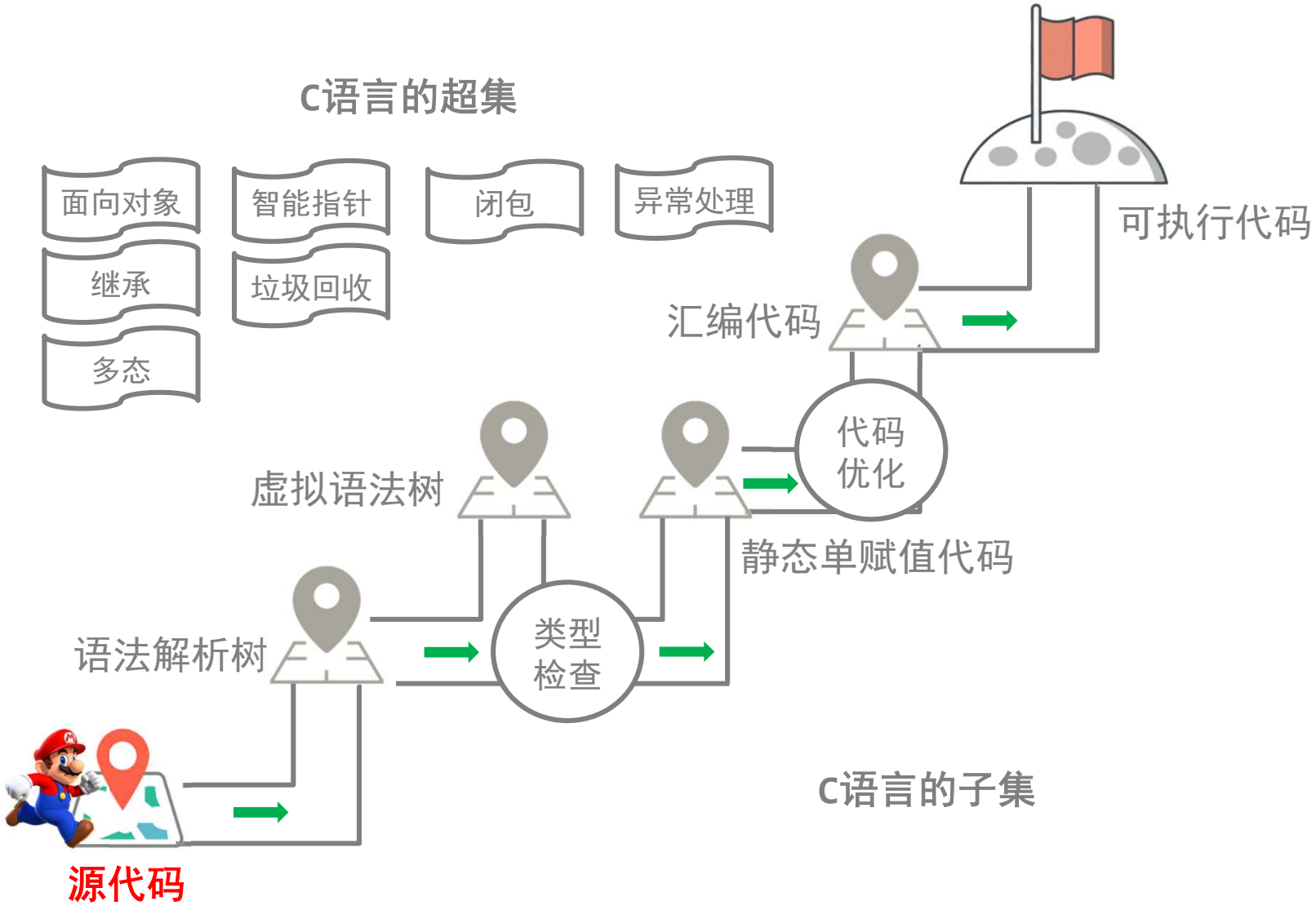
Lecture 1

编程语言

徐 辉

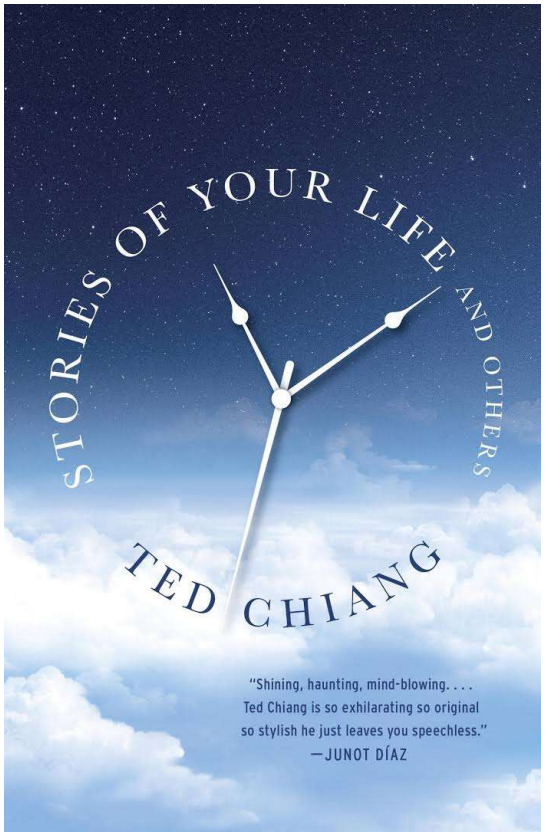
xuh@fudan.edu.cn

学习地图


C语言的超集
面向对象
智能指针
闭包
异常处理
继承
垃圾回收
多态
可执行代码
汇编代码
代码
优化
虚拟语法树
静态单赋值代码
类型
检查
语法解析树
C语言的子集
源代码

沃尔夫假说：Sapir-Whorf hypothesis

人类的思维模式会受其使用语言的影响



教学大纲

- 一、编程语言范式
- 二、编程语言发展史
- 三、编程语言流行语

一、编程语言范式

主要编程范式Programming Paradigm

- 命令式（Imperative）编程
- 函数式（Functional）编程
- 声明式（Declarative）编程
- 面向对象（Object-oriented）

命令式编程

- 命令式编程语言是接近计算机硬件实现的语言设计
  - 很容易翻译成计算机指令
  - 执行顺序很关键
  - 代表语言：Fortran、Basic、C
  - 主要组成：
    - 赋值语句：内存存取、运算、以及更复杂的运算赋值形式
    - 循环语句：for、while等
    - 条件/非条件跳转语句

```
int fact(int n) {
    int sofar = 1;
    while (n>0) sofar *= n--;
    return sofar;
}
```

函数式编程

- 将电脑运算视为函数运算，避免或减少副作用.
  - 副作用：除了返回值以外对主调用函数产生的影响。
    - 如修改全局变量或参数。
  - 主要特点：
    - 单赋值
    - 多用递归实现
  - 代表语言：Haskell、ML、Lisp等
  - 基于λ演算（λ-calculus）的思想

```
(defun factorial (n)
  (if (= n 0) 1
      (* n (factorial (- n 1))) ) )

(loop for i from 0 to 16
   do (format t "~D! = ~D~%" i (factorial i)) )
```

Lisp代码

函数式编程：匿名函数实现递归

Lisp代码（Y Combinator）

```
(defun YCombinator (f)
    (funcall #'(lambda (x) (funcall f
                                    #'(lambda (y) (funcall (funcall x x) y))))
             #'(lambda (x) (funcall f
                                    #'(lambda (y) (funcall (funcall x x) y))))))
(funcall
    (YCombinator
         #'(lambda (f)
             #'(lambda (n) (if (eq n 0) 1 (* n (funcall f (1- n))))))))
 10)
```

声明式编程

- 描述目标的性质，而非流程，避免或减少副作用。
- 包括以下子类：
  - 函数式编程：如Haskell、ML、Lisp
  - 领域专属语言：如SQL、XML
  - 逻辑式编程：如Prolog
  - 约束式编程：如SMT-Lib

```
fact(X,1) :-
    X =:= 1.
fact(X,Fact) :-
    X > 1,
    NewX is X - 1,
    fact(NewX,NF),
    Fact is X * NF.
```

Prolog代码

```
(set-logic QF_LIA)
(declare-const result Int)
(define-fun-rec
   fac ((x Int)) Int (
      ite (<= x 1)  1
      (* x (fac (- x 1)))
   )
)
(assert (= (fac 4) result))
(check-sat)
(get-value(result))
```

SMT-Lib代码

面向对象编程

- 将对象（类的实例）视作程序的基本单元。
  - 将数据和方法封装在对象中；
  - 类可以包含子类，子类可继承父类的数据和方法；
  - 不同的子类表现出多态性，如同一方法的效果多样性。

```
class Num {
    int val;
public:
    Num(int val) {this->val = val;}
    int factorial() {
        int i, r = 1;
        for (i = 1; i <= this->val; i++) {
            r = r*i;
        }
        return r;
    }
};

Num obj(10);
obj.factorial();
```

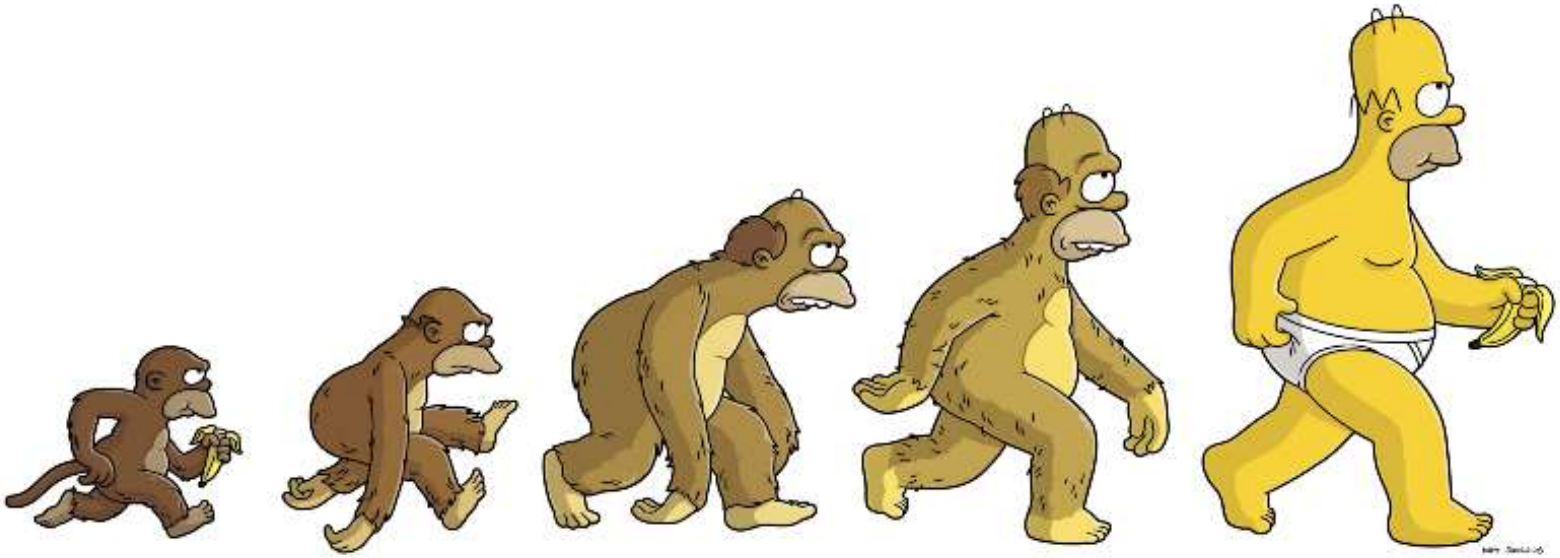
语言影响编程习惯

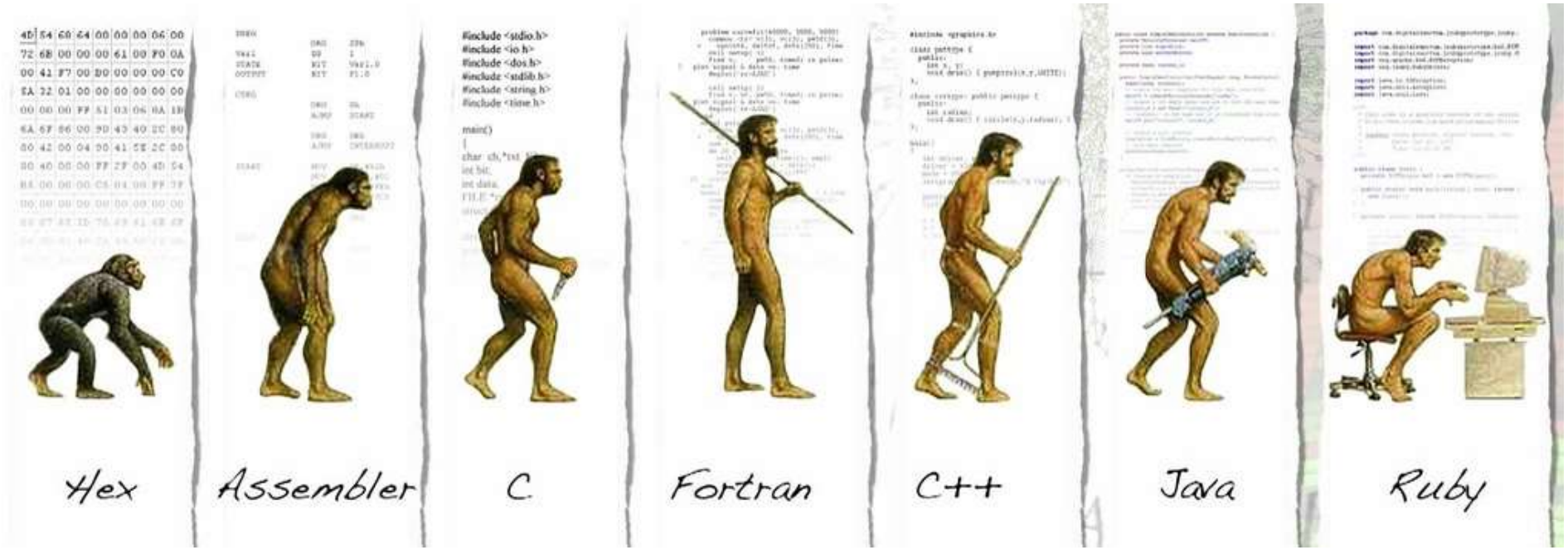
- 语言特性决定了其用法
  - 面向对象：大量使用对象
  - 函数式编程：大量使用小的无副作用的函数
  - 声明式编程：在逻辑问题域内进行搜索

如何对语言进行分类？

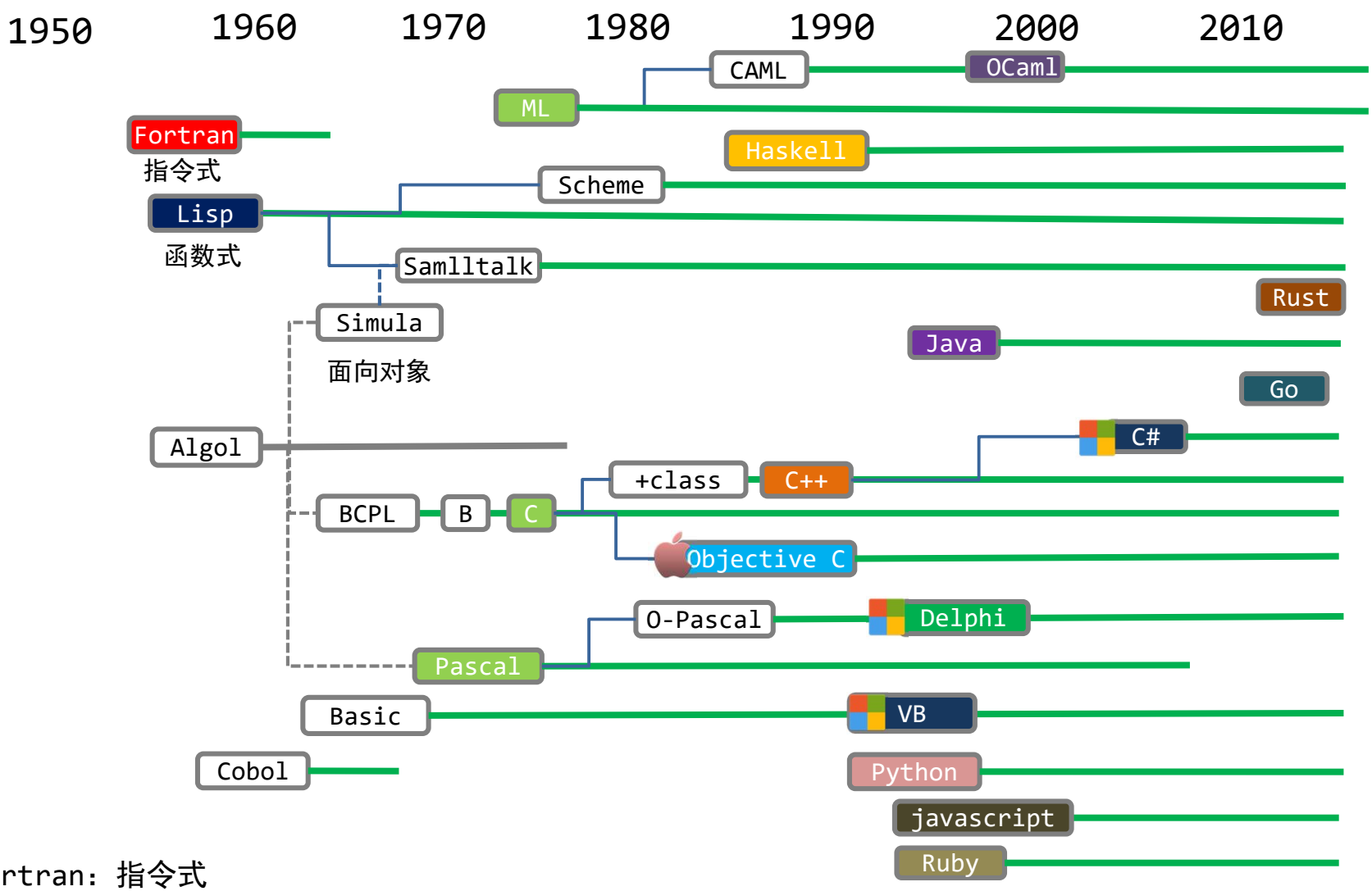
- 语言并非纯粹的属于某一编程范式，而是多范式的。
- 很多语言范式或特性的概念是模糊的
  - 坏问题：X语言是函数式编程语言吗？
  - 好问题：X语言支持函数式编程吗？怎样支持的？

二、编程语言发展简史


MACHINE
ASSEMBLY
PROCEDURAL
OBJECT ORIENTED
FUNCTIONAL


Hex
Assembler
C
Fortran
C++
Java
Ruby

编程语言发展简史

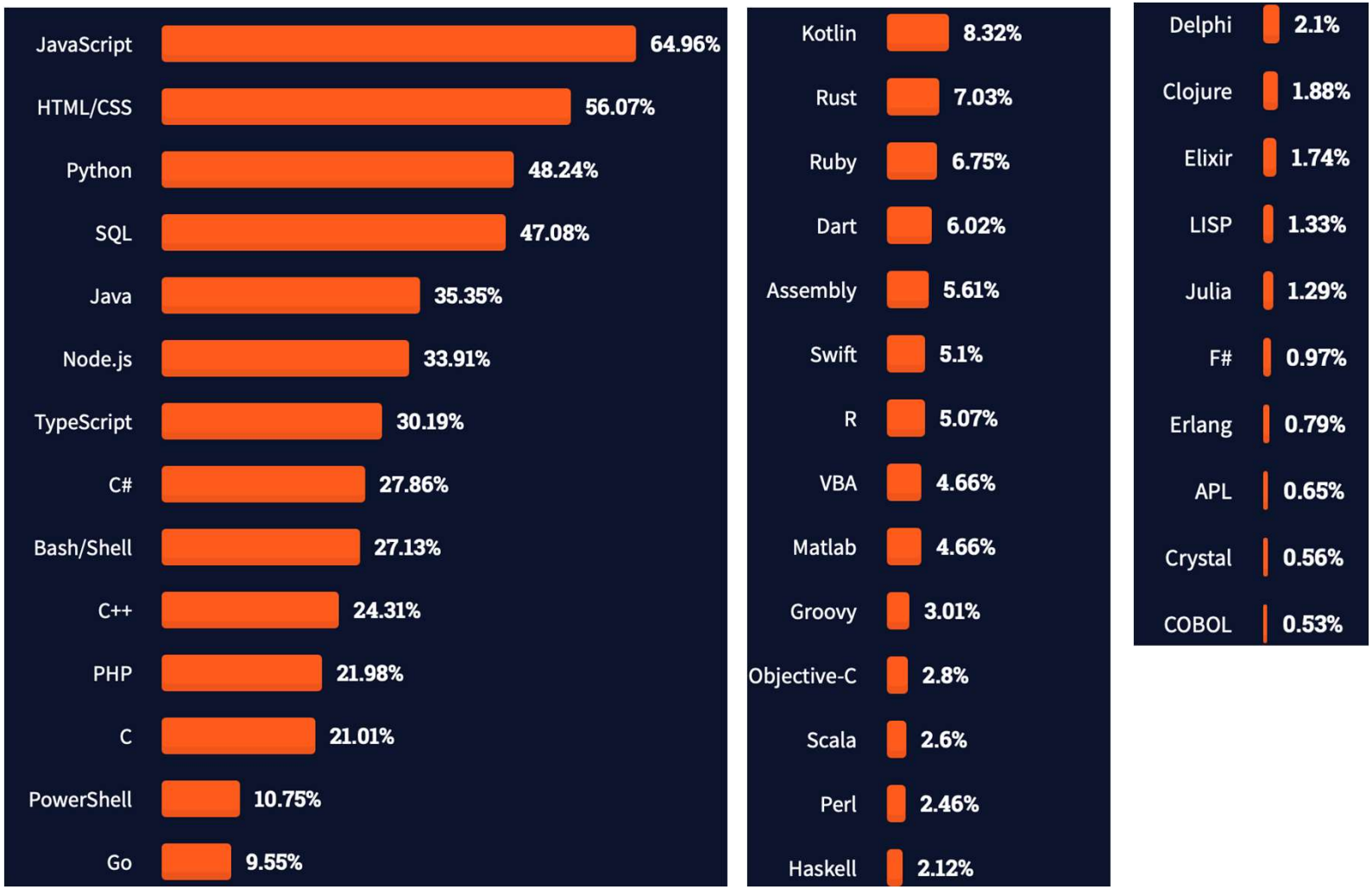


Fortran：指令式
Lisp：函数式
Algol：指令式
Simula：面向对象

编程语言发展

- 新的语言不断涌现，旧的语言出现各种方言
- 广泛使用的语言并非创新性强，如Java（1995年）
  - 借鉴了很多C++的概念
  - C++是对C语言的扩充，结合了Simula、ML等语言思想
  - C语言由B语言发展来，B语言继承Algol的特性
- 未广泛使用的语言并非创新性弱，如Algol
  - 最早的编程语言之一，Algol 58、Algol 60、Algol 68
  - 首次引入了很多被广泛借鉴的编程思想

Stackoverflow语言使用统计排名



https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof

编程语言是否还有研究价值？

- 编程语言历史学术会议HOPL（History of programming languages）
  - 2021年是第四届
- 1996年1月，第二界HOPL会议达成的基本共识是编程语言研究已结束，因为所有重要的概念基本都发明了。
  - 但在此之后，Java诞生，并取得巨大成功
- 语言发展的推动力：
  - 语言是有寿命的，不断发展迭代；
  - 新语言刚开始因为某些不可替代的特性在10-20年间活跃；
  - 在此之后通常需要变化、升级来适应需求；
  - 有些语言（如Cobol）因为仅因维护需要还有人在使用。

https://hopl4.sigplan.org/

主要编程范式的诞生：可计算性问题

大卫.希尔伯特
（数学的完备性问题）
1900年

- 完备性：所有数学命题都可以用一组有限的公理证明或证否；
- 一致性：可以证明的都是真命题。

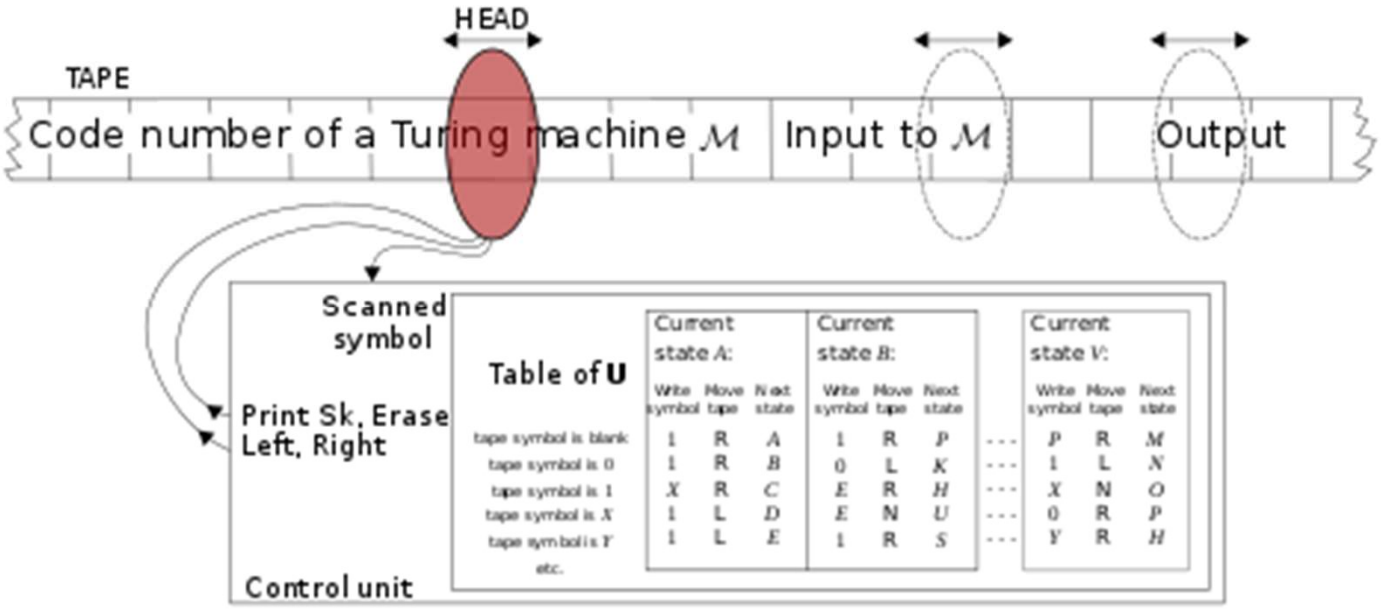
库尔特.哥德尔
（不完备定理）
1931年

- 反例：“这个命题是不可证的”
- 如果完备性成立，则一致性不成立；
- 如果一致性成立，则完备性不成立。

邱奇和图灵
（可计算性）
1936年

- 可判定性：哪些问题是在有限时间内可以计算的？
- 邱奇：可计算函数和Lambda演绎
- 可计算函数可以等价为图灵机能计算的函数
  - 《论可计算数及其在判定问题中的应用》

图灵机启发命令式编程



- 通用图灵机：
  - 一条无限长的纸带TAPE
  - 一个读写头HEAD
  - 一个状态寄存器
  - 一套控制规则TABLE：根据当前的寄存器值和读写头所指的格子上的符号确定下一步的动作

Lambda演绎启发函数式编程

组成	形式	举例	
变量	x		
抽象	(λx.M)	f(x)= x + 2 => λx.x + 2	函数定义
应用	(M,N)	f(1) => (λx.x + 2) 1	函数应用

可以接受函数作为　函数的输入参数或返回值

(λf.f 1)(λx.x+2)

应用β reduction ⇨ (λx.x + 2) 1

应用β reduction ⇨ 1 + 2

利用不动点实现匿名递归

$(\lambda x.x\ x)(\lambda x.x\ x)$

应用β reduction ⇨ $(\lambda x.x\ x)(\lambda x.x\ x)$

β-normal form

$X = (\lambda x.f(x\ x))(\lambda x.f(x\ x))$

应用β reduction ⇨ $X = f(\lambda x.f(x\ x)\ \lambda x.f(x\ x))$

替换 ⇨ $X = f(X)$

Y Combinator：$\lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$

特性：对于任意输入参数（函数f）都返回不动点：

$$Yf = f(Yf) = f(f(Yf)) = f(f(\ldots f(Yf)\ldots))$$

匿名递归实现阶乘

令 F := λf.λx.(if x == 0 then 1 else x * f (x-1))

```
YF 3
F(YF) 3
λf.λx.(if x == 0 then 1 else x * f (x-1)) (YF) 3
if 3 == 0 then 1 else 3 * (YF)(3-1)
3 * (YF) 2
3 * F(YF) 2
3 * (λf. λx.(if x == 0 then 1 else x * f (x-1)) (YF) 2)
3 * (if 2 == 0 then 1 else 2 * (YF)(2-1))
3 * (2 * (YF) 1)
6 * (YF) 1
6 * F(YF) 1
6 * (λf. λx.(if x == 0 then 1 else x * f (x-1)) (YF) 1)
6 * (if 1 == 0 then 1 else 1 * (YF)(1-1))
6 * (YF) 0
6 * F(YF) 0
6 * (λf. λx.(if x == 0 then 1 else x * f (x-1)) (YF) 0)
6 * (if 0 == 0 then 1 else 0* (YF)(0-1))
6 * 1
6
```

现在你可以理解这段代码了吗？

Lisp代码（Y Combinator）

```
(defun YCombinator (f)
    (funcall #'(lambda (x) (funcall f
                                    #'(lambda (n) (funcall (funcall x x) n))))
             #'(lambda (x) (funcall f
                                    #'(lambda (n) (funcall (funcall x x) n))))))
(funcall
    (YCombinator
         #'(lambda (f)
             #'(lambda (n) (if (eq n 0) 1 (* n (funcall f (1- n))))))))
 10)
```

不同的编程语言的等价性

- 邱齐-图灵理论决定了哪些问题是可计算的：
  - 图灵机与Lambda算子能解决的问题集合等价；
  - 不同的通用编程语言（general purpose language）能解决的问题集合等价；
    - 如C、Java、Lisp
  - 领域特定语言（domain-specific language）的表达能力一般是受限的；
    - 如SQL、HTML、CSS
  - 高级语言和汇编语言能解决的问题集合等价。

三、编程语言流行语

Programming Language Buzzwords

编译执行和解释执行

- 编译执行
  - 通过编译器将源代码翻译为可以直接运行的机器码；
  - 如C/C++、Rust等语言。
- 解释执行
  - 在目标机器上将源代码翻译成计算机指令并执行；
  - 如javascript、python等各种脚本语言；
  - 平台无关，但存在一定的运行时解释开销。
- 混合模式
  - 如Java等语言通过编译器将源代码翻译为中间代码；
  - 平台无关，省略代码解析开销。

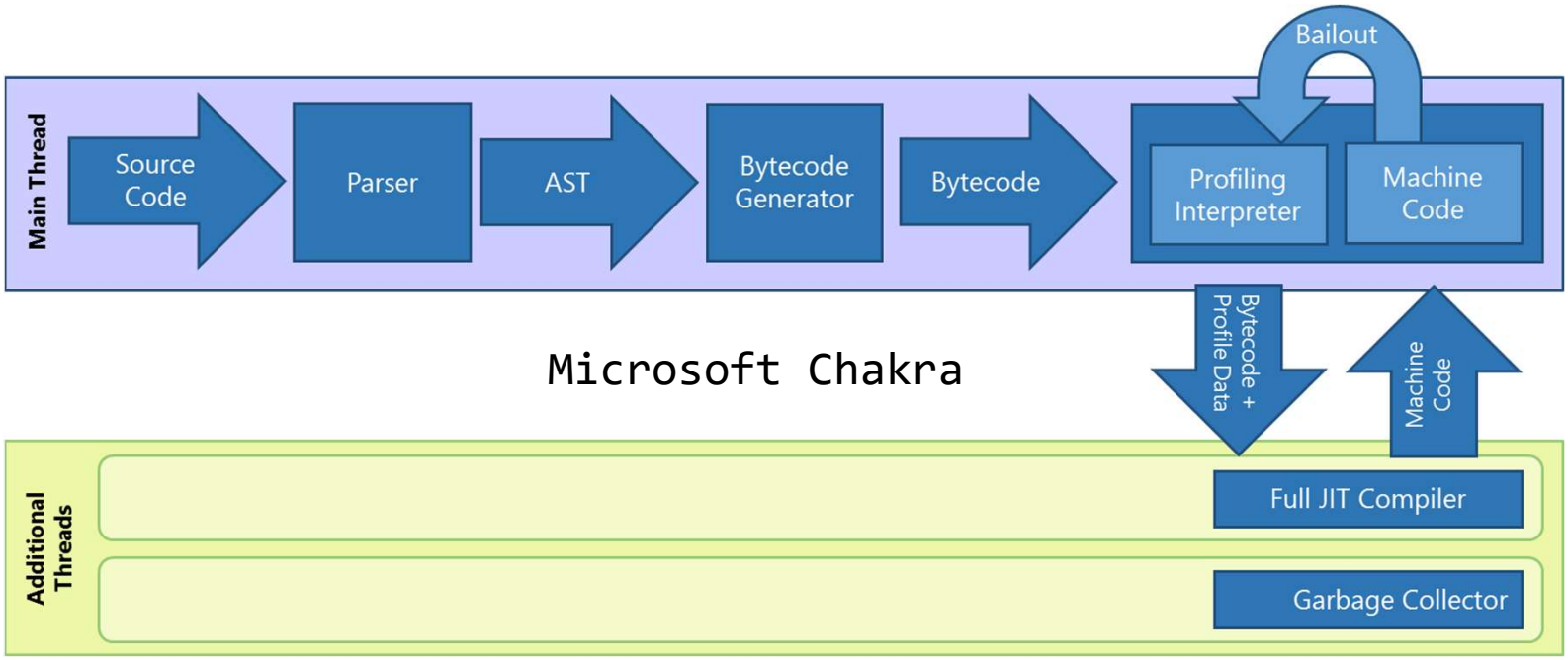
运行时编译just-in-time (JIT) compilation

- 为了降低解释执行的运行时开销，JIT编译器将一部分需要频繁运行的代码直接编译为机器码。
- 在javascript、JVM等解释执行引擎中广泛使用。
- 将全部代码提前编译为机器码的技术称为Ahead-of-Time（AOT）compilation。
  - 传统C/C++等编译执行语言
  - 将Java代码直接编译为机器码
    - 安卓智能手机

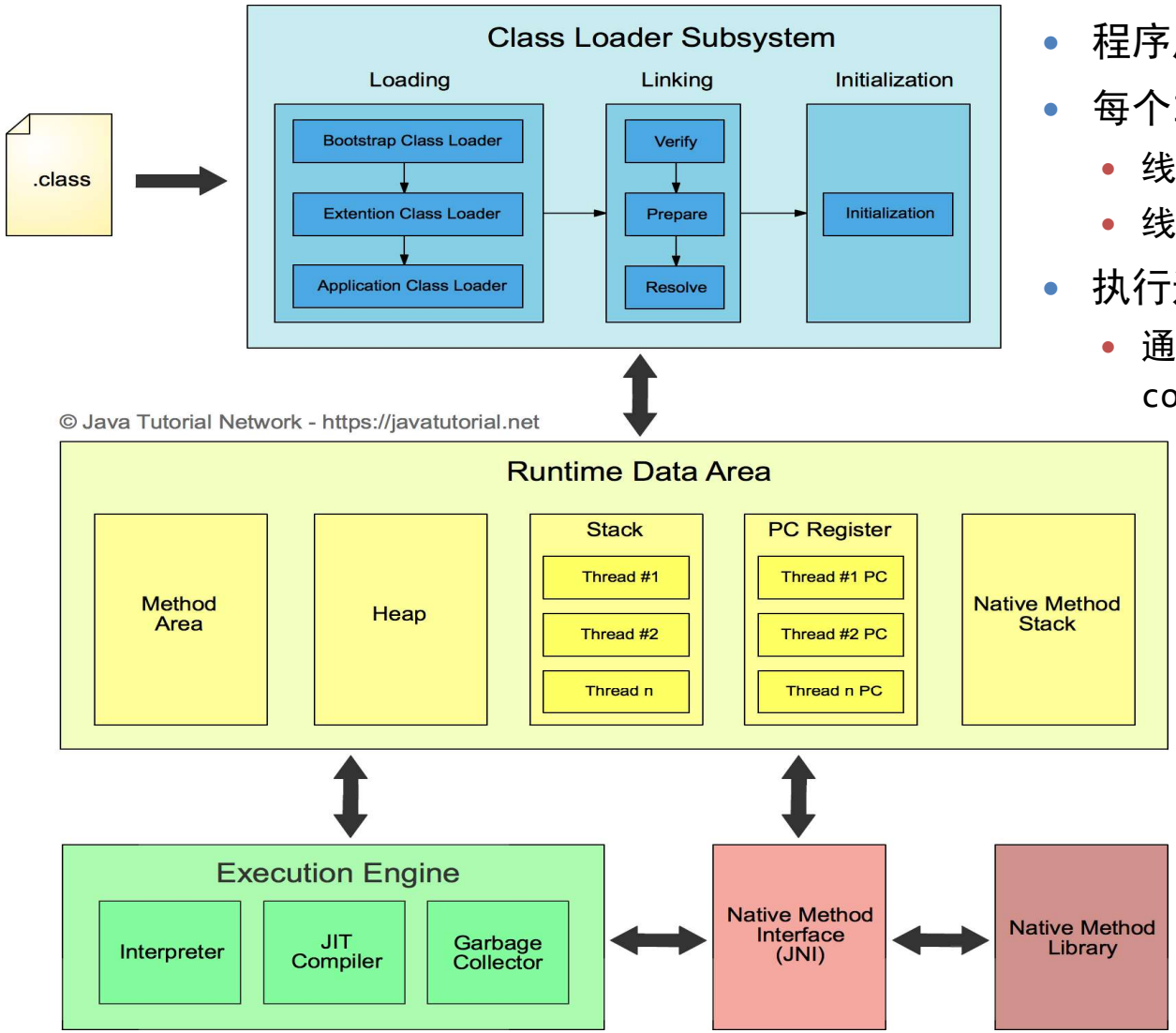
实例：JavaScript引擎

- 如微软的Chakra和Google 的V8。
- 通过Profiler监控代码运行，分析可通过JIT优化的代码。
  - 如循环内部的代码；
  - 优化是有风险的，尤其js是动态类型，可能存在类型问题；
  - 通过Bailout回退到解释执行。

```
for (var i = 0; i < arr.length; i++) {
    sum += arr[i];
}
```



实例：Java虚拟机



- 程序启动时进行加载、链接、和初始化。
- 每个JVM维护独立的运行时环境。
  - 线程间共享Method和Heap
  - 线程独享的stack和register
- 执行过程为解释执行。
  - 通过JIT将重复运行的代码翻译成native code

静态类型和动态类型

- 静态类型（static typing）
  - 编译时确定数据类型；
  - 通过类型检查避免运行时错误；
  - 如C/C++、Java等语言；
  - 一般写程序时需要为变量指定具体的类型，但不绝对。
- 动态类型
  - 运行时确定数据类型；
  - 如Python等各种语言；
  - 一般写程序时不用为变量指定具体的类型。

运行时环境和垃圾回收

- 运行时环境（RTE）：软件在系统中运行所需要的环境，一般指动态库依赖等。
  - 如C语言的libc，Java的JRE等；
  - 与具体的操作系统交互，完成内存管理等功能；
  - 有些语言如C、Rust可以不使用运行时（no_std）。
    - 在裸机环境运行
- 垃圾回收：自动内存（堆）管理机制
  - 用户程序和垃圾回收器管理共同的内存空间；
  - 如Java、Go、Python、Ruby等语言；
  - 垃圾回收依赖虚拟机吗？不是。

内容总结

- 主要编程范式：
  - 命令式、函数式、声明式、面向对象
- 编程语言发展简史
  - 为什么会有命令式和函数式编程
- 一些基本概念
  - 编译执行、解释执行、JIT、AOT
  - 静态类型和动态类型
  - 运行时环境和垃圾回收

练习

- 什么是编程语言？
  - 《A Programming Language》- 1962年
    - Applied mathematics is largely concerned with the design and analysis of explicit procedures for calculating the exact or approximate values of various functions. Such explicit procedures are called algorithms or programs.Because an effective notation for the description of programs exhibits considerable syntactic structure, it is called a programming language.
- 自然语言可以编程吗？

练习

- 从下列代码可以看出C++和Java存在哪些语言差异？

```
class MyString {
private:
    char* s;
    int size;

public:
    MyString(char* c) {
        size = strlen(c);
        s = new char[size + 1];
        strcpy(s, c);
    }

    char* get() {
        return s;
    }
    ~ MyString() {
        delete[] s;
    }
};

int main(){
    char in[] = {'C','+','+','\n'};
    MyString myStr = MyString(in);
    std::cout<<myStr.get()<<std::endl;
}
```

```
public class MyString {
    private char[] s;
    public MyString(char[] in){
        s = in;
    }
    public char[] get(){
        return s;
    }
    public static void main(String[] args) {
        MyString myStr = new MyString(
                            "java".toCharArray());
        System.out.println(myStr.s);
    }
}
```

参考文献

- http://www2.cs.uidaho.edu/~jeffery/courses/210/lecture.html#lecture13
- https://medium.com/@ayanonagon/the-y-combinator-no-not-that-one-7268d8d9c46