

# 本科生毕业设计（论文）外文翻译

|  |  |
| --- | --- |
| **外文原文题目：** | Substrate Document |
| **中文翻译题目：** | Substrate文档 |

**树状区块链性能测试及基于Rust的重写**

|  |  |
| --- | --- |
| 学 院： | 计算机学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 07111905 |
| 学生姓名： | 傅泽 |
| 学 号： | 1120192062 |
| 指导教师： | 陆慧梅 |

# **第一章 外文原文**

## 1.1 Quick Start

All of the Substrate tutorials and how-to guides require you to build and run a Substrate node in your development environment.

To help you set up a working environment quickly, the Substrate Developer Hub (https://github.com/substrate-developer-hub/) maintains several \_templates\_ for you to use.

For example, the Developer Hub substrate-node-template (https://github.com/substrate-developer-hub/substrate-node-template/tags/) is a snapshot of the main Substrate node-template binary that includes everything you need to get started with a functional node and a core set of features.

This \_Quick start\_ assumes that you are setting up a development environment for the first time and want to try out running a single blockchain node on your local computer.

To keep things simple, you'll connect to the local node using a web browser and look up a balance for a predefined sample account.

### 1.1.1 Before you begin

Before you begin, verify the following:

- You have an internet connection and access to an interactive shell terminal on your local computer.

- You are generally familiar with software development and using command-line interfaces.

- You have the Rust compiler and toolchain installed.

You can check whether you have Rust installed by running the rustup show command.

If Rust is installed, this command displays version information for the toolchain and compiler.

If Rust is not installed, the command doesn't return any output.

For information about installing Rust, see [Install](/install).

### 1.1.2 Build the node template

1. Clone the node template repository by running the following command:

|  |
| --- |
| git clone https://github.com/substrate-developer-hub/substrate-node-template |

In most cases, you can clone the main branch to get the latest code.

However, you can use the --branch command-line option if you want to work with a Substrate branch that is compatible with a specific Polkadot version.

Click Tags (https://github.com/substrate-developer-hub/substrate-node-template/tags) to see the list of branches that are compatible with specific Polkadot versions.

1. Change to the root of the cloned directory:

|  |
| --- |
| cd substrate-node-template |

If you want to save your changes and make this branch easy to identify, you can create a new branch by running a command similar to the following:

|  |
| --- |
| git switch -c my-branch-v0.9.29 |

1. Compile the node template:

|  |
| --- |
| cargo build --package node-template --release |

Because of the number of packages involved, compiling the node can take several minutes.

### 1.1.3 Start the node

1. Verify that your node is ready to use and see information about the command-line options available by running the following command:

|  |
| --- |
| ./target/release/node-template --help |

The usage information displays the command-line options you can use to:

- start the node

- work with accounts

- modify node operations

1. View account information for the predefined Alice development account by running the following command:

|  |
| --- |
| ./target/release/node-template key inspect //Alice |

The command displays the following account information:

|  |
| --- |
| Secret Key URI //Alice is account:  Network ID: substrate  Secret seed: 0xe5be9a5092b81bca64be81d212e7f2f9eba183bb7a90954f7b76361f6edb5c0a  Public key (hex): 0xd43593c715fdd31c61141abd04a99fd6822c8558854ccde39a5684e7a56da27d  Account ID: 0xd43593c715fdd31c61141abd04a99fd6822c8558854ccde39a5684e7a56da27d  Public key (SS58): 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY  SS58 Address: 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY |

3. Start the node in development mode by running the following command:

|  |
| --- |
| ./target/release/node-template --dev |

In development mode, the chain doesn't require any peer computers to finalize blocks.

As the node starts, the terminal displays output about the operations performed.

If you see messages that blocks are being proposed and finalized, you have a running node.

|  |
| --- |
| ... Idle (0 peers), best: #3 (0xcc78…5cb1), finalized #1 ...  ... Starting consensus session on top of parent ...  ... Prepared block for proposing at 4 (0 ms) ... |

### 1.1.4 Connect to the node

1. Create a simple HTML file with JavaScript and the Polkadot-JS API (https://polkadot.js.org/docs/) to interact with the blockchain.

For example, create an index.html file that uses JavaScript and HTML to:

- take an account address as input

- look up the account balance using an onClick event

- display the balance for the account as output

This sample index.html (/assets/quickstart/index.html) provides a simple example of how to use JavaScript, the Polkadot-JS API, and HTML to get an account balance.

2. Copy and paste the sample code (https://github.com/substrate-developer-hub/substrate-docs/blob/main/static/assets/quickstart/index.html) for the Quick start: Get Balance application into a new file in your text editor and save the file on your local computer.

3. Open the index.html file in a web browser.

4. Copy and paste the SS58 Address for the Alice account in the input field, then click Get Balance.

### 1.1.5 Stop the node

1. Go to the terminal that displays blockchain operations.

1. Stop the local blockchain and clear all state by pressing ctrl-c.

### 1.1.6 Where to go next

In this \_Quick start\_, you learned how to compile and run a single blockchain node on your local computer.

To start learning how to customize its features, explore the following resources:

- Architecture

- Runtime development

- Rust for Substrate

## 1.2 Fundamentals

The topics in Fundamentals explain many of the core principles and unique features of the Substrate development environment and highlights some of the design decisions available to you as a blockchain builder.

Substrate offers a modular and flexible library of tools that empower you to select, compose, modify, and reuse components to suit the specific purpose of the blockchain you want to create, whether that is a private network or publish a blockchain that can interact with other blockchains through the Polkadot network.

The topics in Fundamentals are intended to help you learn what's possible when you build a Substrate-based blockchain and how Substrate can help you build a blockchain that best serves your specific project requirements or business model.

Before you start building, though, you want to make sure you are in the right place.

- [Blockchain basics] (/fundamentals/blockchain-basics/) provides context about the complexity associated with blockchain development and introduces common blockchain concepts, components, and terminology.

- [Why Substrate?] (/fundamentals/why-substrate/) highlights the key benefits of developing with Substrate that most blockchain and smart contract platforms can't provide.

- [What is Substrate?] (/fundamentals/what-is-substrate/) explains the core principles behind the design of Substrate and how the design decisions informed the technologies used.

- [Architecture] (/fundamentals/architecture/) describes the key components of the Substrate node architecture and how these components relate to the design and architecture of your custom blockchain.

- [Networks and blockchains] (/fundamentals/node-and-network-types/) defines the network topology for different blockchain deployment scenarios and how they apply to Substrate-based blockchains.

- [Runtime development] (/fundamentals/runtime-development/) highlights the importance of the Substrate runtime and introduces the core application interfaces and primitives required for Substrate runtime development.

- [Consensus] (/fundamentals/consensus/) describes the most common consensus models and the types of consensus you can implement for a Substrate blockchain.

- [Transactions and block basics] (/fundamentals/transaction-types/) introduces transaction types and the components that make up a block.

- [Transaction lifecycle] (/fundamentals/transaction-lifecycle/) explains how transactions are received, queued, and executed to eventually be included in a block.

- [State transitions and storage] (/fundamentals/state-transitions-and-storage/) describes how the state changes processed in the runtime are stored and managed using trie data structures and a key-value database.

- [Accounts, addresses, and keys] (/fundamentals/accounts-addresses-keys/) explains the relationship between accounts, addresses, and keys and how they are used.

- [Rust for Substrate] (/fundamentals/rust-basics/) highlights the specific Rust features—including traits, generics, associated types, and macros—that you should be most familiar with to build a Substrate-based blockchain.

- [Offchain operations] (/fundamentals/offchain-operations/) explores reasons for handling some operations offchain and alternatives for performing those offchain operations.

- [Light clients in Substrate Connect] (/fundamentals/light-clients-in-substrate-connect/) describes how to use Substrate Connect to integrate a light client into your applications and enable interaction with any Substrate-based chain.

- [Cross-consensus messaging] (/fundamentals/xcm-communication/) provides an overview of cross-consensus communication and the cross-consensus messaging (XCM) format.

After you digest the information in these introductory sections, you'll be ready to start designing, building, and testing your own custom blockchain solution.

### 1.2.1 Blockchain basics

A blockchain is a decentralized ledger that records information in a sequence of blocks.

The information contained in a block is an ordered set of instructions that might result in a change in state.

In a blockchain network, individual computers—called nodes—communicate with each other to form a decentralized peer-to-peer (P2P) network.

There is no central authority that controls the network and, typically, each node that participates in block production stores a copy of the blocks that make up the canonical chain.

In most cases, users interact with a blockchain by submitting a request that might result in a change in state, for example, a request to change the owner of a file or to transfer funds from one account to another.

These transactions requests are gossiped to other nodes on the network and assembled into a block by a block author.

To ensure the security of the data on the chain and the ongoing progress of the chain, the nodes use some form of consensus to agree on the state of the data in each block and on the order of transactions executed.

#### 1.2.1.1 What is a blockchain node?

At a high level, all blockchain nodes require the following core components:

- Data storage for the state changes recorded as a result of transactions.

- Peer-to-peer networking for decentralized communication between nodes.

- Consensus methodology to protect against malicious activity and ensure the ongoing progress of the chain.

- Logic for ordering and processing incoming transactions.

- Cryptography for generating hash digests for blocks and for signing and verifying the signatures associated with transactions.

Because of the complexity involved in building the core components a blockchain requires, most blockchain projects start with a complete copy of an existing blockchain code base so that developers can modify existing code to add new features instead of writing everything from scratch.

For example, the Bitcoin repository was forked to create Litecoin, ZCash, Namecoin and Bitcoin Cash.

Similarly, the Ethereum repository was forked to create Quorum, POA Network, KodakCoin, and Musicoin.

However, most blockchain platforms are not designed to allow for modification or customization.

As a result, building a new blockchain by forking has serious limitations, including limitations such as scalability that are inherent in the originating blockchain code.

Before you explore how Substrate alleviates many of the limitations associated with other blockchain projects, it's important to understand some of the common properties that most blockchains share.

By learning about how most blockchains operate, you'll be better prepared to see how Substrate provides alternatives and capabilities for building a blockchain best suited to your needs.

#### 1.2.1.2 State transitions and conflicts

A blockchain is essentially a [state machine] (https://en.wikipedia.org/wiki/Finite-state\_machine).

At any point in time, the blockchain has a current internal state.

As inbound transactions are executed, they result in changes to state so the blockchain must transition from its current state to a new state.

However, there can be multiple valid transitions that would result in different future states, and the blockchain must select a single state transition that can be agreed upon.

To agree on the state after a transition, all operations within a blockchain must be deterministic.

For the chain to progress successfully, a majority of the nodes must agree on all of the state transitions, including:

- The initial state of the chain, called the genesis state or genesis block.

- The series of state transitions that result from executed transactions that are recorded in each block.

- A final state for the block to be included in the chain.

In centralized networks, a central authority can choose between mutually exclusive state transitions.

For example, a server configured as the primary authority might record changes to state transition in the order it sees them or use a weighting process to choose between competing alternatives when a conflict arises.

In a decentralized network, the nodes see transactions in different orders, so they must use a more elaborate method to select transactions and choose between conflicting state transition.

The method that a blockchain uses to batch transactions into blocks and to select which node can submit a block to the chain is called the blockchain's consensus model or consensus algorithm.

The most commonly-used consensus model is called the proof-of-work consensus model.

With the proof-of-work consensus model, the node that completes a computational problem first has the right to submit a block to the chain.

For a blockchain to be fault tolerant and provide a consistent view of state even if some nodes are compromised by malicious actors or network outages, some consensus models require at least two-thirds of the nodes to agree on state at all time.

This two-thirds majority ensures that the network is fault tolerant and can withstand some network participants behaving badly, regardless of whether the behavior is intentional or accidental.

#### 1.2.1.3 Blockchain economics

All blockchains require resources—processors, memory, storage, and network bandwidth—to perform operations.

The computers that participate in the network—the nodes that produce blocks—provide these resources to blockchain users.

The nodes create a distributed, decentralized network that serves the needs of a community of participants.

To support a community and make a blockchain sustainable, most blockchains require users to pay for the network resources they use in the form of transaction fees.

The payment of transaction fees requires user identities to be associated with accounts that hold assets of some type.

Blockchains typically use tokens to represent the value of assets in an account and network participants purchase tokens outside of the chain through an exchange.

Network participants can then deposit the tokens to enable them to pay for transactions.

#### 1.2.1.4 Blockchain governance

Some blockchains allow network participants to submit and vote on proposals that affect network operations or the blockchain community.

By submitting and voting on proposals—referenda—the blockchain community can determine how the blockchain evolves in an essentially democratic process.

On-chain governance is relatively rare, however, and to participate, a blockchain might require users to maintain a significant stake of tokens in an account or to be selected as a representative for other users.

#### 1.2.1.5 Applications running on a blockchain

Applications that run on a blockchain—often referred to as decentralized applications or dApps—are typically web applications that are written using front-end frameworks but with backend \*\*smart contracts\*\* for changing the blockchain state.

A smart contract is a program that runs on a blockchain and executes transactions on behalf of users under specific conditions.

Developers can write smart contracts to ensure that the outcome of programmatically-executed transactions is recorded and can't be tampered with.

Yet, with smart contracts alone, developers don't have access to some underlying blockchain functionality—such as the consensus, storage, or transaction layers—and instead, abide by a chain's fixed rules and restrictions.

Smart contract developers often accept these limitations as a tradeoff that enables faster development time with fewer core design decisions to make.

#### 1.2.1.6 Where to go next

All blockchains share some common characteristics.

Substrate—while not a blockchain itself—is a blockchain builders' toolkit with a modular framework of components to create a custom blockchain.

With Substrate, you can take the common blockchain components—like storage and consensus and cryptography—and combine them to use the functions they provide as-is or modify them to suit the purpose of your project.

You can explore the following resources to learn more.

**Tell me**

- [Fundamentals](/fundamentals/)

- [Why Substrate?](/fundamentals/why-substrate/)

- [Architecture](/fundamentals/architecture/)

- [Networks and blockchains](/fundamentals/node-and-network-types/)

**Guide me**

- [Build a local blockchain] (/tutorials/get-started/build-local-blockchain/)

- [Simulate a network] (/tutorials/get-started/simulate-network/)

- [Add trusted nodes] (/tutorials/get-started/add-trusted-nodes/)

If you prefer to explore code directly, you can start building in the [Substrate Playground] (https://docs.substrate.io/playground/) or consult the API reference to get details about the Rust crates you use.

### 1.2.2 Why Substrate?

Blockchain development is complex.

It involves sophisticated technologies—including advanced cryptography and distributed network communication—that you have to get right to provide a secure platform for applications to run on and for users to trust.

There are hard problems around scale, governance, interoperability, and upgradeability to address.

The complexity creates a high barrier to entry for developers to overcome.

With this in mind, the first question to answer is: what do you want to build?

Substrate isn't a perfect fit for every use case, application, or project.

However, Substrate might be the perfect choice if you want to build a blockchain that is:

- tailored to a very specific use case

- able to connect and communicate with other blockchains

- customizable with predefined composable modular components

- able to evolve and change with upgrades over time

Substrate is a Software Development Kit (SDK) specifically designed to provide you with all of the fundamental components a blockchain requires so you can focus on crafting the logic that makes your chain unique and innovative.

Unlike other distributed ledger platforms, Substrate is:

- [Flexible] (#flexible)

- [Open](#open)

- [Interoperable] (#interoperable)

- [Future-proof] (#future-proof)

- [Where to go next] (#where-to-go-next)

#### 1.2.2.1 Flexible

Most blockchain platforms have very tightly coupled, opinionated, sub-systems that are quite hard to decouple.

There is risk as well in a chain based on a fork of another blockchain that those non-obvious couplings can fundamentally undermine the blockchain system itself.

Substrate is a fully modular blockchain framework that lets you compose a chain with explicitly decoupled components, by selecting the network stack, consensus model, or governance method that suits your project or by creating your own components.

With Substrate, you can deploy a blockchain that's designed and built for your specifications, but that can also evolve with your changing needs.

#### 1.2.2.2 Open

All of the Substrate architecture and tooling is available under [open-source licensing] (https://github.com/paritytech/substrate#license).

Core components of the Substrate framework use open protocols such as libp2p and jsonRPC while empowering you to decide how much you want to customize your blockchain architecture.

Substrate also has a large, active, and helpful [builder community] (https://substrate.io/ecosystem/) contributing to the ecosystem.

Contributions from the community enhance the capabilities available for you to incorporate into your own blockchain as it evolves.

#### 1.2.2.3Interoperable

Most blockchain platforms provided limited ability to interact with other blockchain networks.

All Substrate-based blockchains can interoperate with the other blockchains through [Cross-Consensus Messaging] (https://wiki.polkadot.network/docs/learn-crosschain) (XCM).

Substrate can be used to create chains as independent networks (solo chains), or tightly coupled to a [relay chain] (https://wiki.polkadot.network/docs/learn-architecture#relay-chain) to share its security as a [parachain] (https://wiki.polkadot.network/docs/learn-parachains).

#### 1.2.2.4 Future-proof

Substrate is built to be upgradeable, composable, and adaptable.

The state transition logic—the Substrate runtime—is a self-contained WebAssembly object.

Your nodes can be given the ability to completely change the runtime itself under specific conditions, inducing a runtime upgrade \_network wide\_.

Thus "forkless" upgrades are possible, as no action is required in most cases for nodes to operate with this new runtime.

Over time, your network's runtime protocols can seamlessly, and perhaps radically, evolve with the needs of your users.

#### 1.2.2.5 Where to go next

Substrate provides a unique and powerful framework for building blockchain networks and blockchain applications.

- You can get started with a pre-configured node and start a solo chain with no code.

- You can use Substrate to build a custom solo chain that runs independent of any other blockchains.

- You can couple your Substrate chain to a relay chain like Polkadot or Kusama.

Depending on your background and interests, explore the following resources to learn more.

**Tell me**

- [Blockchain basics] (/fundamentals/blockchain-basics/)

- [Architecture] (/fundamentals/architecture/)

- [Runtime development] (/fundamentals/runtime-development)

- [Networks and blockchains] (/fundamentals/node-and-network-types/)

- [Install] (/install/)

**Guide me**

- [Build a local blockchain] (/tutorials/get-started/build-local-blockchain/)

- [Simulate a network] (/tutorials/get-started/simulate-network/)

- [Add trusted validators] (/tutorials/get-started/add-trusted-nodes/)

If you prefer to explore code directly, you can start building in the [Substrate Playground] (https://docs.substrate.io/playground/) and consult the [API reference (Rustdocs)] (https://paritytech.github.io/substrate/master) to get details about the Rust crates within Substrate.

### 1.2.3 What is Substrate?

[Why Substrate] (/fundamentals/why-substrate/) introduced some of the core characteristics and benefits of the Substrate development environment.

If those benefits—flexibility, open source, interoperability, and upgradeability—are important to you, you might want to dig deeper into what Substrate is and into why it's designed the way it is.

With this in mind, let's look more closely at what Substrate is.

#### 1.2.3.1 Multi-chain vision

Substrate is a Rust-based framework for building blockchain applications using modular and extensible components to support a multi-chain block space.

An important part of learning how to use this framework is understanding that it envisions and embraces a multi-chain universe.

Instead of a single chain with limited features and scalability, Substrate is designed to allow for specialization, optimization, and scalability.

In this vision of the future—a multi-chain future—blockchains are specialized for different purposes and built to address specific problems or use cases.

Substrate is designed to make blockchains easy to customize and upgrade, so that projects can evolve over time and adapt to new opportunities.

Instead of a single dominant chain with an inflexible protocol, Substrate is designed to allow for experimentation, coordination, and collaboration across multiple chains.

But if the future of the blockspace consists of many different blockchains performing specialized tasks, how does one go about building those blockchains?

Building a blockchain from scratch is expensive, time-consuming, and error-prone. It requires an investment of capital and advanced expertise in multiple technologies and disciplines, Making mistakes can be prohibitively costly.

It is, inherently, not for the faint of heart.

With so many barriers to entry, how then is a multi-chain future possible?

And if you were to overcome all of these difficulties to build a blockchain, how would you upgrade it to keep it relevant as society and technology evolve?

#### 1.2.3.2 Core design principles

Before Substrate, a lot of resources were expended trying to get this blockchain of the multi-chain future right, trying to build a blockchain that was future-proof and upgradeable, with all the right components.

All of that work—time, money, and energy—resulted in a realization that well thought-out choices made today would likely become the mistakes of tomorrow. If the blockchain relied on some specific technology or assumption, over time that technology or assumption would likely hamper and, ultimately, stifle innovation.

That realization, along with the need to provide tools for a multi-chain future, led to the development of Substrate. The following defining principles emerged as core design decisions for the Substrate framework:

- Rust as the core programming language for the codebase.

- WebAssembly as an execution environment for application logic.

- Extensive use of layered abstractions, generic implementations, and flexible application programming interfaces as primary coding practices and the separation of libraries into distinct architectural components.

Each of these design decisions plays an important role in how Substrate operates and how you can use Substrate to build specialized composable blockchains using reusable and customizable components.

#### 1.2.3.3 Why Substrate uses Rust

There are several important benefits that are the direct result of selecting Rust as the underlying programming language for the Substrate framework.

Although it might be somewhat more complex to learn, Rust is a highly performant system-level programming language designed for building bullet-proof, mission-critical, and production-grade software.

- Rust has a sophisticated memory management system that relies on ownership rules rather than memory allocation and garbage collection.

These ownership rules enable the compiler to verify every variable you use and every memory address you reference to avoid memory leaks.

- The Rust compiler also prevents you from introducing the kinds of coding mistakes that make systems vulnerable.

- Rust provides abstractions—like traits and generics—that make code more flexible without imposing performance overhead.

- The compiler performs static checks on your application source code while the code is being compiled.

When you execute the compiled code, the performance is comparable to the fastest programming languages currently available.

- Compiled Rust code has a relatively small footprint and can run on embedded devices with support for any operating system.

In addition to all of these other benefits, Rust code can be compiled to WebAssembly and WebAssembly is at the heart of what makes Substrate flexible and its runtime upgradeable.

#### 1.2.3.4 WebAssembly and upgradeability

Most components in a Substrate blockchain are customizable or support multiple implementations for maximum flexibility.

However, storing the application logic onchain as a WebAssembly binary is one of the few attributes of Substrate that you can’t change.

If you don't use WebAssembly for the application logic, you're no longer using Substrate.

Substrate-based chains are designed to be upgradable by default.

This is made possible by encoding the application logic of your blockchain—the specialized business logic—as a WebAssembly binary and storing that binary as a part of the blockchain state.

In the same way you can execute a transaction to change an account balance, you can execute a transaction to change the WebAssembly code.

After the block containing that transaction is produced, the next block reflects the updated blockchain state.

#### 1.2.3.5 Abstractions and generics

In addition to the core application logic in the WebAssembly binary, there are many other components of a Substrate-based blockchain that you can customize.

For example, there are multiple implementations for the consensus, networking, and database layers, including support for the following:

- Multiple consensus engines (BABE/Grandpa/AURA)

- Multiple network protocols (QUIC, TCP)

- Multiple database implementations (ParityDB, RocksDB)

In the Polkadot ecosystem, there are numerous examples of parachain projects that have customized different aspects of the Substrate node.

In many cases, those ecosystem applications require access to the core components of the chain—such as the consensus layer or transaction pool—wouldn’t be possible if they were written as smart contract applications.

At the application layer, you have even more flexibility.

For example, you can choose between an accounts-based ledger format or UTXO.

You have full control over how to implement transaction fees or define custom incentives around resource usage.

With Substrate, you can build virtually any type of blockchain you can imagine by customizing and extending the framework to suit your needs.

### 1.2.4 Architecture

As noted in [Blockchain basics] (/fundamentals/blockchain-basics), a blockchain relies on a decentralized network of computers—called nodes—that communicate with each other.

Because the node is a core component of any blockchain, it’s important to understand what makes a Substrate node unique, including the core services and libraries that are provided by default and how the node can be customized and extended to suit different project goals.

#### 1.2.4.1 High level overview

In a decentralized network, all nodes act as both clients that request data and as servers that respond to requests for data.

Conceptually and programmatically, the Substrate architecture divides operational responsibilities along similar lines.

The following diagram illustrates this separation of responsibilities in simplified form to help you visualize the architecture and how Substrate provides a modular framework for building blockchains.

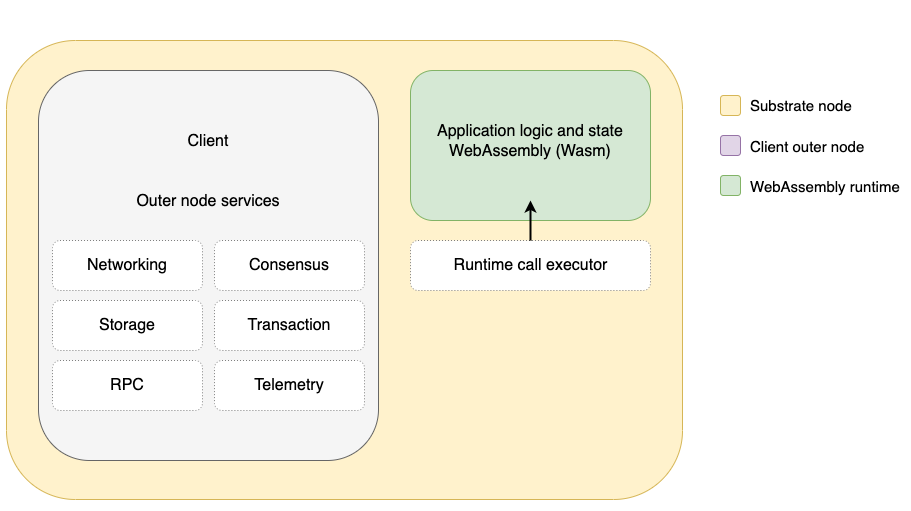


Figure 1-1 Substrate Architecture

At a high level, a Substrate node provides a layered environment with two main elements:

- A client with outer node services that handles network activity such as peer discovery, managing transaction requests, reaching consensus with peers, and responding to RPC calls.

- A runtime that contains all of the business logic for executing the state transition function of the blockchain.

**Outer node**

The outer node is responsible for activity that takes place outside of the runtime.

For example, the outer node is responsible for handling peer discovery, managing the transaction pool, communicating with other nodes to reach consensus, and answering RPC calls or browser requests from the outside world.

Some of the most important activities that are handled by the outer node involve the following components:

- [Storage] (/fundamentals/state-transitions-and-storage/): The outer node persists the evolving state of a Substrate blockchain using a simple and highly efficient key-value storage layer.

- [Peer-to-peer networking] (/fundamentals/node-and-network-types/): The outer node uses the Rust implementation of the [libp2p network stack] (https://libp2p.io/) to communicate with other network participants.

- [Consensus] (/fundamentals/consensus/): The outer node communicates with other network participants to ensure they agree on the state of the blockchain.

- [Remote procedure call (RPC) API] (/build/remote-procedure-calls/): The outer node accepts inbound HTTP and WebSocket requests to allow blockchain users to interact with the network.

- [Telemetry] (/maintain/monitor/): The outer node collects and provides access to node metrics through an embedded [Prometheus](https://prometheus.io/) server.

- [Execution environment] (/build/build-process/): The outer node is responsible for selecting the execution environment—WebAssembly or native Rust—for the runtime to use then dispatching calls to the runtime selected.

Performing these tasks often requires the outer node to query the runtime for information or to provide information to the runtime.

This communication is handled by calling specialized [runtime APIs] (/reference/runtime-apis/).

**Runtime**

The runtime determines whether transactions are valid or invalid and is responsible for handling changes to the blockchain's state transition function.

Because the runtime executes the functions it receives, it controls how transactions are included in blocks and how blocks are returned to the outer node for gossiping or importing to other nodes.

In essence, the runtime is responsible for handling everything that happens on-chain.

It is also the core component of the node for building Substrate blockchains.

The Substrate runtime is designed to compile to [WebAssembly (Wasm)] (/reference/glossary#webassembly-wasm) byte code.

This design decision enables:

- Support for forkless upgrades.

- Multi-platform compatibility.

- Runtime validity checking.

- Validation proofs for relay chain consensus mechanisms.

Similar to how the outer node has a way to provide information to the runtime, the runtime uses specialized [host functions] (https://paritytech.github.io/substrate/master/sp\_io/index.html) to communicate with the outer node or the outside world.

#### 1.2.4.2 Light client nodes

A light client or light node is a simplified version of a Substrate node that only provides the runtime and current state.

Light nodes enable users to connect to a Substrate runtime directly using a browser, browser extension, mobile device, or desktop computer.

With a light client node, you can use RPC endpoints written in Rust, JavaScript, or other languages to connect to the WebAssembly execution environment to read block headers, submit transactions, and view the results of transactions.

#### 1.2.4.3 Where to go next

Now that you have an overview of the Substrate architecture and core node components, explore the following topics to learn more.

- [Networks and blockchains] (/fundamentals/node-and-network-types)

- [Transactions and block basics] (/fundamentals/transaction-types)

- [Transaction lifecycle] (/fundamentals/transaction-lifecycle/)

- [State transitions and storage] (/fundamentals/state-transitions-and-storage/)

- [Runtime APIs] (/reference/runtime-apis/)

### 1.2.5 Networks and blockchains

When thinking about building a blockchain, it's useful to consider that boundaries are what define a network.

For example, a set of computers connected to a single router can be considered a home network.

A firewall might be the boundary that defines an enterprise network.

Smaller, isolated networks can be connected to wider area networks through a common communication protocol.

Similarly, you can think of a blockchain network as being defined by its boundaries and its isolation from or communication with other blockchains.

As a blockchain builder's toolkit, Substrate enables you to develop any type of blockchain you can imagine and to define its boundaries based on your application-specific requirements. With this flexibility in mind, one of the decisions you need to make is the type of network you want to build and the role that different nodes might play in that network.

#### 1.2.5.1 Network types

Substrate-based blockchains can be used in different types of network architecture.

For example, Substrate blockchains are used to build the following network types:

- Private networks that limit access to a restricted set of nodes.

- Solo chains that implement their own security protocol and don't connect or communicate with any other chains.

Bitcoin and Ethereum are examples of non-Substrate based solo chains.

- Relay chains that provide decentralized security and communication for other chains that connect to them.

Kusama and Polkadot are examples of relay chains.

- Parachains that are built to connect to a relay chain and have the ability to communicate with other chains that use the same relay chain.

Because parachains depend on the relay chain to finalize the blocks produced, parachains must implement the same consensus protocol as the relay chain they target.

#### 1.2.5.2 Node types

Blockchains require network nodes to be synchronised to present a consistent and up-to-date view of the blockchain state.

Each synchronised node stores a copy of the blockchain and keeps track of incoming transactions.

However, keeping a full copy of an entire blockchain requires a lot of storage and computing resources and downloading all of the blocks from genesis to the most recent isn’t practical for most use cases.

To make it easier to maintain the security and integrity of the chain but reduce the resource requirements for clients wanting access to blockchain data, there are different types of nodes that can interact with the chain:

- [Full nodes] (#full-nodes)

- [Archive nodes] (#archive-nodes)

- [Light client nodes] (#light-client-nodes)

**Full nodes**

Full nodes are a critical part of the blockchain network infrastructure and are the most common node type.

Full nodes store blockchain data and, typically, participate in common blockchain operations, such as authoring and validating blocks, receiving and verifying transactions, and serving data in response to user requests.

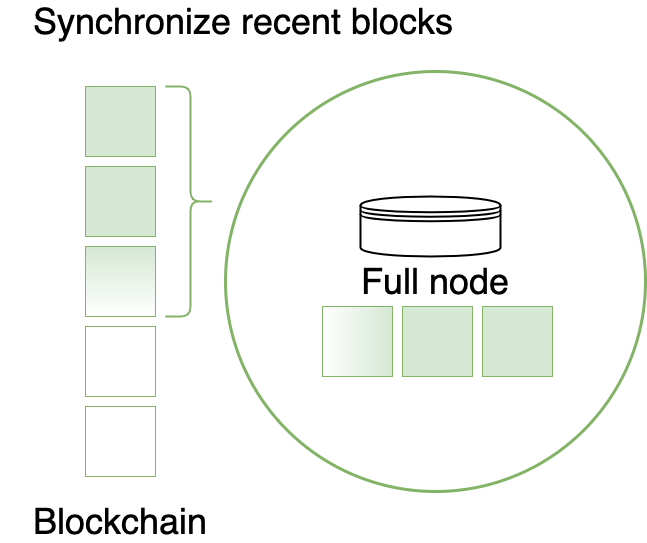


Figure 1-2 Full node

By default, full nodes are configured to store only the most recent 256 blocks and to discard state older than that—with the exception of the genesis block—to prevent the full node from growing indefinitely and consuming all available disk space.

You can configure the number of blocks a full node retains.

Although older blocks are discarded, full nodes retain all of the [block headers] (/reference/glossary/#header) from the genesis block to the most recent block to validate that the state is correct.

Because the full node has access to all of the block headers, it can be used to rebuild the state of the entire blockchain by executing all of the blocks from the genesis block.

Thus it requires much more computation to retrieve information about some previous state, and an archive should generally be used instead.

Full nodes allow you to read the current state of the chain and to submit and validate transactions directly on the network.

By discarding state from older blocks, a full node requires much less disk space than an archive node.

However, a full node requires far more computational resources to query and retrieve information about some previous state.

If you need to query historical blocks, you should purge the full node then restart it as an archive node.

**Archive nodes**

Archive nodes are similar to full nodes except that they store all past blocks with complete state available for every block.

Archive nodes are most often used by utilities—such as block explorers, wallets, discussion forums, and similar applications—that need access to historical information.

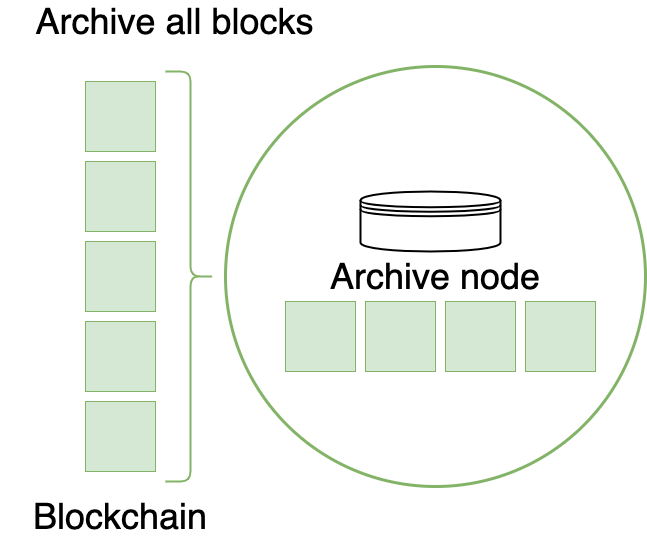


Figure 1-3 Archive node

Because archive nodes retain historical state, they require a lot of disk space.

Because of the disk space required to run them, archive nodes are less common than full nodes.

However, archive nodes make it convenient to query the past state of the chain at any point in time.

For example, you can query an archive node to look up an account balance in a certain block or to see details about a transaction resulted in a specific state change.

These types of queries are faster and more efficient when you run them on the data in an archive node.

**Light client nodes**

Light client nodes enable you to connect to a Substrate network with minimal hardware requirements.

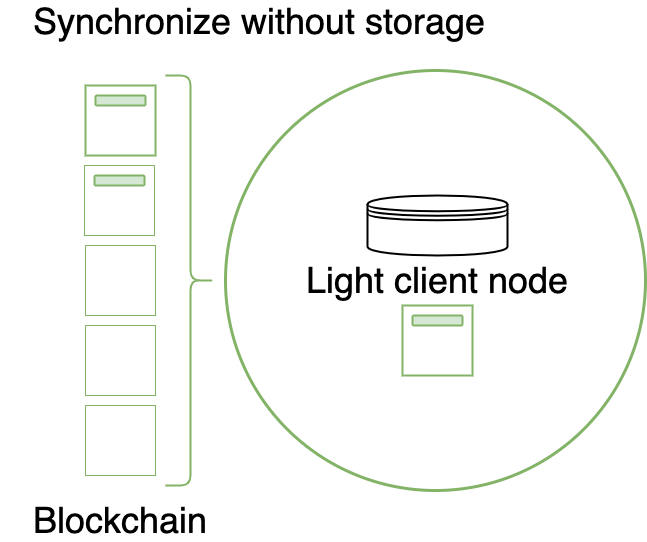


Figure 1-4 Light client node

Because light client nodes require minimal system resources, they can be embedded into web-based applications, browser extensions, mobile device applications, or internet of things (IoT) devices.

Light client nodes provide a runtime and access to the current state through RPC endpoints.

The RPC endpoints for light client nodes can be written in Rust, JavaScript, or other languages and used to read block headers, submit transactions, and view the results of transactions.

Light client nodes don't participate in blockchain or network operations.

For example, light client nodes aren't responsible for block authoring or validation, gossipping transactions or reaching consensus.

The light client node doesn't store any past blocks, so it can't read historical data without requesting it from a node that has it.

#### 1.2.5.3 Node roles

Depending on the command-line options you specify when you start a node, nodes can play different roles in the progression of the chain and can provide different levels of access to the on-chain state.

For example, you can limit the nodes that are authorized to author new blocks and which nodes can communicate with peers.

Peer nodes that aren't authorized as block producers can import new blocks, receive transactions, and send and receive gossip about new transactions to other nodes.

Nodes can also be prevented from connecting to the broader network and restricted to communicating with specific nodes.

#### 1.2.5.4 Where to go next

You can use Substrate to build virtually any type of network—from a completely self-contained and private solo-chain to your own relay chain ecosystem or compatible parachains.

To take a deeper dive into networks and nodes types, explore the following topics.

- [Build a local blockchain] (/tutorials/get-started/build-local-blockchain/)

- [Simulate a network] (/tutorials/get-started/simulate-network/)

- [Add trusted nodes] (/tutorials/get-started/add-trusted-nodes/)

- [Authorize specific nodes] (/tutorials/get-started/authorize-specific-nodes/)

# 外文译文

## 2.1 快速开始

所有Substrate教程都是引导您在自己的开发环境中搭建并运行一个Substrate节点的操作教程。为了让您能够快速地设置好开发环境，[Substrate开发中心] (https://github.com/substrate-developer-hub/)维护了数份模板任君选择。例如，开发中心提供的[Substrate节点模板] (https://github.com/substrate-developer-hub/substrate-node-template/tags/)即是一份Substrate节点模板主程序的快照，它提供了一个可以工作的节点和一系列核心功能，足够您快速上手开发之用了。

这份快速开始指南假定您是第一次搭建开发环境，以试图在本地运行一个单链区块链的节点。为使过程尽可能简单，您将通过网络浏览器连接到您的本地节点，并在一个给定的样本账户库中查询账户余额。

### 2.1.1 开始之前

在您开始之前，请依次执行下列检查：

- 确认拥有互联网连接，并在本地计算机上准备一个交互式终端

- 您对于软件开发和命令行交互界面较为熟悉

- 本地计算机上已经安装Rust编译器和工具链

为检查Rust是否安装，您可以在终端中运行rustup show指令。若Rust已经安装，您将看到关于编译器和工具链的版本信息；否则，该指令将不会有任何回显。有关安装Rust的事宜，请参见[安装] (https://docs.substrate.io/install/)。

### 2.1.2 构建节点模板

1. 将节点模板的仓库通过下列指令克隆到本地：

|  |
| --- |
| git clone https://github.com/substrate-developer-hub/substrate-node-template |

在绝大多数情况下，您只需克隆main分支即可获得最新的代码。不过，您也可以使用--branch选项选择特定分支以搭配特定版本的Polkadot使用。关于不同分支和不同版本的Polkadot的适配情况，请参阅[Tags] (https://github.com/substrate-developer-hub/substrate-node-template/tags)。

2. 在终端中，切换工作目录到克隆下来的仓库文件夹中：

|  |
| --- |
| cd substrate-node-template |

若您想让自己的分支更有辨识度，或是更易于保存属于您自己的修改，您可以使用形如下列指令来创建一个新的分支：

|  |
| --- |
| git switch -c my-branch-v0.9.29 |

3. 编译节点模板：

|  |
| --- |
| cargo build --package node-template --release |

由于所用的软件包数量较多，编译可能花费长达数分钟才能完成。

### 2.1.3 启动节点

1. 运行下列指令，确认您的节点已经就绪，并查看可用的命令行选项：

|  |
| --- |
| ./target/release/node-template --help |

通过上述命令获得的信息，我们可以知晓用于下列目的的命令行选项：

- 启动节点

- 处理账户

- 修改节点操作

2. 运行下列指令，查看预定义的开发账号Alice的信息：

|  |
| --- |
| ./target/release/node-template key inspect //Alice |

该指令将显示如下信息：

|  |
| --- |
| Secret Key URI //Alice is account:  Network ID: substrate  Secret seed: 0xe5be9a5092b81bca64be81d212e7f2f9eba183bb7a90954f7b76361f6edb5c0a  Public key (hex): 0xd43593c715fdd31c61141abd04a99fd6822c8558854ccde39a5684e7a56da27d  Account ID: 0xd43593c715fdd31c61141abd04a99fd6822c8558854ccde39a5684e7a56da27d  Public key (SS58): 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY  SS58 Address: 5GrwvaEF5zXb26Fz9rcQpDWS57CtERHpNehXCPcNoHGKutQY |

3. 通过下列指令，您可以以开发模式启动节点：

|  |
| --- |
| ./target/release/node-template --dev |

在该模式下，该区块链不需要任何对等计算机以封装区块。节点启动后，终端将显示其所执行的操作的输出信息。若您看到形如“区块正在被提交/封装”，则说明您的节点已经开始正常运行：

|  |
| --- |
| ... Idle (0 peers), best: #3 (0xcc78…5cb1), finalized #1 ...  ... Starting consensus session on top of parent ...  ... Prepared block for proposing at 4 (0 ms) ... |

### 2.1.4 连接到节点

1. 创建一个内含JavaScript以及[Polkadot-JS API] ([Overview | polkadot{.js}](https://polkadot.js.org/docs/))的HTML文件，以和节点互动。

在本例中，我们将创建一个名为index.html的文件，其内含的JavaScript以及HTML代码将可以：

- 选择一个账号作为输入

- 使用onClick事件来查阅账号信息

- 作为以上两步骤的输出，显示该账号的余额信息

这份[示例index.html] ([Quick start: Get Balance (substrate.io)] (https://docs.substrate.io/assets/quickstart/))提供了使用JS、Polkadot-JS API和HTML进行以上步骤的简单演示。

2. 复制粘贴[上述示例HTML文件的源代码] ([substrate-docs/index.html at main · substrate-developer-hub/substrate-docs (github.com)] (https://github.com/substrate-developer-hub/substrate-docs/blob/main/static/assets/quickstart/index.html))至您的代码编辑器中，并保存为index.html至本地。

3. 在浏览器中打开index.html。

4. 复制账号Alice的SS58地址到网页中的输入框中，然后点击Get Balance按钮。

### 2.1.5 关闭节点

1. 找到之前显示区块链操作信息的终端。

2. 键盘输入Ctrl + C以终止节点工作。

### 2.1.6 下一步做什么？

在本节快速开始指南中，您学习了在本地计算机上编译并运行一个单链区块链的方法。若想学习各种功能的个性化设置，请参阅以下资源：

- [架构] (https://docs.substrate.io/fundamentals/architecture/)

- [运行时开发] (https://docs.substrate.io/fundamentals/runtime-development/)

- [为Substrate服务的Rust] (<https://docs.substrate.io/fundamentals/rust-basics/)>

## 2.2 基础知识

基础知识章节的话题将围绕Substrate开发环境的核心准则和独特功能特性展开；此外，我们也将强调数个可供区块链开发者选择的设计选项。

Substrate提供了一组模块化的、灵活的工具库，为您在开发不同特定用途的区块链时随意选择、组合、修改并复用。无论是在私有网络中运行，抑或是经由Polkadot与其他区块链联动，Substrate均能胜任。

本章的话题将帮助您了解：通过Substrate能够搭建起什么样的区块链系统？这些系统能够贴合您的实际需求的秘诀又何在？

在我们正式使用Substrate开始开发之前，请您了解以下板块的知识：

- [区块链基础] (/fundamentals/blockchain-basics/) 向您解释了开发区块链系统的复杂程度，并介绍了常见的区块链概念、组件和术语。

- [为何选择Substrate？] (/fundamentals/why-substrate/) 强调了Substrate所提供的，其他大部分区块链与智能合约平台所不具有的关键优点。

- [Substrate是什么？] (/fundamentals/what-is-substrate/) 解释了Substrate设计背后的核心理念，以及设计决策对我们使用的技术的影响。

- [架构] (/fundamentals/architecture/) 描绘了Substrate中节点架构的关键组件，以及这些组件如何与您自己的个性化区块链设计架构相关联。

- [网络与区块链] (/fundamentals/node-and-network-types/) 定义了在不同区块链部署场景下的不同拓扑结构，以及它们在基于Substrate的区块链中的实现。

- [运行时开发] (/fundamentals/runtime-development/) 强调了Substrate运行时的重要性，介绍了Substrate运行时的核心应用接口和运行之的最低要求。

- [共识] (/fundamentals/consensus/) 介绍最常见的共识模型，以及您能够在Substrate区块链上使用的共识模型的种类。

- [交易与区块基础] (/fundamentals/transaction-types/) 介绍交易的种类，以及构成一个区块的组件。

- [交易周期] (/fundamentals/transaction-lifecycle/) 介绍交易是如何被接收、排队处理，并最终封装进区块的。

- [状态转换与存储] (/fundamentals/state-transitions-and-storage/) 描述在运行时中，状态如何转换与存储，以及使用字典树和键值对加以管理。

- [账户、地址与密钥] (/fundamentals/accounts-addresses-keys/) 介绍了账户、地址与密钥的相互关系以及它们的用途。

- [为Substrate所用的Rust] (/fundamentals/rust-basics/) 强调了Rust特有的traits（特征）、泛型，关联类型和宏。熟练掌握这些知识，您就能顺利地构建基于Substrate的区块链了。

- [离链操作] (/fundamentals/offchain-operations/) 揭示了一部分操作最好离链操作的原因，以及它们的平替。

- [Substrate连接中的轻量级客户端] (/fundamentals/light-clients-in-substrate-connect/) 介绍了使用Substrate连接在您的应用程序中嵌入一个轻量级客户端的方法，以与其他基于Substrate的链互交互。

- [跨共识通信] (/fundamentals/xcm-communication/) 一份有关跨共识通信及其通信格式的概述。

在您消化了以上章节的知识之后，您就已经完全准备好设计、构建、测试属于您的区块链了。

### 2.2.1 区块链基础

区块链是一个去中心化的分布式账本，它以一系列的区块记录信息。区块包含的信息是一组有序的指令集合，这些指令可能导致状态的变化。

在区块链网络中，各个计算机（称为节点）相互通信，形成了去中心化的点对点（P2P）网络。在这样的网络中，并不存在所谓中央权威机构控制；每个参与区块生成的节点，都存储着组成所有节点公认的区块链区块的副本。

多数情况下，用户通过提交请求来与区块链交互，这些请求可能导致区块链的状态转变。例如，一个请求可能改变一份文件的所有权，或者在两个账户之间转移资产。这些交易请求将由网络中的其他节点传播，并由区块生产者（block author）封装成一个区块。

为了确保链上数据的安全性和链上正在进行的事务的进度，节点使用某种形式的共识，就每个区块中的数据状态和已完成执行的事务的顺序达成一致。

#### 2.2.1.1 什么是区块链节点？

抽象至一定高度后，所有的区块链节点都可以认为由以下四部分组成：

- 数据存储区，它们记录交易之后产生的不同状态

- 点对点网络，用于去中心化地与其他节点通信

- 共识算法，用于保护自己免受恶意攻击者的攻击，确保链上事务顺利执行

- 为收到的交易进行排序、处理的算法逻辑

- 加密算法，用于为区块生成哈希摘要，为交易进行签名，以及验证收到的交易的签名

由于搭建区块链系统所需的核心组件的实现复杂度非常大，大多数区块链项目都是基于一份现成的代码作为基础，这样一来，程序员就可以通过修改已有代码来增添新功能，而非自己从零实现一个了。

例如，Litecoin、ZCash、Namecoin和Bitcoin Cash都是从比特币的代码仓库中fork出去的。相似地，Quorom，POA Network，KodakCoin和Musicoin是从以太坊的代码仓库fork出去的。

然而，大多数区块链平台并不是为个性化定制区块链而准备的。其结果就是，仅仅通过fork已有的仓库来构建新区块链有非常严重的局限性，包括但不限于原区块链代码带来的可延展性差等问题。

在我们探索Substrate在区块链项目中缓解上述问题的妙招之前，您很有必要了解一些大部分区块链共有的常见属性。通过学习大部分区块链的操作方法，您将更好地了解Substrate如何为构建最适合您需求的区块链提供替代方案和功能。

#### 2.2.1.2 状态转换与冲突

区块链的本质是[状态机] ((https://en.wikipedia.org/wiki/Finite-state\_machine))。

在状态机运行的任意时刻，它都拥有一个内部的状态。

当执行收到的交易时，它们会导致状态的更改，因此区块链必须从当前状态转换到新状态。

但是，可能存在多个有效转换，这些转换将导致不同的新状态，区块链必须在其中选择一个，使之可以与其他节点产生共识。

为了就状态转换之后的新状态达成共识，所有区块链中的操作都必须是确定的。为使链能够正常运转，居多数的节点必须就状态转换之后的新状态达成共识，这包括：

- 这条链的初始状态，又被称作“创世状态”或者“创世区块”

- 每一块区块中记录的，因发生交易而产生的状态变化

- 链中即将加入的，最终的状态

在中心化的网络中，“中心权威机构”可以在互斥状态转换之间进行选择。例如，配置为主要授权机构的服务器可能会按照看到它们的顺序，记录状态转换的更改；或者在冲突发生时，使用加权的算法在相互冲突的方案之间进行选择。

区块链用于将事务批量处理成区块并选择哪个节点可以向链提交区块的方法，称为区块链的共识模型或共识算法。最常用的共识模型被称为工作量证明共识模型。在工作量证明共识模型中，第一个完成计算问题的节点才有权向链提交区块。

为了使区块链具有容错能力并令所有节点拥有一致的状态，一些共识模型要求：即使某些节点受到恶意行为者干扰或网络中断威胁，至少三分之二的节点也必须在任何时候都对状态达成一致。

这三分之二的多数节点确保了网络是容错的，并且可以承受一些网络参与者的不良行为，无论这些行为是故意的还是偶然的。

#### 2.2.1.3 区块链经济

所有区块链都需要资源来执行各种操作，例如处理器、内存、存储和网络带宽。参与网络的计算机（生产区块的节点）会向区块链用户提供这些资源。这些节点创建了一个分布式的、去中心化的网络，以满足网络用户（社区）的需求。

为了维护社区运转并使区块链可持续发展，大多数区块链要求用户以手续费的形式为他们使用的网络资源付费。要支付手续费，用户的身份就必须与持有某种类型资产的账户相关联。区块链通常使用“代币”来代表账户中的资产价值，用户通过“交易所”在链外购买代币。这样一来，用户就可以获得代币，为他们的交易付款了。

#### 2.2.1.4 区块链管理方法

一些区块链允许用户就影响网络运营或区块链社区的问题提交提案，并为它们投票。通过提交和投票提案，区块链社区可以民主地决定区块链的发展方向。

然而，链上管理相对罕见；用户需要在账户中持有大量代币，或者被选为“用户代表”，才有可能参与管理。

#### 2.2.1.5 区块链上运行的应用程序

运行在区块链上的应用程序（通常被称为去中心化应用程序或dapps）通常是使用前端框架编写的web应用程序，但它们能在后端使用智能合约来更改区块链状态。

所谓智能合约，乃是在区块链上运行的程序，在特定条件下，它们能代表用户执行交易。开发人员可以编写智能合约，以确保程序执行的事务的结果被记录下来，并且不会被篡改。

然而，仅仅通过智能合约，开发人员无法访问一些底层的区块链功能——例如共识层、存储层或交易层——相反，开发人员必须遵守链的固定规则和限制。智能合约开发人员通常将这些限制作为一种权衡，以缩短开发时间，不必过多留意核心设计上的决策。

#### 2.2.1.6 下一步该学什么？

所有的区块链都有其共通之处。Substrate（虽然它本身不是个区块链）是一组区块链开发者的工具套件，提供了模块化的框架组件来创造个性化的区块链。有了Substrate的帮助，您就可以随意使用常见的区块链组件（例如存储，共识和加密组件），将它们加以组合甚至改进，以最佳地贴合您的需求。您可以参阅以下链接以了解更多信息。

**请告诉我更多！**

- [基础知识] (/fundamentals/)

- [为何选择Substrate？] (/fundamentals/why-substrate/)

- [架构] (/fundamentals/architecture/)

- [网络与区块链] (/fundamentals/node-and-network-types/)

**请引导我实践！**

- [搭建本地区块链] (/tutorials/get-started/build-local-blockchain/)

- [模拟网络] (/tutorials/get-started/simulate-network/)

- [加入可信节点] (/tutorials/get-started/add-trusted-nodes/)

若您偏爱直接上手研究代码，请移步[Substrate Playground] (https://docs.substrate.io/playground/)或者参阅API参考信息，以了解您将使用的Rust crates。

### 2.2.2 为何选择Substrate？

进行区块链开发非常复杂。您必须正确掌握一系列复杂的技术（例如高级密码学和分布式网络通信），以便为应用程序运行和用户信任提供一个安全的平台。在拓展性、管理、组件间协作和可升级性，都是程序员需要面对的难题。这种复杂性为开发人员创造了一个相当高的门槛。

既然如此，我们不禁想问出第一个问题：我们到底想构建一个什么东西？

Substrate并非放之四海而皆准的万金油，但如果您想构建这样的项目，那Substrate大概率是您的最佳选择：

- 您的项目用途具有很高的特殊性

- 您的项目需要和其他区块链互动

- 您的项目涉及到预定义的、可组合的模块化组件

- 您的项目需要拥有随着时间升级的能力

Substrate是一个软件开发工具包（SDK），它能为您提供构建区块链所需的所有基本组件，您只需专注于制作使您的链独特和创新的逻辑即可。

和其他的分布式账本不同的是，Substrate具有以下特点：

- [灵活] (#flexible)

- [开源] (#open)

- [可协作] (#interoperable)

- [可升级] (#future-proof)

- [可衔接的（Where to go next）] (#where-to-go-next)

#### 2.2.2.1 灵活

大多数区块链平台都有非常紧密耦合的子系统，并且很难解耦。倘若其中的某几个子系统存在风险，这些难以发现的耦合可能严重破坏区块链系统本身。

相比之下，Substrate是一个完全模块化的区块链框架，通过选择适合您的项目的网络堆栈、共识模型或管理方法或创建自己的组件，您可以使用显式解耦的组件组成一个链。有了Substrate，您可以部署根据您的规范设计和构建的区块链，但它也可以随着您不断变化的需求而发展。

#### 2.2.2.2 开源

所有的Substrate体系结构和工具都可在遵循[开源许可协议] (https://github.com/paritytech/substrate#license)的前提下使用。Substrate框架的核心组件均使用像libp2p和jsonRPC这样的开源协议，并且允许您决定如何定制您的区块链架构。

Substrate也有一个活跃、乐于助人的大型[构建者社区] (https://substrate.io/ecosystem/)，源源不断地为软件生态做出贡献；这些贡献允许您在开发自己的区块链时，向其引入新的增强功能。

#### 2.2.2.3 可协作

大多数区块链平台与其他区块链网络的交互能力有限。与之不同的是，所有基于Substrate的区块链都可以通过[跨共识消息传递] (https://wiki.polkadot.network/docs/learn-crosschain) (XCM)与其他区块链进行互操作。

Substrate可用于创建作为独立网络（单链）的区块链，或与[中继链](https://wiki.polkadot.network/docs/learn-architecture#relay-chain)紧密耦合以作为[平行链（parachain）] (https://wiki.polkadot.network/docs/learn-parachains)共享其安全性。

#### 2.2.2.4 可升级

Substrate为可升级、可组合和可适应性而生。

状态转换逻辑（Substrate运行时，下称运行时）是一个自包含的WebAssembly对象，而您的节点拥有在特定条件下完全改变运行时本身的能力，从而在网络范围内进行运行时升级。因此，“无分叉”升级（forkless upgrades）是可能的，因为在大多数情况下，节点不需要任何操作就可以使用这个新的运行时。随着时间的推移，网络的运行时协议可以无缝地（也许是根本地）随着用户的需求而发展。

#### 2.2.2.5 下一步学什么？

Substrate提供了一套用于构建区块链和链上应用程序的、独树一帜的强力框架。

- 您可以从一个预定义的节点入手，启动一条没有节点的单链

- 您可以使用Substrate构建一条独立于其他区块链的定制单链

- 您可以将您的Substrate区块链耦合入一条接力链，例如Polkadot或者Kusama

根据您的知识储备和兴趣点，您可选择参阅以下链接：

**请告诉我更多！**

- [区块链基础] (/fundamentals/blockchain-basics/)

- [架构] (/fundamentals/architecture/)

- [运行时开发] (/fundamentals/runtime-development)

- [网络与区块链] (/fundamentals/node-and-network-types/)

- [安装] (/install/)

**请引导我实践！**

- [搭建本地链] (/tutorials/get-started/build-local-blockchain/)

- [模拟网络] (/tutorials/get-started/simulate-network/)

- [添加可信的验证器] (/tutorials/get-started/add-trusted-nodes/)

若您偏爱直接上手研究代码，请移步[Substrate Playground] (https://docs.substrate.io/playground/)或者参阅[API参考信息] (https://paritytech.github.io/substrate/master)，以了解您将使用的Rust crates。

### 2.2.3 Substrate是什么？

[为什么选择Substrate？] (/fundamentals/why-substrate/) 介绍了一些Substrate开发环境的核心特点和优势。

如果这些优势（灵活性、开源、可协作和可升级性）对您很重要，那么您一定会想知道Substrate是什么，以及我们为何如此设计它的。

话不多说，让我们进入正题吧。

#### 2.2.3.1 多链视角

Substrate是一个基于rust的框架，它使用模块化和可扩展的组件构建区块链应用程序，以支持多链的区块空间（multi-chain block space）。学习如何使用这个框架的重点在于理解这个概念：Substrate设想并包含了一个“多链宇宙”（multi-chain universe）。

Substrate不是一个功能受限，可拓展性差的单链；相反，它专门设计为能够满足特定领域的细分需求，并具有相当的可优化和可扩展性。未来，在这样多链结构的加持下，区块链可以广泛用于不同的应用场景，并用于解决特定的问题或用例。

Substrate旨在使区块链易于定制和升级，以便项目可以随着时间的推移而发展以紧跟潮流。与使用不甚灵活的协议的单一主导链不同，Substrate的设计允许跨多个链进行实验、协调和协作。

但是，如果区块空间由许多不同的区块链组成，执行特定的任务，我们应当如何构建这些区块链呢？从头构建多链非常昂贵、耗时，而且容易出错。它需要在多个技术和学科上的金钱投资和先进的专业知识，犯错的代价可能会令人望而却步。这样的代价对于大多数不愿冒进的人来说可谓不可接受。

有如此高的准入门槛，那么多链结构的未来应当何去何从？如果我们要克服所有这些困难来建造一个区块链，应当如何升级它以保持它与社会和技术发展的相关性?

#### 2.2.3.2 核心设计原则

在Substrate之前，人们花费了大量的资源，试图设计一个多链结构的未来区块链，建立一个具有所有正确组件的、面向未来的、可升级的区块链。然而，所有这些花费的时间、金钱和精力最终导向了一个结论：当下做出的深思熟虑的选择很可能成为未来的绊脚石。如果区块链依赖于某种特定的技术或假设，随着时间的推移，这种技术或假设可能会阻碍并最终扼杀创新。

以上结论，以及为未来的多链提供工具的需求，促成了Substrate的问世。以下定义原则是Substrate框架的核心设计决策:

- 将Rust作为代码库的核心编程语言。

- 将WebAssembly作为应用程序逻辑的执行环境。

- 广泛使用分层抽象、泛型实现和灵活的API作为主要的编码实践，并将库分离为不同的体系结构组件（architecture component）。

以上设计决策，都在Substrate的运作方式以及通过可重用和可定制的组件构建专门的可组合区块链方面，发挥着重要作用。

#### 2.2.3.3 Substrate为何选择了Rust？

尽管学习起来可能有些复杂，选择Rust作为Substrate框架的底层编程语言有几个重要的好处，它是一种高性能的系统级编程语言，用于构建高鲁棒性、关键性（mission-critical）的生产级软件：

- Rust有一个复杂的内存管理系统，它依赖于所有权规则，而不是手动内存分配和垃圾收集。这些所有权规则使编译器能够正确处理您使用的每个变量和引用的每个内存地址，以避免内存泄漏。

- Rust编译器可以防止引入存在潜在安全漏洞的编码错误。

- Rust提供了抽象——比如特征和泛型——使代码更加灵活，而不会增加性能开销。执行经过编译的程序时，其性能可与目前最快的编程语言相媲美。

- 编译后的Rust代码占用空间相对较小，可以在支持任意操作系统的嵌入式设备上运行。

除了所有这些好处之外，Rust代码还可以编译为WebAssembly，而WebAssembly是使Substrate拥有灵活性和运行时可升级性的核心。

#### 2.2.3.4 WebAssembly和可升级性

Substrate区块链中的大多数组件都是可定制的，或者支持多种实现以获得最大的灵活性。但是，将程序逻辑以WebAssembly二进制文件的形式存储在区块链上是少数几个不能更改的Substrate属性之一。如果您没有使用WebAssembly来表达程序逻辑，那么您使用的就不再是Substrate了。基于Substrate的区块链拥有与生俱来的可升级性。

这是通过将区块链的应用程序逻辑（专门的业务逻辑）编码为WebAssembly二进制文件，并将该文件存储为区块链状态的一部分来实现的。就像您可以发起一个交易来更改帐户余额一样，您也可以发起一个更改WebAssembly代码的交易。在生成包含该交易的区块之后，下一个区块将能反映更新后的区块链状态。

#### 2.2.3.5 抽象与泛化

除了WebAssembly二进制文件中的核心应用程序逻辑之外，您还可以定制许多其他基于Substrate的区块链的组件。例如，共识层、网络层和数据库层均存在多种实现，包括:

- 多种共识引擎（BABE/Grandpa/AURA）

- 多种网络协议（QUIC，TCP）

- 多种数据库实现（ParityDB，RocksDB）

在Polkadot的生态中，有许多对Substrate节点进行定制了的平行副链项目示例。在许多情况下，这些生态中的应用程序有访问链的核心组件（如共识层或交易池）的需求；如果它们被编写为智能合约应用程序，就不可能实现。

在网络的应用层中，您甚至拥有更大的灵活性。例如，您可以在基于帐户的分类帐格式（account-based ledger）或UTXO格式之间进行选择。无论是交易手续费的实现，还是定义围绕资源使用的自定义激励措施，掌控权均在您手中。

综上所述，使用Substrate，您可以通过定制和扩展框架来满足您的需求，从而构建您可以想象的任何类型的区块链。

### 2.2.4 架构

正如我们在[区块链基础] (/fundamentals/blockchain-basics)一章中所说，一个区块链依赖于去中心化的网络，在网络中，数台电脑（亦称为节点）之间可以相互通信。

由于任意区块链中必然包含节点这一核心概念，我们有必要了解让Substrate节点之于其他普通节点的独到之处，例如默认提供的核心服务和库，以及改造并拓展节点使之适用于多种工况的方法。

#### 2.2.4.1 高层次概述

在一个去中心化的网络中，所有的节点不仅充当客户端（向网络请求数据），也能发挥服务器的作用（对网络请求作出响应）。从概念上来说（同时也可以从编程的角度来说），Substrate架构也按照类似的方式划分节点的操作职责。

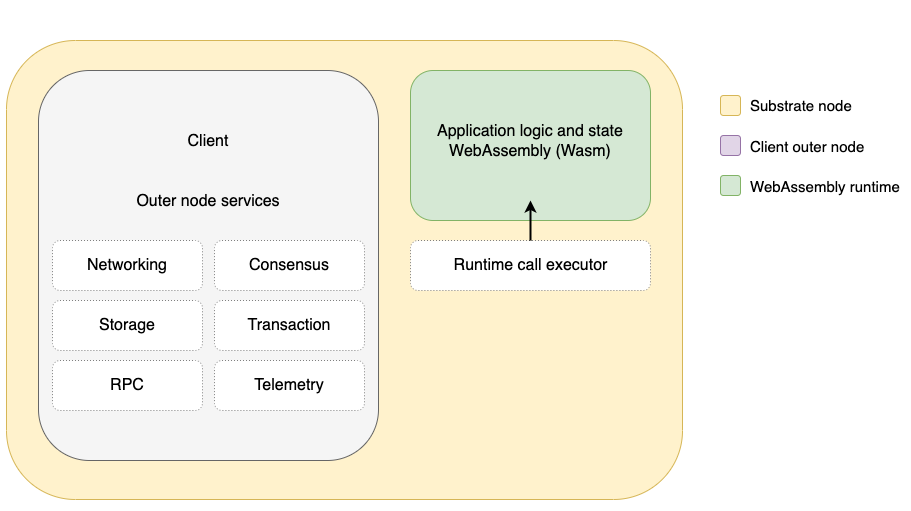
下图简要说明了这种职责的划分，以帮助您直观了解Substrate架构，以及Substrate如何为构建区块链提供模块化框架。

图2-1 Substrate架构

高层次概括地说，Substrate节点提供了一个层次化的环境，其中包含两个主要的元素：

- 一个带有外部节点服务（outer node services）的客户端，用于处理网络活动，如发现对等节点、管理交易请求、与对等节点达成共识以及响应RPC调用。

- 一个包含了执行区块链的状态转换函数的所有业务逻辑的运行时。

#### 2.2.4.2 外部节点

外部节点负责运行时外部发生的活动。例如，外部节点负责发现对等节点、管理交易池、与其他节点通信以达成共识，以及响应来自“外部世界”的RPC调用或浏览器请求。

外层节点处理的一些最重要的活动包括以下几种：

- [存储] (/fundamentals/state-transitions-and-storage/): 使用简单高效的键值对存储层，保存Substrate区块链不断变化的状态。

- [点对点网络] (/fundamentals/node-and-network-types/): 使用[libp2p network stack](https://libp2p.io/)的Rust实现与其他的网络参与者通信。

- [共识] (/fundamentals/consensus/): 与其他网络参与者通信，以确保他们对区块链的状态达成共识。

- [Remote procedure call（RPC） API] (/build/remote-procedure-calls/): 接收入站的HTTP和WebSocket请求，以便区块链用户与网络交互。

- [维护节点度量] (/maintain/monitor/): 通过内嵌的[Prometheus](https://prometheus.io/)服务器收集并提供节点度量相关的信息。

- [执行环境] (/build/build-process/): 为运行时选择要使用的执行环境（WebAssembly或本地的Rust）然后将调用分派给所选的环境。

执行这些任务通常需要外部节点查询运行时以获取信息或向运行时提供信息。这种通信通过调用专门的[runtime APIs] (/reference/runtime-apis/)来处理。

#### 2.2.4.3 运行时

运行时确定交易是否有效，并负责处理区块链的状态转换函数的更改。因为运行时能执行它接收到的函数，所以它可以控制如何将交易包含在区块中，以及如何将区块返回到外部节点以传播或导入到其他节点。从本质上讲，运行时负责处理区块链上发生的所有事情，它也是构建Substrate区块链节点的核心组件。

Substrate运行时被设计成编译为[WebAssembly (Wasm)] (/reference/glossary#webassembly-wasm)字节码的形式。这样的设计决策有如下益处：

- 支持无分叉的升级。

- 多平台适配。

- 可以检查运行时有效性。

- 已验证可用于中继链共识机制。

与外部节点向运行时提供信息的方式类似，运行时使用专门的[主机函数（host function）] (https://paritytech.github.io/substrate/master/sp\_io/index.html)与外部节点或外部世界通信。

#### 2.2.4.4 轻量级（light）客户端节点

轻量级客户端或轻量级节点是Substrate节点的简化版本，仅提供运行时和当前状态。它允许用户使用浏览器、浏览器扩展、移动设备或家用电脑直接连接到Substrate运行时。

有了轻量级客户端节点，您就可以使用用Rust、JavaScript或其他语言编写的RPC端口（endpoint）连接到WebAssembly执行环境，以读取区块头（block header）、提交交易并查看其的结果。

#### 2.2.4.5 下一步学什么？

在您已经对Substrate体系结构和核心节点组件有了基本的认知后，我们建议参阅以下主题以了解更多信息：

- [网络和区块链] (/fundamentals/node-and-network-types)

- [交易和区块基础] (/fundamentals/transaction-types)

- [交易生命周期] (/fundamentals/transaction-lifecycle/)

- [状态转变和存储] (/fundamentals/state-transitions-and-storage/)

- [Runtime APIs] (/reference/runtime-apis/)

### 2.2.5网络与区块链

在考虑构建区块链时，我们必须要认识到：定义一个网络离不开边界（boundaries）这一概念。例如，连接到单个路由器的一组计算机可以被认为是一个家庭网络，而防火墙可能是企业网络的边界。通过公共通信协议，较小的孤立网络也可以连接到更大型的网络。

类似地，我们可以认为，区块链网络乃是由其边界，和与其他区块链的关系（孤立、或有所通信）定义的。

作为区块链工程师的工具包，Substrate能够用于开发您能想到的任何类型的区块链，并根据特定于应用程序的需求定义其边界。考虑到这种灵活性，您需要决定您想要构建的网络类型是什么，以及不同节点在该网络中可能扮演的角色。

#### 2.2.5.1 网络类型

基于Substrate的区块链可用于不同类型的网络架构，例如下列的数种：

- 私有网络 仅能访问受限的数个节点的网络。

- 单链网络 拥有各自的安全协议实现，并不与其他区块链网络通信。比特币和以太坊就是这样的网络，不过他们不基于Substrate开发。

- 中继链（Relay chains）网络 为连接到它们的其他网络提供去中心化的安全保障和通信机制。

- 平行多链（Parachains）网络 为连接到中继链而生，也能与使用同一个中继链的其他网络通信。因为平行多链依赖中继链来封装产生的区块，所以平行多链必须使用与它们的目标中继链相同的共识协议。

#### 2.2.5.2 节点类型

区块链要求网络节点同步，以呈现一致且最新的区块链状态视图。每个同步节点存储区块链的副本，并跟踪收到的交易。然而，保存整个区块链的完整副本需要大量的存储空间和计算量，并且下载从创世块到最新区块的所有区块对于大多数区块链的应用场景来说是不实际的。

为了更容易维护链的安全性和完整性，同时减少想要访问区块链数据的客户端对资源的消耗，有不同类型的节点可以与链交互:

- [完全节点（Full nodes）] (#full-nodes)

- [归档节点（Archive nodes）] (#archive-nodes)

- [轻量级客户端节点（Light client nodes）] (#light-client-nodes)

**完全节点**

完全节点是区块链网络基础设施的关键部分，也是最常见的节点类型；其负责存储区块链数据，并且一般会参与常见的区块链操作，例如创建和验证区块，接收和验证交易，以及响应用户请求。

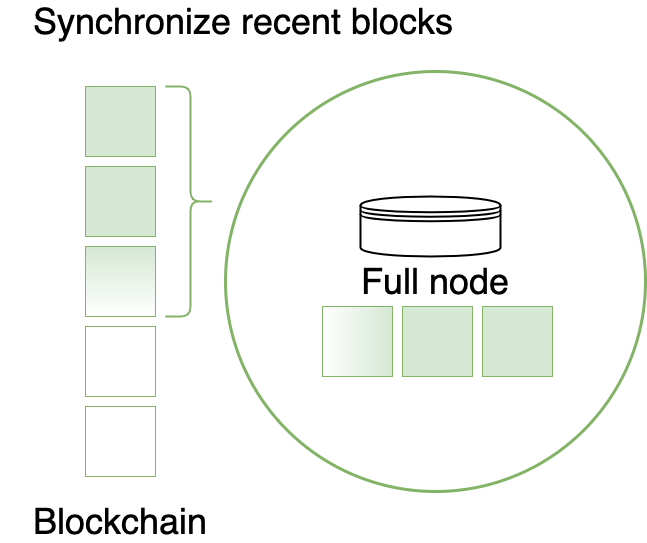


图2-2 完全节点

默认情况下，完全节点被配置为只存储最近的256个块，并丢弃比这更早的状态（创世块除外），以防止全节点无限增长进而消耗所有的磁盘空间。不过，我们也可以配置完全节点保留的区块数量。

虽然较久远的区块被丢弃了，但完全节点会保留从创世块到最近区块的所有[块报头] (/reference/glossary/#header)，以对状态的正确性进行验证。完全节点可以访问所有的块标头，所以可以通过执行从创世块开始的所有块（executing all of the blocks from the genesis block）来重建整个区块链的状态。检索有关过往状态的信息需要很大的计算量，故通常应该使用存档以代替上述过程。

完完全节点允许您读取链的当前状态，并直接在网络上提交和验证交易。由于丢弃了久远区块中的状态，完整节点所需的磁盘空间比归档节点少得多。不过，一个完全节点需要更多的计算资源来查询和检索关于某些先前状态的信息。如果需要查询历史区块，应该清空这个完全节点，然后将其作为存档节点重新启动。

**归档节点**

归档节点类似于完全节点，不过它们会存储所有过去的区块，每个区块都存储有完整的状态信息。这种节点往往在需要访问历史信息的场景下使用（如块浏览器、钱包、论坛和类似的应用程序）。

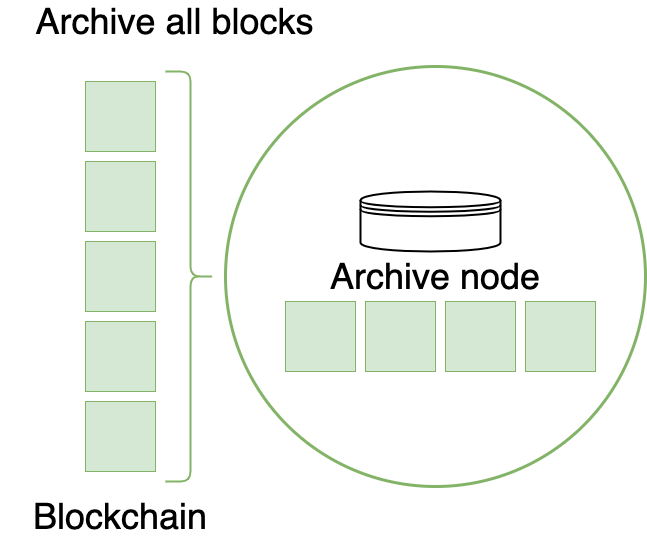


图2-3 归档节点

由于归档节点保存着历史状态，它们将消耗大量磁盘空间，这也是它们不如完全节点常见的原因。不过，归档节点可以方便地查询区块链在任何时间点的过去状，例如查找某个区块中的帐户余额，或者查看导致指定状态更改的交易的详细信息。当在归档节点存储的数据上运行这些类型的查询时，它们会更快、更高效。

**轻量级客户端节点**

轻量级客户端节点使您能够以最低的硬件要求连接到Substrate网络。

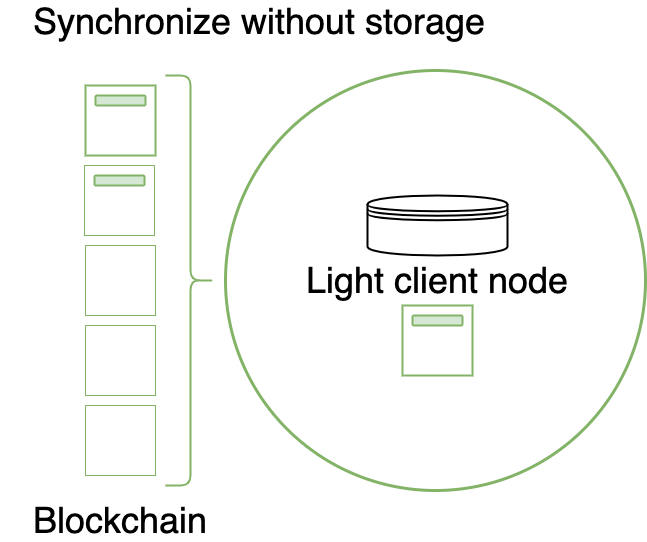


图2-4 轻量级客户端节点

因为轻量级客户端节点需要的系统资源最少，所以它们可以嵌入到基于web的应用程序、浏览器扩展、移动设备应用程序或物联网（IoT）设备中。这类节点通过RPC端口提供运行时和对当前链状态的访问。轻量级客户端节点的RPC端口可以用Rust、JavaScript或其他语言编写，用于读取区块头、提交交易和查看其结果。

轻量级客户端节点不参与区块链和网络操作。例如，轻客户端节点不负责产生或验证区块，也不会传播交易或达成共识。它们也不会存储任何过去的块，因此如果不向拥有历史数据的节点请求，它就无法读取历史数据。

#### 2.2.5.3 节点规则（Node roles）

根据启动节点时指定的命令行选项，节点可以在区块链的运行过程中扮演不同的角色，并可以提供对链上状态的不同级别的访问。例如，您可以指定哪些节点有权生产新区块，哪些节点可以与对等节点通信。未被授权生产区块的对等节点可以导入（import）新区块，接收交易，并向其他节点发送和接收有关新交易的消息（gossip）。除此之外，您还可以阻止节点连接到更广泛的网络，并限制节点与特定节点通信。

#### 2.2.5.4 下一步学什么？

您可以使用Substrate构建几乎任何类型的网络——从完全自包含的私有单链到您自己的中继链生态系统或兼容的平行副链。

请参阅以下页面，以更深入地研究网络和节点类型：

- [构建本地链] (/tutorials/get-started/build-local-blockchain/)

- [模拟网络] (/tutorials/get-started/simulate-network/)

- [添加可信节点] (/tutorials/get-started/add-trusted-nodes/)

- [认证指定节点] (/tutorials/get-started/authorize-specific-nodes/)