

复旦大学计算机科学技术学院



编程方法与技术

C.6. JavaScript的let/const

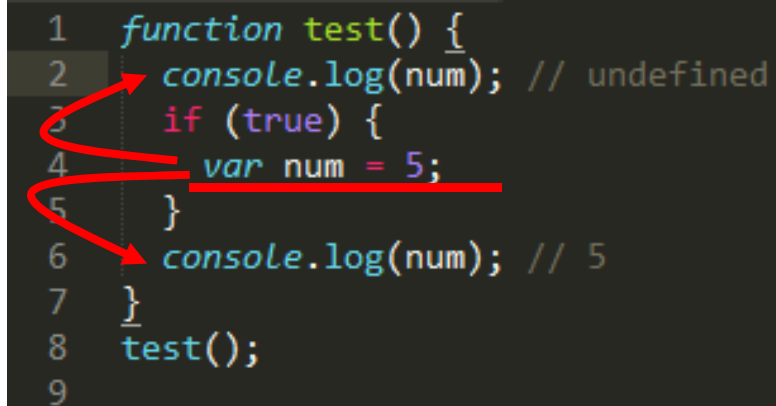
周扬帆

2021-2022第一学期

let和const

□ var存在变量提升

```
1  function test() {  
2    console.log(num); // undefined  
3    if (true) {  
4      var num = 5;  
5    }  
6    console.log(num); // 5  
7  }  
8  test();  
9
```



let关键字

□ 声明块内作用域

- 不存在变量提升

```
1 function test() {  
2   if (true) {  
3     let num = 5;  
4   }  
5   console.log(num); // ReferenceError: num is not defined  
6 }  
7 test();  
8
```

□ 不能重复声明

```
1 function test() {  
2   let num = 1;  
3   let num = 5; // SyntaxError: Identifier 'num' has already been declared  
4 }  
5 test();  
6
```

const关键字

- ❑ 声明常量
- ❑ 块级作用域
- ❑ 不存在变量提升
- ❑ 不能重复声明

```
1  const PI = 3.1415;  
2  PI // 3.1415  
3  
4  PI = 3;  
5  // TypeError: Assignment to constant variable.
```

复旦大学计算机科学技术学院



编程方法与技术

C.7. 编程语言与人机交互

周扬帆

2021-2022第一学期

编程语言

□ 编程语言

- 用户：程序员
- 目标：完成对机器的编程(programming)

□ 低级语言/底层语言

- 机器码
- 汇编语言

□ 高级语言

- C语言
- JAVA语言
- JS语言
- Python语言

机器码

❑ 机器码

- 直接对应机器的操作: opcode

❑ 可编程机器

- Jacquard loom (雅卡尔提花机): 1804



en.wikipedia.org/wiki/Jacquard_machine

■ 打孔编程

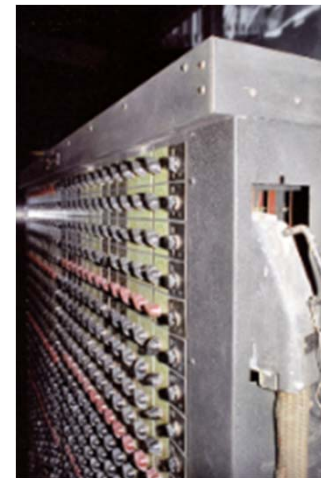
编程码

□ 机器码

- 直接对应机器的操作: opcode

□ 可编程机器

- ENIAC (第一台可编程通用电子计算机): 1945



en.wikipedia.org/wiki/ENIAC

- 开关电路板编程

机器码

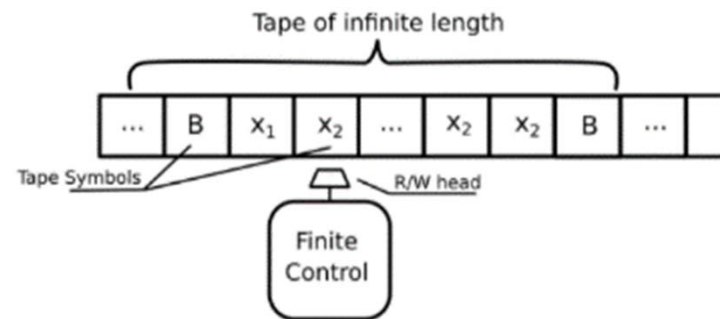
□ 机器码

- 直接对应机器的操作: opcode
- 通用、可复用
- 机器码够不够了?



<https://www.newscientist.com/people/alan-turing/>

- Turing completeness
- Church-Turing thesis



机器码

□ 机器码

- 直接对应机器的操作：opcode
- 通用、可复用
- 机器码够不够了？
- 编程效率低
 - 需要程序员理解比较“机械”的规则
 - 容易犯错

□ 汇编语言



汇编语言

□ 汇编语言

- 人类容易理解的方式，去“指导”机器工作
- 机器指令集(instruction set) \leftrightarrow 汇编指令集

```
mov    rsi, 2
mov    rax, 0
mov    rcx, 2
cmp    eax, dword [list+4]
jne    lp
mov    word [list+8], 4
lp:    add    eax, dword [list+rsi*4]
inc    rsi
loop   lp
```

```
[root@linux-server root]# objdump -D a.out | grep -A20 main.:
00000000 <main>:
00000000: 55                push    %ebp
00000001: 89 c5            mov     %esp,%ebp
00000003: 83 ec 08        sub     $0x8,%esp
00000006: 90                nop
00000007: c7 45 fc 00 00 00 00 movl    $0x0,0xffffffffc(%ebp)
0000000c: 89 f6            mov     %esi,%esi
0000000e: 83 7d fc 09      cmpl    $0x9,0xffffffffc(%ebp)
00000011: 7c 02            jlc     00000018 <main+0x10>
00000013: eb 18            jmp     00000018 <main+0x30>
00000015: 03 ec 0c        sub     $0xc,%esp
00000018: 68 08 05 04 08   push    $0x00040508
0000001b: e8 93 fe ff ff   call    00000018 <main+0x38>
0000001e: 03 c4 10        add     $0x10,%esp
00000020: 8d 45 fc        lea     0xffffffffc(%ebp),%eax
00000023: ff 00            incl    (%eax)
00000025: cb c1            jmp     00000018 <main+0x10>
00000027: 90                nop
00000028: b0 00 00 00 00   mov     $0x0,%eax
0000002b: c9                leave   %eax
0000002c: c3                ret
```

- 人写指令 \leftarrow **汇编器** \rightarrow 机器码
- 是因为某牛人发明了汇编器，于是才有了汇编语言吗？

汇编语言

□ 汇编语言

- 人类容易理解的方式，去“指导”机器工作
- 机器指令集(instruction set) \leftrightarrow 汇编指令集
- 更方便编程
- 编程效率低
 - 需要程序员理解“机器”的规则
 - 容易犯错
- 思考：为什么还需要学？

□ 高级语言

- C/Java/JS/Python

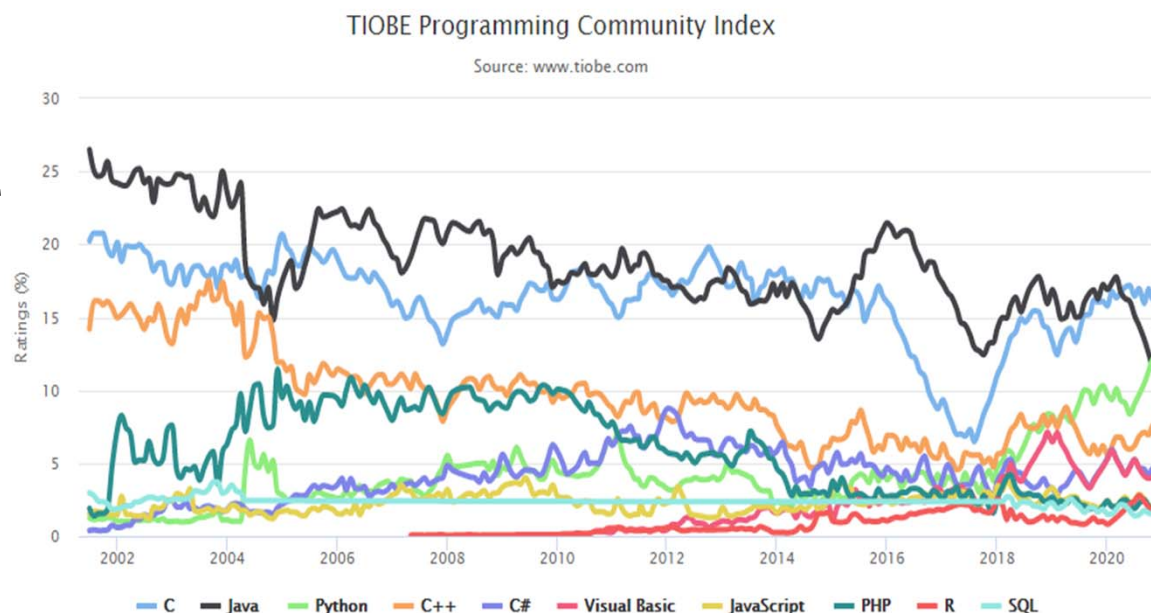
高级语言

□ 高级语言

- 人类更容易理解的方式，去编排机器的工作
- 什么是更容易理解的方式
 - 程序员的计算任务导向
 - 不管机器说什么语言，程序员只和它说人话

■ 数目？

- ~400: GitHub
- ~700: 维基百科
- ~8000: HOPL



高级语言：C

□ 和机器说人话(C语言) (1970前后开发)

- 数据抽象: int/double ...
- 运算抽象 $+ - * /$...
- 过程抽象 if/else, for ...
- 编程便利 函数/结构体 ...
- 好处: 方便、不易犯错



wikipedia.org



高级语言：C

□ 和机器说人话(C语言) (1970前后开发)

- 数据抽象: int/double ...
- 运算抽象 $+ - * /$...
- 过程抽象 if/else, for ...
- 编程便利 函数/结构体 ...
- 好处: 方便、不易犯错



wikipedia.org

□ 高级语言 ← 编译器/运行时 → 机器码

- 问题: 是因为人类技术突飞猛进, 某大师发明了C编译器导致C的流行吗?

高级语言：C

□ 高级语言 ← 编译器/运行时 → 机器码

- 问题：是因为人类技术突飞猛进，某大师发明了C编译器导致C的流行吗？

□ 过往的高级语言

- 1940年代：Plankalkül观念 (Konrad Zuse)
- 1950年代：
 - FORTRAN (John W. Backus) 数值计算
 - LISP (John McCarthy) 函数式编程
 - ALGOL (ACM, GAMM) 结构化编程/规范语言Spec
- 1960年代：
 - Simula (Ole-Johan Dahl) 面向对象

高级语言：C

- 高级语言 ← 编译器/运行时 → 机器码
 - 问题：是因为人类技术突飞猛进，某大师发明了C编译器导致C的流行吗？
- 1970年代：成熟的思想，完善的理论，只缺两个程序员（误）
- 解决Unix系统的编程需求
 - 写操作系统
 - 针对硬件特性写内核
 - 写操作系统需要什么特别的东西？
 - 直接对地址进行操作
 - 指针是好的，于是就有指针😊

高级语言：C

□ 指针的好处

- 灵活访问硬件/内存
- 效率高，对内存的操作编程方便

□ C语言流行

- native的语言，方便写各种**unix上的应用软件**
 - 通过C进行系统调用，如read/write
- **硬件大发展**的年代

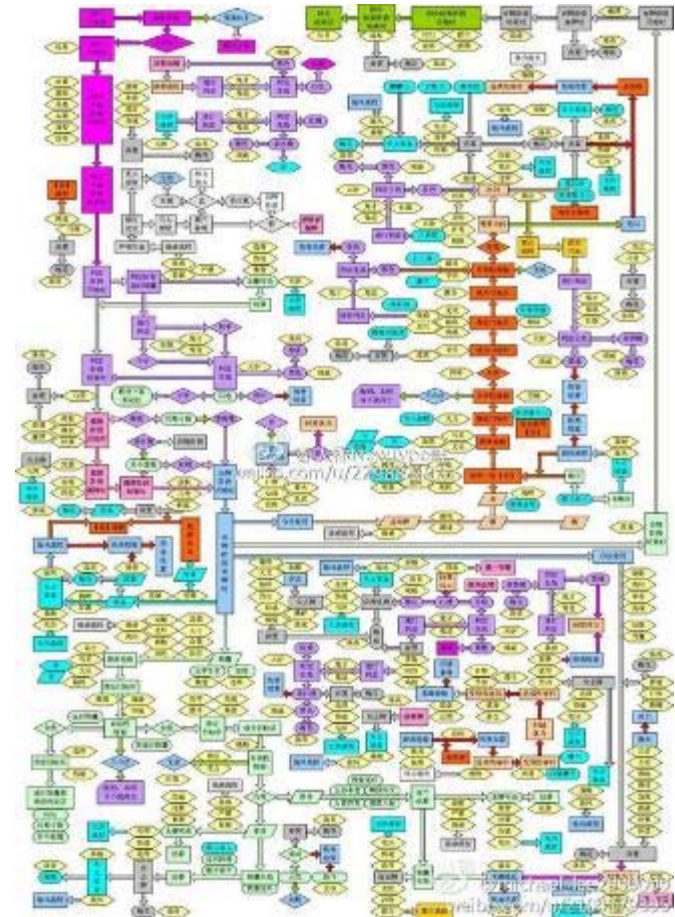
□ C语言的问题

- 软件越来越**复杂**，对复用、模块化解耦、可扩展性、避免犯错等等的需求越来越高

高级语言：C

□ C语言的问题

- 软件越来越**复杂**，对复用、模块化解耦、可扩展性、避免犯错等等的需求越来越高
- 比如malloc忘记free
- 针对任务过程编程
 - 各个环节都可能使用某个数据结构
 - 如果更改了？
- ...
- 需求变了：程序（**业务代码**）复杂
 - 程序员的合作模式 – 团队编程
 - 人容易犯错
 - 程序需要升级、复用
 - ...



高级语言：Java

- **需求变了：程序复杂**
 - 程序员的合作模式 – 团队编程
 - 人容易犯错
 - 程序需要升级、复用
- **Java**
 - 1990年代初期：James A. Gosling
 - **简单、面向对象、易学易用**
 - 风格类似于C/C++
 - 提供丰富的类库
 - 完全面向对象
 - **鲁棒、安全**



wikipedia.org

高级语言：Java

□ Java的流行

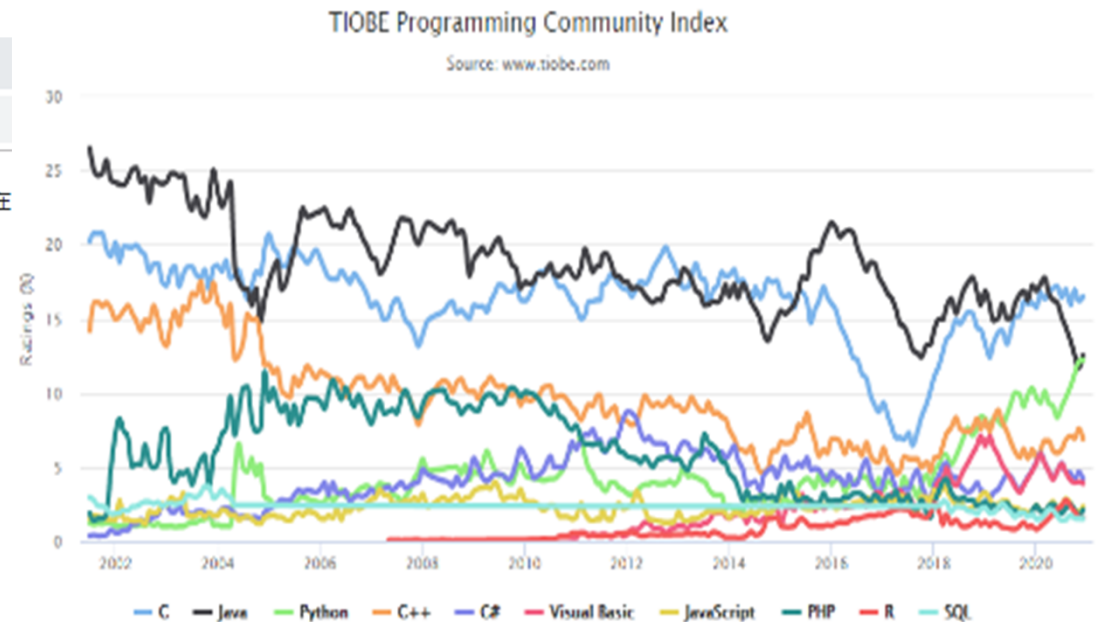
+

n [US] | https://www.java.com/zh_CN/about/

从笔记本电脑到数据中心，从游戏控制台到科学超级计算机，从手机到互联网，Java 无处不在



- 97% 的企业桌面运行 Java
- 美国有 89% 的桌面（或计算机）运行 Java
- 全球有 900 万 Java 开发人员
- 开发人员的头号选择
- 排名第一的部署平台
- 有 30 亿部移动电话运行 Java
- 100% 的蓝光播放器附带了 Java
- 有 50 亿张 Java 卡在使用
- 1.25 亿台 TV 设备运行 Java
- 前 5 个原始设备制造商均提供了 Java ME



Java结构化编程

□ 分支关键字

- `if else`
- `switch case break`

□ 循环关键字

- `while`循环
- `for`循环
- `do while`循环
- `break continue`控制循环过程

□ 调用和返回`return`

□ 异常处理`try catch finally`

- 回头细讲

纯结构化编程

非结构化编程?
有什么不好?

Java类型抽象

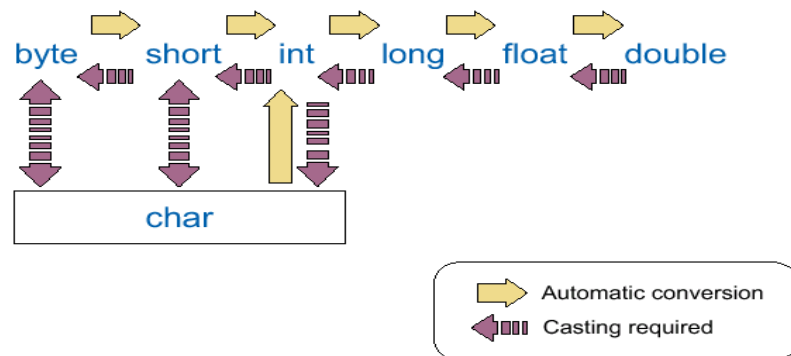
□ 8种基本类型

- 整形: **int** 4字节 默认0
- 短整形: **short** 2字节 默认0
- 长整形: **long** 4字节 默认0
- 浮点形: **float** 4字节 默认0.0f
- 双精度浮点形: **double** 8字节 默认0L
- 字符形: **char** 2字节 默认 'u0000'
- 字节形: **byte** 1字节 默认0
- 布尔形: **boolean** 1个bit 默认False
(由于填充的关系, 通常是1字节)

Type Name	Size (in bits)
Integrals:	
byte	8
short	16
int	32
long	64
Floating Points:	
float	32
double	64
Characters:	
char	16
Booleans:	
boolean	n/a

Java类型抽象

- 指定存储大小
- 默认值
- Casting的规矩
 - 低级(规模"小")的类型可以自动转换为高级(规模"大")的类型
 - 高级(规模"大")的类型需**显式**地转换为低级(规模"小")的类型



Java类型抽象

□ 扩展数据类型

- 数组、对象-类

□ 数组

- 元素都是同一种类型
- 长度在创建时确定，并保持不变
 - `char s[] = new char[20];`
 - 运行时越界检查
 - 解决C语言指针容易导致的bug
- 对避免犯错的好处？

Java类型抽象

□ 类-对象

■ 面向对象

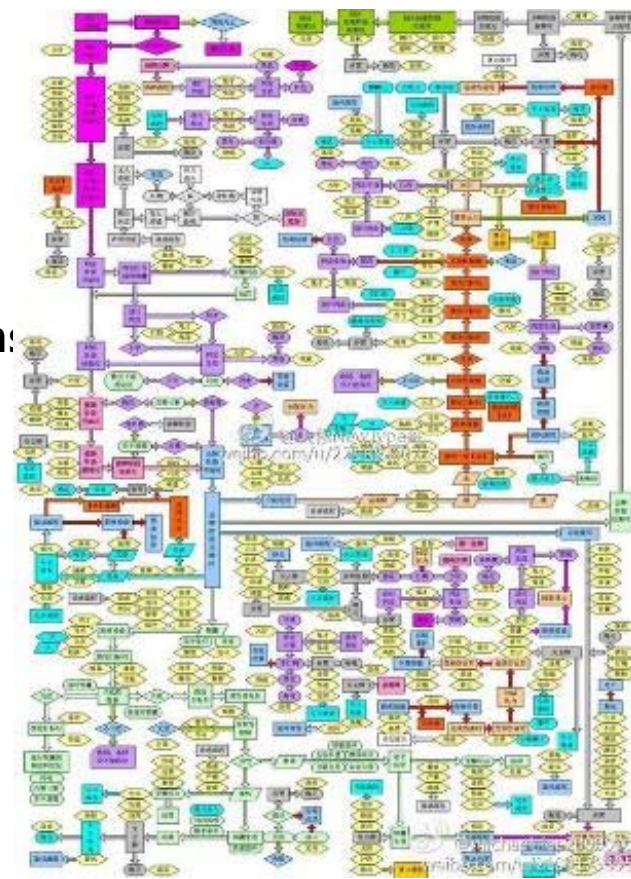
- 封装：把针对数据的逻辑代码，和数据实现在一起
- 继承
- 多态
- 其他：构造函数/数据访问保护

□ 开闭原则

- Software entities (classes, modules, function: should be open for extension but closed for modification

□ 里式替换法则

- 子类必须实现父类的属性(替换父类)



Java避免程序员犯错

□ 语言的设计层面及编译检查

- 让你做不了容易出错的事
 - 没有指针、有自动内存管理
 - 数据类型抽象(如boolean)
 - 面向对象的支持
 - 不能乱cast

□ 运行时

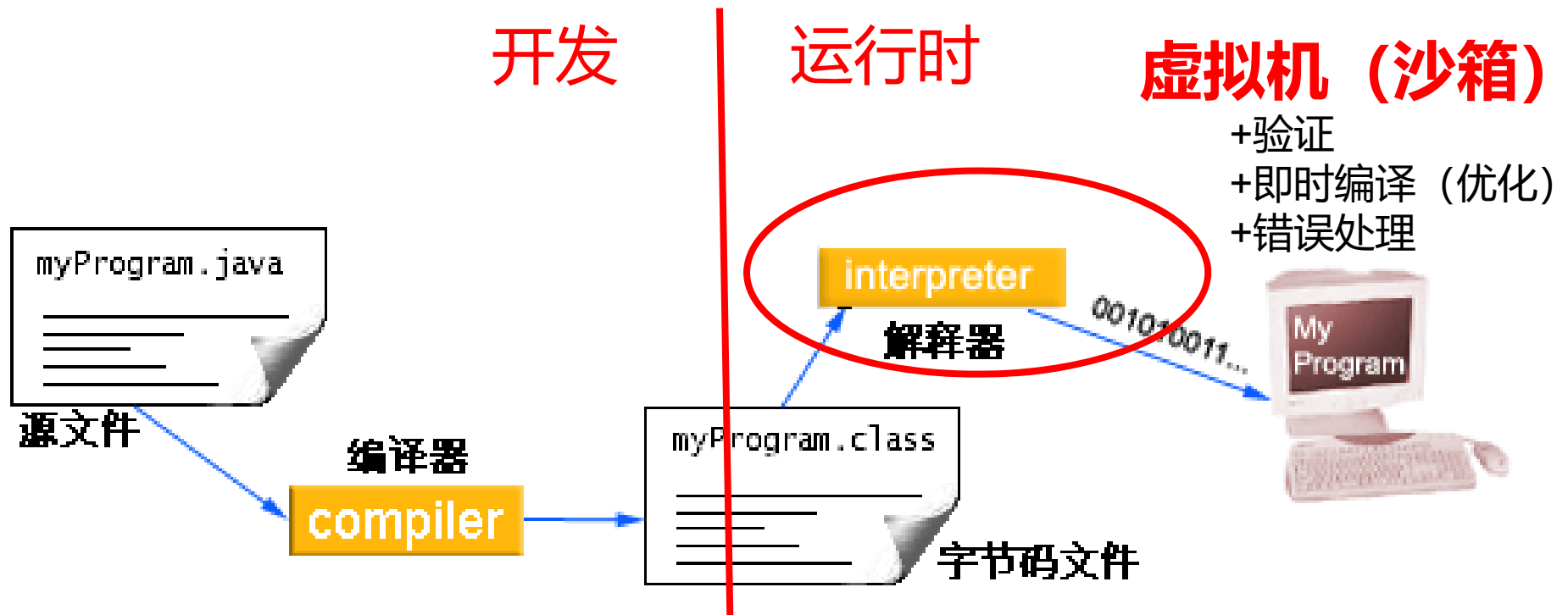
- 数组越界
- 文件找不到了, U盘拔出来了

Development

Runtime

问题在哪里被发现更好?

Java工作模式



□ Java也不是纯解释性的

■ 预测，即时编译 → 优化

运行时保护

- ❑ 代码检查、内存保护
 - 没有野指针
- ❑ 内存管理
 - 垃圾回收（可达性分析），只new不用delete
- ❑ 运行时错误检查
 - Exception Handling
- ❑ ...

Java异常

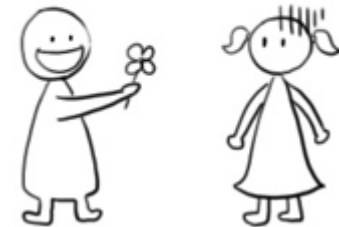
- **程序运行时会遇见很多异常**
 - 文件找不到
 - 读写文件时发生IO错误
 - 网络连接失败
 - 参数非法
 - 空引用
 - 数组越界
- **通过异常的捕捉处理、防止程序崩溃**

Java异常的捕捉

□ try/catch/finally

```
try {  
    //可能会抛出异常的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
catch (XXException e) {  
    //异常处理的语句  
}  
finally {  
    //最后需要执行的语句  
}
```

try ...



catch ...



异常捕捉示例

```
public static void copy(String sFile, String dFile) {  
    File srcFile = new File(sFile);  
    File destFile = new File(dFile);  
    FileInputStream fin = null;  
    FileOutputStream fout = null;  
    try {  
        fin = new FileInputStream(srcFile);  
        if (!destFile.exists()) {  
            destFile.createNewFile();  
        }  
        fout = new FileOutputStream(destFile);  
        byte[] bytes = new byte[1024];  
        while (fin.read(bytes) != -1) {  
            fout.write(bytes);  
            fout.flush();  
        }  
    } catch (FileNotFoundException e) {  
        System.out.println("Can find the source file: " + sFile);  
    } catch (IOException e) {  
        System.out.println("IO Exception caught.");  
    }  
    ...  
}
```


Java异常

❑ Checked exceptions

- 提供机制，强制程序员写异常处理
- 什么是需要强制的?

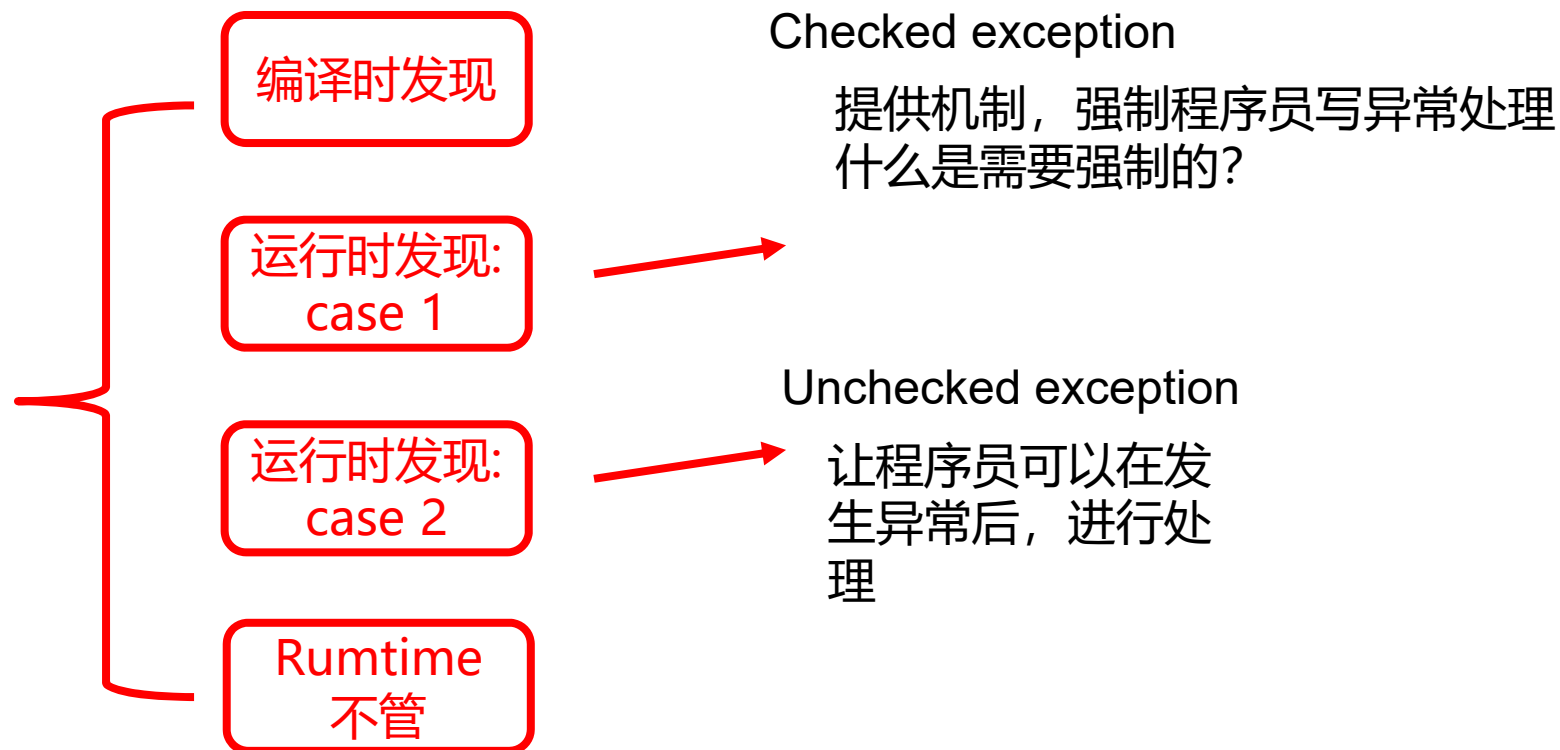
程序员

❑ Unchecked exceptions

环境

- 提供机制，让程序员可以在发生异常后，进行处理

Safety威胁的处理



高级语言：Java

- Java严格、甚至有点繁琐的语法
 - 避免程序员犯猪头错误
 - 适合大规模的业务代码
- Java方便团队合作
 - API, 框架
 - 命名空间(package)
 - 复用、扩展
 - 静态类型检查、强类型
 - ...
- 解决了业务代码编程语言正确的需求

高级语言：JavaScript

- 1995年Netscape公司开发
 - Brendan Eich
- 动机
 - HTML页面都是静态的，无法动态改变
 - “看上去与Java足够相似，但是比Java简单”

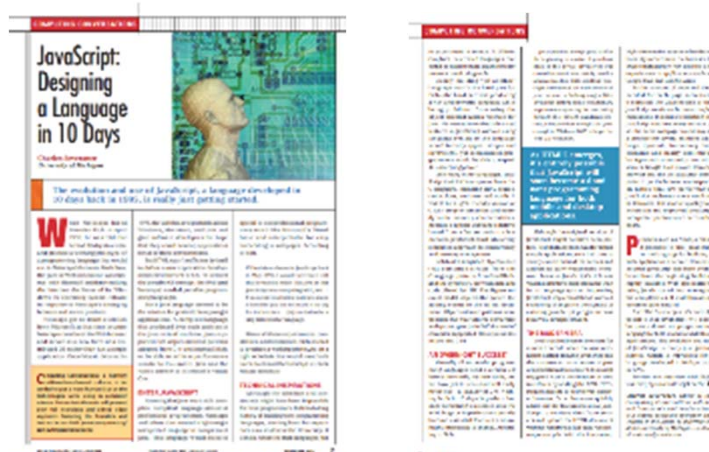


高级语言：JavaScript

□ 做法

- 借鉴C语言的基本语法
- 借鉴Java语言的数据类型和内存管理;
- 借鉴Scheme语言以函数为核心的思路
- 借鉴Self语言实现面向对象-原型(prototype)继承机制

- Charles Severance, "JavaScript: Designing a Language in 10 Days", Computer, vol. 45, no. , pp. 7-8, Feb. 2012



高级语言：JavaScript

□ 弱类型，动态类型检查

■ 数组可以多个类型

→ [0, "hello", null, undefined, false, 1, [1], {a: 1}]

■ 比较运算符 ☺

• 红色：===

• 橙色：==

• 蓝色：只有 >=

如 "-1" >= []

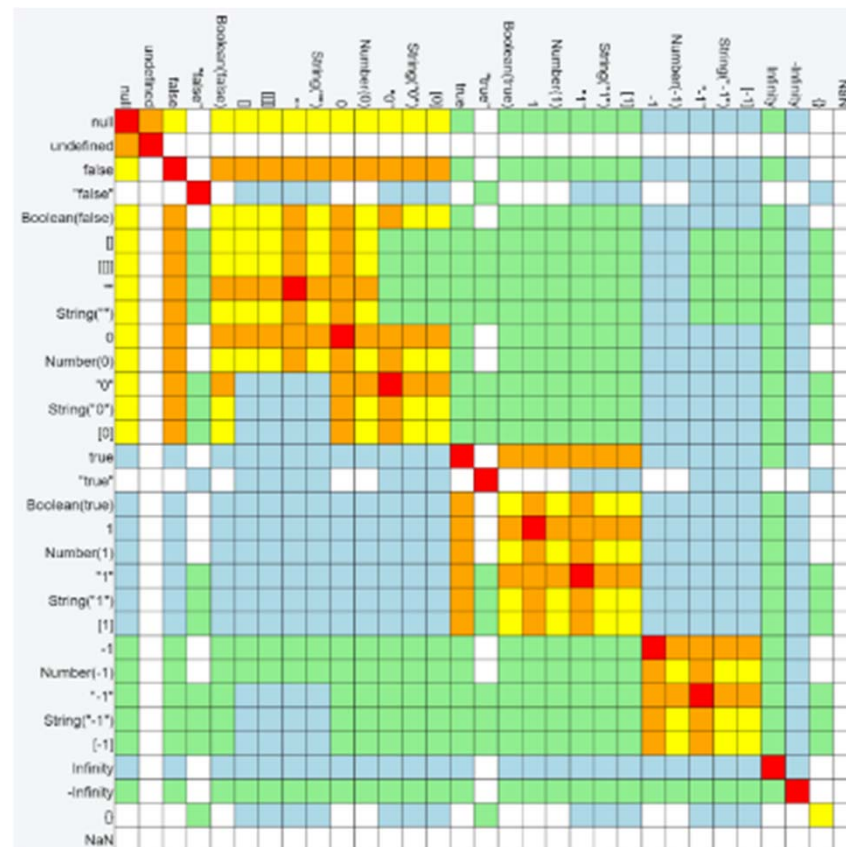
• 绿色：只有 <=

如 "1" <= false

• 黄色：<= 和 >= 同时成立，

== 不成立，

如 [1] 和 "1"



高级语言：JavaScript

□ var定义的变量

■ 函数作用域

```
var value = 'local';  
var func = function() {  
    //省略若干代码  
    if (false) {  
        var value = 'func_local';  
    }  
    console.log(value);  
}  
func();  
console.log(value);
```

undefined

local

级高语言：JavaScript

□ this指针绑定问题

```
var a = 10;
var obj = {
  a : 2,
  foo: function() {
    console.log(this.a);
    function test() {
      console.log(this.a);
      console.log(a);
    }
    test();
  }
};
obj.foo();
```

2

undefined

10

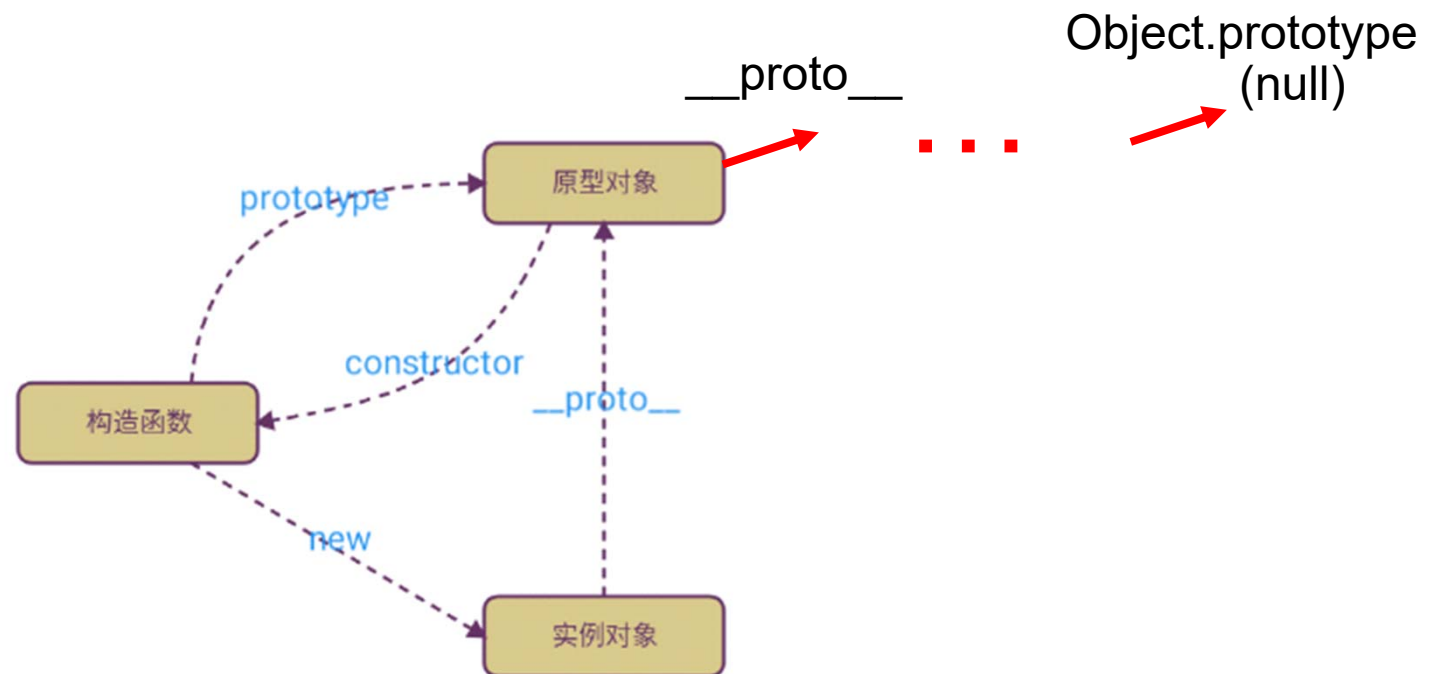
□ this引用的指向和函数的调用方式有关

- 普通的函数调用、对象的方法调用、构造函数调用、apply/call调用

高级语言：JavaScript

□ 对象继承：原型继承范式

■ 通过构造函数的prototype实现继承



```
Dog.prototype = new Animal();  
var pp = new Dog();
```

高级语言：JavaScript

□ 寄生组合继承

```
function derive(o) {  
  function F() {  
  }  
  F.prototype = o;  
  return new F();  
}  
function Animal(gender){  
  this.gender = gender;  方法和数据剥离  
}  
Animal.prototype.getName = function(){ return 'Animal';};  
function Dog(gender){  
  Animal.call(this, gender);  拷贝数据  
  // ...  
}  
var proto = derive(Animal.prototype);  “虚” 的原型，只负责连接方法  
proto.constructor = Dog;  
Dog.prototype = proto;
```

高级语言：JavaScript

❑ ECMAScript 2015 (ES6)

■ class关键字简化继承编程

```
1 class Animal {
2   constructor(name, sex) {
3     this.name = name;
4     this.sex = sex;
5   }
6
7   getName() {
8     return this.name;
9   }
10 }
11
12 class People extends Animal {
13   constructor(name, sex) {
14     super(name, sex);
15   }
16 }
17
18 var Chris = new People('Chris', 'male');
19 Chris.getName(); // Chris
```

■ 箭头函数把this绑定在词法上下文

```
getName () {
  var name = () => {
    console.log(this.name);
  }
  name();
}
```

■ let/const

→ 块级作用域

→ 不能重复声明

高级语言：JS → TS

- ❑ ECMAScript 2015 (ES6) → ES11 (2020)
- ❑ Node.js
 - 甚至后端的业务代码也常用js实现
- ❑ NPM 包管理器
 - npm -install jquery
 - 解决复用 – 库依赖
- ❑ 但是
 - JS是动态类型检查语言
- ❑ TypeScript
 - JS超集（完全兼容），静态类型检查
- ❑ JS/TS: usability导向的发展

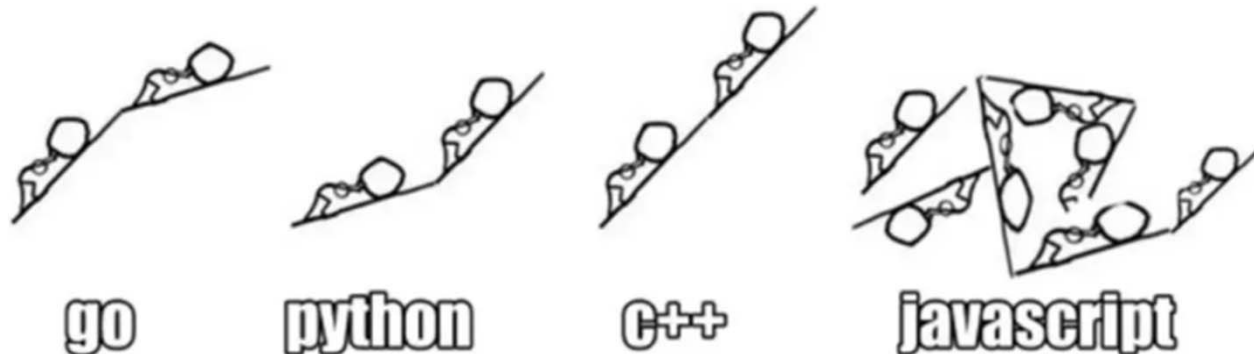
高级语言：Python

- ❑ **2010年前**
 - 谁会更流行: Perl/Python/Ruby
- ❑ **现在**
 - 大部分人的第一语言
 - C语言?
- ❑ **Python流行是因为它有某种特别强大的技术发明吗?**
- ❑ **用户的选择**
 - 写小型程序方便
 - 简单、易学, API多, 适合数据分析
 - 大型程序?

编程语言

□ 编程技术的发展，都是在解决程序员的问题

- 程序员是编程语言的用户
- 需求驱动
 - 正确地了解需求
 - 解决正确的需求
- 不是用甩别人几条街技术来获得流行
 - 都是用well-accepted的理论、思想（官科）
- Usability决定了成败
- C、Java、JS/TS、Python等等





感谢聆听，欢迎交流



X2 D4025
zyf@fudan