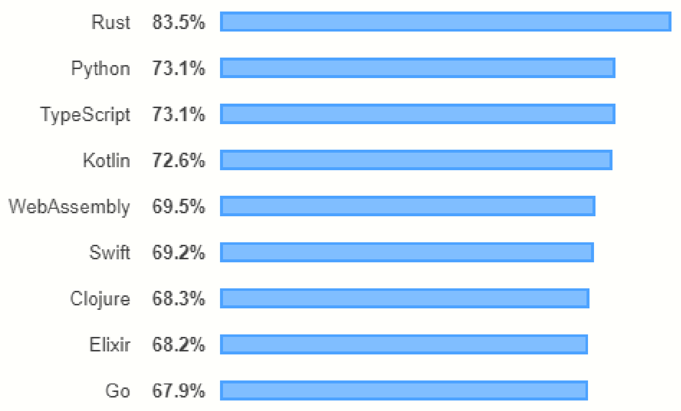
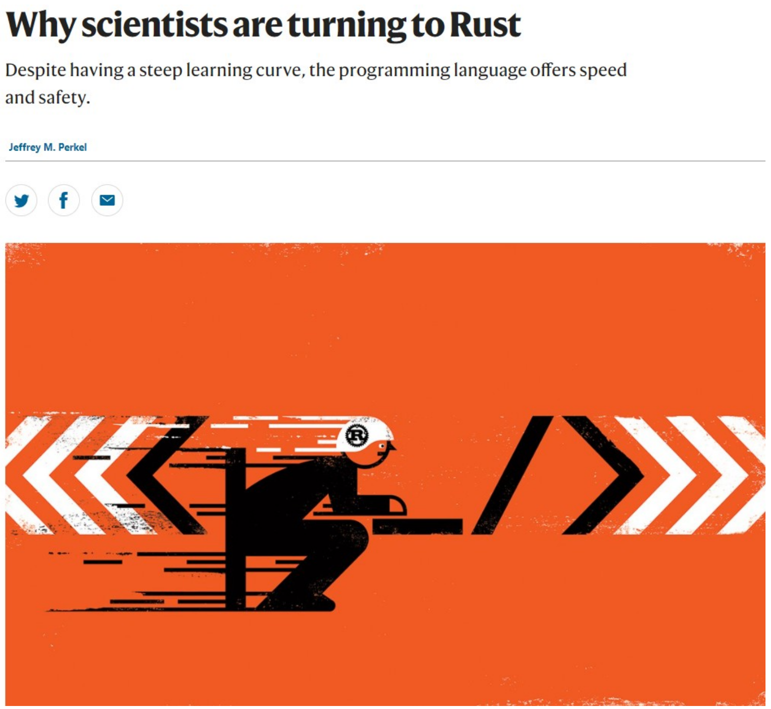
**Yijun Yu ( 俞一峻 )**

可信编程首席专家
华为可信软件工程和开源实验室
英国开放大学计算机通信系

# Rust 带来越来越多的学术创新

- Rust 在 Stack Overflow 连续五年被评为“最受开发者欢迎”的编程语言
- 统计在编程语言（PLDI，POPL），软件工程（ICSE，ASE）等学术顶会近年论文发表情况，增长趋势十分明显
- 12 月在 Nature 发表的《Why Scientists are Turning to Rust》将掀起更大的热情去研究

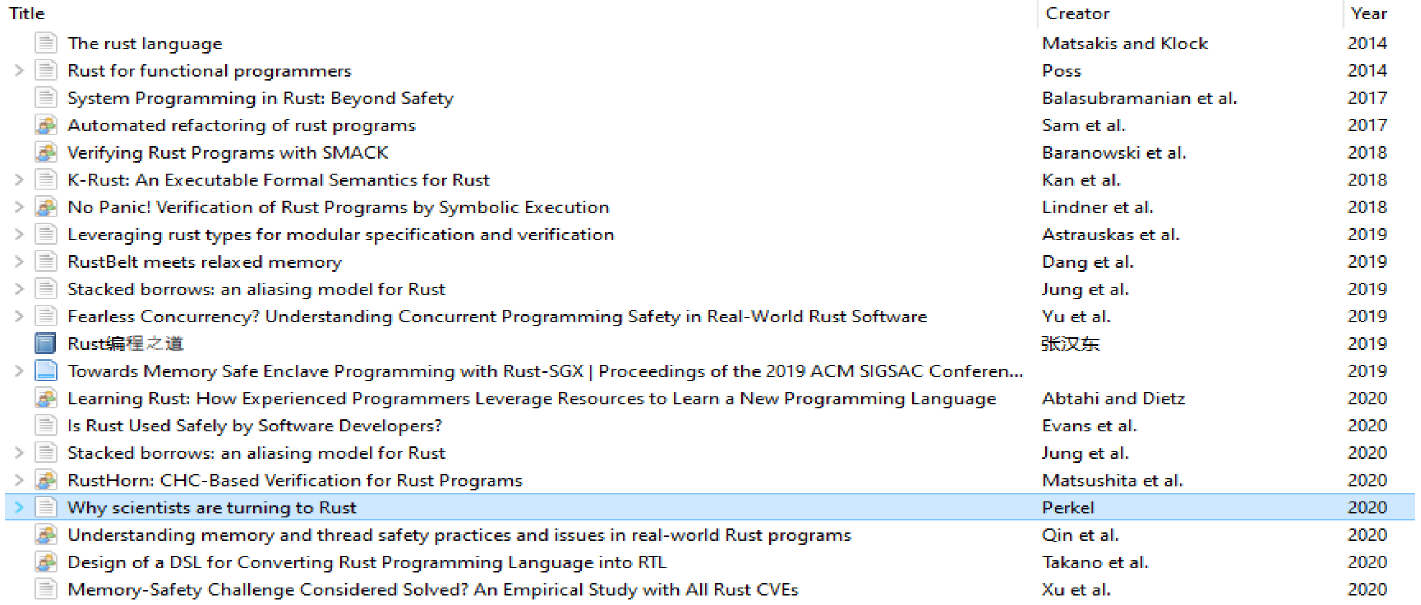




## Why scientists are turning to Rust

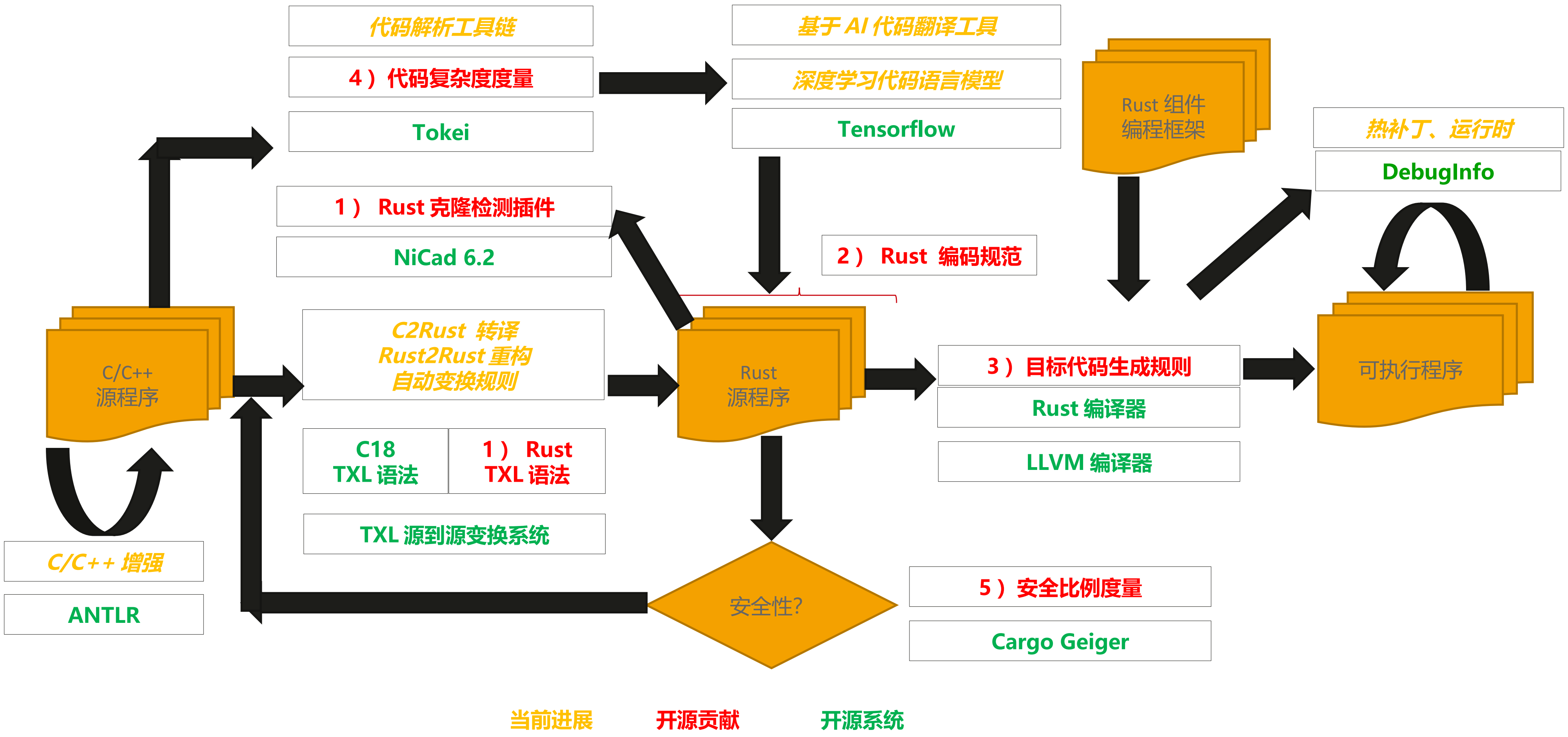
Despite having a steep learning curve, the programming language offers speed and safety.

Jeffrey M. Perkel



| Title | Creator | Year |
|---|---|---|
| The rust language | Matsakis and Klock | 2014 |
| Rust for functional programmers | Poss | 2014 |
| System Programming in Rust: Beyond Safety | Balasubramanian et al. | 2017 |
| Automated refactoring of rust programs | Sam et al. | 2017 |
| Verifying Rust Programs with SMACK | Baranowski et al. | 2018 |
| K-Rust: An Executable Formal Semantics for Rust | Kan et al. | 2018 |
| No Panic! Verification of Rust Programs by Symbolic Execution | Lindner et al. | 2018 |
| Leveraging rust types for modular specification and verification | Astrauskas et al. | 2019 |
| RustBelt meets relaxed memory | Dang et al. | 2019 |
| Stacked borrows: an aliasing model for Rust | Jung et al. | 2019 |
| Fearless Concurrency? Understanding Concurrent Programming Safety in Real-World Rust Software | Yu et al. | 2019 |
| Rust编程之道 | 张汉东 | 2019 |
| Towards Memory Safe Enclave Programming with Rust-SGX \| Proceedings of the 2019 ACM SIGSAC Conferen... |  | 2019 |
| Learning Rust: How Experienced Programmers Leverage Resources to Learn a New Programming Language | Abtahi and Dietz | 2020 |
| Is Rust Used Safely by Software Developers? | Evans et al. | 2020 |
| Stacked borrows: an aliasing model for Rust | Jung et al. | 2020 |
| RustHorn: CHC-Based Verification for Rust Programs | Matsushita et al. | 2020 |
| Why scientists are turning to Rust | Perkel | 2020 |
| Understanding memory and thread safety practices and issues in real-world Rust programs | Qin et al. | 2020 |
| Design of a DSL for Converting Rust Programming Language into RTL | Takano et al. | 2020 |
| Memory-Safety Challenge Considered Solved? An Empirical Study with All Rust CVEs | Xu et al. | 2020 |

# 华为对 Rust 语言和技术的一些探索



# Rust/LLVM 编译器

- AArch64 ILP32 Big Endian 是 ARM 优化设计的芯片指令集。
- 针对该指令集， LLVM 编译器无法生成目标代码。
- 因为 Rust 编译器基于 LLVM 实现，这是我们向产品部门技术转化的技术困难
  。
- Rust Team 核心成员加入后，项目组改进了：
  - LLVM 编译器，实现了 ILP32 大端目标代码输出：（aarch64_be-unknown-linux-gnu_ilp32 ）
  - Rust 编译器，实现了构建工具链一站式的交叉编译代码输出：cargo build --target aarch64_be-unknown-linux-gnu_ilp32
- 完成标准类库的目标代码输出，在华为内部 Rust 产品上得到验证。
- 对 Rust 编译器，项目组将继续改进、增加语言特性，比如 AOP 等

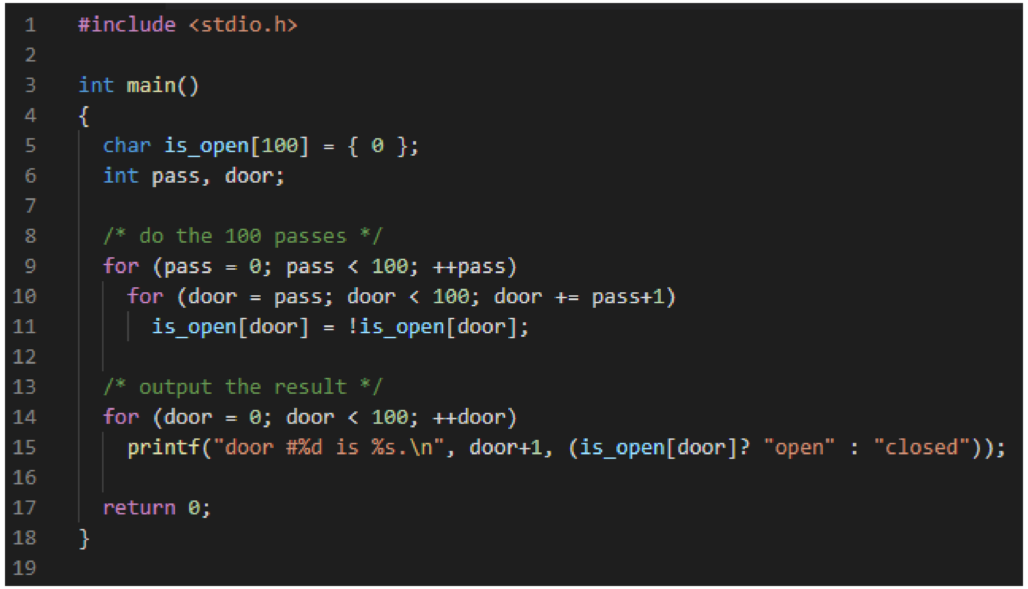
# 支撑用 Rust 开发嵌入式系统组件

提供 Rust 语言的并发编程框架，简化用户异步编程，提供丰富的运行时库。

**关键能力：**

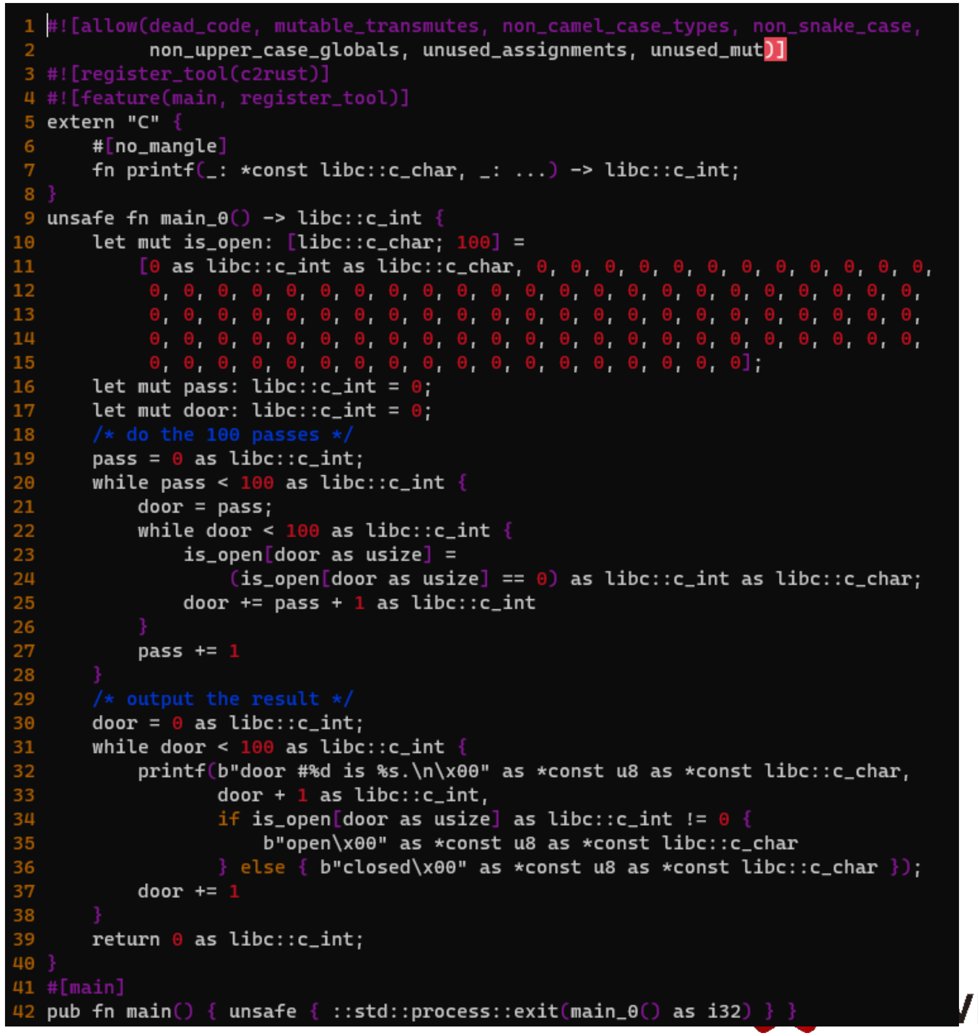
1. **编程模型**：
   提供 Actor 编程模型，利用 Rust 实现同步编程异步运行的能力。
2. **调度框架**：
   利用 Rust 的 future 与 await 机制实现用户态的无栈协程调度，及其容错机制、协程同步原语等
   。
3. **通信框架**：
   提供 Actor 之间的消息互通，支撑各类通信协议，实现 Rust 组件与非 Rust 组件的互通。
4. **运行时**：
   提供高性能、高可靠性的运行时库，提供 Rust 的热补丁能力。

# 可信代码平滑迁移：从 C/C++ 到 Rust



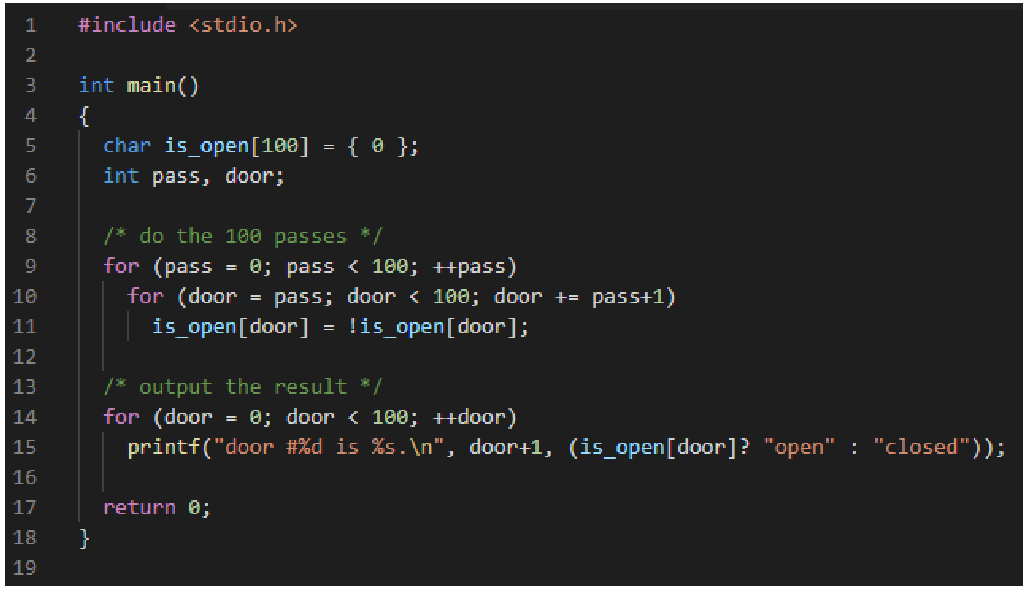
```c
#include <stdio.h>

int main()
{
  char is_open[100] = { 0 };
  int pass, door;

  /* do the 100 passes */
  for (pass = 0; pass < 100; ++pass)
    for (door = pass; door < 100; door += pass+1)
      is_open[door] = !is_open[door];

  /* output the result */
  for (door = 0; door < 100; ++door)
    printf("door #%d is %s.\n", door+1, (is_open[door]? "open" : "closed"));

  return 0;
}

```

c2rust.com



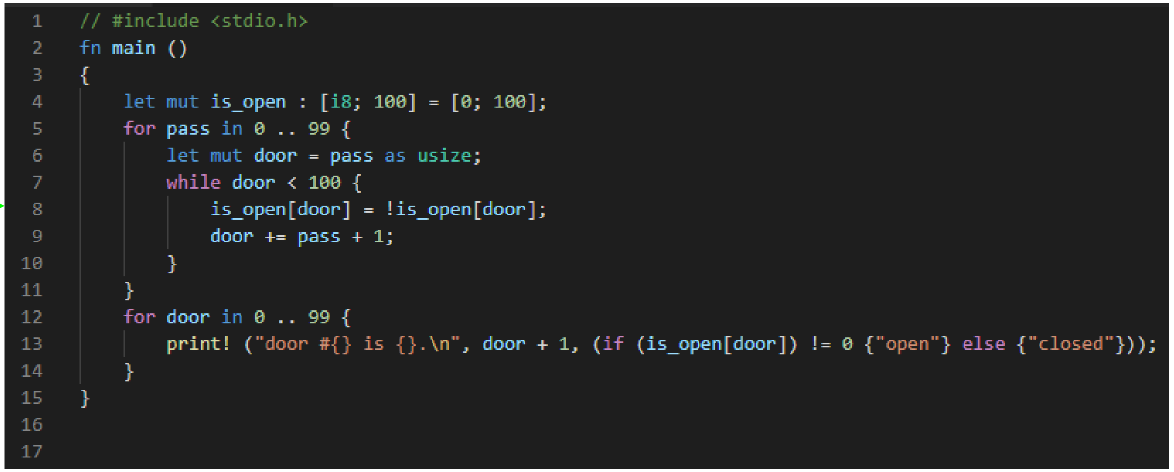
```rust
#![allow(dead_code, mutable_transmutes, non_camel_case_types, non_snake_case,
         non_upper_case_globals, unused_assignments, unused_mut)]
#![register_tool(c2rust)]
#![feature(main, register_tool)]
extern "C" {
    #[no_mangle]
    fn printf(_: *const libc::c_char, _: ...) -> libc::c_int;
}
unsafe fn main_0() -> libc::c_int {
    let mut is_open: [libc::c_char; 100] =
        [0 as libc::c_int as libc::c_char, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    let mut pass: libc::c_int = 0;
    let mut door: libc::c_int = 0;
    /* do the 100 passes */
    pass = 0 as libc::c_int;
    while pass < 100 as libc::c_int {
        door = pass;
        while door < 100 as libc::c_int {
            is_open[door as usize] =
                (is_open[door as usize] == 0) as libc::c_int as libc::c_char;
            door += pass + 1 as libc::c_int
        }
        pass += 1
    }
    /* output the result */
    door = 0 as libc::c_int;
    while door < 100 as libc::c_int {
        printf(b"door #%d is %s.\n\x00" as *const u8 as *const libc::c_char,
               door + 1 as libc::c_int,
               if is_open[door as usize] as libc::c_int != 0 {
                   b"open\x00" as *const u8 as *const libc::c_char
               } else { b"closed\x00" as *const u8 as *const libc::c_char });
        door += 1
    }
    return 0 as libc::c_int;
}
#[main]
pub fn main() { unsafe { ::std::process::exit(main_0() as i32) } }
```

# 可信代码平滑迁移： Rust TXL 语法



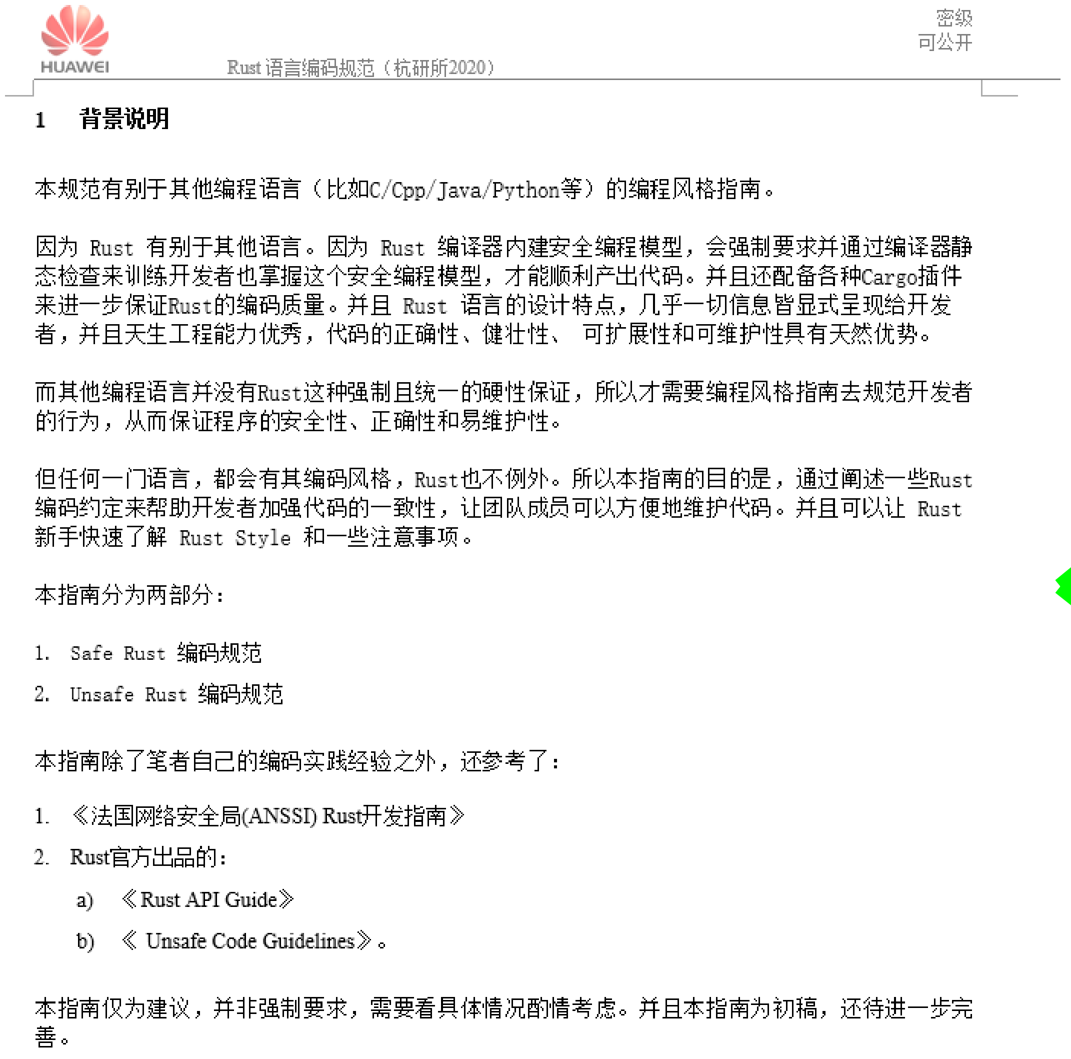
```c
#include <stdio.h>

int main()
{
  char is_open[100] = { 0 };
  int pass, door;

  /* do the 100 passes */
  for (pass = 0; pass < 100; ++pass)
    for (door = pass; door < 100; door += pass+1)
      is_open[door] = !is_open[door];

  /* output the result */
  for (door = 0; door < 100; ++door)
    printf("door #%d is %s.\n", door+1, (is_open[door]? "open" : "closed"));

  return 0;
}

```

TXL



```rust
// #include <stdio.h>
fn main ()
{
    let mut is_open : [i8; 100] = [0; 100];
    for pass in 0 .. 99 {
        let mut door = pass as usize;
        while door < 100 {
            is_open[door] = !is_open[door];
            door += pass + 1;
        }
    }
    for door in 0 .. 99 {
        print! ("door #{} is {}.\n", door + 1, (if (is_open[door]) != 0 {"open"} else {"closed"}));
    }
}


```

# Rust 社区建设： Rust 编码规范



HUAWEI　Rust 语言编码规范（杭研所2020）

密级
可公开

## 1　背景说明

本规范有别于其他编程语言（比如C/Cpp/Java/Python等）的编程风格指南。

因为 Rust 有别于其他语言。因为 Rust 编译器内建安全编程模型，会强制要求并通过编译器器静态检查来训练开发者也掌握这个安全编程模型，才能顺利产出代码。并且还配备各种Cargo插件来进一步保证Rust的编码质量。并且 Rust 语言的设计特点，几乎一切信息皆显式呈现给开发者，并且天生工程能力优秀，代码的正确性、健壮性、 可扩展性和可维护性具有天然优势。

而其他编程语言并没有Rust这种强制且统一的硬性保证，所以才需要编程风格指南去规范开发者的行为，从而保证程序的安全性、正确性和易维护性。

但任何一门语言，都会有其编码风格，Rust也不例外。所以本指南的目的是，通过阐述一些Rust编码约定来帮助开发者加强代码的一致性，让团队成员可以方便地维护代码。并且可以让 Rust 新手快速了解 Rust Style 和一些注意事项。

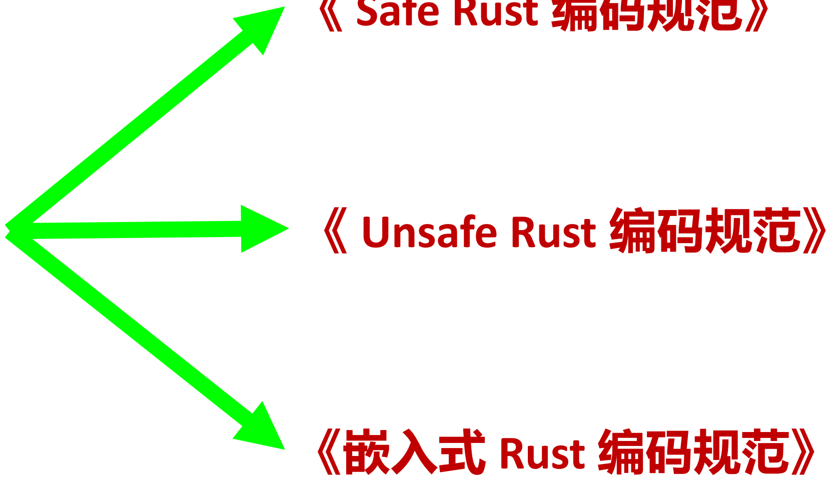
本指南分为两部分：

1. Safe Rust 编码规范
2. Unsafe Rust 编码规范

本指南除了笔者自己的编码实践经验之外，还参考了：

1. 《法国网络安全局(ANSSI) Rust开发指南》
2. Rust官方出品的：
   a) 《Rust API Guide》
   b) 《 Unsafe Code Guidelines》。

本指南仅为建议，并非强制要求，需要看具体情况酌情考虑。并且本指南为初稿，还待进一步完善。



《 Safe Rust 编码规范》

《 Unsafe Rust 编码规范》

《嵌入式 Rust 编码规范》

# Rust 社区建设：代码度量工具 Tokei

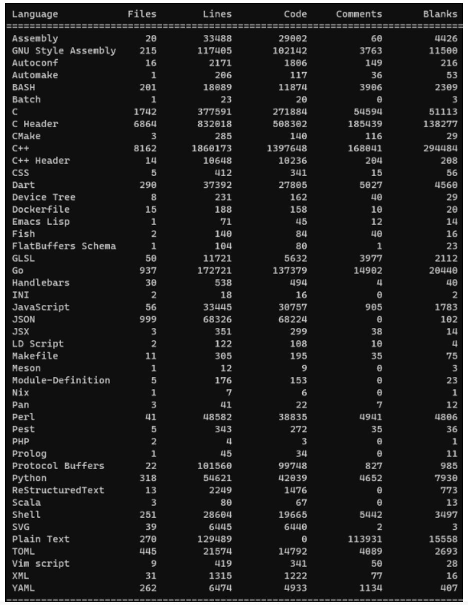
**Tokei** 作为开源代码度量工具，能够针对项目中不同编程语言（识别多达２００多种）统计代码行和复杂度等有用信息。 https://github.com/XAMPPRocky/tokei

例如， Google Fuschia 使用 Tokei 工具统计 C /C++， Rust 等编程语言的代码行数（右上图），呈现出三种编程语言代码量的演化（见右下图）。

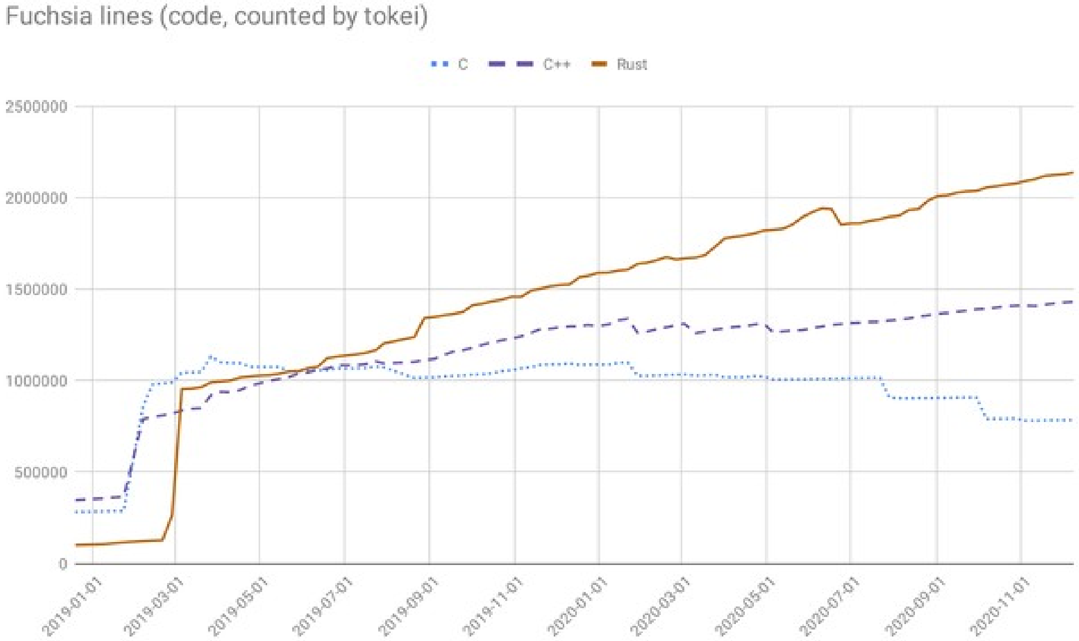
但是， Tokei 的统计代码行的数据结构使用了多个串行的计数器，当项目含有很多文件时，这些计数任务必须串行进行。项目组需要对多个语言统计代码度量，实现了如下功能：

- 修改了 Tokei 的数据结构，把影响并行化的计数器改写为可并行化的参数，更改了部分相关的 API
- 修改了 Tokei 的命令参数，根据识别出来的程序代码文件类型增加了批处理等。

前者是对 Tokei 项目的改进，后者是对 Tokei 项目的扩充。



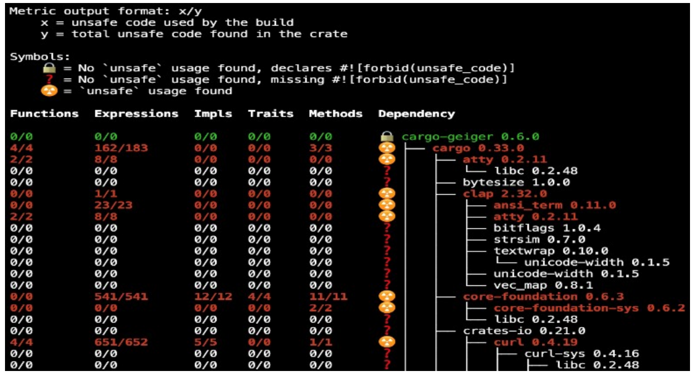
```
Language            Files        Lines         Code     Comments       Blanks
=============================================================================
Assembly               20        33488        29002           60         4426
GNU Style Assembly    215       117405       102142         3763        11500
Autoconf               16         2171         1806          149          216
Automake                1          206          117           36           53
BASH                  201        18089        11874         3906         2309
Batch                   1           23           20            0            3
C                    1742       377591       271884        54594        51113
C Header             6864       832018       508302       185439       138277
CMake                   3          285          140          116           29
C++                  8162      1860173      1397648       168041       294484
C++ Header             14        10648        10236          204          208
CSS                     5          412          341           15           56
Dart                  290        37392        27805         5027         4560
Device Tree             8          231          162           40           29
Dockerfile             15          188          158           10           20
Emacs Lisp              1           71           45           12           14
Fish                    2          140           84           40           16
FlatBuffers Schema      1          104           80            1           23
GLSL                   50        11721         5632         3977         2112
Go                    937       172721       137379        14902        20440
Handlebars             30          538          494            4           40
INI                     2           18           16            0            2
JavaScript             56        33445        30757          905         1783
JSON                  999        68326        68224            0          102
JSX                     3          351          299           38           14
LD Script               2          122          108           10            4
Makefile               11          305          195           35           75
Meson                   1           12            9            0            3
Module-Definition       5          176          153            0           23
Nix                     1            7            6            0            1
Pan                     3           41           22            7           12
Perl                   41        48582        38835         4941         4806
Pest                    5          343          272           35           36
PHP                     2            4            3            0            1
Prolog                  1           45           34            0           11
Protocol Buffers       22       101560        99748          827          985
Python                318        54621        42039         4652         7930
ReStructuredText       13         2249         1476            0          773
Scala                   3           80           67            0           13
Shell                 251        28604        19665         5442         3497
SVG                    39         6445         6440            2            3
Plain Text            270       129489            0       113931        15558
TOML                  445        21574        14792         4089         2693
Vim script              9          419          341           50           28
XML                    31         1315         1222           77           16
YAML                  262         6474         4933         1134          407
-----------------------------------------------------------------------------
```



Fuchsia lines (code, counted by tokei)

# Rust 社区建设：安全代码度量工具 Cargo-Geiger

**cargo-geiger** 作为开源安全代码度量工具，能够针对项目中的 unsafety 关键字，分类统计函数，表达式，结构，实现，接口，相关性等的不安全要素个数（参见右图1）。
https://github.com/rust-secure-code/cargo-geiger

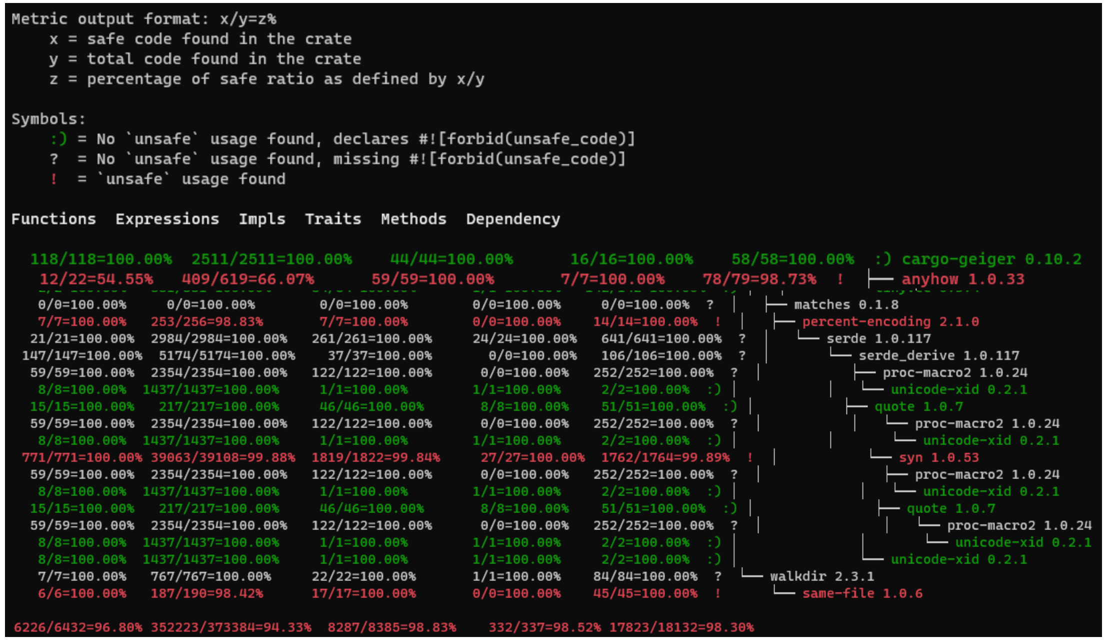


```
Metric output format: x/y
    x = unsafe code used by the build
    y = total unsafe code found in the crate

Symbols:
    🔒 = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?  = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    ☢  = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

0/0        0/0          0/0    0/0     0/0      🔒 cargo-geiger 0.6.0
4/4        162/183      0/0    0/0     3/3      ☢  ├── cargo 0.33.0
2/2        8/8          0/0    0/0     0/0      ☢  │   ├── atty 0.2.11
0/0        0/0          0/0    0/0     0/0      ?  │   │   └── libc 0.2.48
0/0        0/0          0/0    0/0     0/0      ?  │   ├── bytesize 1.0.0
0/0        1/1          0/0    0/0     0/0      ☢  │   ├── clap 2.32.0
0/0        23/23        0/0    0/0     0/0      ☢  │   │   ├── ansi_term 0.11.0
2/2        8/8          0/0    0/0     0/0      ☢  │   │   ├── atty 0.2.11
0/0        0/0          0/0    0/0     0/0      ?  │   │   ├── bitflags 1.0.4
0/0        0/0          0/0    0/0     0/0      ?  │   │   ├── strsim 0.7.0
0/0        0/0          0/0    0/0     0/0      ?  │   │   ├── textwrap 0.10.0
0/0        0/0          0/0    0/0     0/0      ?  │   │   │   └── unicode-width 0.1.5
0/0        0/0          0/0    0/0     0/0      ?  │   │   ├── unicode-width 0.1.5
0/0        0/0          0/0    0/0     0/0      ?  │   │   └── vec_map 0.8.1
0/0        541/541      12/12  4/4     11/11    ☢  │   ├── core-foundation 0.6.3
0/0        0/0          0/0    0/0     2/2      ☢  │   │   ├── core-foundation-sys 0.6.2
0/0        0/0          0/0    0/0     0/0      ?  │   │   └── libc 0.2.48
0/0        0/0          0/0    0/0     0/0      ?  │   ├── crates-io 0.21.0
4/4        651/652      5/5    0/0     1/1      ☢  │   │   ├── curl 0.4.19
0/0        0/0          0/0    0/0     0/0      ?  │   │   │   ├── curl-sys 0.4.16
0/0        0/0          0/0    0/0     0/0      ?  │   │   │   │   ├── libc 0.2.48
```

图 1. Geiger 安全报告

但是，该统计不反映代码中安全要素的比例。通过改进 **cargo-geiger**，项目组实现了如下功能：

- **统计安全代码要素的个数；**
- **报告安全代码要素的百分比（参见右图2）。**



```
Metric output format: x/y=z%
    x = safe code found in the crate
    y = total code found in the crate
    z = percentage of safe ratio as defined by x/y

Symbols:
    :)  = No `unsafe` usage found, declares #![forbid(unsafe_code)]
    ?   = No `unsafe` usage found, missing #![forbid(unsafe_code)]
    !   = `unsafe` usage found

Functions  Expressions  Impls  Traits  Methods  Dependency

  118/118=100.00%  2511/2511=100.00%   44/44=100.00%      16/16=100.00%    58/58=100.00%  :) cargo-geiger 0.10.2
   12/22=54.55%   409/619=66.07%     59/59=100.00%      7/7=100.00%    78/79=98.73%  !  ├── anyhow 1.0.33
    0/0=100.00%     0/0=100.00%       0/0=100.00%       0/0=100.00%     0/0=100.00%  ?  │   │ ├── matches 0.1.8
    7/7=100.00%   253/256=98.83%      7/7=100.00%       0/0=100.00%    14/14=100.00%  !  │   │ ├── percent-encoding 2.1.0
   21/21=100.00%  2984/2984=100.00%   261/261=100.00%     24/24=100.00%   641/641=100.00%  ?  │     └── serde 1.0.117
  147/147=100.00%  5174/5174=100.00%    37/37=100.00%      0/0=100.00%   106/106=100.00%  ?  │         └── serde_derive 1.0.117
   59/59=100.00%  2354/2354=100.00%   122/122=100.00%     0/0=100.00%   252/252=100.00%  ?  │             ├── proc-macro2 1.0.24
    8/8=100.00%  1437/1437=100.00%     1/1=100.00%      1/1=100.00%     2/2=100.00%  :) │             │   └── unicode-xid 0.2.1
   15/15=100.00%   217/217=100.00%    46/46=100.00%      8/8=100.00%    51/51=100.00%  :) │             ├── quote 1.0.7
   59/59=100.00%  2354/2354=100.00%   122/122=100.00%     0/0=100.00%   252/252=100.00%  ?  │             │   └── proc-macro2 1.0.24
    8/8=100.00%  1437/1437=100.00%     1/1=100.00%      1/1=100.00%     2/2=100.00%  :) │             │       └── unicode-xid 0.2.1
  771/771=100.00% 39063/39108=99.88%  1819/1822=99.84%    27/27=100.00%  1762/1764=99.89%  !  │             └── syn 1.0.53
   59/59=100.00%  2354/2354=100.00%   122/122=100.00%     0/0=100.00%   252/252=100.00%  ?  │                 ├── proc-macro2 1.0.24
    8/8=100.00%  1437/1437=100.00%     1/1=100.00%      1/1=100.00%     2/2=100.00%  :) │                 │   └── unicode-xid 0.2.1
   15/15=100.00%   217/217=100.00%    46/46=100.00%      8/8=100.00%    51/51=100.00%  :) │                 ├── quote 1.0.7
   59/59=100.00%  2354/2354=100.00%   122/122=100.00%     0/0=100.00%   252/252=100.00%  ?  │                 │   └── proc-macro2 1.0.24
    8/8=100.00%  1437/1437=100.00%     1/1=100.00%      1/1=100.00%     2/2=100.00%  :) │                 │       └── unicode-xid 0.2.1
    8/8=100.00%  1437/1437=100.00%     1/1=100.00%      1/1=100.00%     2/2=100.00%  :) │                 └── unicode-xid 0.2.1
    7/7=100.00%   767/767=100.00%    22/22=100.00%      1/1=100.00%    84/84=100.00%  ?  └── walkdir 2.3.1
    6/6=100.00%   187/190=98.42%     17/17=100.00%      0/0=100.00%    45/45=100.00%  !      └── same-file 1.0.6

6226/6432=96.80% 352223/373384=94.33%  8287/8385=98.83%    332/337=98.52% 17823/18132=98.30%
```

图 2. Geiger 安全代码报告（改进后）

产品线 Rust 项目可以定期运行 Geiger 工具，输出度量报告。

# Rust 前沿研究进展：智能化代码学习

- **从存量代码学习算法分类任务：**
  C/C++ 数据集 104 类算法题， 52,000 程序
  >基于语法树达到 94% 算法分类准确度 [AAAI'16]
  >基于抽象语法树达到 98% 算法分类准确度 [ICSE'19]
  >基于胶囊网络达到 98.5% 算法分类准确度 [AAAI'21]
- **跨语言学习算法分类任务：**
  GitHub 数据集， Rosetta Code 数据集
  >基于图学习达到 86% 算法分类准确度 [SANER'19]
- **跨任务自学习：**以上多个数据集
  >算法分类、代码推荐、补全、搜索 [ICSE'21]

[Workshop][NeurIPS'19] **TreeCaps:Tree-Structured Capsule Networks for Program Source Code Processing**, by Vinoj JAYASUNDARA, Nghi D. Q. BUI, Lingxiao JIANG, David LO, in Thirty-fourth Conference on Neural Information Processing Systems (NeurIPS), Workshop on Machine Learning for Systems, 2019, Vancouver, Canada

[Rank A][ASE'19] **AutoFocus: Interpreting Attention-based Neural Networks by Code Perturbation**, by Nghi D. Q. BUI, Yijun YU, Lingxiao JIANG, in Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE 2019), Research Track, New Ideas Papers, San Diego, California, United States, 2019

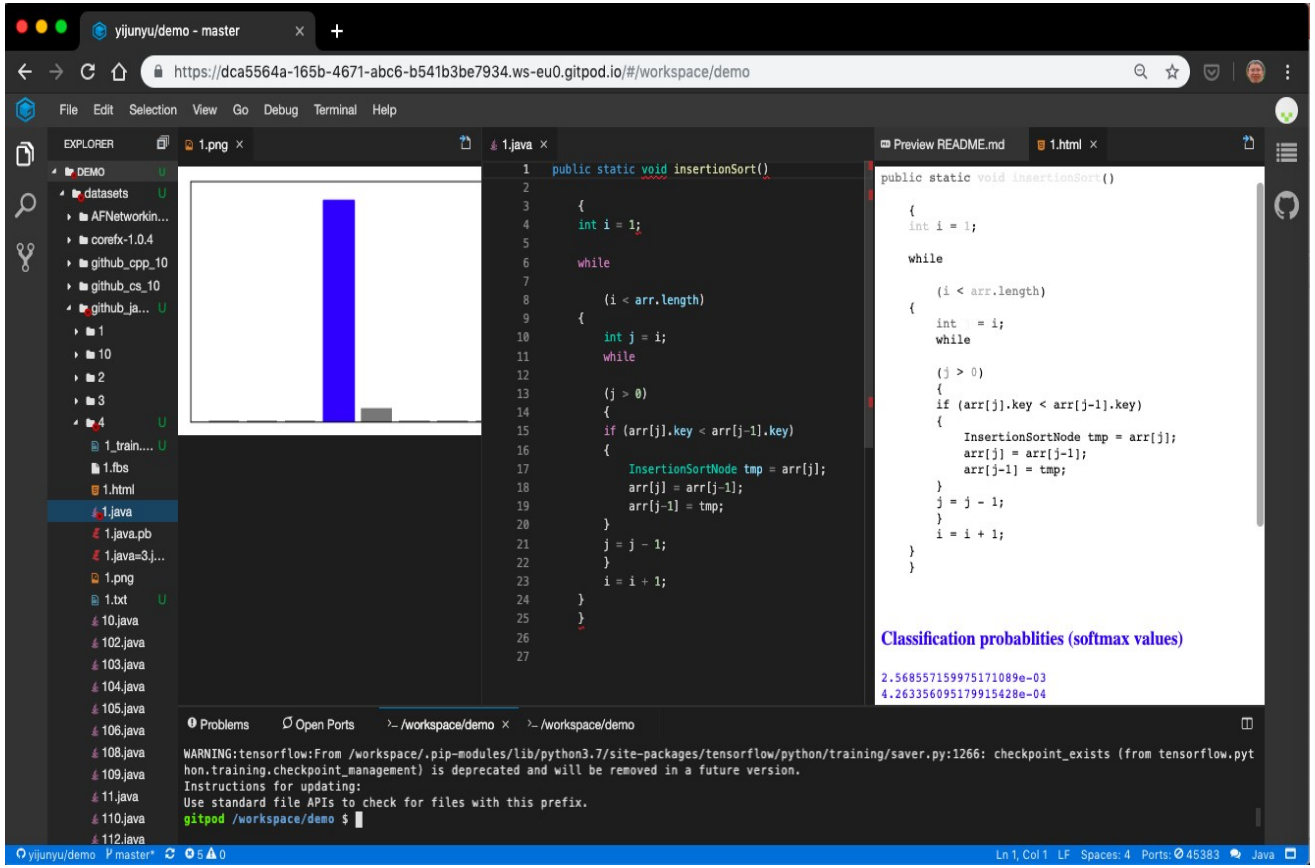
[Rank A*][ESEC/FSE'19] **SAR: Learning Cross-Language API Mappings with Little Knowledge** , by Nghi D. Q. Bui, Yijun Yu, Lingxiao Jiang , accepted at the IEEE/ACM 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), Research Track, Tallinn, Estonia, 2019

[Rank A*][ICSE'19] **Towards Zero Knowledge Learning for Cross-Language API Mappings** , by Nghi D. Q. BUI, in Proceedings of the IEEE/ACM 41th International Conference on Software Engineering (ICSE): ACM Student Research Competition Track (SRC), Montreal, Canada, 2019 - (**Bronze Medal**)

[Rank B][SANER'19] **Bilateral Dependency Neural Networks for Cross-Language Algorithm Classification**, by Nghi D. Q. Bui, Yijun Yu, Lingxiao Jiang, in the 26th edition of the IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), Research Track, Zhejiang University in Hangzhou, February 24-27, 2019

[Rank A*][ICSE'18] **Hierarchical Learning of Cross-Language Mappings through Distributed Vector Representations for Code** , by Nghi D. Q. Bui, Lingxiao JIANG, in Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE): New Ideas and Emerging Technologies Results Track ( NIER), Gothenburg, Sweden, 2018 - (**ACM SIGSOFT Distinguished Paper Award**)

[Workshop][NL4SE-AAAI'17] **Cross-Language Learning for Program Classification Using Bilateral Tree-Based Convolutional Neural Networks** , by Nghi D. Q. Bui, Lingxiao JIANG, and Yijun YU . In the proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI) Workshop on NLP for Software Engineering, New Orleans, Lousiana, USA, 2018.



```
public static void insertionSort()

    {
    int i = 1;

    while

        (i < arr.length)
    {
        int j = i;
        while

        (j > 0)
        {
        if (arr[j].key < arr[j-1].key)
        {
            InsertionSortNode tmp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = tmp;
        }
        j = j - 1;
        }
        i = i + 1;
    }
    }
```

```
public static void insertionSort()

    {
    int i = 1;

    while

        (i < arr.length)
    {
        int j = i;
        while

        (j > 0)
        {
        if (arr[j].key < arr[j-1].key)
        {
            InsertionSortNode tmp = arr[j];
            arr[j] = arr[j-1];
            arr[j-1] = tmp;
        }
        j = j - 1;
        }
        i = i + 1;
    }
    }
```

Classification probablities (softmax values)

```
2.568557159975171089e-03
4.263356095179915428e-04
```

```
WARNING:tensorflow:From /workspace/.pip-modules/lib/python3.7/site-packages/tensorflow/python/training/saver.py:1266: checkpoint_exists (from tensorflow.pyt
hon.training.checkpoint_management) is deprecated and will be removed in a future version.
Instructions for updating:
Use standard file APIs to check for files with this prefix.
gitpod /workspace/demo $
```

IDE 集成插件