sub0

Rust 在 Substrate 开发框架中的使用

孙凯超

内容

- Rust 简介
- Rust 特性
- Why blockchain
- 什么是Substrate
- Substrate Runtime 组件
- Substrate 应用开发

Bugs from Chrome

Stable Channel Update for Desktop

Thursday, October 10, 2019

The Stable channel has been updated to 77.0.3865.120 for Windows, Mac, and Linux. This will roll out over the coming days/weeks. A list of all changes is available in the log.

Security Fixes and Rewards

Note: Access to bug details and links may be kept restricted until a majority of users are updated with a fix. We will also retain restrictions if the bug exists in a third party library that other projects similarly depend on, but haven't yet fixed.

This update includes 8 security fixes. Below, we highlight fixes that were contributed by external researchers. Please see the Chrome Security Page for more information.

[$20500][1005753] **High** CVE-2019-13693: Use-after-free in IndexedDB.
Reported by Guang Gong of Alpha Team, Qihoo 360 on 2019-09-19
[$TBD][1005251] **High** CVE-2019-13694: Use-after-free in WebRTC.
Reported by bananapenguin on 2019-09-18
[$15000][1004730] **High** CVE-2019-13695: Use-after-free in audio.
Reported by Man Yue Mo of Semmle Security Research Team on 2019-09-17
[$7500][1000635] **High** CVE-2019-13696: Use-after-free in V8.
Reported by Guang Gong of Alpha Team, Qihoo 360 on 2019-09-04

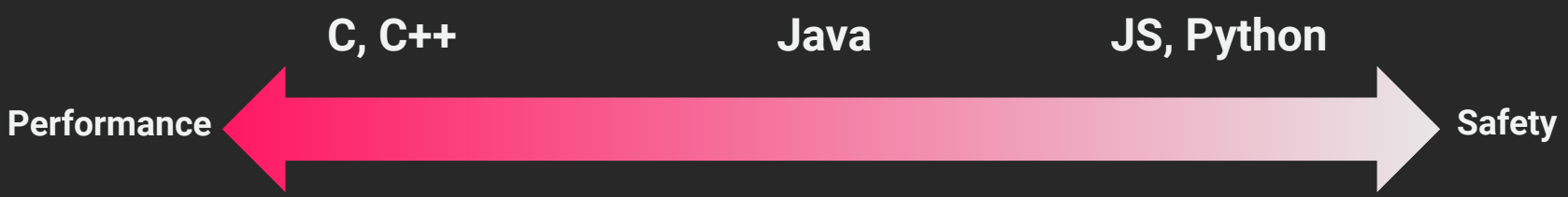
[$2000][990849] **High** CVE-2019-13697: Cross-origin size leak.
Reported by Luan Herrera @lbherrera_ on 2019-08-05

We would also like to thank all security researchers that worked with us during the development cycle to prevent security bugs from ever reaching the Stable channel.

As usual, our ongoing internal security work was responsible for a wide range of fixes:

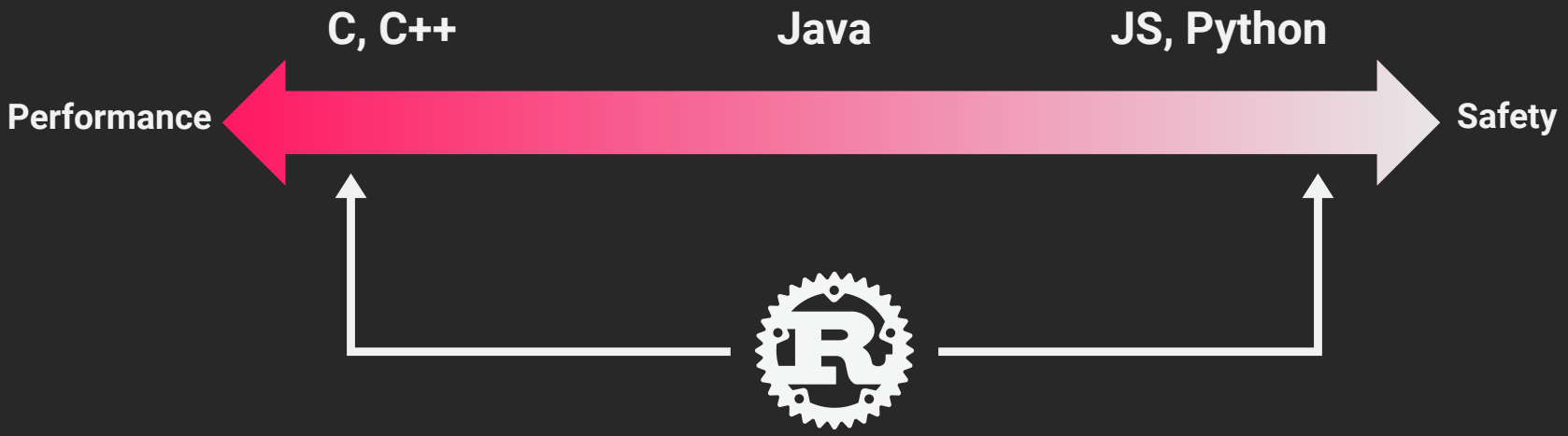
- [1011875] Various fixes from internal audits, fuzzing and other initiatives

Rust 简介

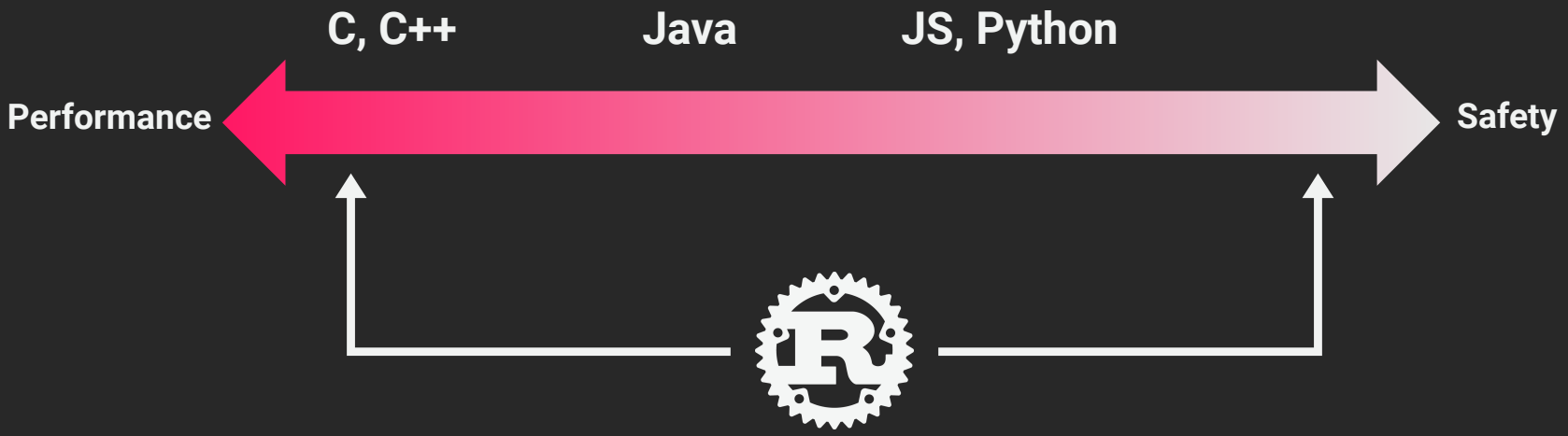


parity

Rust 简介



Rust 简介



Rust 简介

A language empowering everyone to build reliable and efficient software.

- 2015年发布Rust 1.0
- 内存安全、高效和并发
- 无GC
- 支持函数式编程
- 强大友好的编译器

parity

Rust 特性 - 内存安全

Rust 的内存安全是由Ownership来保证，在编译期间检查，规则：

- 每个值都有一个owner
- 同一时间只有一个owner
- 当owner离开作用域之后，值被丢弃

Rust 特性 - ownership

```
fn print_sum(v: Vec<i32>) {
    println!("{}", v[0] + v[1]);

}


fn main() {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }

    print_sum(v);

    println!("Compile error: {}", v);

}
```

Rust 特性 - ownership

```
fn print_sum(v: Vec<i32>) {
    println!("{}", v[0] + v[1]);

}
```

deallocation

```
fn main() {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }

    print_sum(v);

    println!("Compile error: {}", v);

}
```

create resource

move ownership

error!

Rust 特性 - copy

```
fn print_sum(a: i32, b: i32) {
    println!("{}", a + b);

}

fn main() {
    let a = 35;
    let b = 42;

    print_sum(a, b);


    println!("We still have {} and {}", a, b);

}
```

Rust 特性 - copy

```
fn print_sum(a: i32, b: i32) {
    println!("{}", a + b);

}
```

deallocation

```
fn main() {
    let a = 35;
    let b = 42;

    print_sum(a, b);


    println!("We still have {} and {}", a, b);

}
```

create resource

copied

no error

deallocation

Rust 特性 - borrow ownership

```
fn print_sum(v: &Vec<i32>) {
    println!("{}", v[0] + v[1]);

}

fn main() {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }


    print_sum(&v);

    println!("We still have v: {}, {}, ...", v[0], v[1]);

}
```

Rust 特性 - borrow ownership

```
fn print_sum(v: &Vec<i32>) {
    println!("{}", v[0] + v[1]);

}
```

drop reference

```
fn main() {
    let mut v = Vec::new();
    for i in 1..1000 {
        v.push(i);
    }


    print_sum(&v);

    println!("We still have v: {}, {}, ...", v[0], v[1]);

}
```

create resource

borrow v

no error

deallocation

Rust 特性 - 并发

Ownership 使并发变得简单：

- Message passing
  - fn send<T: Send>(chan: &Channel<T>, t: T);
  - fn recv<T: Send>(chan: &Channel<T>) -> T;
- Lock
  - fn mutex<T: Send>(t: T) -> Mutex<T>; // create a new mutex
  - fn lock<T: Send>(mutex: &Mutex<T>) -> MutexGuard<T>; // acquire the lock
  - fn access<T: Send>(guard: &mut MutexGuard<T>) -> &mut T; // access the data protected by the lock

Rust in blockchain

Why Rust:

- All the above features
- WebAssembly

例子:

- Substrate
- Libra

parity

Why blockchain?

web 2.0 开发

后端：

- 开发语言：Java, Ruby
- 框架：Spring，Rails
- 数据库：Postgres，MySQL
- 自动化测试
- CI / CD
- 部署云服务：AWS，阿里云

前端：HTML, Javascript, CSS

- React
- Vue
- Angular

web 2.0

中心应用的问题:

- 难以保证不作恶
- 用户隐私问题
- 代码安全
- 隐藏成本高
- 服务不可用
- ……

web 2.0 Vs web 3.0

中心应用的问题:

- 难以保证不作恶
- 用户隐私问题
- 代码安全
- 隐藏成本高
- 服务不可用
- ……

去中心应用 – 解决方案

- 分叉、链上治理
- 用户拥有数据主权
- 开源可审查
- 分享权益
- 永不离线
- ……

parity

Chain-less

协议:

- IPFS
- Matrix
- BitTorrent

特点:

- DHT
- 自驱动
- 数据存储、索引

parity

Chain-less Vs blockchain

协议：

- IPFS
- Matrix
- BitTorrent

特点：

- DHT
- 自驱动
- 数据存储、索引

协议：

- Bitcoin
- Ethereum
- Polkadot

特点：

- 区块结构，链式存储
- 激励机制
- 数据交易

parity

Smart contract

特点:

- Gas 费用
- 沙盒环境
- 链上存储租赁
- 状态回滚

Smart contract Vs application chain

smart contract 特点:

- Gas 费用
- 沙盒环境
- 链上存储租赁
- 状态回滚

app chain特点:

- Runtime 安全有开发者完全负责
- 获取链上所有状态
- 高度定制化，包括共识，通证，交易方式

parity

Substrate简介

一个开源、模块化、可扩展的区块链开发框架，涵盖了区块链的核心组件：

- Database layer
- P2P
- PoS
- Transaction pool
- Full / light client
- Runtime modules

parity

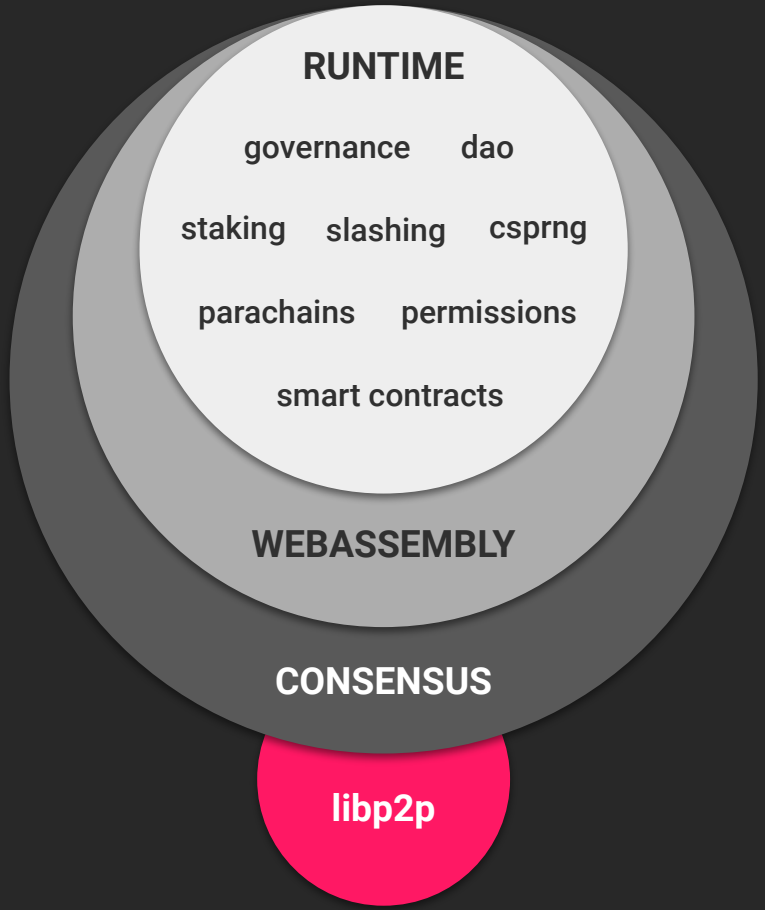
Rust

编译为Wasm

Wasm在链上存储

Rust

编译成本地可执行文件



Substrate Runtime 组件

Substrate Runtime Module Library 提供一系列的即插即用的功能，如资产管理、共识、合约、自治等。

你也可以开发自己的Runtime组件。

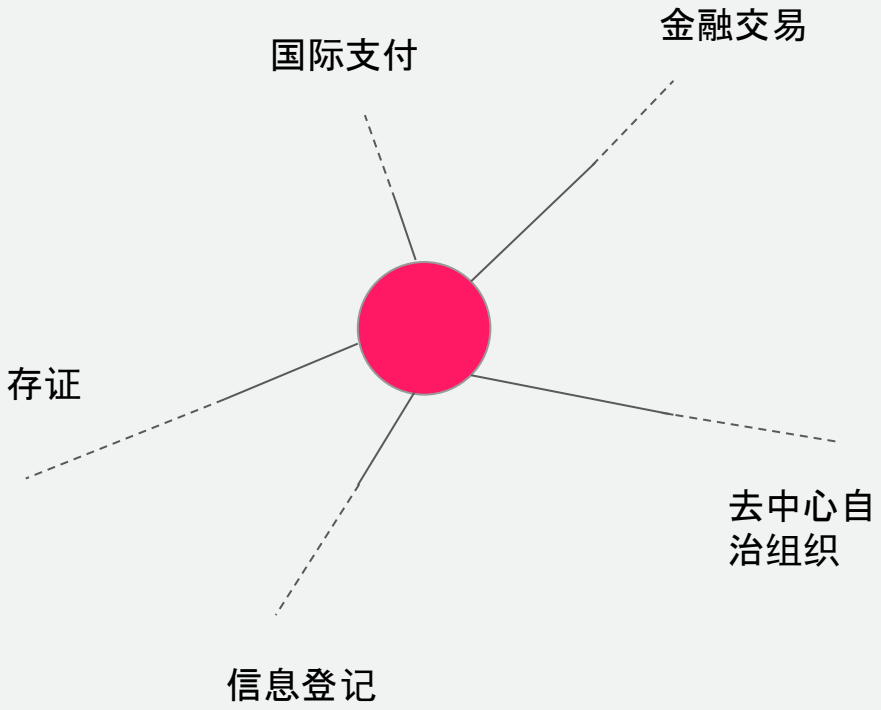
Substrate Runtime Module Library (SRML)			
assets	aura	balances	consensus
contract	council	democracy	executive
treasury	grandpa	indices	metadata
session	staking	sudo	system
timestamp	finality-grandpa	and more	...

parity

应用场景

去中心化技术的特点包括，永不离线、开源审查、数据加密、保护隐私、分享权益等。

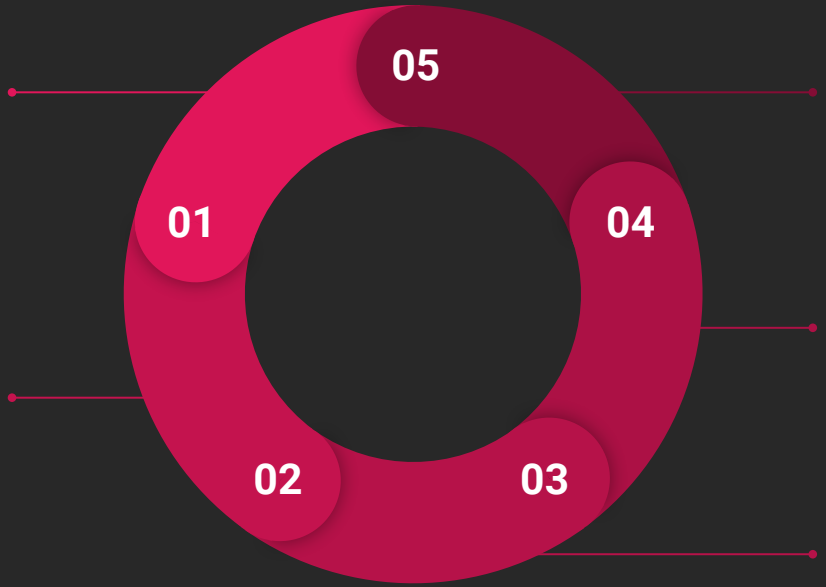
随着区块链技术的发展，交易成本、确认时间、能源消耗、安全性、互通性都有极大地提升。



应用链开发 - 房产登记交易平台

房主登记房产信息

房管局认证登记信息



房管局授权交易

买房购买锁定房产

房主出售房产

parity

应用链开发 - 存储管理

```
pub struct Property<Hash> {
  id: Hash,
  size: u64,
  certificate_no: u64,
  is_authenticated: bool
}
```

parity

应用链开发 - 存储管理

```
decl_storage! {
  trait Store for Module<T: Trait> as RealEstateStorage {
     Nonce: u64;
     Properties get(property): map T::Hash =>
Property<T::Hash>;
     AllPropertiesArray: map u64 => T::Hash;
     PropertyOwner get(property_owner): map T::Hash =>
T::AccountId;
     Managers get(manager): map u64 => T::AccountId;
     ManagersIndex: map T::AccountId => u64;
     ManagerNonce: u64;
     ……
  }
}
```

应用链开发 - 功能函数

```
pub fn record_property(origin, size: u64, certificate_no: u64) -> Result {
    let sender = ensure_signed(origin)?;
    let nonce = <Nonce<T>>::get();
    let random_seed = <system::Module<T>>::random_seed();
    let random_hash = (random_seed, &sender, nonce).using_encoded(<T as
system::Trait>::Hashing::hash);

    let property = Property {
       id: random_hash,
       size: size,
       certificate_no: certificate_no,
       is_authenticated: false
    };
    <Properties<T>>::insert(random_hash, property);
    <AllPropertiesArray<T>>::insert(nonce, random_hash);
    <PropertyOwner<T>>::insert(random_hash, sender);
    <Nonce<T>>::mutate(|n| *n += 1);
    Ok(())
}
```

parity

应用链开发 - 事件定义和触发

```
decl_event!(
  pub enum Event<T> where
  <T as system::Trait>::AccountId,
  <T as system::Trait>::Hash,
  <T as balances::Trait>::Balance
  {
     Authenticated(AccountId, Hash, bool),
     …….
  }
);


pub fn authenticate(origin, property_id: T::Hash, is_authenticated: bool) ->
Result {
  …….
  Self::deposit_event(RawEvent::Authenticated(sender, property_id,
is_authenticated));
  Ok(())
}
```

parity

应用链开发 - 开发原则

- 安全检查优先
- 线下处理计算密集的任务
- 文件存储采用其它方案，如 IPFS

应用源码 :kaichaosun/substrate-real-estate-node

parity

Demo

Help Contribute!

官方文档：substrate.dev

知乎专栏：《Substrate区块链开发》

Questions?