

```
int i = 67;  
a = a + i;  
把i变成一个“67”字串，再合并
```

```
System.out.println("Result = " + i);
```

String对象的+

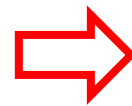
□ String类的 +

```
String a = "12345";  
a = a + "67";
```

```
a = (new StringBuilder(a)).append("67").toString();
```

■ 注意潜在的开销

```
for(1m次) {  
    String X = Y + Z;  
}
```



```
for(1m次) {  
    StringBuilder.append  
}
```

```
a = "12345" + "67"; ??
```

字符串类

java中的方法的参数是基本类型的时候是按值传递的，引用类型是按引用传递的，

但是对于string来说，

究竟是怎么传递的？

string也是引用类型，但是为什么还有传递参数的时候是传值的，

而似乎StringBuffer是按引用传递的

这是为什么？？？

```
public class Test
{
    public static void test(String str)
    {
        str = "world";
    }
    public static void main(String[] args)
    {
        String str1 = new String("hello");
        test(str1);
        System.out.println(str1); //str1的值并没有改变，说明在test(str1)中是传值的，
    }
}
```

```
public class Test
{
    public static void test(StringBuffer str)
    {
        str.append(" world");
    }
    public static void main(String[] args)
    {
        StringBuffer str1 = new StringBuffer("hello");
        test(str1);
        System.out.println(str1); //但是这里的str1却改变了，成了hello world，是不是说这里的
        //传递的是对象的引用？？？
    }
}
```

String对象

```
String a = "a";  
String b = "b";  
String c = "a";  
String ab1 = a + b;  
String ab2 = "ab";  
String ab3 = a.concat(b);  
String ab4 = a + b;  
String ab5 = "a" + "b";  
System.out.println(a == c);           true  
System.out.println(ab1 == ab2);        false  
System.out.println(ab1 == ab3);        false  
System.out.println(ab1 == ab4);        false  
System.out.println(ab1 == ab5);        false  
System.out.println(ab2 == ab3);        false  
System.out.println(ab2 == ab4);        false  
System.out.println(ab2 == ab5);        true  
System.out.println(ab3 == ab4);        false  
System.out.println(ab3 == ab5);        false  
System.out.println(ab4 == ab5);        false
```

String的intern方法

□ String可以创建在常量池或者堆上

```
String a = "abcd";  
String a = new String("abcd");
```

□ intern方法

- 如果常量池中不存在这个字串，将之加入常量池(JDK1.6)或将引用加入常量池(JDK1.7+)，返回常量池中该字串引用
- 如果常量池中存在这个字串，返回常量池中该字串引用

```
String a = "abcd";  
String b = new String("abcd");  
String c = (new String("abcd")).intern();  
System.out.println(a == b); //b == c ?, a == c ?
```

- 好处？ 坏处？

String的intern方法

□ intern方法

- 如果常量池中不存在这个字串，将之加入常量池(JDK1.6)或将引用加入常量池(JDK1.7+)，返回常量池中该字串引用

→ JDK1.7+, 常量池存引用, JVM偷懒不值拷贝

→ 好处?

```
String s1 = new String("1") + new String("1");  
System.out.println(s1.intern() == s1);           true  
String s2 = "11";  
System.out.println(s2 == s1);                     true
```

```
String s1 = new String("1") + new String("1");  
String s2 = "11";  
System.out.println(s2 == s1.intern());           true  
System.out.println(s2 == s1);                     false
```


String对象的intern

```
public static void main(String[] args) {  
    String s1 = "AB";  
    String s2 = new String("AB");  
    String s3 = "A";  
    String s4 = "B";  
    String s5 = "A" + "B";  
    String s6 = s3 + s4;  
    System.out.println(s1 == s2);  
    System.out.println(s1 == s5);  
    System.out.println(s1 == s6);  
    System.out.println(s1 == s6.intern());  
    System.out.println(s2 == s2.intern());  
}
```

String对象的intern

```
public static void main(String[] args) {  
    String s1 = "AB";  
    String s2 = new String("AB");  
    String s3 = "A";  
    String s4 = "B";  
    String s5 = "A" + "B";  
    String s6 = s3 + s4;  
    System.out.println(s1 == s2);  
    System.out.println(s1 == s5);  
    System.out.println(s1 == s6);  
    System.out.println(s1 == s6.intern());  
    System.out.println(s2 == s2.intern());  
}
```

String对象

各种初(wu)级(liao)面试题

- (1) 现在当有人问 `String str = new String("abc");`创建了几个对象, 常量池有abc字段是1个, 常量池没有"abc"字段则是2个。
- (2) `String str="abc";`创建了几个对象 (如果常量池里面已经有对象了就是0个。如果没有就是1个) ;
- (3) `new String("abc").intern();`创建了几个对象 (如果常量池里面已经有该字符串对象了就是1个, 如果没有就是两个)

```
String b = new String("ab" + "cd");  
String c = b.intern();  
System.out.println(c == b);
```

false

```
String d = "";  
String b = new String("ab" + "cd");  
String a = "a" + d + "b";  
System.out.println(a == a.intern());
```

true

```
String a = "";  
String b = new String("ab" + a + "cd");  
String c = b.intern();  
System.out.println(c == b);
```

true

```
String d = "";  
String b = new String("ab" + d + "cd");  
String a = "a" + d + "b";  
System.out.println(a == a.intern());
```

false

JavaScript的字符串

□ 引擎(Runtime)决定, 基本一样

□ V8

- 常量字符串一定分配在data space数据区空间

```
var a = "hello";  
// a is Internalized  
  
var b = a;  
// b is Internalized  
same pointer, assignment  
  
var c = a + "world"; // #world is Internalized  
// c is not Internalized  
  
function subject() {  
    return "world"; // #world is Internalized  
};  
var d = a + subject();  
// d is not Internalized  
same pointer, internalized
```

- 字符串被设置为对象属性名时会被尝试改造为常量化版本

→ var obj = {}; obj[c] = 1;

//obj.helloworld === 1

//Object.keys(obj)[0] === "helloworld"

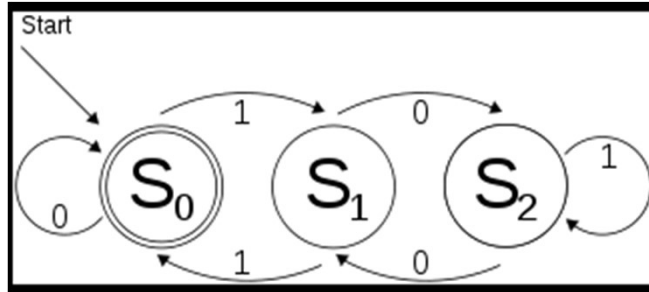
StringTokenizer类

- StringTokenizer类用于字符串的词法分析
- String的split方法
 - 正则表达式?

字符串分析

□ 正则表达式?

- 有限状态机



- <https://www.runoob.com/regexp/regexp-tutorial.html>

□ Java

- java.util.regex包, Pattern和Matcher类
- <https://www.runoob.com/java/java-regular-expressions.html>

□ JavaScript

- `var a = /[1-9][0-9]*/; String b = '12'; b.match(a);`

思考

- ❑ 理解String、StringBuffer、StringBuilder的优缺点，思考它们的应用场景
- ❑ 理解String的存储方式和intern

复旦大学计算机科学技术学院



编程方法与技术

4.5. 课堂练习: String

周扬帆

2021-2022第一学期

课堂练习目的

- **熟悉面向对象的思维**
 - 面向字符串，开发相关的操作
- **熟悉java基本语法**
- **练习自己设计测试用例**
- **练习测试程序性能的简单方法**

课堂练习

□ 实现自己的字符串类MyString

- 用私有成员变量final char [] value存储字符串

- 简单方法的实现

 - MyString(char [] v): 将 v 的内容复制到成员 value

 - length(): 返回字符串的长度

 - getValue(): 返回一个新的数组，拷贝字符串出来

 - concat(char [] v): 把value和v连起来，形成新的字符串

- 不太简单的方法的实现

 - indexOf(char [] v): 找子串，返回第一次出现时第一个元素的index

 - replace(char [] v1, char [] v2): 将value里出现v1的所有部分替换为v2

课堂练习

□ 和String比效率（下面是简单例子）

```
long startTime;  
long endTime;  
MyString a;  
startTime = System.currentTimeMillis(); //获取当前时间  
for(int i = 0; i < TIMES; i++) { //TIMES为常量  
    a = new MyString (chars); //chars为char数组  
    a.indexOf(target); //target为char数组  
}  
endTime = System.currentTimeMillis();  
System.out.println("MyString: "+ (endTime - startTime) + "ms");
```

课堂练习

□ 提交

- MyString代码
- 性能测试代码 (MyStringTest) 、设计测试用例测试 replace和indexOf效率与String区别
- 测试报告 (一页纸以内)

□ 评分

- MyString的代码正确性
- 性能测试的实现
 - 正确性：看代码，说服力：看那一页纸
- MyString性能的高低不作为评分标准
 - 再差的算法都不扣分，只要实现了功能

□ ddl: Oct-20-23:59 (下周三晚)