拥抱未来语言Rust|第二回 第一性原理看类型系统

原创 三角兽 三角兽 2024-03-14 12:05 四川

如何为各位看官呈现一篇既有深度又浅显易懂，还不落俗的关于类型系统的文章。思考许久，终于找到切入点，决定带大家从上帝视角审视通用类型系统，避免陷入某个特定语言的实现细节。在此，用**“5 Why”**法，去探寻类型系统的**“第一性原理”**。

------ 10个追问 ------

第一问：什么是**类型系统**？

答：是编程语言中的一个组成部分，它定义了语言中数据的类型，以及这些类型之间的操作规则。

第二问：Rust、Java、Python、汇编等几乎所有计算机语言**为什么需要类型系统**？

答：类型系统用于区分不同的数据种类，确保类型的正确性。

第三问：为什么需要**区分不同的数据种类**？

答：编译阶段，类型检查保证类型安全；运行阶段，CPU需要知道一定的信息，才能正确地执行机器指令，而这信息隐藏在编译结果产生的操作码中的。例如读写某个变量，需要确定读写的长度和对应内存的位置。在语言层面，数据类型对应着长度，变量的名称对应着内存的位置。

第四问：为什么CPU的执行需要确定**操作码**？

答：因为不同的操作码，对应着CPU中不同的的物理逻辑电路，例如+和-操作，对应着全加电路；*和/对应着乘法电路和除法电路。

第五问：为什么会有**逻辑电路**？

答：因为实现计算机的基础是逻辑电路，而逻辑电路的理论基础是布尔逻辑，它是逻辑学的一个分支。

第六问：什么是**逻辑**？

答：逻辑是一种用于分析和评估论证或推理的形式化体系，通常由命题及命题之间建立的推理关系组成

第七问：什么是**命题**？

答：通常形式是，用于判断客观或主观事实是否成立的陈述语句，由这些语句组成命题。通过判断命题的真假，形成推理，构成逻辑。从这个角度讲，逻辑能力就是将事实从模糊中分离出来，进行真假判断或推理的能力。

第八问：什么**语言**(这里指人类的语言)?

答：语言是沟通的桥梁，它起源生物群体对客观事实的心理描摹所形成的**“观念”**，这些**“观念”**在群体中需要传播和交流，于是在群体中达成共识，形成了最初的语言。

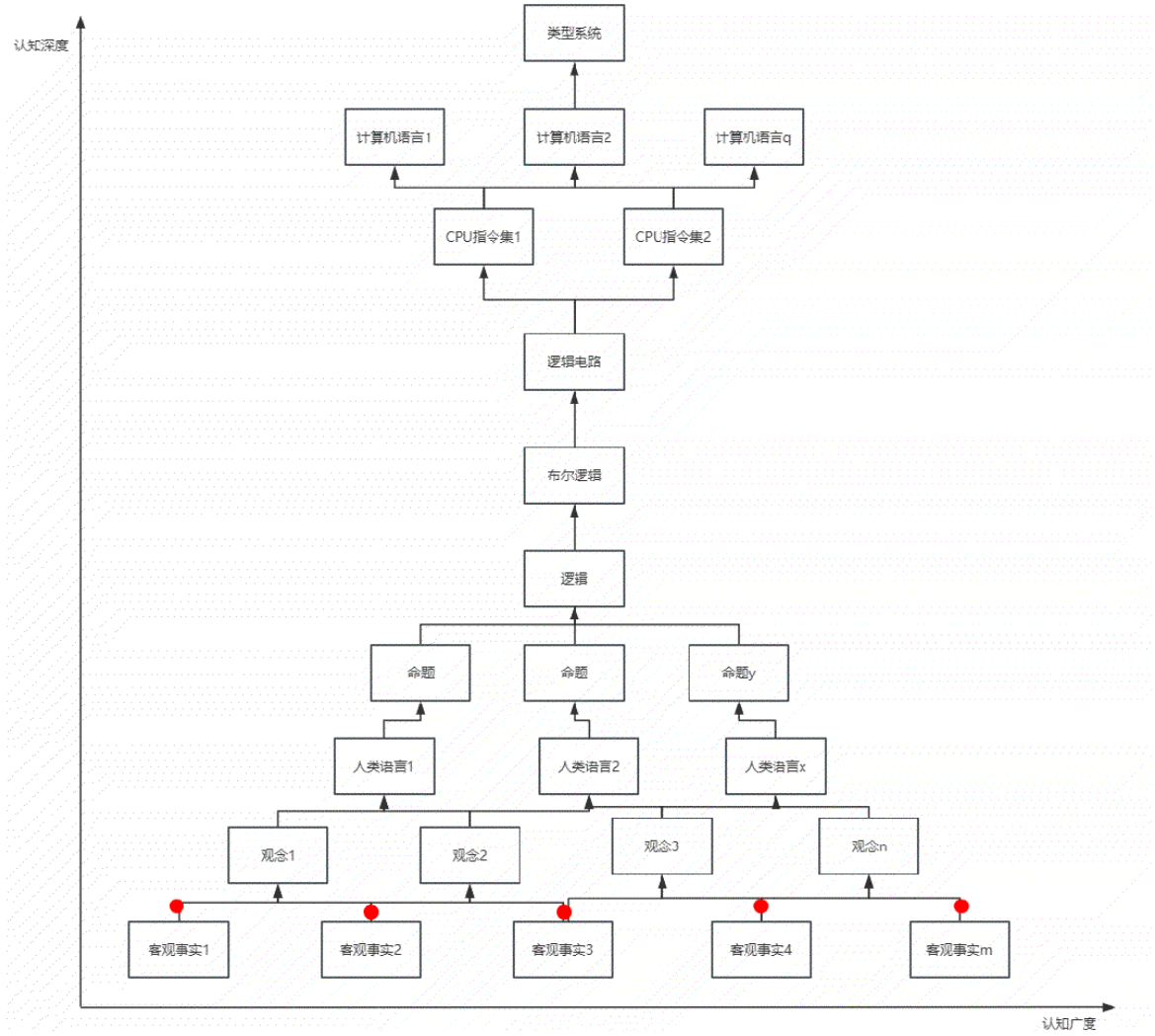
第九问：什么是**客观事实**?

答：世界本来就存在的东西。例如，这个世界存在**“猫”**这个事实，经过人类的**“心理描摹”**，有了**“猫”**的**“观念”**，群体之间需要进行沟通和交流，于是有了**“猫”**这个词语，词语的组合就形成了语言，但语言不一定就是有逻辑的语言，经常听到毫无逻辑的言论，就是这个意思。语言形成命题，组成逻辑链。再到后面的布尔逻辑，逻辑电路，计算机编程语言，及类型系统，是不是就连起来了。

第十问：……我还是别问了，哈哈哈。

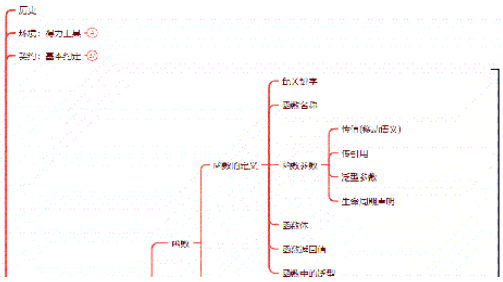
“5 Why”法不一定要5个Why，只要获取到想要的答案就可以停止了，上面其实到第四问就可以停止了，因为已经知道类型系统是干什么以及为什么需要类型系统。

类型系统是整个人类认知体系扩充的必然产物，咳咳咳，绕得有点远了，踩刹车。



------ 我准备要讲的类型系统 ------

Rust拥有最复杂的类型系统，没有之一，比我学过的Scala的类型系统还复杂(这也是很多人从入门到放弃的原因之一)，不信请看下面的脑图(图太长，做成了滚动动画**:)**)。别说图了，人看完都糊了，哈哈哈。



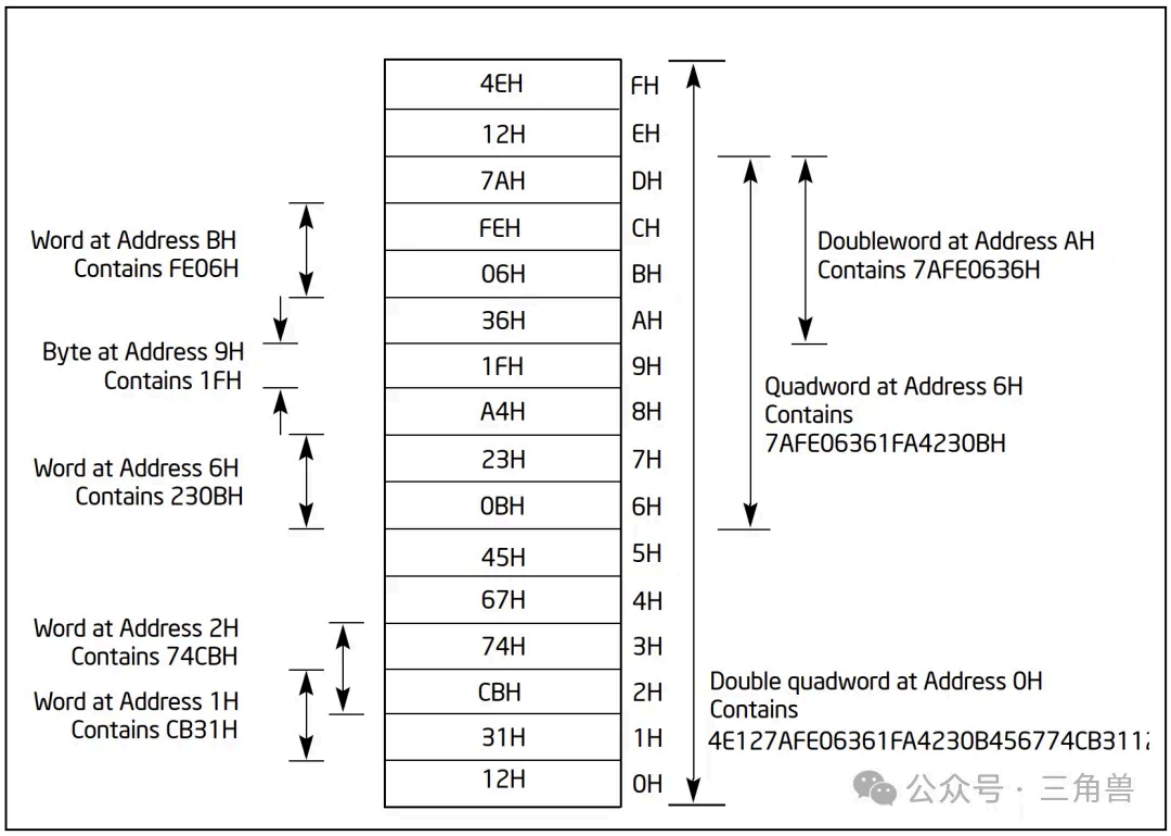
我不打算按脑图来讲，不然会被喷，内容太多显得很啰嗦，安排的60回根本就写不完，即便能写完也猴年马月去了，哈哈哈。

咱们基于**“第一性原理”**，推导和理解Rust的类型系统(其实对所有语言是通用的)，大部分类型及其意义是可以演绎出来的(学原理的重要性)。

------ 推导出类型系统 ------

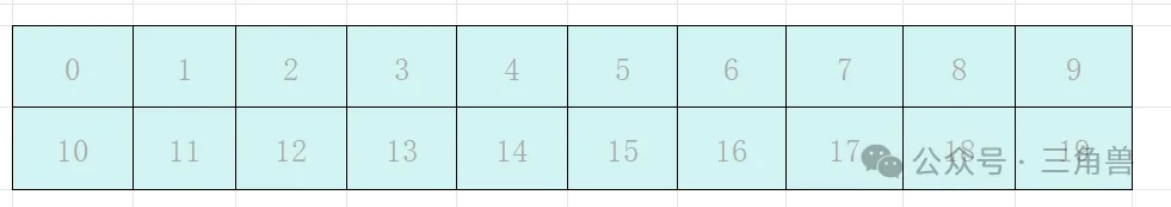
前面提到了，类型系统是人类认知发展的必然产物，不同计算机语言拥有的类型系统，其底层对应的"第一性原理"应该是类似的，接下来我们就推导一套最简单的类型系统。

------ 认识内存 ------

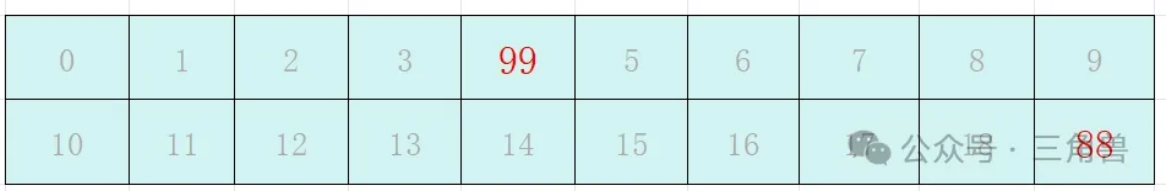


内存可以看做是一个按最小单元为1字节划分的表格，下方是低地址，上方是高地址。按内容的存放方式，可分为小端字节序和大端字节序。小端字节序是指数值的低位存储在内存低地址；大端字节序指数值的低位存储在内存的高位地址，按此规则，上图是小端字节序存储的。

按内存为4Gb计算。这个表格拥有4 * 1024 * 1024 * 1024个存储单元，数量是相当庞大的，为了简化推理，假设我们内存容量为20字节，画成表格就是20个单元格，单元格里面使用十进制数字表示内存地址。



假定现在还没有语言规范，也没有类型系统，描述对内存单元的存取操作，例如往19号格子存入88，往4号格子存入99，如何做呢?



基于这种需求，设计指令，如下：

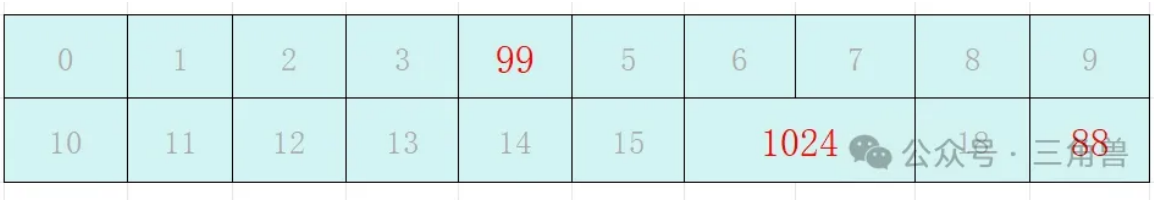
```
mov $88 (19)
mov $99 (4)
```

这两条指令表示：

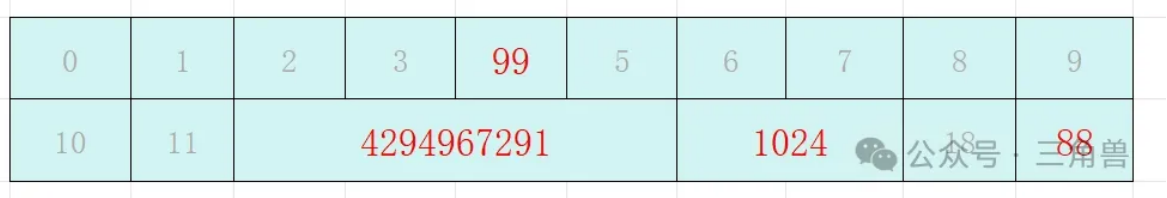
```
往19号格子 存入数字88
往4号格子  存入数字99
```

这里需要注意，每一个单元格的大小是一个字节，即8位，能够表示的数字范围如果是有符号的话，为-128 ~ 127，如果为无符号位为0 ~ 256区间。

如果有一个需求，需要存储1024，怎么办？很显然，一个格子存不下，那我们就用两个格子。



如果要存储我的工资4294967291怎么办呢？哈哈，过于夸张了。那我就用四个格子呗。



此时，前面的汇编语法就得重新设计了，指令需要表明占用的字节数和数据类型。对于占用一个字节的整数，我们用movb表示，对于占用两个字节的整数，我们用movw表示，对于占用四个字节的整数我们用movl表示，以此类推可以设计出更多的指令。

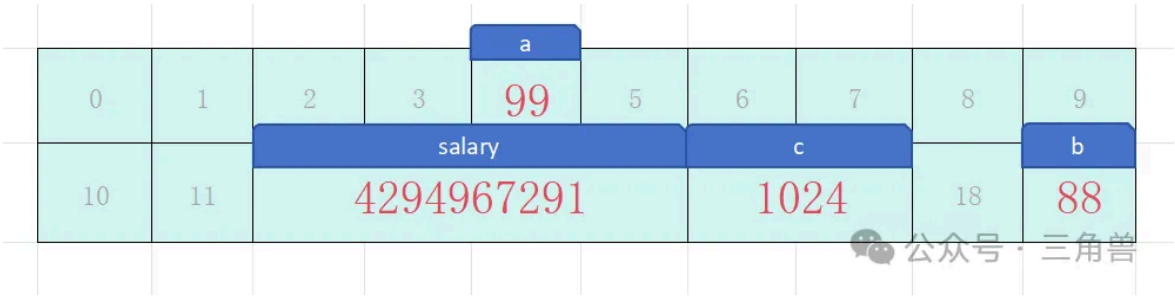
```
movb $88 (19)
movb $99 (4)
movw $1024 (16)
movl $4294967291 (12)
```

对应着如下的描述：

```
往19号格子 存入数字 88 ， 占用1个字节
往4号格子  存入数字 99 ， 占用1个字节
往16号格子 存入数字 1024 ， 占用2个字节
往12号格子 存入数字 4294967291 ， 占用4个字节
```

不知不觉，类型系统就有了一个雏形。但是还有优化的地方，对于高级语言，肯定是不能直接指定内存地址的，都是用一个标号替代具体的内存地址，高级语言中定义的变量在编译链接阶段，被转换为虚拟内存地址，经过MMU内存管理单元最终转换为真实的物理内存地址。

回过头看上面的表格，现在我们的内存已经显得非常凌乱了，还能记起来12号格子存的什么内容吗？那下面我们来优化类型体系，给类型体系加上一个标签试一试。用标签表示格子位置，这样就不用记住那些无意义的格子编号了。



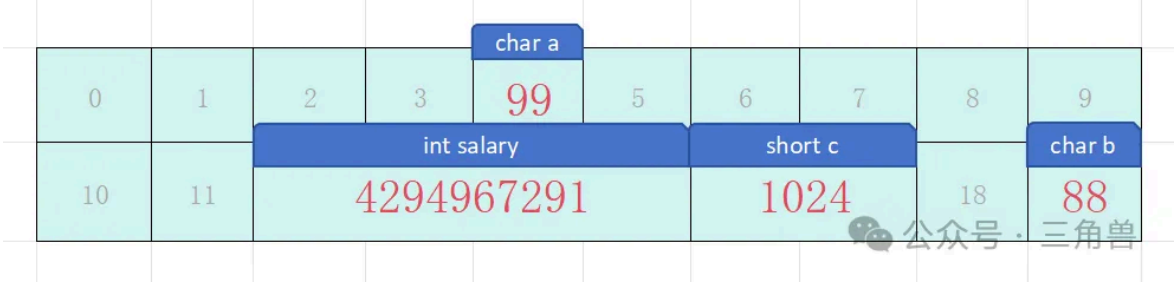
有了标签之后，往格子里面存储数据就不用指定内存网格地址了，高级语言在编译阶段，会将“标签”转换为虚拟地址，最终映射到真实的物理内存上，经过“标签”的抽象，使得程序员无需关心底层具体内存地址的细节，不得不说，真是一个伟大的发明。

那上面定义的汇编又可以改一改了。

```
movb $88 b
movb $99 a
movw $1024 c
movl $4294967291 salary
```

程序员只管定义“标签”(高级语言里面的变量)，剩下的就是编写计算逻辑，后面的事儿就交给编译器了(编译器可真累)。至此我们完成了数据的存储，通过定义标签和定义的指令，知道将值存储到内存的哪个位置和占用多少字节。那反过来，如果我们要取出刚刚存入的salary标签的值，怎么做呢？

具体的描述应该是，找到salary标签所在的内存地址并取出四个字节返回，标签对应了内存地址，但怎么确定要取回几个字节呢？显然我们的类型系统还需要一个“标识”，用于指明一次性操作几个字节，没错这就是高级语言里面的类型了，例如int，byte等。



带类型带标签的写法，其实就是和大多数高级语言类似的语法了，下面是C语言的写法。

```
char a = 99;
char b = 88;
short c = 1024;
int salary = 4294967291;
```

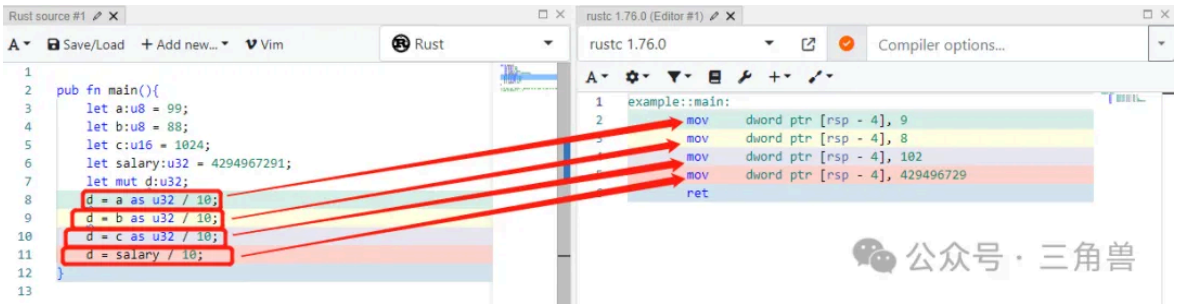
而对于Rust，写法也是类似的：

```
let a:u8 = 99;
let b:u8 = 88;
let c:u16 = 1024;
let salary:u32 = 4294967291;
```

借助在线工具[Compiler Explorer]

```
https://godbolt.org
```

可以方便地看到Rust和编译后汇编语言的对应关系。



高级语言中定义的变量其实是给程序员看的，CPU执行的时候变量已经变成地址了，类型也会变成具体的操作指令。在高级语言中的类型系统比我们定义的“初级系统”复杂和丰富得多，涵盖整型、浮点型、布尔类型、字符类型、数组、元组、结构体、枚举、trait特征、泛型、集合、裸指针及智能指针等等。

不知不觉，我们已经进入到了Rust的类型系统了，哈哈哈。后面几个回合，我打算挑重点来写，因为学会了重点，其它都是可以演绎的。

另外文章里面也不打算嵌入过多的代码，在我看来非常影响排版和浏览体验，只会贴出关键代码并加以注释，所有的项目代码，可以点击**“阅读原文”**进入GitHub查看，代码中的注解更

详细哦，大家顺手点个星，来个**“一键三连”**就是对我最大的鼓励和支持，兽兽在此感谢各位看官了。

------ 关于本系列 ------

本系列的开篇词看这

三角兽新系列！拥抱未来语言Rust

番外篇看这里

拥抱未来语言Rust|番外篇 我的成长故事

前面的基础章节看这里

拥抱未来语言Rust|第一回 环境准备

更多的资料和项目源码可以点击**阅读原文**

最后，祝大家都能追更到系列结束，只要你敢持续追更，并且把每一回的内容搞懂，我就敢让你在系列结束后说一句，我对Rust很熟悉。

您的**一键三连**，就是我持续更新和走下去最大的动力，兽兽在此**鞠躬感谢！**

另外，本系列完全免费，希望大家能多多传播给同样喜欢的人！我们下回见。

拥抱未来语言Rust 61 # 技术的第一性原理 3

拥抱未来语言Rust · 目录 ≡

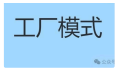
< 上一篇
拥抱未来语言Rust|第一回 环境准备

下一篇 >
拥抱未来语言Rust|第三回 基本约定

阅读原文

别找了，Rust设计模式都在这里了，设计模式之工厂模式

三角兽



性能炸裂！10万行数据耗时500毫秒，使用rust的nom框架实现时序数据库协议解析(下)

三角兽