서브스트레이트 시작하기

tiny.cc/substrate-getting-started

나만의 첫 커스텀 블록체인을 위한
서브스트레이트 개발 가이드

Parity Technologies
parity.io | @paritytech

Get help: tiny.cc/substrate-technical

이 프레젠테이션 원본은 아래링크에서:

tiny.cc/substrate-getting-started

발표자가 남긴 링크들과 부연설명들을 찾아 보세요

서브스트레이트를 시작하기 위한 커맨드

```
curl https://getsubstrate.io -sSf | bash -s -- --fast
```

(~15-20 min)

parity

서브스트레이트가 뭔가요?

서브스트레이트는
오픈소스이고 모듈형이며 확장가능한 블록체인 개발 프레임워크입니다.

서브스트레이트는 무엇인가요?

서브스트레이트는 블록체인 코어 개발에 필요한 모든 리소스들을 제공합니다:

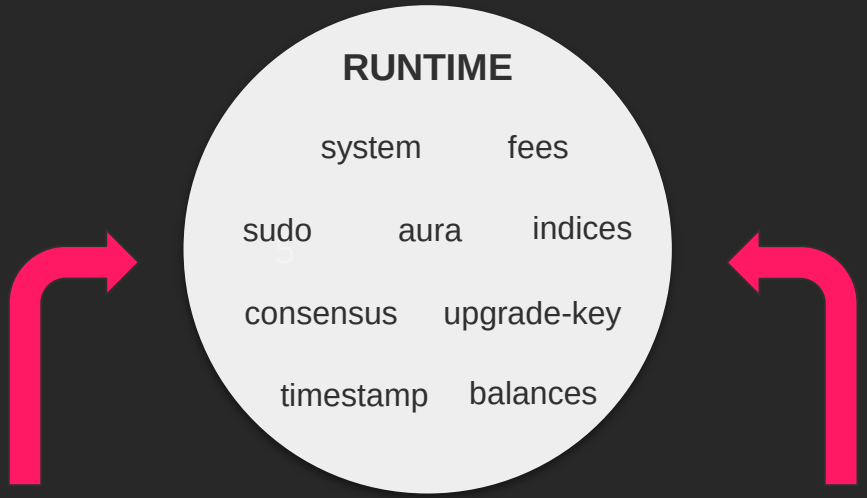
- 데이터베이스 레이어
- 네트워킹 레이어
- 컨센서스 엔진
- 트랜젝션 큐
- 런타임 모듈들을 위한 라이브러리들

그리고 이 모든 것들이 커스텀 및 확장이 가능합니다.

런타임은 무엇인가요?

런타임은 **블록체인의 상태를 변화시키는 로직**들을 말합니다, i.e. the State Transition Function.

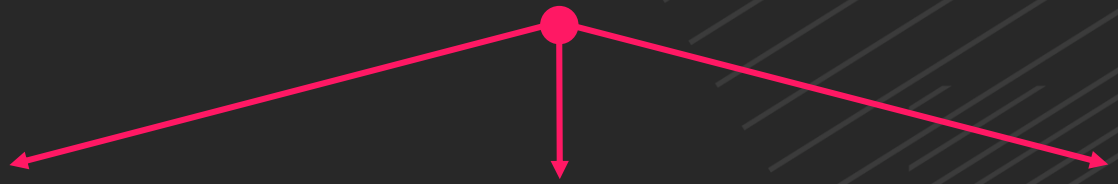
그리고 런타임은 **런타임 모듈들로 만들어지고요.**



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

서브스트레이트 개발을 위한 패키지들

→ tiny.cc/substrate-package ←



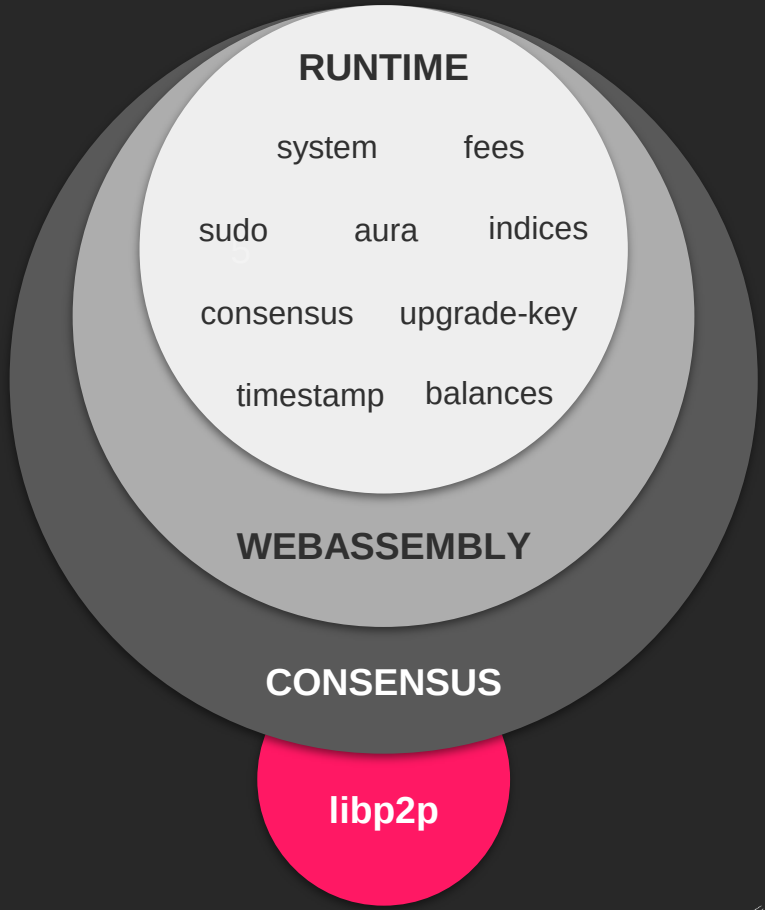
Substrate Dependencies

Substrate Node Template

Substrate UI (React)

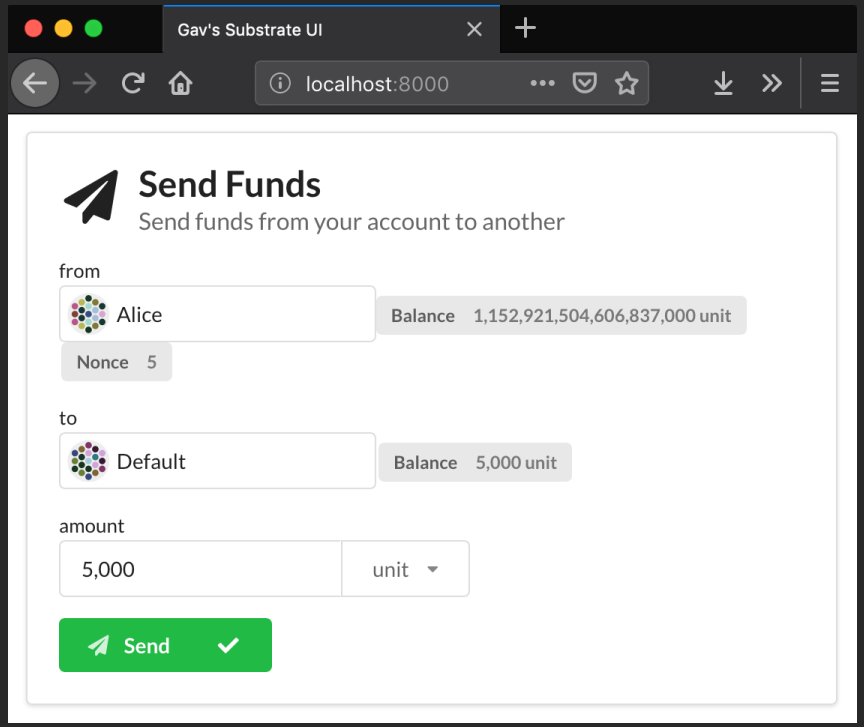
Substrate Node Template

- 작동하는 서브스트레이트 풀노드
- 포함된 SRML:
  - Accounts, Balances, Fees, Runtime Upgrades, and more…
- SRML에서 모듈들을 쉽게 분리하고 포함이 가능
- 나만의 모듈 제작가능



Substrate UI

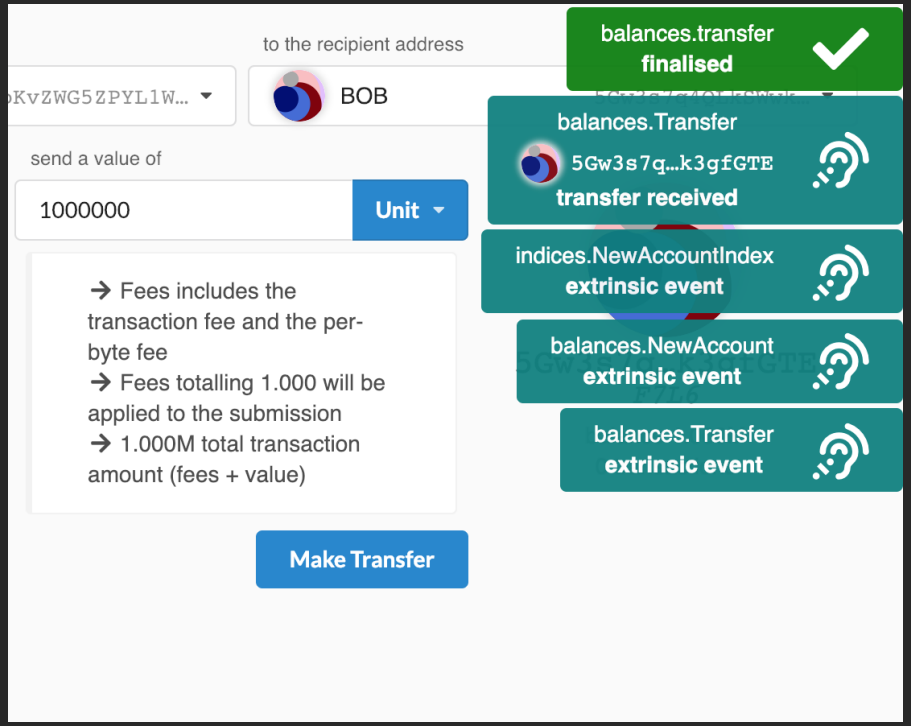
- 아래를 위해 미리 만들어진 리액트 UI:
  - 어카운트 생성
  - 전송 트랜젝션 만들기
  - 런타임 업그레이드
- “Bonds”라는 라이브러리를 사용하여 서브스트레이트와 연결
- 빠르게 UI 기능을 확장하고 개발하는데 용이함
- 커스텀 UI 개발에 활용가능



parity

Polkadot UI

- 일반화되며 실제 사용되고 있는 UI
- 새 기능을 테스트하는 데 사용
- 일반화 된 툴에서 주로 사용되는 내역:
  - 트랜젝션 생성
  - storage 읽기
  - 이벤트 열람
  - 그리고 더....
- 개발에 용이함



parity

서브스트레이트에서 쓰이는 Rust 기본

parity

런타임 에러 핸들링

- 당신의 런타임은 절대 never 에러로 터지지 않습니다:
  - 복구불가능한 에러가 Rust에서 발생하면 그 쓰레드를 아예 없애버립니다.
- 대신 “안전하게” 이를 처리할 수 있게끔 에러를 미리 명시적으로 설정해줘야 합니다.
- 예를 들자면, safe math에서:

```
// BAD
let a = u8::max_value() + 1; // What should Rust do?
// GOOD
let a = u8::max_value().checked_add(1).ok_or("Overflow!");
```

Rust 언어는 명시적입니다.

개발할 때 Rust한테 에러가 뜨는 모든 케이스에 대해 어떻게 할건지 알려줘야 되니까요.

```
// Computes addition, returning the max value if overflow.
pub fn saturating_add(self, rhs: u8) -> u8;
// Computes addition, returning wrapped value if overflow.
pub fn wrapping_add(self, rhs: u8) -> u8;
// Computes addition, returning `None` if overflow.
pub fn checked_add(self, rhs: u8) -> Option<u8>;
```

Null 대신 Option

- Option은 값이 있을 때나 없을 때 모두를 대비하여 코드를 짜게 해줍니다.

```
// Definition of Option type
enum Option<T> {
    Some(T),
    None,
}
```

```
let a = u8::max_value().checked_add(1)
a == None // True
let b = u8::max_value().checked_sub(1)
b == Some(254) // True
```

parity

Panic 대신 Result

- Result는 가능한 에러를 Option에서 더 표현을 할 수 있게 해줌으로서 옵션보다 더 풍부한 표현을 제공해줍니다.

```
// Definition of Result type
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```
// Result in Substrate found in support::dispatch::Result
pub type Result = result::Result<(), &'static str>;
```

parity

쉽게 보여주는 에러 핸들링

```
fn useless_adder(origin, x: u8, y: u8) -> Result {
    let a = match x.checked_add(y) {
        Some(v) => v,
        // Function caught an error, return Err(“message”)
        None => return Err("Overflow occurred")
    };
    // Function ran successfully, return Ok(())
    Ok(())
}
```

parity

쉽게 보여주는 Option 핸들링

```
// These two expressions are equivalent
let a = match x.checked_add(y) {
    Some(v) => v,
    // Function caught an error, return Err()
    None => return Err("Overflow occurred")
};

let a = x.checked_add(y).ok_or("Overflow occurred");
```

쉽게 보여주는 Result 핸들링

```
// These two expressions are equivalent
let sender = match ensure_signed(origin) {
    Ok(s) => s,
    Err(e) => return Err(e),
};

// Note the question mark (?) operator
let sender = ensure_signed(origin)?;
```

매크로들

`decl_storage!` `decl_module!` `decl_event!`

- 러스트 코드를 함축한 러스트 코드
- 모듈을 만들 때 쓰임
- Rust에서 정의되지 않은 커스텀 문법 제작 가능
- 그런데 소스코드 읽기는 어려움
- 마법같이 다루시면 됩니다.

런타임 개발 기본

선검증 후기록

- 이더리움처럼 처리하지 않는 “나쁜 트랜젝션”
- 이더리움: 스테이트는 되돌아오고, storage는 건드리지도 않는데 수수료는 내야됨
- 서브스트레이트: `Err`값이 반환될 시 state값은 변화하지 않음
- 필요한 상황:
  - 트랜젝션이 실패했는데도 늘어나는 nonce값 -> 불필요한 gas비 지불
  - “out of gas”에서도 gas비를 지불해야 함
- “sub-functions”들을 고려한 패턴을 만드는 경우 고려하지 않을 수가 없음

모듈 스켈레톤

```
use support::{decl_module, decl_storage, decl_event,...};
pub trait Trait: system::Trait {...}

decl_storage! {...}
decl_module! {...}
decl_event! {...}
impl<T: Trait> Module<T> {...}
```

parity

Generic Types import 및 정의

```
pub trait Trait: system::Trait {
    type Event: From<Event<Self>> + Into<<Self as system::Trait>::Event>;
}
```

...and it inherits from `system::Trait`:

```
// From `system` SRML Module
pub trait Trait: 'static + Eq + Clone {
    type Origin: ...
    type BlockNumber: ...
    type Hash: ...
    type AccountId: ...
    … and more
}
```

Storage 선언

```
decl_storage! {
    trait Store for Module<T: Trait> as TemplateModule {
        // Here we are declaring a StorageValue, `SomeValue` as a u32
        // `get(some_value)` defines a getter function
        // Getter called with `Self::some_value()`
        SomeValue get(some_value): u32;
        // Here we are declaring a StorageMap from an AccountId to a Hash
        // Getter called with `Self::some_map(account_id)`
        SomeMap get(some_map): map T::AccountId => u32;
    }
}
```

parity

Events 선언

```
decl_event!(
    pub enum Event<T>
    where
        <T as system::Trait>::AccountId
    {
        // Event `ValueStored` deposits values of type `AccountId` and `u32`
        ValueStored(AccountId, u32),
    }
);
```

parity

Dispatchable Functions(이벤트를 보내는 함수) 선언

```
decl_module! {
  pub struct Module<T: Trait> for enum Call where origin: T::Origin {
    fn deposit_event<T>() = default; // The default deposit_event definition

    pub fn store_value(origin, input: u32) -> Result {
      let sender = ensure_signed(origin)?; // Check for transaction
      <SomeValue<T>>::put(input); // Put a value into a StorageValue
      <SomeMap<T>>::insert(sender, input); // Insert key/value in StorageMap
      Self::deposit_event(RawEvent::ValueStored(sender, input)); // Emit Event
      Ok(()) // Return Ok at the end of a function
}}}
```

parity

Public and Private 함수 선언

```
impl<T: Trait> Module<T> {
  fn mint(to: T::AccountId, id: T::Hash) -> Result {}
  fn transfer(from: T::AccountId, to: T::AccountId, id: T::Hash) -> Result {}
}
```

These can also be called from other modules if made `pub`

Custom Struct 선언

```
use parity_codec::{Encode, Decode};

#[derive(Encode, Decode, Default, Clone, PartialEq)]
#[cfg_attr(feature = "std", derive(Debug))]
pub struct Kitty<Hash, Balance> {
    id: Hash,
    price: Balance,
    gen: u64,
}
```

parity

Kitty by David Revoy

Substrate Collectables 워크샵

- ➔ 로컬 서브스트레이트를 돌리고
- ➔ 런타임 모듈 개발과 모범 개발 답안 제공
- ➔ UI로 블록체인 앱 제작
- ➔ 최소한의 러스트 사용

tiny.cc/**substrate-workshop**

parity

다음에 할 일들!

- ✓ 서브스트레이트 패키지들이 말하는 대로 따라해보세요!
  - ➢ tiny.cc/substrate-package
- ✓ Riot에 있는 Substrate Technical channel(영문)이나 Polkadot korea(국문)에 물어보세요!
  - ➢ tiny.cc/substrate-technical(영문) https://t.me/PolkadotKR(국문)
- ✓ 서브스트레이트 런타임 모듈 라이브러리를 뜯어보고 분석해보세요!
  - ➢ tiny.cc/substrate-srml
- ✓ **서브스트레이트로 개발해보세요!**

parity

도움이 되는 링크들

- tiny.cc/substrate-getting-started (this presentation)
- tiny.cc/substrate-package
- tiny.cc/substrate-docs
- tiny.cc/substrate-workshop
- tiny.cc/substrate-technical

Questions?

tiny.cc/substrate-technical