

# 面向IoT的程序设计语言

Deeplang

杨文章

wenzhang5800@gmail.com

2020-12-5



1. 项目背景介绍
2. Deeplang 设计目标
3. Deeplang 特性提案
4. Deeplang 工程实现
5. 未来的方向



Deeplang是由华为编程语言实验室与浙江大学竺可桢学院合作共建的语言设计与实现的开源课题。由华为实验室研究员指导，多所大学高低年级同学参与实践。

2020/4/29  
项目组 init

2020/10/29  
hello world

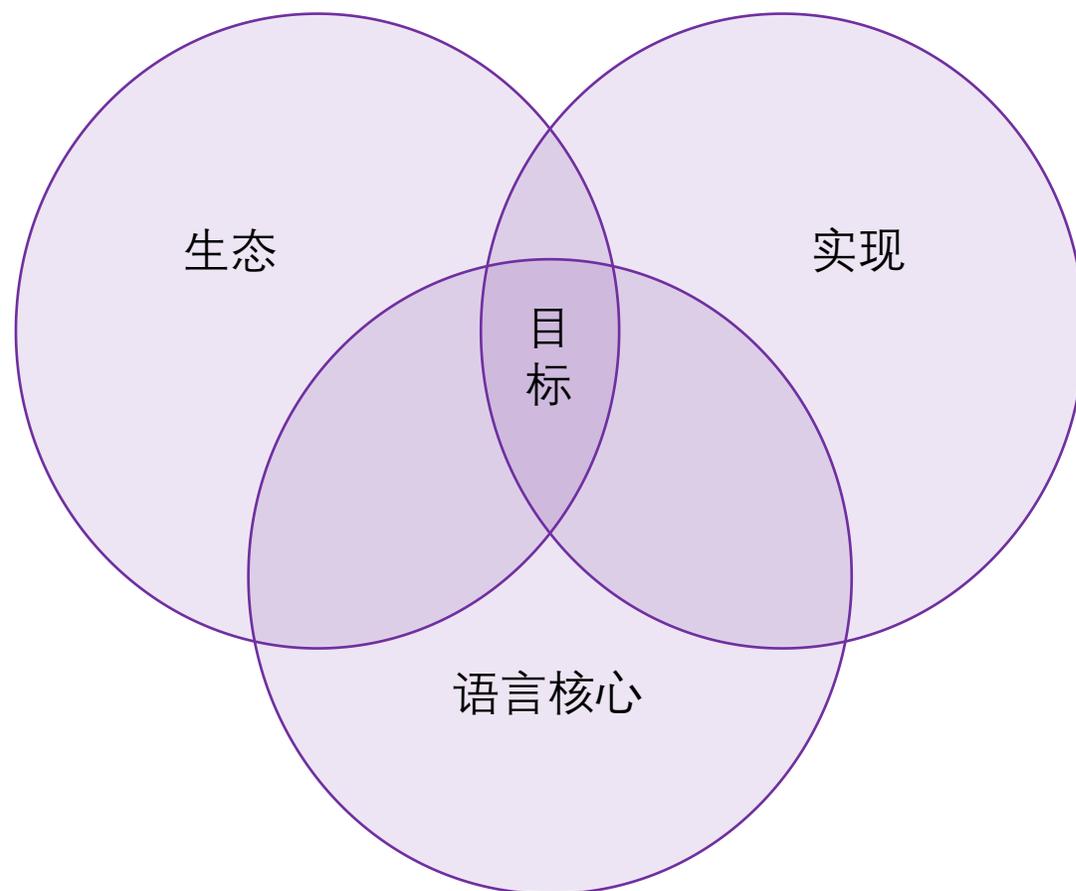
2020/12/05  
第一次公开  
报告

2020/5/07  
第一次CCB  
会议

2020/11/29  
官网建立  
<http://deeplang.org/>

2021/\*/\*  
TODO

1. 项目背景介绍
2. Deeplang 设计目标
3. Deeplang 特性提案
4. Deeplang 工程实现
5. 未来的方向



# Deeplang



Empowering everyone to build reliable and efficient software.



Combine object-oriented and functional programming in one concise, high-level language.



There should be one—and preferably only one—obvious way to do it



More easy and more safe.

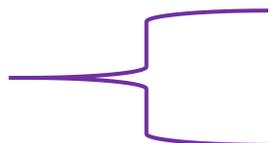


1. 项目背景介绍
2. Deeplang 设计目标
3. Deeplang 特性提案
4. Deeplang 工程实现
5. 未来的方向

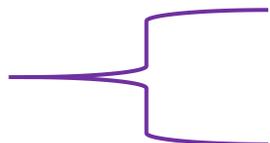
## 安全区域

Rust的内存安全学习曲线非常陡峭

Expert



Domain developer

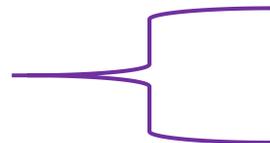


```
extern "C" {  
    fn abs(input: i32) -> i32;  
}  
  
fn main() {  
    unsafe {  
        println!("Absolute value of -3 according to C: {}", abs(-3));  
    }  
}
```

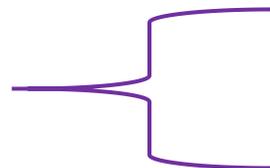
```
safe {  
    fun abs(input: i32) -> i32;  
}  
  
fun main() -> () {  
    puts(abs(-3).toString());  
}
```

# 代数数据类型

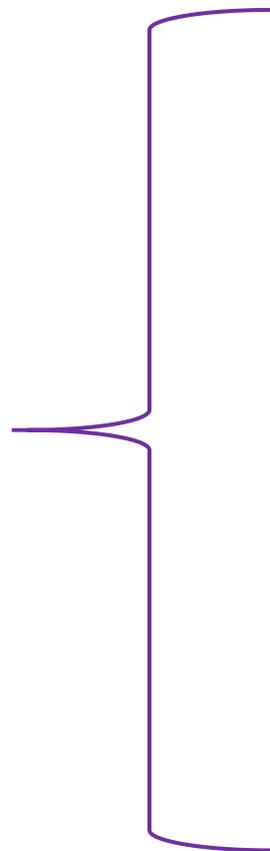
trait



Sum type



Product type



```
type Show {  
    show: This -> string  
}
```

```
type List [  
    Nil,  
    Cons(head: i32, tail: List)  
]
```

```
type Person (  
    age: i32,  
    name: string  
) impl Show {  
    fun show(this: This) -> string {  
        return "xxx"  
    }  
}
```

// desugar

```
type Person [  
    Person(age: i32, name: string)  
] impl Show {  
    fun show(this: This) -> string {  
        return "xxx"  
    }  
}
```

## 静态结构化类型

类型的相等测试由类型真实的结构而不是名字来决定。

C++ java

```
class Foo {  
    method(input: string): number { ... }  
}  
class Bar {  
    method(input: string): number { ... }  
}  
let foo: Foo = new Bar(); // ERROR!!
```

deeplang

```
type Show {  
    show: This -> string  
}  
  
type Foo () impl {  
    fun show(this: This) -> string {  
        return "xxx";  
    }  
}  
  
let foo: Show = new Foo();
```

## 委托

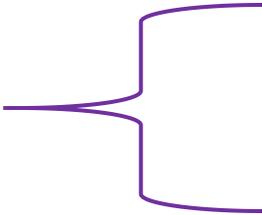
将一个子字段field的成员都委托到当前层级。

```
type Animal (  
    weight: i32  
)  
  
type Person (  
    age: i32,  
    name: string,  
    as animal: Animal    //delegate  
)  
  
let p: Person = new Person();  
p.weight // p.animal.weight
```

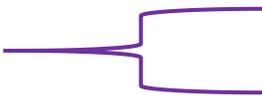
## 鸭子类型

采用组合的方式组织程序

不需要 impl Quack



满足结构即可通过类型检查



```
type Quack {
    quack: () -> ()
}

type Duck() impl Quack {
    fun quack() -> () {
        print("quaaaack");
    }
}

type Bird() impl {
    fun quack() -> () {
        print("bird quaaaack");
    }
}

fun sound(animal: Quack) -> () {
    animal.quack();
}

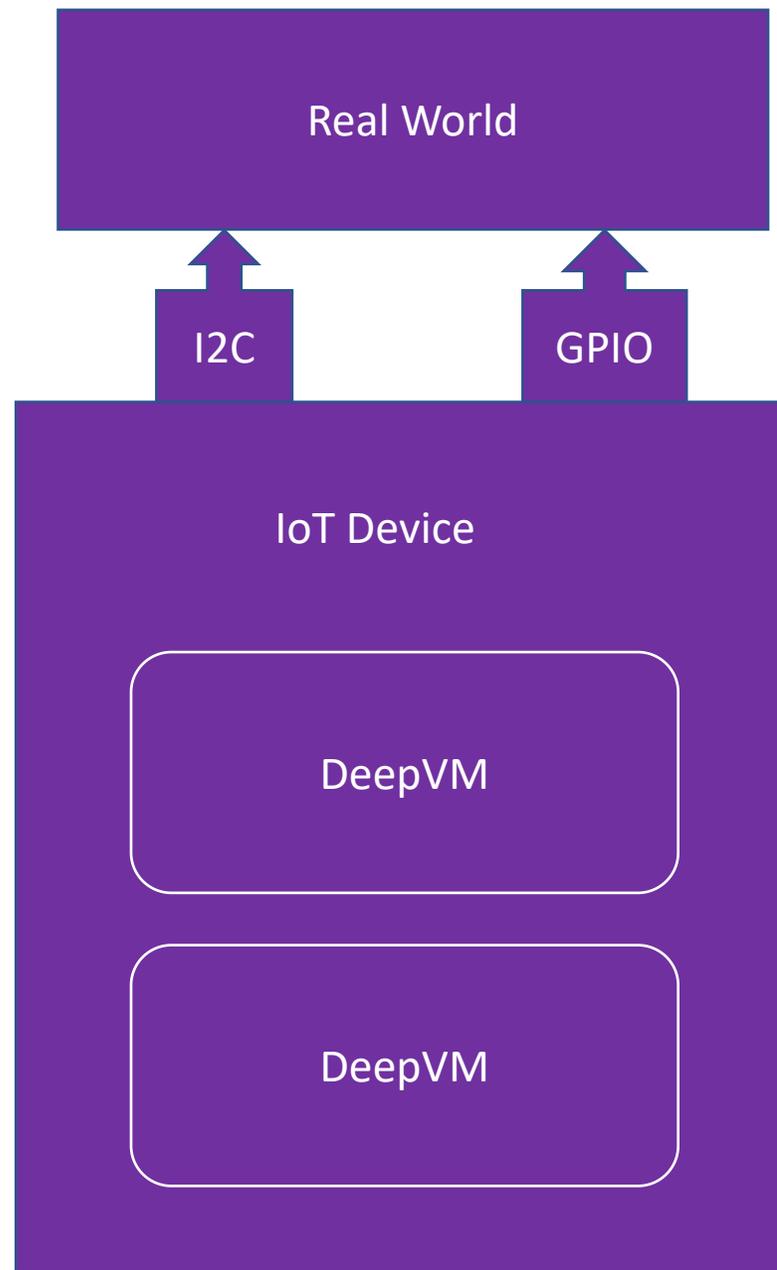
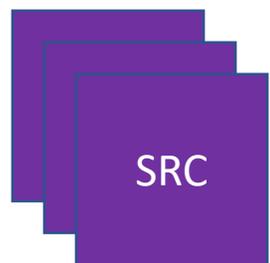
let duck: Duck = new Duck();
let bird: Bird = new Bird();

sound(duck);    // quaaaack
sound(bird);    // bird quaaaack
```



1. 项目背景介绍
2. Deeplang 设计目标
3. Deeplang 特性提案
4. Deeplang 工程实现
5. 未来的方向

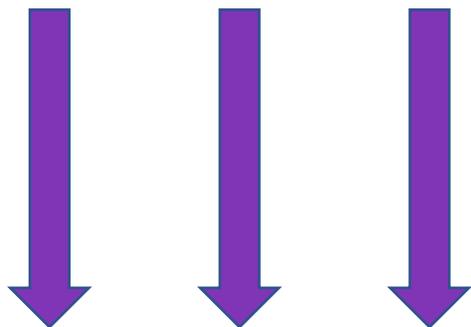
轻量化的编译器，  
不做重量级的优化





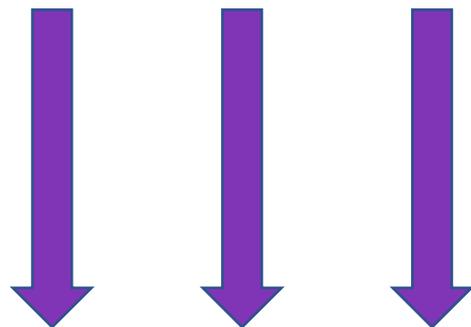
1. 项目背景介绍
2. Deeplang 设计目标
3. Deeplang 特性提案
4. Deeplang 工程实现
5. 未来的方向

1. 继续完善语言内核
2. 手写的更轻量级的lexer和parser
3. 替换wabt库
4. 新的后端, 如riscv native



Deep Compiler

1. 针对arm, riscv平台优化后端运行效率
2. jit



DeepVM

1. 拥抱IoT平台, 如鸿蒙, liteOS等。
2. 抽象硬件接口的标准库设计
3. 支持图形化debug
4. language server protocol
5. 包管理器



感谢观看