ブロックチェーンの霧が晴れる

最初のトランザクションから
RpcNodeで作る自分のサービスまで



やさしい言葉で・図と例つき

やさしい言葉で

ブロックチェーンの霧が晴れる

最初のトランザクションから、RpcNodeを使った自分のサービスまで



こんな人のための本です

- 「ブロックチェーン」という言葉は聞いたことがあるが、仕組みはわからない。
- 自分のサイトで暗号資産を受け取りたい。
- プログラムは書くが、ノードに問い合わせたことはまだない。

必要なもの

- 好奇心。予備知識はいりません。
- 例を試すための、ターミナルが使えるパソコン。
- 最初のリクエストは、登録もキーもなしで動きます。

各章の読み方

主人公の疑問　解説　やってみよう　落とし穴　考えてみよう

著者：rpcnode.dev · 2026

目次

部 1. なぜプロバイダーが必要なのか

1. 暗号資産で支払いを受けたい 4
2. 全部自分でやる費用 9
3. プロバイダーと自前ノード 15

部 2. ゼロから学ぶブロックチェーン

4. 仲介者のいないお金と記録 21
5. ブロックとチェーン 26
6. 鍵、アドレス、署名 31
7. トランザクションの旅 37
8. ネットワークはどう合意するのか 42
9. コイン、トークン、スマートコントラクト 48

部 3. ノードとRPC

10. ノードの種類 54
11. ノードとの話し方 59
12. 自前ノードの内側 65
13. プロバイダーの選び方 71

部 4. ネットワークの仕組み

14. Bitcoinとその親戚たち 77
15. Ethereum：アカウント、ガス、イベント 82
16. Ethereumの親戚と「2階」 88
17. TRON：エネルギーとステーブルコイン 93
18. Solana：すべてはアカウントに 99
19. TON：電話ではなく手紙 105
20. 別のやり方をするネットワーク 111
21. どのネットワークを選ぶか 118

部 5. RpcNode

22. RpcNodeでの最初の5分 125
23. リクエストに何が起きるのか 131
24. 履歴とアーカイブデータ 137
25. プラン、呼び出しの値段、上限 143
26. アドレスのイベント：Webhookとソケット 149
27. お支払いとトランザクションの追跡 155

部 6. 実践

28. お店を組み立てる 161
29. 何もかもうまくいかないとき 168
30. 安全対策と公開前チェックリスト 175

部 7. この先へ

31. DeFi：仲介者のいない金融 182

巻末

おわりに 189
付録A　用語集 191
付録B　ネットワーク早見表 193
付録C　リクエスト早見表 195

第1章　暗号資産で支払いを受けたい。何が必要？

始まりは一人のお客さんでした

アルチョムは小さなネットショップを開いています。コーヒー豆を焙煎して、郵便で送る店です。カード払いも、銀行振込も、代金引換も、もうちゃんと動いています。そこへ一通のメールが届きました。

「こんにちは！　2キロ注文したいのですが、暗号資産で払えますか？」

アルチョムに異存はありません。スマートフォンにウォレットを入れ、アドレスを手に入れて、お客さんに送ります。数分後、画面にお金が表示されました。

けれど10件目の注文ともなると、ウォレットを手で確認するのにうんざりしてきます。注文が支払い済みかどうか、サイトが自分で判断してくれたらいいのに。



解説：ノードとは何か

ブロックチェーンについて何か知りたければ、それを保管している相手に聞くしかありません。その相手をノードと呼びます。

誰が誰にいくら送ったかを書き込む、みんなで使うノートを思い浮かべてください。このノートは一つの金庫にしまわれているのではありません。世界中のたくさんの参加者が写しを持ち、つねに互いに照らし合わせています。これがブロックチェーンです。詳しくは第5章で見ていきます。

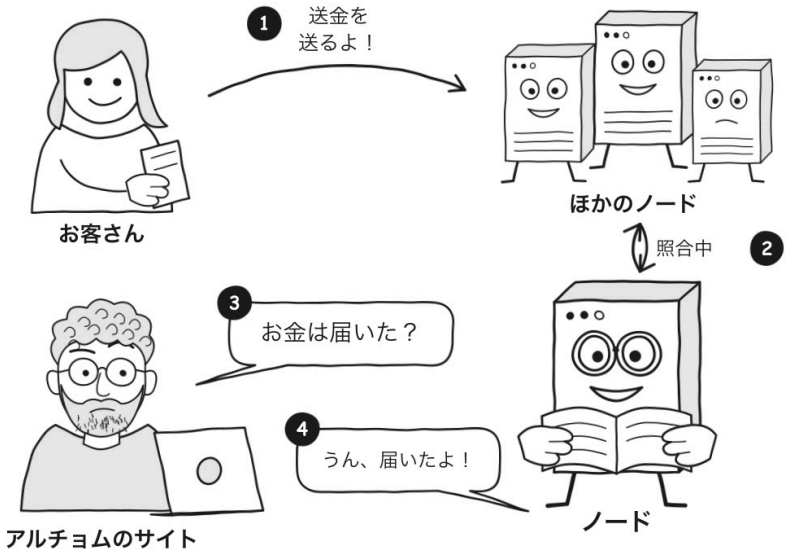
ノードとは、ネットワークのプログラムを動かしているコンピューターです。やることは4つあります。

1. 記録の写しを自分で保管する。

2. ほかのノードから新しい記録を受け取り、ネットワークのルールに沿って確かめる。
3. 自分に送られてきた送金を、先へ伝える。
4. 「残高はいくら？」「送金は通った？」といったプログラムの質問に答える。

ノードにもいろいろあり、すべての履歴を持っていないものもあります。それは第10章で。

アルチョムのサイトは、ブロックチェーンを直接のぞくわけではありません。ノードに質問し、ノードが答えるのです。



やってみよう：ノードへの最初の質問

プログラムは簡単なルールに従ってノードと話します。リクエストを送り、レスポンスを受け取る。そのルールの一つがJSON-RPCです。たとえばEthereumのノードに「このアドレスにはコインがいくつある？」と聞くには、こうします。

```
curl https://ノードのアドレス \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["ウォレットのアドレス","latest"]}'
```

ここで `ノードのアドレス` は「どこを訪ねるか」、`ウォレットのアドレス` は「誰の残高を聞くか」です。`latest` という言葉は「いちばん新しい時点で」という意味です。

ノードはこう答えます。

```
{"jsonrpc":"2.0","id":1,"result":"0xde0b6b3a7640000"}
```

でたらめな文字の並びに見えますが、これは16進数で書かれた数で、1,000,000,000,000,000,000のことです。ノードはコインのいちばん小さな単位で数えます。この単位をweiと呼び、1 ETHはちょうどこの数のweiです。つまり、このアドレスには1 ETHあります。

要するに、仕掛けは「質問と答え」だけです。あとは、この`ノードのアドレス`を誰のものにするかを決めれば済みます。

二つの道：自前のノードか、プロバイダーか

ノードは自分で動かすこともできます。あるいは、誰かがあなたの代わりに運用しているノードにつなぐこともできます。そうしたサービスがプロバイダーで、すぐに使える`ノードのアドレス`を発行してくれます。この本で扱うRpcNodeも、その一つです。

比べる点	自前のノード	プロバイダー
いつから始められるか	インストールと同期のあと。ネットワークによって数時間から数週間	登録すれば数分で
何にお金がかかるか	サーバー、ディスク、通信量、担当者の時間	サービスの利用プラン
誰が更新し、直すのか	あなた	プロバイダー
複数のネットワーク	ネットワークごとにノードが必要	たいていは一つのマイページで
どんなリスクがあるか	寝ている間にノードが遅れたり壊れたりする	他社のサービスに頼ることになる

どちらの道にも出番があります。特別にプライバシーを守りたい、負荷がとても大きい、ネットワークの運営そのものに参加する。そんなときは自前のノードに意味があります。

> アルチョム：「僕はコーヒーを焙煎するのが仕事で、サーバーの番人じゃない。でも、自分でやったら実際いくらかかるんだろう？」

手短に言うと、複数のネットワークを予備サーバーと全履歴つきで運用すれば、費用は数万ドルになります。小さなノードを一つ動かすだけなら、ずっと安く済みます。詳しい計算は第2章で。

落とし穴：「プロバイダーにウォレットの鍵を渡せばいい」

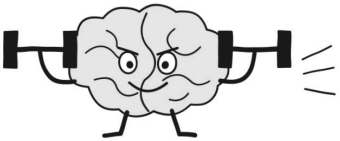
絶対にだめです。残高を知ったり支払いを確認したりするには、ノードにはアドレスだけで十分です。秘密鍵とシークレットフレーズは、お金を動かす力そのものです。知った人はすべて

を持ち去れます。まともなプロバイダーには必要のないものです。鍵については第6章で説明します。



もう一つ注意があります。上のリクエストでわかるのは、ネットワーク固有のコイン、ここではETHだけです。お客さんがUSDTのようなトークンで払った場合、この質問では見えません。トークンは聞き方が違います。第9章で扱います。

考えてみよう



頭の体操をしよう！

1. アルチョムが支払いを確認していますが、ノードはネットワークから1時間遅れています。彼には何が見え、お客さんに何と返事をするでしょうか。
2. お店の注文は1日3件です。もし3000件だったら？　自前のノードかプロバイダーかの選び方は、どう変わるでしょうか。

1問目のヒント：遅れたノードは、1時間前の状態を正直に見せているだけです。

まとめ

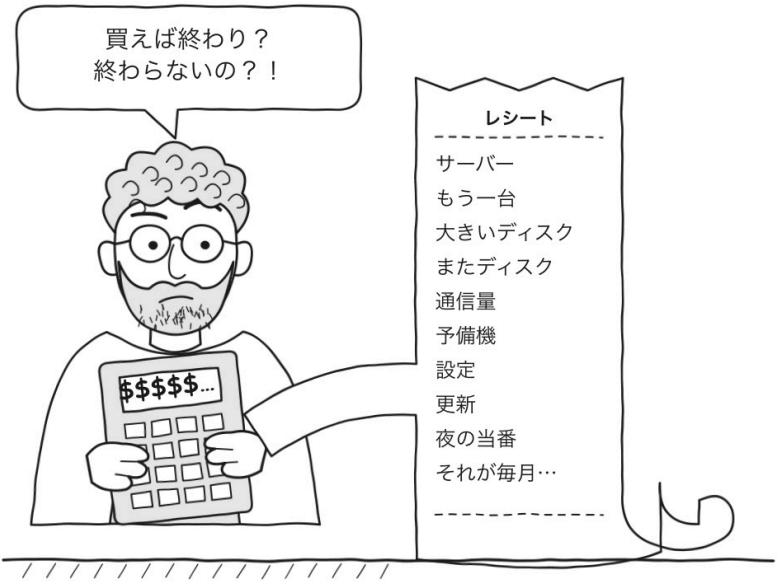
- プログラムは、ノードに質問することでブロックチェーンのことを知ります。
- ノードは自分で持つことも、プロバイダーから借りることもできます。
- ノードに必要なのはアドレスで、あなたの秘密鍵ではありません。

次へ：第2章。アルチョムは電卓を手に、自前のサーバーがいくらになるかを計算します。

第2章　全部自分でやると、いくらかかる？

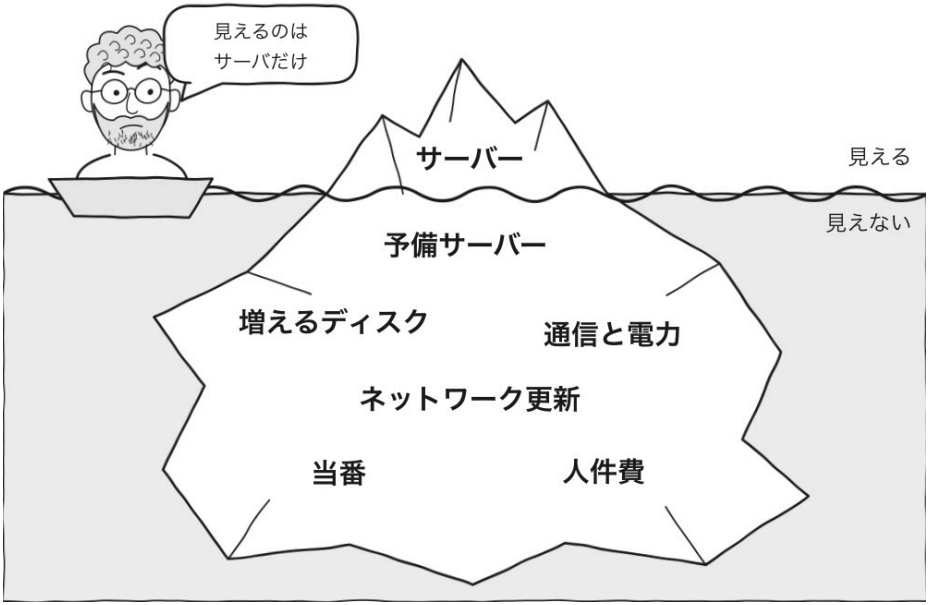
アルチョム、電卓を手にする

アルチョムの考えは単純です。「サーバーを買ってノードを入れれば、支払いは勝手に流れてくる。払うのは一度きりだ」。彼は電卓を開き、項目を書き出し始めました。



解説：費用の中身

サーバーは氷山の一角です。残りは、最初は見えません。



1. サーバー。プロセッサー、メモリー、そして高速なNVMeディスク。
2. 予備のサーバー。一台が壊れたら、直している間はもう一台が答えます。
3. ディスクは増え続ける。ブロックチェーンは新しい記録を足していくだけです。Bitcoinのチェーンは、2026年11月までの1年でおよそ700 GBから786 GBに増えました。
4. 通信量。ノードはつねにデータをやり取りしています。TONのライトサーバーは、ピーク時で月に約16 TBを必要とします。
5. 更新。ネットワークはルールを変えます。古いプログラムはついていけなくなります。
6. 人。誰かがこのすべてを設定し、見守り、夜中に起きなければなりません。

ネットワークごとに必要な機材

これは公式ドキュメントにある通常のノードの要件です。ネットワークの投票に参加するための要件ではありません。

ネットワーク	プロセッサー	メモリー	ディスク
Bitcoin	1 GHz以上	2 GB	推奨は750 GB。ただしチェーン自体がすでに約786 GB
Ethereum	4コア以上	32 GB	NVMe 4 TB
BNB Smart Chain	16コア	64 GB	SSD 3 TB以上。データベースの完全なスナップショットはすでに約7.4 TB、削減版は約1.9 TB
TRON	16コア	32 GB	SSD 3.5〜4 TB以上
Solana（RPCノード）	16コア以上	512 GB以上	NVMeを複数台。アカウント用1 TB以上、台帳用1 TB以上、スナップショット用500 GB以上

おかしな点に気づきましたか。Bitcoinでは、チェーンより小さいディスクが推奨されています。BNBは最低3 TBなのに、完全なスナップショットはその2倍をはるかに超えます。ドキュメントはいつもネットワークに追いつけるわけではないので、ディスクは余裕をもって用意します。

すべての履歴が必要なら？

通常のノードは、現在のことならよく知っています。「1年前の残高は？」と聞くには、アーカイブノードが必要です。これは第10章で扱います。アーカイブはずっと重くなります。TONのアーカイブは16〜20 TB必要です。Ethereumの大きさはプログラムによって大きく違い、2 TBから14 TB以上までさまざまです。以下の計算にアーカイブは含めていません。含めれば、もっと高くなります。

お金を計算する

サーバーは買うこともできますが、ここではレンタルで計算します。ホスティング各社の公開価格表をもとにした2026年11月27日時点の見積もりで、税金と初期費用は含みません。価格は変わります。

サーバーのクラス	中身	月額の目安
中型	8コア、64 GB、NVMe 2 TB×2	$180
大型	16コア、128 GB、NVMe 4〜8 TB	$460〜560
超大型	メモリー512 GB以上	$1,160〜2,350

次に、5つのネットワークです。価格は下限に近いほうを取り、ネットワークごとにサーバーを2台、本番機と予備機を置きます。

ネットワーク	クラス	1台の月額	予備込みの月額
Bitcoin	中型	$180	$360
Ethereum	中型	$180	$360
BNB Smart Chain	大型	$500	$1,000
TRON	大型	$500	$1,000
Solana	超大型	$1,200	$2,400
合計		$2,560	$5,120

> アルチョム：「月5,120ドルって、年にすると61,440ドルじゃないか。しかもこれ、レンタル代だけ！？」

そうです。最初の表の上限の価格を使うと、年におよそ$92,000になります。「数万ドル」というのは、こういうことです。

ネットワーク一つだけなら、ずっと安く済みます。Bitcoinは多くを要求しないので、中型より控えめなサーバーでも足ります。

では、誰が面倒を見るのか

サーバーは自分で自分の世話をしてくれません。「Habr Career」のデータによると、ロシアの中堅DevOpsエンジニアの月給はおよそ16万〜29万ルーブルです。一人で、休日も休暇もなく24時間の当番を務めることはできません。二人にしても、年におよそ380万〜700万ルーブルかかります。

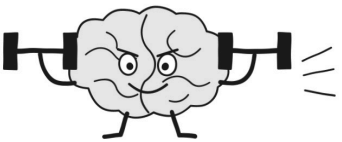
落とし穴：「サーバーを一台買えば終わり」

サーバーが一台だと、そこが壊れれば全部が止まります。ディスクがいっぱいになる、プログラムが固まる、ネットワークが更新される。するとサイトは支払いを確認できません。お客さんはお金を送ったのに、注文は「未払い」のままです。



よく忘れられることがもう一つあります。支払う前に、ホスティング会社の規約を読んでください。暗号資産に関わるプログラムを禁止し、サーバーを止めることがある会社もあります。

考えてみよう



頭の体操をしよう！

1. アルチョムに必要なのはBitcoinだけで、中型サーバーを予備込みで使います。年間いくらになりますか。
2. 5つのネットワークに、1台$500の大型サーバーを使う6つ目を、予備込みで追加しました。年間の費用はいくら増えますか。
3. 支払いをするお客さんは1日3回です。それだけの出費に見合うでしょうか。

答え：1）360×12＝$4,320。2）1,000×12＝$12,000。3）決めるのはあなたです。第3章が助けになります。

まとめ

- 支払いは一度きりではなく、毎月です。
- 5つのネットワークを予備込みで持つと、レンタル代だけで年$60,000を超えます。アーカイブと人件費は別です。
- ディスクは増え、ネットワークは更新され、誰かが当番をしなければなりません。

次へ：第3章。プロバイダーが引き受けてくれることと、それでも自前のノードが必要な場合。

数字の出どころ

すべてのページは2026年11月27日に閲覧しました。

- Bitcoinの要件：bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoinのチェーンの大きさ：api.blockchain.info/charts/blocks-size
- Ethereum: ethereum.org/developers/docs/nodes-and-clients/run-a-node
- BNB Smart Chain: docs.bnbchain.org/bnb-smart-chain/developers/node_operators/node_best_practices; github.com/bnb-chain/bsc-snapshots
- TRON: tronprotocol.github.io/documentation-en/using_javatron/installing_javatron; developers.tron.network/docs/deploy-the-fullnode-or-supernode
- Solana: docs.anza.xyz/operations/requirements
- TON: docs.ton.org/nodes/cpp/run-liteserver; docs.ton.org/nodes/cpp/run-archive-liteserver
- 給与：career.habr.com/professions/devops（2025〜2026年のデータ）

サーバーの価格は、複数のホスティング会社の公開価格表をもとに著者が丸めた見積もりです。

第3章　プロバイダーがしてくれることと、自前のノードが必要なとき

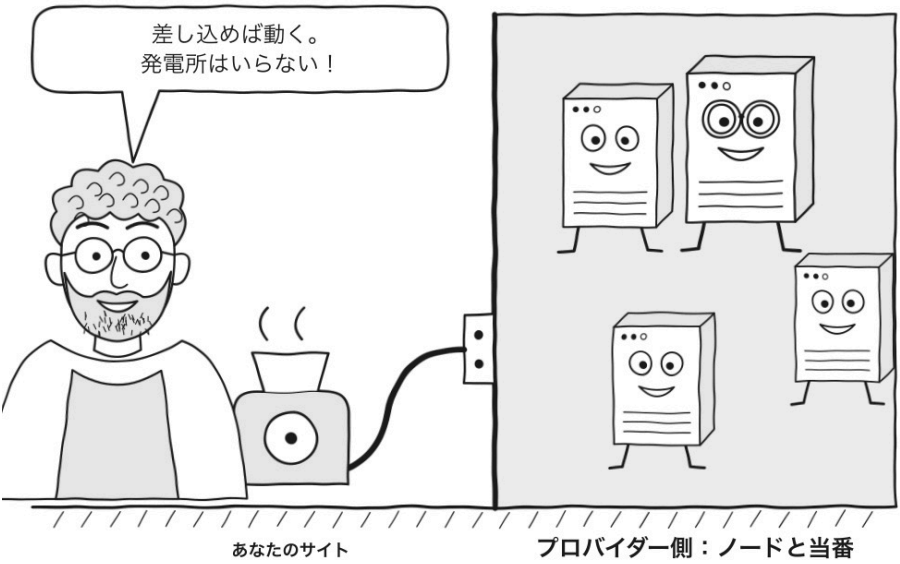
分かれ道のアルチョム

第2章の計算のあと、電卓は画面を下にして伏せられています。5つのネットワークを予備込みで持つと、レンタル代だけで年6万ドル超。コーヒーはそこまで稼いでくれません。



解説：プロバイダーが引き受けること

電気はコンセントから取るもので、やかん一つのために発電所を建てたりしません。ノードも同じです。プロバイダーはノードを自分の側で動かし、あなたには質問を送るためのアドレスを渡します。



ふつう、プロバイダーの側が受け持つ仕事は5つです。

1. 機材とディスク。買うか借りるかして、容量が足りなくなれば増やす。
2. 同期。ノードがネットワークから遅れないようにする。
3. 更新。ネットワークがルールを変えたら、期限内にプログラムを入れ替える。
4. 故障の交代。リクエストは元気なノードに回す。
5. 当番。夜中に起きるのはアルチョムではない。

プロバイダーはいろいろで、約束する内容もさまざまです。このリストのうち実際に何が受けられるのかは、支払う前に確かめます。選び方は第13章で扱います。

あなたの側に残ること

- ウォレットの鍵。持っているのはあなただけです。
- エンドポイントのアドレス。プロバイダーが発行したアドレスをこう呼びます。多くの場合、あなた専用のアクセスキーが埋め込まれています。パスワードのように大切にしてください。他人のリクエストがあなたの利用枠を使ってしまいます。
- あなたのコード。エラーに耐え、リクエストをやり直し、一つの支払いを二重に数えないようにしなければなりません。

どのプランにも出てくる3つの言葉

言葉	意味	自分への質問
1秒あたりのリクエスト数（RPS）	どれくらいの頻度で聞けるか	同時に何人のお客さんが来るか？
利用枠、またはクレジット	一定の期間に全部でどれだけ聞けるか。質問によって値段が違う	月末まで足りるか？
アーカイブアクセス	遠い過去のことを聞けるか	古い履歴は必要か？

やってみよう：本物のノードに質問する

第1章では、アドレスの場所に `ノードのアドレス` と書いてありました。今度は本物を入れてみましょう。RpcNodeには無料の公開ノードがあり、登録もキーもいりません。Ethereumの最新ブロックの番号を聞いてみます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

この章を書いていたとき、ノードはこう答えました。

```
{"jsonrpc":"2.0","id":1,"result":"0x1946393"}
```

`0x1946393` は26,502,035のことです。あなたが試せば、もっと大きな数になるはずです。ネットワークは立ち止まりません。1分後にもう一度リクエストして、自分の目で確かめてください。

公開ノードはみんなで共用するので、制限があります。2026年11月27日時点では、1つのアドレスにつき1秒5リクエストまでで、いちばん「高価な」メソッドは使えません。お試しには十分ですが、お店には専用のエンドポイントが必要です。

費用はいくら

例として、2026年11月27日時点のRpcNodeの最初の2つのプランを挙げます。最新の内容はサイトで確認してください。

	Free	Starter
月額	$0	$19
クレジット	1500万	5000万
1秒あたりのリクエスト数	15	40
エンドポイント	3	5
ネットワーク	EthereumとSolana	すべて
アーカイブアクセス	なし	あり

RpcNodeのノードは3つの地域にあります。フランクフルト、アメリカ、日本です。

> アルチョム：「月5,120ドルに対して19ドル？　どこに落とし穴があるんだ？」

これは別のものです。自前のサーバーは、まるごとあなたのものです。プランで手に入るのは、全体の能力のうち上限つきの取り分です。小さなお店ならその取り分で足り、「発電所」をまるごと買う必要はありません。

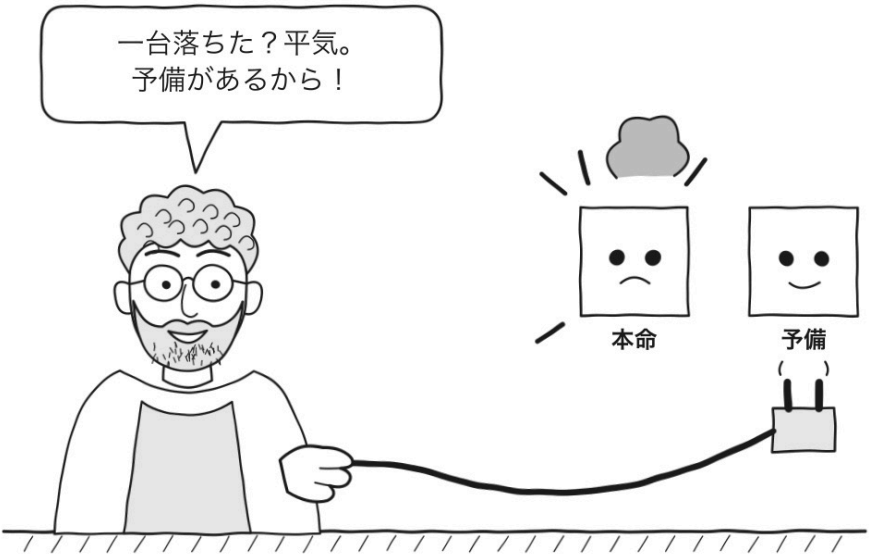
それでも自前のノードが必要なとき

- プライバシー。プロバイダーにはあなたのリクエストが見えます。どのアドレスを調べ、どこから聞いているかです。
- 巨大な負荷。リクエストが多すぎて、プランのほうが自前のサーバーより高くなる。
- ネットワークへの参加。あなたがバリデーターである、またはネットワーク自体を開発している。
- 特別な設定。プロバイダーが提供していないメソッドや動作モードが必要。
- 法律や発注元の要件。データは自社内だけで処理しなければならない。

混ぜる道もあります。主力のネットワークは自前のノード、そのほかと万一の備えはプロバイダー、という形です。

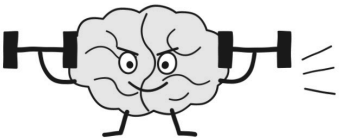
落とし穴：「プロバイダーは絶対に落ちない」

何でも落ちます。サーバーも、通信回線も、プロバイダーのサービスも。支払いが大事なら、答えが返らないときにサイトがどうするかを、前もって決めておきましょう。



- ただ聞けなかっただけなのに、お客さんに「未払いです」と言わない。
- リクエストは後でやり直す。続けざまに100回も繰り返さない。
- 予備のアドレスを持つ。別のプロバイダーか、自前のノードです。

考えてみよう



頭の体操をしよう！

1. アルチョムのサイトは、支払い待ちの注文20件それぞれについて、毎秒1回、1リクエストで支払いを確認します。毎秒15リクエストの枠に収まるでしょうか。
2. ノードが10分間応答しないとき、お客さんの画面には何を表示すべきでしょうか。
3. 自前のノードが必要なのはどちらでしょう。コーヒー店と、ネットワーク自体を開発するチーム。

1問目のヒント：20は15より大きい。確認の回数を減らすか、もっと賢く聞きましょう。方法は第26章で。

まとめ

- プロバイダーは、機材、同期、更新、当番を引き受けます。
- 鍵、エンドポイントのアドレス、障害時のサイトの振る舞いは、あなたの側に残ります。
- 自前のノードが必要なのは、プライバシー、大きな負荷、ネットワークへの参加、特別な要件のためです。

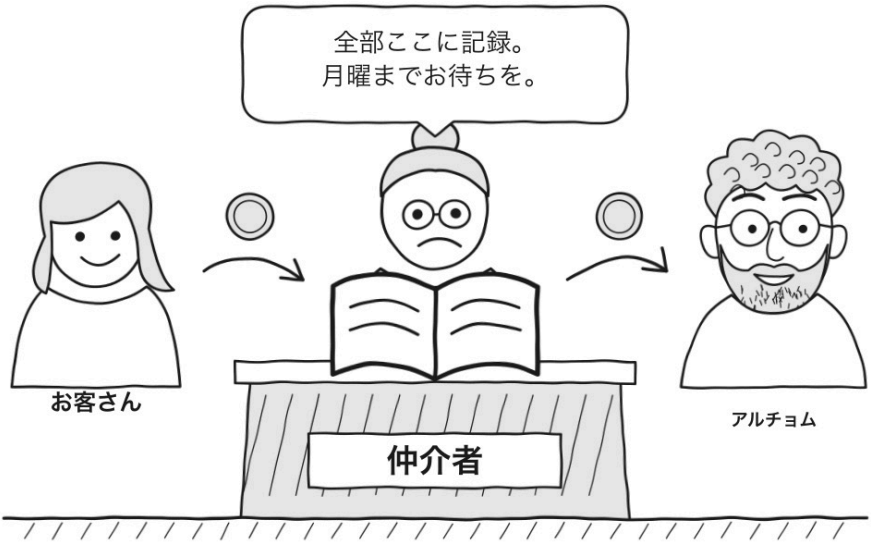
第1部はここまでです。アルチョムはプロバイダーを選び、本物のノードから答えを受け取りました。でも、ノードが告げた26,502,035番の「ブロック」とは何でしょう？　次へ：第2部、第4章。仲介者のいないお金と記録。

第4章　仲介者のいないお金と記録

あなたのお金はどこにある？

アルチョムはカードに送金を受け取りました。紙幣は一枚もどこへも運ばれていません。では何が起きたのでしょう。誰かがお客さんの名前の横の数字を減らし、アルチョムの名前の横の数字を増やしたのです。

現金でないお金とは、記録です。そして記録をつける人を仲介者と呼びます。銀行や決済サービスです。



解説：仲介者はなぜ必要で、何が不便なのか

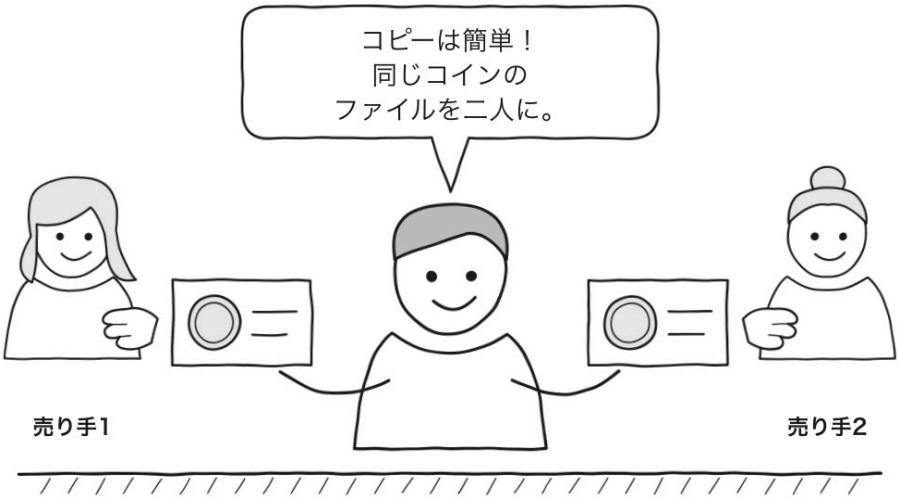
仲介者は役に立つことをたくさんしています。お金が足りるかを確かめます。同じお金を二度使えないようにします。もめごとを調べて、支払いを戻してくれることもあります。

けれども、記録はすべて仲介者の手の中にあります。送金を止めることも、手数料を取ることも、口座を凍結することもできますし、休日は閉まっていたりします。お客さんが外国に住んでいると、とくに目につきます。

> アルチョム：「じゃあ、お客さんがお金をファイルで送ればいい。写真みたいに。仲介者なんて抜きで！」

なぜファイルを送るだけではだめなのか

写真は100回送っても、手元に残ります。お金でそれはできません。一枚のコインで二人に払えてしまうからです。これを二重支払いと呼びます。

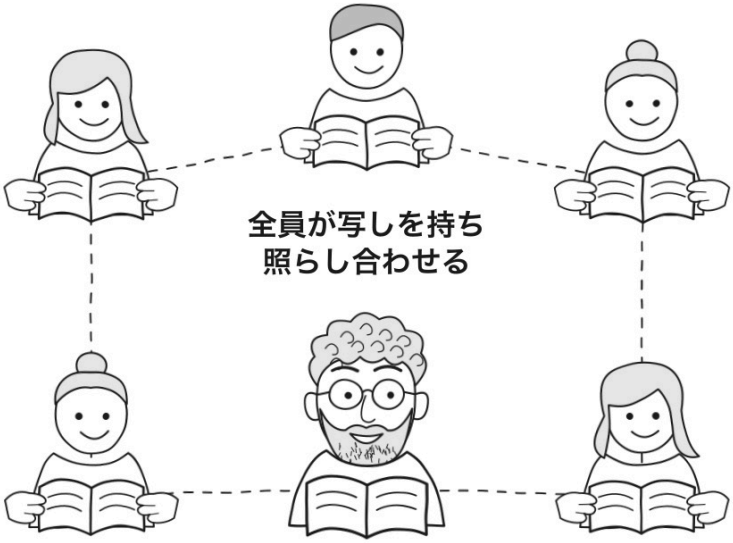


長いあいだ、解決策は一つしかありませんでした。誰か代表者が台帳をつけて、どのコインが今は誰のものかを決める。つまり、また仲介者です。

発想：みんなが記録を持つ

2008年、サトシ・ナカモトという仮名の著者が別の道を示し、2009年1月にBitcoinのネットワークが動き始めました。考え方はこうです。

1. 送金の一覧を一人にしまい込ませない。多くの人が写しを持つ。
2. 新しい送金はすべて、参加者が共通のルールで確かめる。お金はあるか、すでに使われていないか。
3. 参加者は代表者なしで、どの記録が正しいかを合意する。



こうして共同で管理する共通の一覧を、ブロックチェーンと呼びます。記録がどうやって鎖になるのかは第5章で、参加者がどうやって合意するのかは第8章で見ます。

やってみよう：共有ノートごっこ

コードはいりません。友だちを二人呼んで、それぞれに紙を一枚持ってもらいます。最初は全員の紙にこう書いてあります。アーニャはコイン10枚、ボリスは5枚、ヴェラは0枚。ルールは一つだけ。持っている以上は渡せない。

番号	送りたい人	受け付ける？	アーニャ	ボリス	ヴェラ
開始			10	5	0
1	アーニャ → ヴェラに4				
2	ヴェラ → ボリスに1				
3	ボリス → アーニャに7				
4	アーニャ → ボリスに6				

空欄を埋めて、紙を見比べてください。全員が同じなら、代表者なしで合意できたことになります。答えは章の終わりにあります。

ブロックチェーンがくれるもの、くれないもの

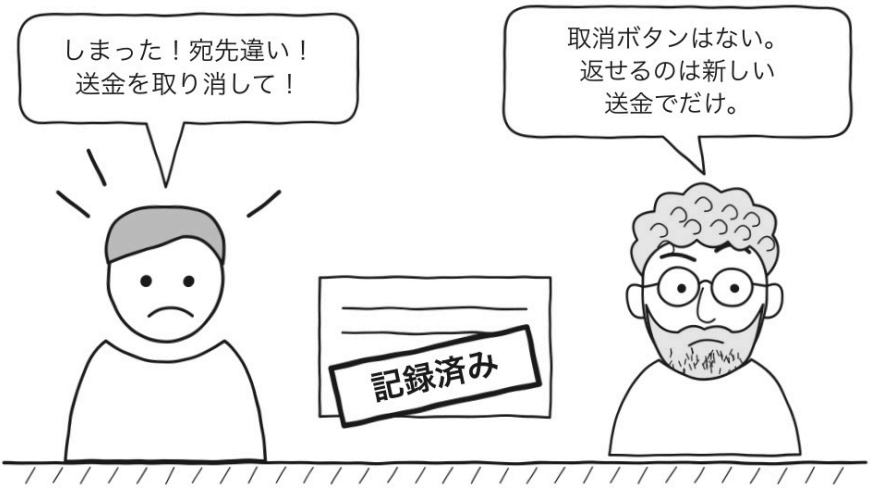
くれるもの	くれないもの
いつでも、何時でも、直接の送金	「取り消し」ボタン
一つの口座を、いろいろな国から使える	安定したコインの値段。相場は大きく変わることがある
誰でも確かめられる記録	秘密。どのアドレスの送金も全員に見える
一覧の保管を誰かに任せて信じる必要がない	なくした鍵を復旧してくれるサポート窓口

どれも値動きが激しいわけではありません。1ドル付近を保とうとするコインもあり、ステーブルコインと呼ばれます。ただし、広く使われているものの背後には発行会社がいて、アドレスにあるお金を凍結できます。詳しくは第9章で。

仲介者が消えてなくなったわけではありません。取引所、ウォレット、プロバイダーがいます。ただし役割は違います。取引所はあなたのコインを預かることがあります。第1部のプロバイダーはお金を預かりません。あなたの質問をノードに伝えるだけです。

落とし穴：「気に入らなければ取り消せばいい」

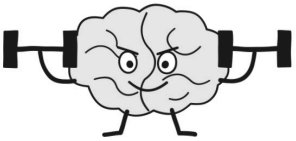
承認された送金は取り消せません。アドレスを間違えれば、お金は行ってしまいます。戻せるのは受け取った人だけです。その人にその気があり、そもそも持ち主のいるアドレスならの話です。



だからアドレスは二度確かめ、大きな金額は、まず少額を送って試します。

ちょっとした話。2010年5月、ピザ2枚の代金に10,000ビットコインが支払われました。その送金は今も共有ノートに記録されています。

考えてみよう



頭の体操をしよう！

1. ゲームの終わりに、ヴェラはコインを3枚持っています。彼女はアーニャに「3枚あげる」、同時にボリスにも「3枚あげる」と書きました。参加者はどうすべきでしょうか。
2. ある参加者は1つ目の送金を、別の参加者は2つ目を受け付けたら？　どちらが正しいのでしょう。
3. 支払いを取り消せるのと取り消せないのと、アルチョムに都合がよいのはどちらでしょう。お客さんにとっては？

ヒント：1）受け付けられるのは片方だけです。2）これがいちばん難しい問いで、第8章はそのための章です。

まとめ

- 現金でないお金は記録であり、誰かがそれをつける必要があります。
- ブロックチェーンは、多くの人が共通のルールで管理する共通の一覧です。
- 仲介者からの自由は、責任も意味します。送金を取り消す人も、鍵を復旧する人もいません。

ゲームの答え：送金1は成立（6、5、4）。2は成立（6、6、3）。3は却下で、ボリスは6枚しかありません（6、6、3）。4は成立（0、12、3）。コインの合計は15枚のままです。

次へ：第5章。ノードが告げた「ブロック」とは何か、そして古い記録をこっそり書き換えられないのはなぜか。

第5章　ブロックとチェーン：古い記録をこっそり書き換えられない理由

あの番号

第3章で、ノードはアルチョムに答えました。最新のブロックは26,502,035番。ブロックとは何でしょう。なぜ番号がついているのでしょう。そして、誰かがノートの記録をこっそり直すのを、何が防いでいるのでしょう。

解説：ブロックはノートの1ページ

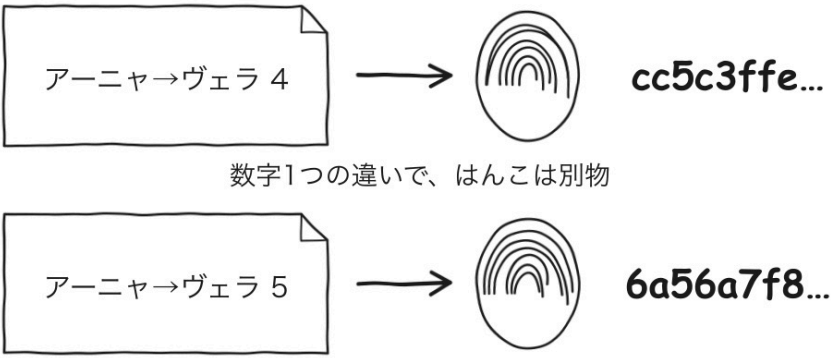
送金は一件ずつ書き込むのではありません。行をページに並べるように、まとめて束にします。このページがブロックです。そこには次のものが書かれています。

1. 送金などの操作。これらをトランザクションと呼びます。
2. 順番を示すページの番号。
3. 作られた時刻。
4. 前のページのはんこ。秘密はすべてここにあります。

はんこ、つまりハッシュ

どんな文章からも、数字と文字でできた短い文字列を計算できます。これがハッシュです。この本では「はんこ」と呼びます。指紋に似ています。

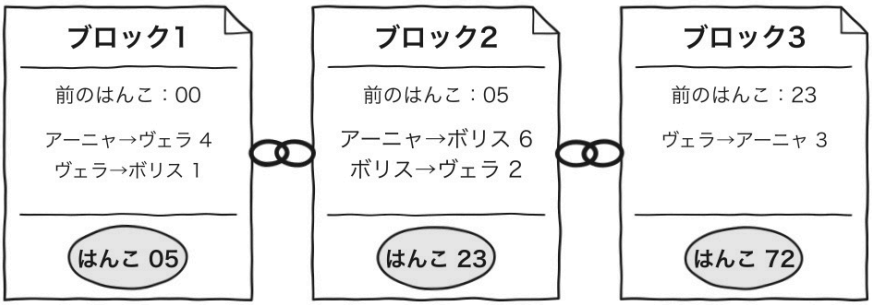
- 同じ文章からは、いつも同じはんこができる。
- 一文字でも変えると、はんこはまったく別のものになる。
- はんこから元の文章は復元できない。
- 同じはんこになる別の文章を探し出すのは、実際には不可能。



これは、絵の中の2行に対する本物のSHA-256ハッシュの先頭部分です。SHA-256はBitcoinで使われています。ほかのネットワークは、別の方法ではんこを計算することもあります。

チェーン

各ページには、前のページのはんこが書かれています。そしてページ自身のはんこは、そこに書かれたすべて、この前のはんこも含めた全部から作られます。

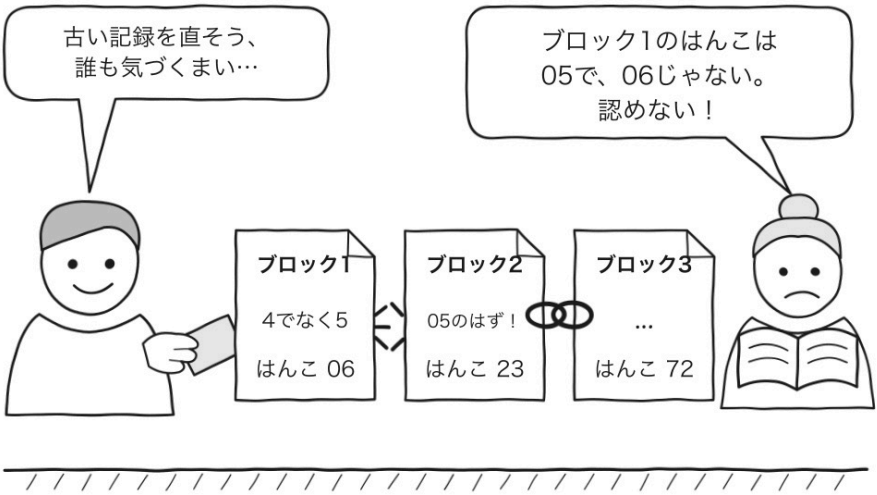


こうしてページはつながり、ブロックの鎖になります。英語でblockchain。名前の由来はこれだけです。

絵の中のはんこは、2桁の数字を使ったおもちゃです。暗算できるようにしてあります。

古い記録をいじるとどうなるか

詐欺師が最初のブロックの4を5に変えます。そのブロックのはんこは変わります。ところが2番目のブロックには、古いはんこが書かれています。鎖が切れて、すぐにばれます。



詐欺師は、その後のはんこをすべて書き直すこともできます。しかし、ほかの参加者はそれぞれの写しを持っていて、彼の版は一致しません。自分の版を認めさせるには、残りのネットワーク全体を力で上回る必要があります。それには莫大な費用がかかります。ネットワークがどの鎖を正しいと決めるのかは、第8章で見ます。

本物のチェーンを見てみよう

これは2026年11月26日のEthereumの3つのブロックです。はんこは長いので、先頭と末尾だけを示します。

ブロック番号	そのはんこ	前のブロックのはんこ	トランザクション数
26,502,033	`0xbe58…d764`	`0x2844…c876`	229
26,502,034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26,502,035	`0xeab2…c904`	`0x7bb7…b7a4`	515

指でたどってみてください。ある行のはんこが、次の行にそのまま現れます。これらのブロックの間隔は、ちょうど12秒ずつでした。Bitcoinでは新しいブロックの出る間隔はもっと長く、平均でおよそ10分に1回です。

いちばん最初のブロックには、前のブロックがありません。Bitcoinの最初のブロックは、2009年1月3日に作られました。

やってみよう：自分のはんこ

おもちゃのルールを考えます。

はんこ =（前のはんこ × 3 + ページ上の全金額）の結果の下2桁。

ブロック	送金額	はんこ	ブロック1の4を5に変えると
1	4と1		
2	6と2		
3	3		

最初のブロックでは、前のはんこは0です。両方の列を埋めてください。いくつのはんこが変わりましたか。

落とし穴：「ブロックに入ったのを見たら、もう永久に安心」

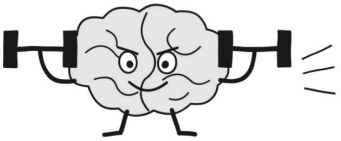
いちばん新しいブロックは、別のブロックに置き換わることがあります。ネットワークの各部分が、まだ合意しきれていないからです。上に積まれたブロックが多いほど、あなたの記録を取り除くのは難しくなります。

これを承認数で数えます。支払いの入ったブロックそのものが1つ目で、次のブロックが積まれるたびに1つ増えます。



どれだけ待つかは、ネットワークと金額によります。コーヒー1杯なら短く、大口の注文なら長く。詳しくは第8章で。

考えてみよう



頭の体操をしよう！

1. このはんこをだましてみてください。ブロック1の4と1を別の数に置き換えて、はんこが変わらないようにします。このルールについて何がわかりますか。
2. 3年前のブロックを変えるのが、昨日のブロックより難しいのはなぜでしょう。
3. 12秒ごとにブロックが出るとすると、1時間で何ブロックになりますか。

答え：1) たとえば3と2。合計が同じです。本物のハッシュなら気づきます。2) 上に積まれたすべてを作り直さなければならないからです。3) 300。

まとめ

- ブロックは、トランザクション、番号、時刻、前のページのはんこが書かれた1ページです。
- 古い記録を変えるとはんこが変わり、鎖が切れます。
- できたばかりのブロックはあまり信用せず、承認を待ちます。

ゲームの答え：はんこは05、23、72。置き換えた後は06、26、81で、3つとも変わりました。

次へ：第6章。送金したのがお金の持ち主本人だと、ネットワークはどうやって知るのでしょう。鍵、アドレス、署名の話です。

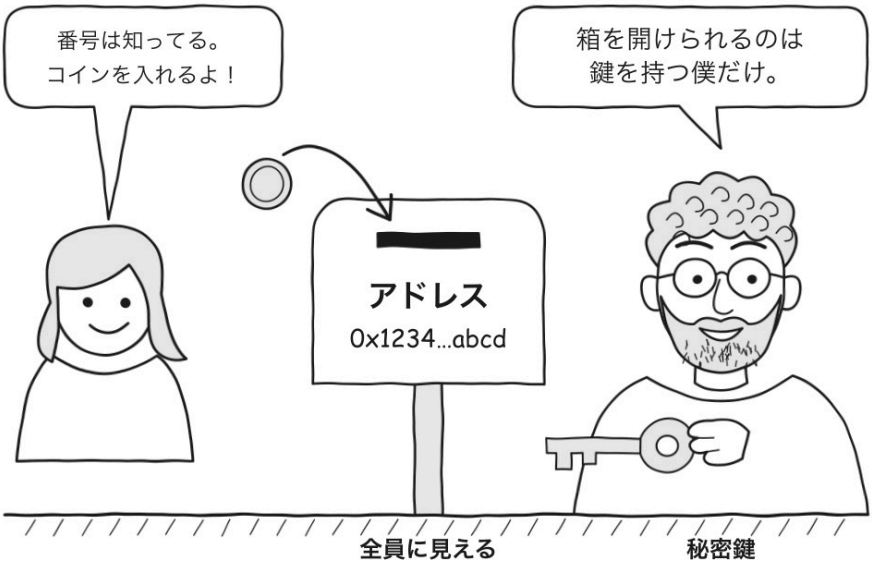
第6章　鍵、アドレス、署名

それがあなたのお金だと、ネットワークはどうやって知るのか

共有ノートに身分証はありません。名前を聞く人も、顔写真を照らし合わせる人もいません。では、誰かが「アルチョムは全財産を私に渡す」と書くのを、何が防いでいるのでしょう。

解説：郵便受けと鍵

道に面した郵便受けを思い浮かべてください。受け口の番号は誰でも知っていて、誰でもコインを投げ込めます。けれど、扉を開けてコインを取り出せるのは、鍵を持っている人だけです。



もの	たとえると	誰に見せるか
アドレス	郵便受けの番号	お金を受け取るためなら、誰にでも
公開鍵	署名を確かめるための見本	全員に見せてよい
秘密鍵	郵便受けの鍵	誰にも見せない

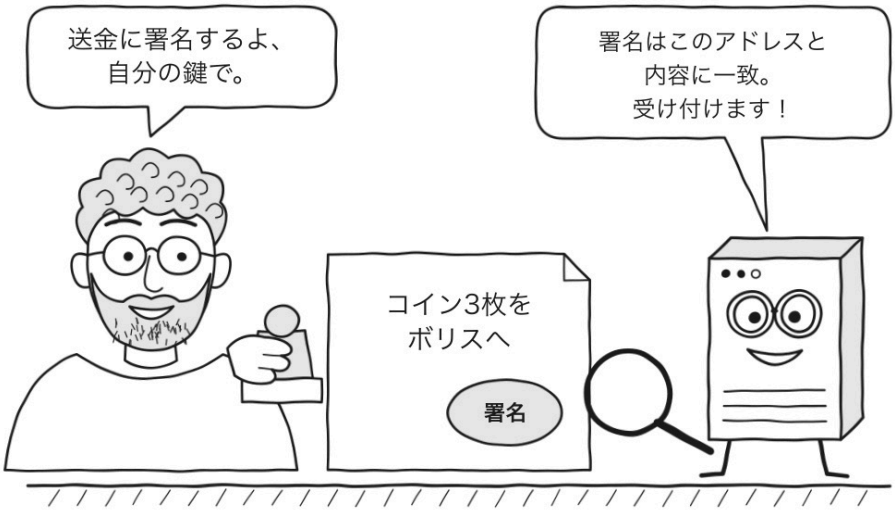
この3つは一方通行でつながっています。秘密鍵から公開鍵を作り、公開鍵からアドレスを作ります。逆向きの計算は、実際には不可能です。ネットワークによっては、アドレスは公開鍵

そのもので、特別な形式で書いてあるだけです。

ウォレットは、鍵を保管するプログラムや機器です。中にコインは入っていません。コインは共有ノートの記録のままです。

署名

コインを渡すとき、持ち主は送金内容を作り、秘密鍵で署名します。このとき鍵はどこにも送りません。ネットワークに出ていくのは、送金内容と署名です。



ノードは2つのことを確かめます。署名がそのアドレスの鍵で作られたものか。そして署名のあとで内容が変えられていないか。ある送金の署名は、別の送金には使えません。同じ送金をネットワークが二度実行することもありません。

やってみよう：おもちゃの署名

すべて電卓で計算できます。秘密鍵は数字の7、公開鍵は3です。秘密なのは最初のほうだけです。

> ルール：選んだ数を掛けていき、1回ごとに33で割った余りを残す。

メッセージ「4」に署名します。秘密鍵が指示する回数、つまり7個の4を掛け合わせます。

4の個数	1	2	3	4	5	6	7
余り	4	16	31	25	1	4	16

各ステップは、前の余りに4を掛けたものです。たとえば3番目は16×4＝64で、33で割った余りは31です。7番目まで進むと、署名16が得られます。

確かめます。署名の16を、公開鍵が指示する回数、つまり3個掛け合わせます。16、次に16×16＝256で余り25、次に25×16＝400で余り4。元のメッセージになりました。署名は本物です！

ここで誰かがメッセージを「5」に直し、署名は16のままにしたとします。確認の結果はやはり4で、5になりません。偽造はばれます。

これは考え方を示しただけです。こんな小さな数なら、小学生でも破れます。本物のネットワークの鍵は巨大な数で、使う数学も違います。

シークレットフレーズ

秘密鍵は長い文字列なので、書き写すのが大変です。そのため多くのウォレットは、12語か24語のシークレットフレーズを出してくれます。このフレーズから、鍵とすべてのアドレスを復元できます。



- フレーズを知る人が、お金の持ち主です。
- フレーズも端末もなくしたら、復旧してくれる人はいません。
- 写真に撮らない。メモアプリやメッセージに保存しない。
- サイトに入力しない。「サポート」に読み上げない。本物のサポートはフレーズを聞きません。

パスワードは変えられますが、秘密鍵は変えられません。漏れてしまったら、急いで新しいアドレスにお金を移すしかありません。

お店にとっての意味

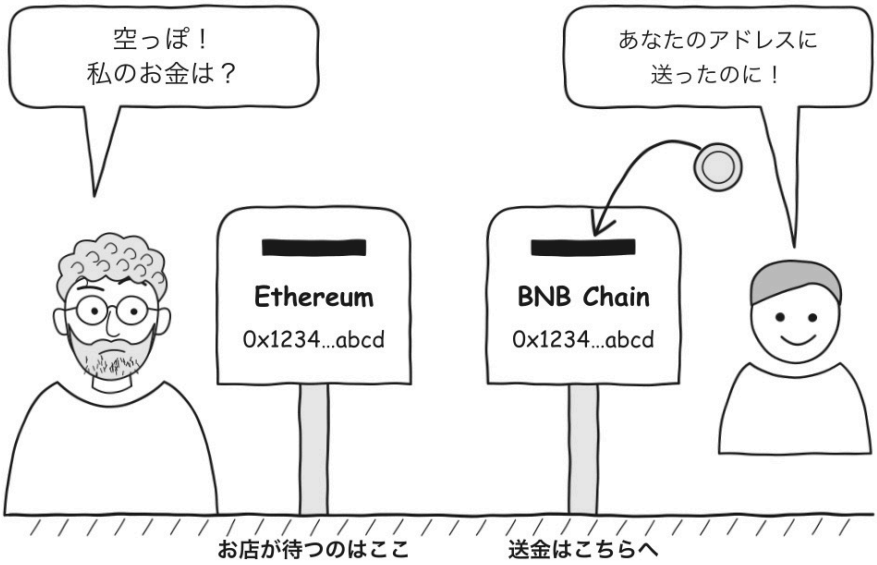
お金を受け取り、支払いを確認するだけなら、アルチョムのサイトに必要なのはアドレスだけです。秘密鍵はお店のサーバーにいりませんし、置くべきでもありません。サイトが破られれば、お金まで持っていかれます。注文ごとに別のアドレスを出すと便利です。これは第28章で。

落とし穴：「アドレスが同じなら届くはず」

アドレスの見た目は、ネットワークによって違います。

ネットワーク	アドレスの先頭
Bitcoin	`1`、`3`、または `bc1`
Ethereumとその仲間のネットワーク	`0x`
TRON	`T`
TON	ふつう `EQ` か `UQ`
Solana	決まった先頭文字はない

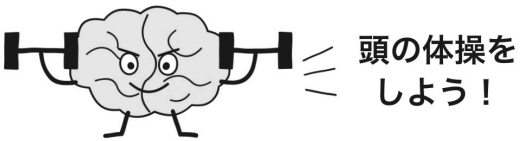
ところが、Ethereum、BNB Smart Chain、Polygonをはじめ多くのネットワークでは、アドレスの見た目が同じです。しかしノートは別々です。お客さんが違うネットワークで送ると、お店には支払いが見えません。



だから請求には、アドレスと金額だけでなく、ネットワークも書きます。

詐欺師の手口をもう一つ。あなたがよく知るアドレスに似たアドレスから、ごく少額を送ってくるのです。あなたが履歴からそのアドレスをコピーすることを狙っています。先頭と末尾の数文字だけでなく、アドレス全体を照らし合わせてください。

考えてみよう



1. メッセージ「2」に鍵7で署名し、鍵3で署名を確かめてください。
2. アルチョムはお店のアドレスを名刺に印刷しました。危険でしょうか。
3. 「ウォレットのサポート」が、止まった送金を戻すためにシークレットフレーズを教えてほしいと言ってきました。どうしますか。

答え：1) 署名は29。確認するとまた2になります。2) それでお金を盗まれることはありませんが、そのアドレスへの入金は誰にでも見えます。3) 何も教えない。詐欺です。

まとめ

- アドレスは誰に見せてもよく、秘密鍵とシークレットフレーズは誰にも見せません。
- 署名は、鍵の持ち主が送金を許可したこと、内容が変えられていないことを証明します。
- ネットワークは、アドレスと同じくらい大事です。

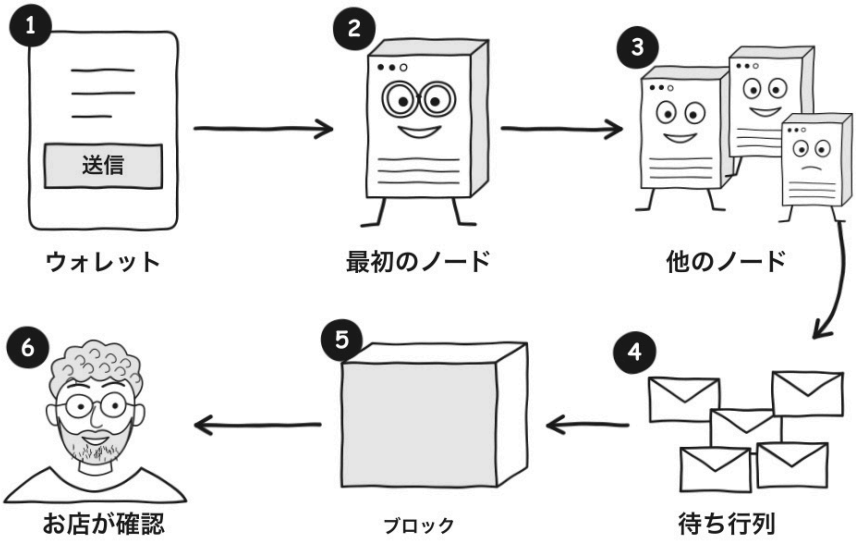
次へ：第7章。送金に署名しました。「送信」ボタンを押してからブロックに記録されるまで、何が起きるのでしょう。

第7章　トランザクションの旅：「送信」ボタンからブロックまで

「送りました！」「こちらには何も」

お客さんはボタンを押して「送りました」と書いてきます。アルチョムが見ても、お金はありません。誰かがだましているのでしょうか。おそらく誰もだましていません。送金がまだ旅の途中なのです。

解説：6つの停留所



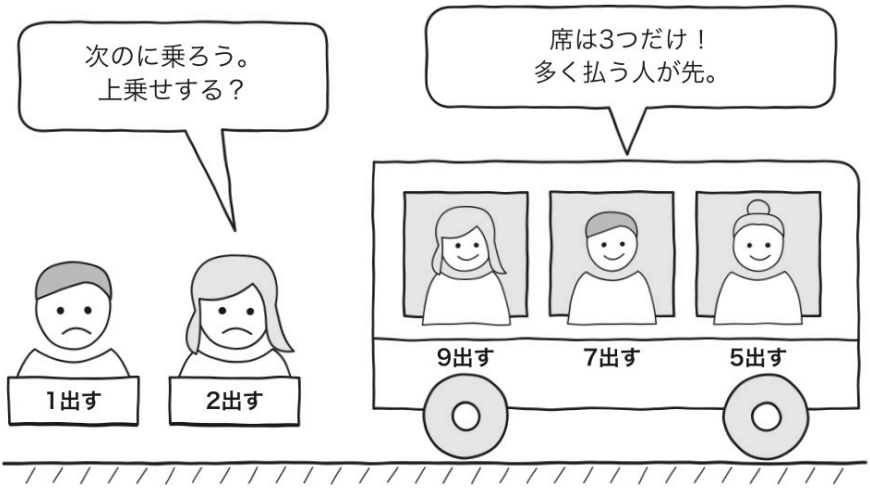
1. ウォレットが送金内容を作り、鍵で署名します。
2. 最初のノードが受け取り、署名は正しいか、お金は足りるかを確かめます。
3. そのノードはほかのノードへ渡し、そこからさらに先へ広がります。
4. 送金は待ち行列で待ちます。英語ではmempoolといいます。
5. ページをまとめる番の参加者が、送金をブロックに入れます。
6. お店がそれを見つけ、承認を待ちます。

呼び名や細部はネットワークによって違います。たとえば共通の待ち行列がなく、送金がブロックをまとめる参加者へ直接向かうものもあります。

送金には一つひとつ「はんこ」があります。トランザクションハッシュです。荷物の追跡番号のようなもので、これを使えば送金が今どこにあるかがわかります。

手数料：バスの座席

ブロックには全部は入りません。座席が3つのバスの停留所に、5人が待っているところを想像してください。



ブロックに入るために、送る人はネットワークに手数料を払います。乗りたい人が多いほど、座席は高くなります。少ししか出さない人は次のバスを待ち、ときには結局乗れないこともあります。

- 手数料を払うのは送る側で、お店の収入にはなりません。
- 手数料は送金額ではなく、送金の「重さ」とネットワークの混み具合で決まります。
- この「重さ」の測り方はネットワークごとに違います。本書の第4部で扱います。

本物の送金を見てみよう

これはEthereumのブロック26,502,085にある、ふつうの送金です。トランザクションハッシュを指定して、ノードに聞きました。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getTransactionReceipt",
       "params":["送金のハッシュ"]}'
```

答えは長いので、大事なところをふつうの数に直して示します。

フィールド	値	意味
`status`	`0x1`	正常に実行された
`blockNumber`	26,502,085	どのブロックに入っているか
`gasUsed`	21,000	単純な送金の「重さ」
手数料	0.000043 ETH	「重さ」×1単位の値段
送金額	0.000041 ETH	送金そのものの情報から読み取る

気づきましたか。このときは、送った額より多い手数料を払っています。手数料は、中にいくら入っているかを気にしません。

この値段はたえず変わります。自分で聞いてみてください。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_gasPrice","params":[]}'
```

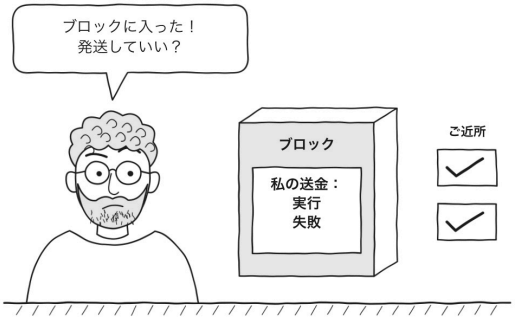
ノードは「重さ」1単位の現在の値段をweiで答えます。ちなみに同じ日、RpcNodeのBitcoin公開ノードでは、約83,000件の送金が待ち行列で待っていました。

送金に起こりうること

状態	意味	お店がすること
待機中	まだ待ち行列にいる	待つ。商品は送らない
ブロック内	記録されたが、承認がまだ少ない	成功したかを確かめて、待つ
承認済み	上に十分な数のブロックが積まれた	注文を発送する
実行失敗	ブロックにはあるが、送金は成立しなかった	支払いと数えない
行列から脱落、または置き換え	ブロックに入らなかった	お客さんにウォレットを確認してもらう

落とし穴：「ブロックに入ったら支払い完了」

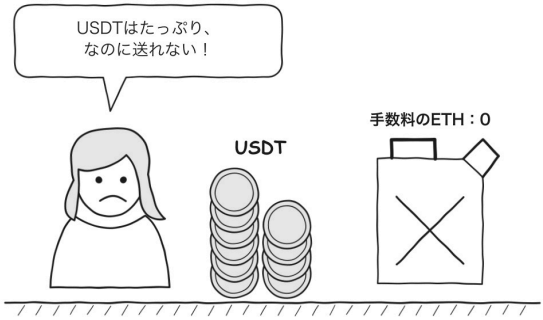
Ethereumとその仲間のネットワークでは、送金がブロックに入っても、実行に失敗することがあります。たとえばお客さんのトークンが足りなかった場合です。手数料は引かれるのに、お金はあなたに届きません。



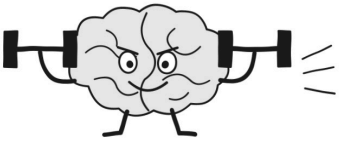
だから、ブロックだけでなく結果も見ます。上の例では`status`フィールドです。さらに受取人、金額、コイン、ネットワークも照らし合わせます。

もう一つの落とし穴：空のタンク

手数料は、ネットワーク固有のコインで取られます。Ethereumでは、USDTを送る場合でもETHです。TRONでは、TRXとネットワーク特有のリソースで支払います。



考えてみよう



頭の体操をしよう！

1. ブロックには送金が3件入ります。待ち行列では、座席に4、9、1、7、2の値がつきました。最初のブロックに乗るのは誰でしょう。
2. お客さんがトランザクションハッシュと「成功」と出た画面の写真を送ってきました。コーヒーを発送するのに十分でしょうか。
3. お客さんはEthereumに100 USDTを持っていますが、ETHはまったくありません。何をすればよいでしょう。

答え：1) 9、7、4。2) いいえ。画面の写真は偽造できます。ハッシュは自分の側で確かめましょう。3) まず手数料用のETHを少し手に入れます。

まとめ

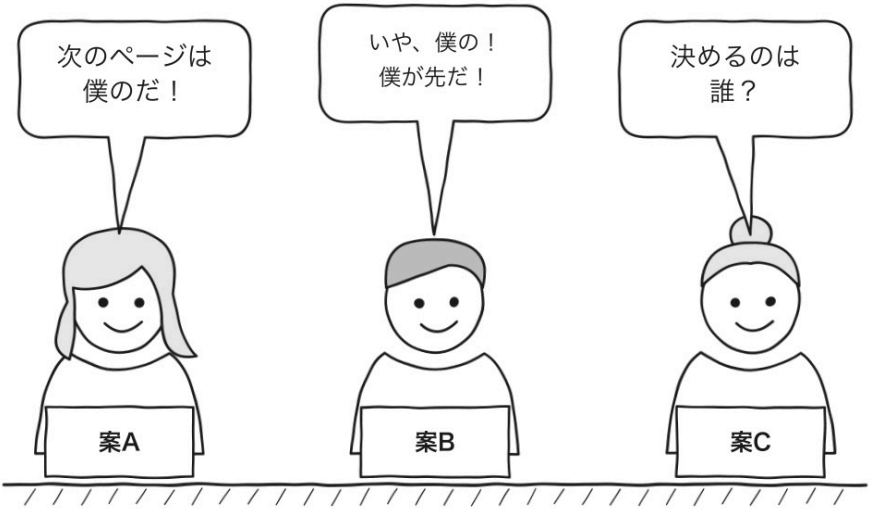
- 「送った」と「受け取った」の間には道のりがあります。ノード、待ち行列、ブロック、承認です。
- 手数料はブロック内の座席を買うもので、金額には左右されません。
- 支払いとは、正しいネットワークで、正しいコインが、十分な承認を得て、送金に成功したことです。

次へ：第8章。誰がブロックをまとめるのか、そして、どうやって全員がこの鎖こそ正しいと認めるのか。

第8章　ネットワークはどうやって合意するのか

次のページは誰が書く？

共有ノートに代表者はいません。何千もの参加者が写しを持ち、誰もが次のページを提案できます。どれを本物とすればよいのでしょう。



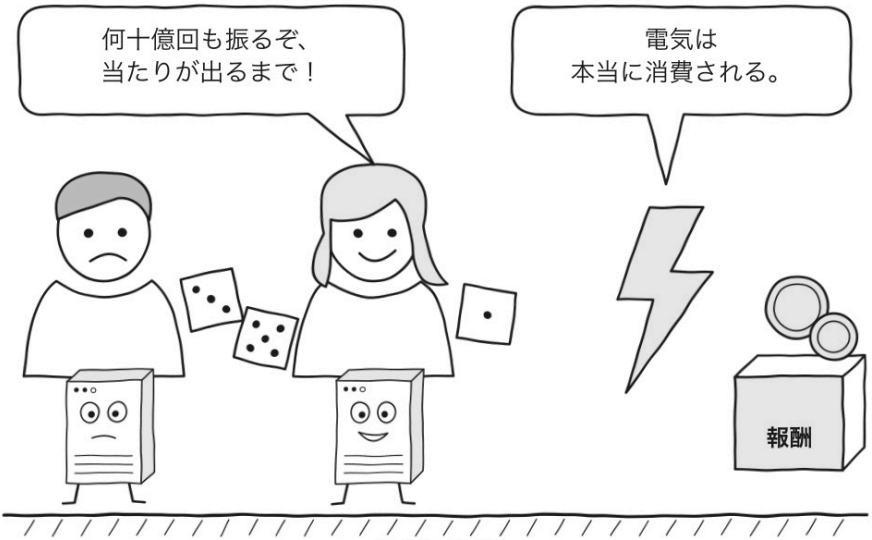
解説：コンセンサス

参加者が一つの答えにたどり着くためのルールを、コンセンサスと呼びます。目的は、正直に振る舞うより、だますほうが高くつくようにすることです。

単純に票を数えるわけにはいきません。詐欺師がプログラムを100万個立ち上げれば、100万票が手に入ります。だから、1票には何らかの元手が必要です。主な方法は2つです。

方法1：作業の証明

英語でProof of Work、その作業はマイニングと呼ばれます。参加者は競って問題を解きます。頭の良さでは解けず、総当たりするしかない問題です。珍しい目の組み合わせが出るまで、サイコロを振り続けるようなものです。



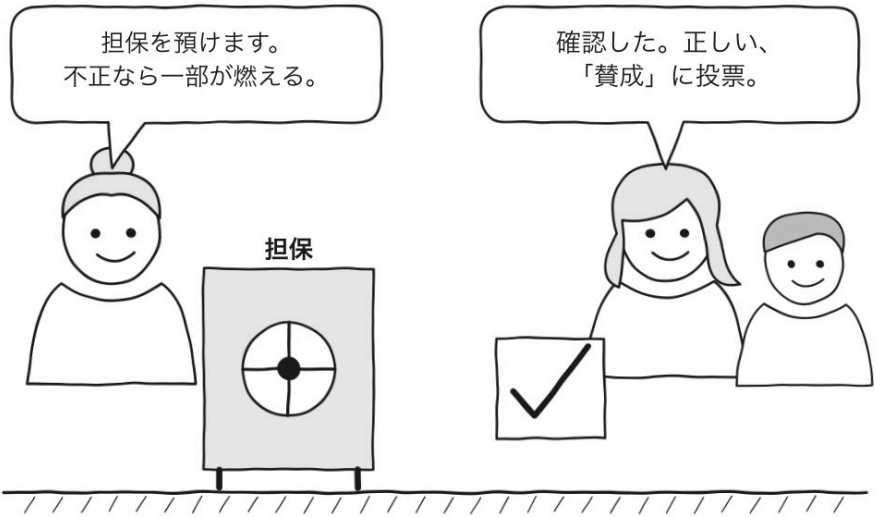
作業の証明

勝者はブロックを追加し、報酬を受け取ります。新しいコインと手数料です。答えを確かめるのは簡単ですが、見つけるには費用がかかります。Bitcoinはこの方式です。

- ブロックは平均で10分に1回現れます。
- 2016ブロックごとに、このペースが保たれるよう問題の難しさが調整されます。
- 1ブロックにつき新しく発行されるコインは今は3.125 BTCで、報酬はおよそ4年に一度、半分になります。

方法2：持ち分の証明

英語でProof of Stake、またはステーキングです。参加者は電気の代わりに、コインを担保として預けます。こうした参加者をバリデーターと呼びます。バリデーターは順番にブロックを提案し、ほかの者が確かめて投票します。



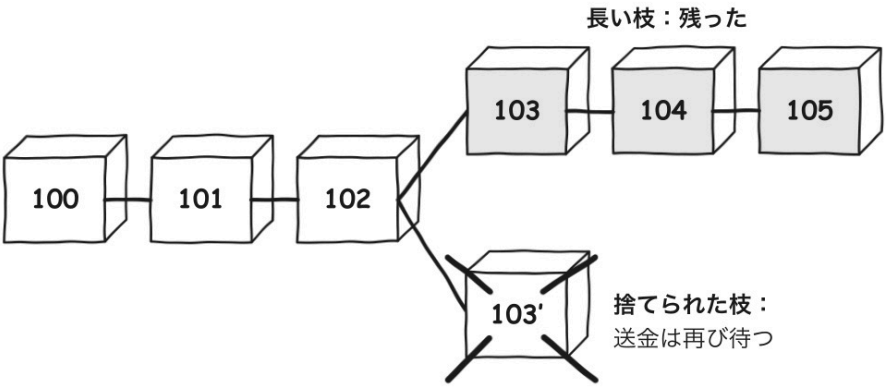
持ち分の証明

重大な違反をすると担保の一部が没収されて燃やされます。これをスラッシングといいます。Ethereumは2022年9月15日にこの方式へ移行し、ネットワークの電力消費は約99.95%減りました。自分でバリデーターになるには32 ETHが必要です。

ほかの形もあります。TRONでは、コインの保有者が選んだ27人の代表がブロックを書きます。BNB Smart Chainでも、限られた数のバリデーターが働いています。

ページが同時に2つできたら

ネットワークは大きく、知らせは一瞬では届きません。同じ番号のブロックが2つ現れ、鎖がしばらく二股に分かれることがあります。やがて片方の枝が勝ち、もう片方は捨てられます。これを再編成（リオーグ）と呼びます。



捨てられたブロックの送金は、跡形もなく消えるわけではありません。ふつうは待ち行列に戻り、別のブロックに入ります。ただし、入らないこともあります。第5章の承認数が必要なのはこのためです。

絵では長い枝が勝っています。正確には、Bitcoinではより多くの作業が注がれた枝が、ステーキングのネットワークではより多くの担保の票を集めた枝が選ばれます。

ファイナリティ：もう取り消されないとき

ファイナリティとは、それを過ぎるとブロックが置き換わらなくなる時点のことです。はっきり宣言するネットワークもあれば、ブロックが積まれるごとに確かさが増していくだけのネットワークもあります。

ネットワーク	記録が確定とみなされるおおよその目安
Bitcoin	はっきりした印はない。大きな金額には6承認、約1時間が勧められる
Ethereum	ふつう13〜15分。もっとかかることもある
TRON	ふつう約1分
BNB Smart Chain	約1秒、またはもっと速い
Solana	2026年の更新後は、1秒未満を想定した設計
TON	約1秒

2026年11月27日時点の各ネットワークのドキュメントによる情報です。これらの数字は更新のたびに変わります。

やってみよう：3つの「新しさ」を聞く

Ethereumのノードには、いちばん新しいブロック以外も聞けます。第1章の `latest` には仲間が2つあります。`safe`（もう信頼できる）と `finalized`（確定した）です。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getBlockByNumber",
       "params":["finalized",false]}'
```

`finalized` を `safe` や `latest` に替えて、番号を比べてください。この章を書いていたとき、ノードはこう答えました。

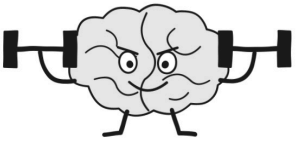
聞いたもの	ブロック番号	最新からの遅れ
`latest`	26,502,104	—
`safe`	26,502,070	34ブロック、約7分
`finalized`	26,502,038	66ブロック、約13分

落とし穴：「どのネットワークも同じ」

「承認を1つ待てば大丈夫」は危険なルールです。同じ1承認でも、ネットワークによって確かさはまったく違います。ネットワークと金額を見て決めましょう。

- 注文が高額なほど、長く待つ。
- 大きな金額では、ファイナリティがあるネットワークならそれを待つ。
- 取り戻せないものを、できたばかりのブロックの送金を根拠に渡さない。

考えてみよう



頭の体操をしよう！

1. 共有ノートで「プログラム1つにつき1票」の投票ができないのはなぜでしょう。
2. Bitcoinのブロックは10分に1回出ます。6承認を待つには平均どれくらいかかりますか。3承認なら？
3. 持ち帰りのコーヒーを1杯買うお客さんと、豆を200 kg買うお客さん。アルチョムはどちらでファイナリティを待つべきでしょう。

答え：1）プログラムは元手がいらず、いくつでも動かせるからです。2）約1時間と約30分。3）200 kg買うお客さんです。

まとめ

- コンセンサスとは、参加者が代表者なしで一つの鎖を選ぶためのルールです。
- 1票には、作業（マイニング）か担保（ステーキング）という元手がかかります。
- 新しいブロックは置き換わることがあります。確定はあとから、ネットワークごとのやり方で訪れます。

次へ：第9章。コイン、トークン、スマートコントラクト。USDTがETHと同じではないのはなぜか。

第9章　コイン、トークン、スマートコントラクト

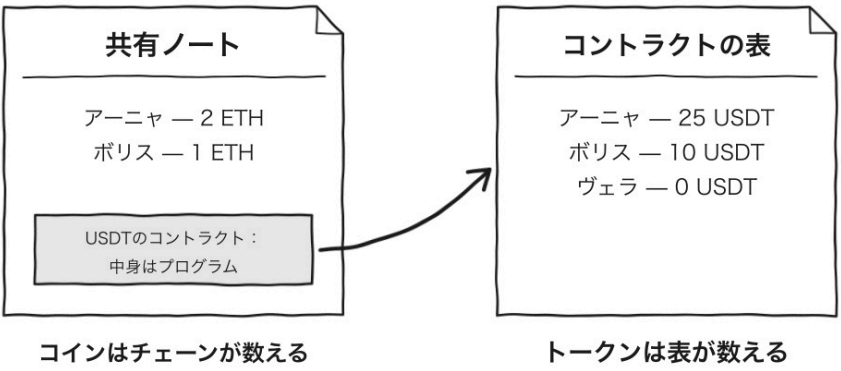
「払ったのに、そちらは空っぽ？」

お客さんがEthereumで25 USDTを送ってきました。アルチョムは第1章のようにノードに聞きますが、残高は変わっていません。お金は消えたのでしょうか。いいえ。聞き方が違っていただけです。

解説：コインとトークン

ネイティブコインは、ネットワーク自身が管理するコインです。EthereumのETH、TRONのTRX、BNB Smart ChainのBNBがそうです。手数料はこれで払います。

トークンは、ネットワークの中に住むプログラムがつけている記録です。USDTはトークンです。

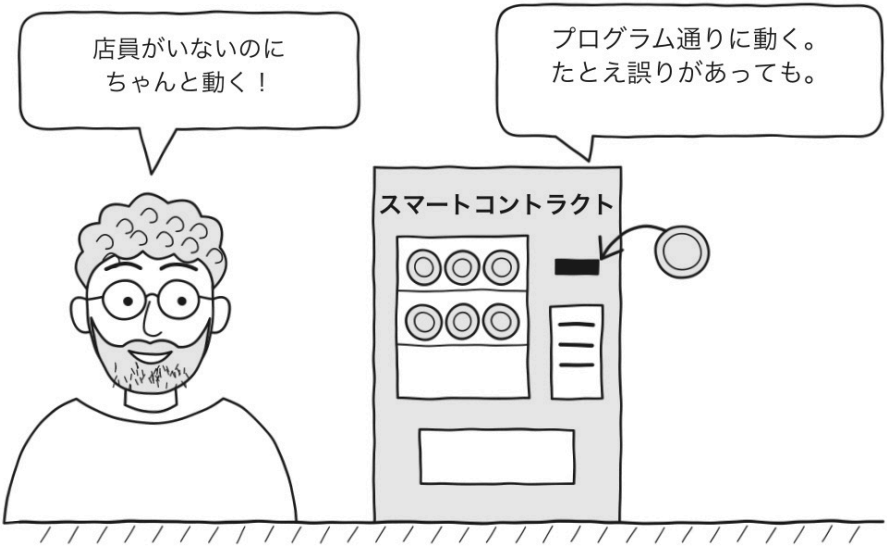


	ネイティブコイン	トークン
誰が記録をつけるか	ネットワーク自身	スマートコントラクト
Ethereumでの例	ETH	USDT、USDC
手数料の支払いに使うもの	そのコイン自身	ネットワークのネイティブコイン
残高の聞き方	`eth_getBalance`	コントラクトへの`eth_call`

トークンの仕組みはネットワークによって違います。ここではEthereumとその仲間を見ていきます。

スマートコントラクト：店員のいない自販機

スマートコントラクトは、ブロックチェーンに書き込まれたプログラムです。自動販売機に似ています。お金を入れてボタンを押せば、あらかじめ決めたルールどおりの結果が出ます。どのノードも同じように実行します。



- 署名を交わす契約書ではなく、あくまでプログラムです。
- ふつうは変更できませんが、変更できる余地を自分に残す作者もいます。
- プログラムに、特別な権限を持つ「持ち主」がいることがあります。
- プログラムの誤りはお金の損失になりえて、巻き戻してくれる人はいません。

やってみよう：トークンの残高

`eth_call`というメソッドは、「何も変えずにコントラクトに聞く」という意味です。ノートに何も書き込まないので、この質問に手数料はかかりません。USDTのコントラクトに、コントラクト自身のアドレスにUSDTがいくらあるかを聞いてみます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0x70a08231000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"},
  "latest"]}'
```

`to`は、どのコントラクトに聞くかです。`data`は3つの部分をつなげたものです。

部分	意味
`0x70a08231`	どの質問か：「残高はいくら？」（ `balanceOf` ）
24個のゼロ	決まった長さにするための埋め草
40文字	誰の残高か： `0x` を除いたアドレス

この章を書いていたとき、ノードはこう答えました。

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000014d0a9aa6c5"}
```

ゼロのあとに `14d0a9aa6c5` があります。これは1,430,402,016,965です。EthereumのUSDTは小数点以下が6桁なので、コントラクトのアドレスには1,430,402.016965 USDTあることになります。

なぜそんなところにあるのでしょう。どうやら大部分は「宛先違い」の送金です。受取人のアドレスの代わりに、コントラクトのアドレスを貼り付けた人たちがいたのです。送った人がふつうの方法で取り戻す手段はありません。

小数点はどこにあるか

コントラクトが保存するのは整数だけです。何桁目で区切るかは、 `decimals` という別の属性が教えます。

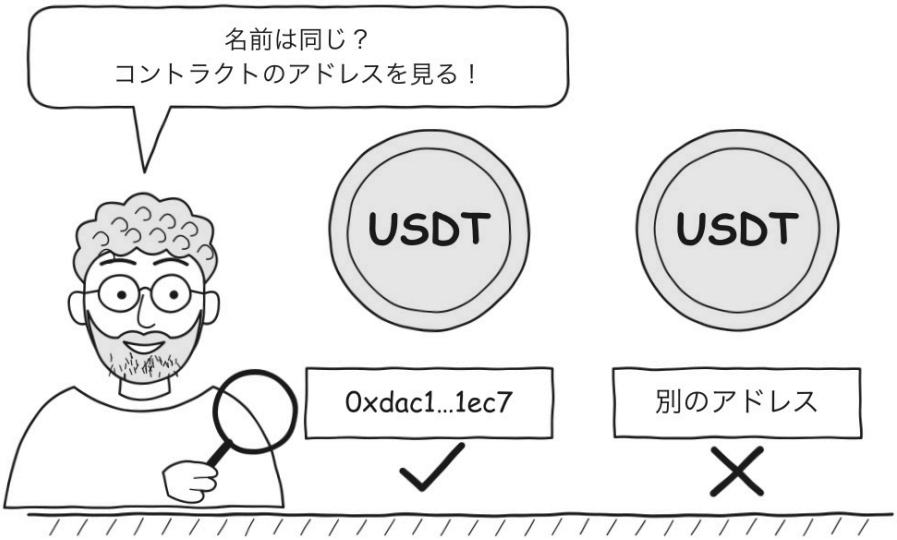
対象	小数点以下の桁数
ETH、ネイティブコイン	18
EthereumのUSDT	6
BNB Smart Chainのトークン「Tether USD」（USDT）	18

名前は同じなのに、桁数は3倍です。取り違えれば、1兆倍の間違いになります。しかも、このBNB Smart Chainのトークンのアドレスは、Tether社のサイトにあるUSDTのコントラクト一覧に載っていません。名前が同じでも、発行元が同じとは限りません。

2026年11月27日に、公開ノードへのリクエストと tether.to/en/supported-protocols のページで確認しました。

落とし穴：偽物のトークン

自分のトークンに「USDT」と名づけることは、誰にでもできます。名前は何の証明にもなりません。本物のUSDTは、ネットワークとコントラクトアドレスの組で見分けます。



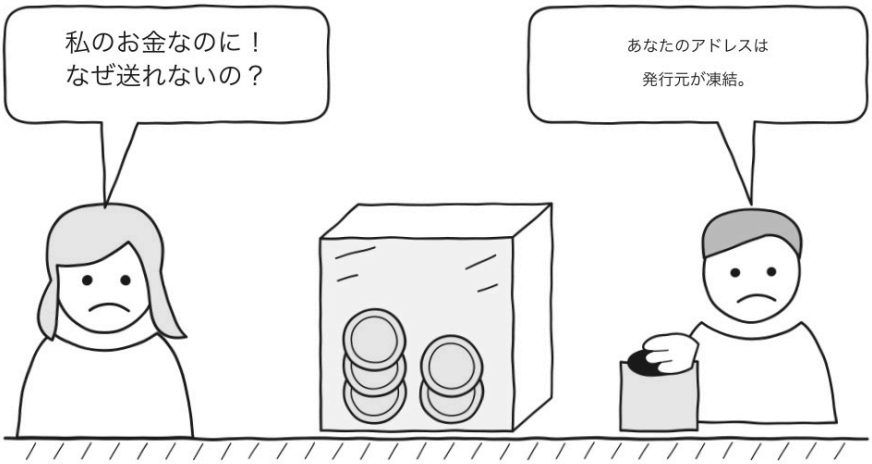
だからお店は、まとめて全部を確かめます。

- ネットワーク。
- トークンのコントラクトアドレス。
- 受取人。
- 正しい桁数で読んだ金額。
- 成功したことと承認数。

ステーブルコイン：便利だが、持ち主がいる

ステーブルコインは、1ドル付近を保とうとするコインです。USDTの場合、それは発行会社の約束であって、自然の法則ではありません。

発行会社には権限もあります。Tetherの利用規約には、トークンを凍結する権利がはっきり書かれています（2026年2月26日版、tether.to/en/legal）。



コインは見えるが動かない

もう一つの落とし穴：「使うことを許可してください」

サイトが許可（`approve`）への署名を求めることがあります。「このコントラクトは私のトークンを使ってよい」という内容です。詐欺師に金額の上限なしで許可すると、後で新しい署名なしに持っていかれます。シークレットフレーズを知る必要すらありません。

> ルール：何に署名するのかわからないなら、署名しない。

考えてみよう



頭の体操をしよう！

1. EthereumのUSDTについて、ノードが25,000,000という数を返しました。何USDTでしょうか。
2. お客さんが、名前はUSDTでもコントラクトアドレスが違うトークンを送ってきました。注文は支払い済みでしょうか。
3. アルチョムはEthereumに100 USDTと0 ETHを持っています。仕入れ先に送れるでしょうか。

答え：1）25。2）いいえ。3）いいえ。手数料を払うものがありません。第7章を参照。

まとめ

- コインはネットワークが管理し、トークンはスマートコントラクトが管理します。
- トークンは名前ではなく、ネットワークとコントラクトアドレスで見分けます。
- 金額は、小数点以下の桁数を考えて読みます。

第2部はここまでです。アルチョムは、ブロック、鍵、手数料、トークンが何かを知りました。それでもノードは、彼にとってまだ中の見えない箱です。次へ：第3部、第10章。ノードにはどんな種類があるのか。

第10章　ノードの種類

「あのとき、いくらあった？」

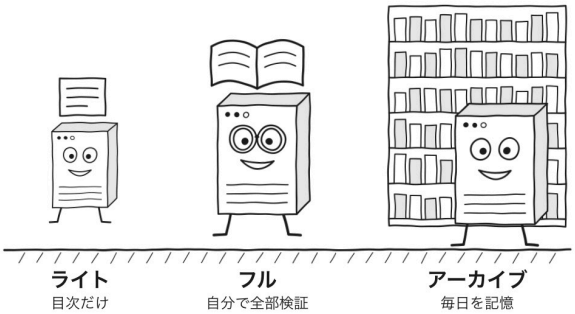
アルチョムは帳簿を突き合わせていて、数か月前にお店のアドレスにいくらあったかを知りたくなりました。ノードは今日のことなら立派に答えるのに、過去のことは断ります。ブロックチェーンは全部を保存しているのではなかったのでしょうか。

解説：記録と残高

ここには2つの別のものが隠れています。

- 履歴は記録そのもの。ブロックと送金です。
- 状態は、ある瞬間に誰が何を持っているか。残高とコントラクトのデータです。

銀行に似ています。取引の一覧はありますが、去年の1分ごとの残高が別々に保管されているわけではありません。知りたければ計算し直す必要があります。ノードの違いは、このうちどこまでを覚えているかです。

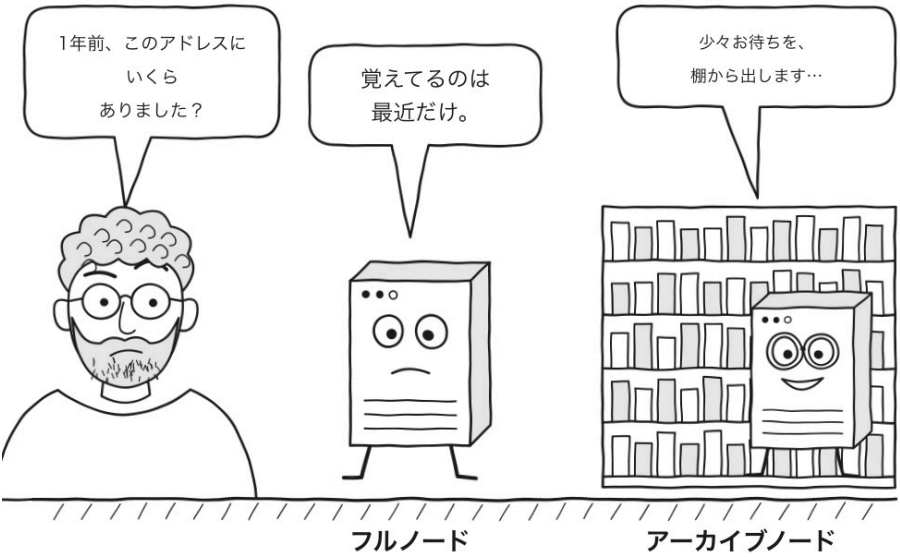


3つの主な種類

フルノードは、ネットワークのルールでブロックを検証し、新しい状態を保管します。Ethereumではふつう直近の約128ブロック、つまりおよそ25分ぶんです。正確な深さは、プログラムと設定によります。

アーカイブノードは、最初からすべてを検証し、取り込んだものを削除しません。どのブロック時点の状態も覚えていますが、その代わりディスクがはるかに多く必要です。

ライトノードは、ブロックの見出し（ヘッダー）だけを取り込み、詳しいことは他のノードに聞きます。よい考えですが、2026年11月27日の時点で、Ethereumには日常的に使える状態のライトノードはありません。



プルーニングノード：確かめて捨てる

Bitcoinにはディスクを節約するモードがあります。プルーニングノード（pruned）はすべてを検証したあと、古いブロックを削除します。権限の欠けないフルノードであることに変わりはなく、消したものを渡せないだけです。Bitcoin Coreの開発者によれば、10 GBでも足ります。

Ethereumにも同じような機能ができました。2025年7月から、ノードのプログラムは、いちばん古いブロック、つまり2022年9月のステーキング移行より前のブロックの中身を保管しないことができます。これは設定で、どこでも有効になっているわけではありません。そのため「フル」ノードでも、ごく古い記録は覚えていないことがあります。

やってみよう：過去を聞く

同じアドレスの残高を2回聞きます。まず現在の分です。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x2a"}
```

`0x2a` は42 wei、ほんのわずかです。今度は `"latest"` を `"0xe4e1c0"`、つまり15,000,000番のブロックに置き換えてください。公開ノードはこう答えます。

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
 "message":"archive data is not available on the public endpoint,
 use a personal endpoint"}}
```

日本語にすると「公開エンドポイントではアーカイブデータは使えません。個人用エンドポイントを使ってください」。故障ではなく、無料サービスの境界線です。RpcNodeでは、アーカイブアクセスは有料プランにあり、Freeにはありません（第3章）。

どれくらいの重さか

ノード	ディスクの目安
Bitcoin、プルーニング	10 GBから
Bitcoin、フル	約900 GB
Ethereum、フル	約2 TB以上
Ethereum、アーカイブ	プログラムによって2〜14 TB以上

Ethereumのデータは、ノードの運用を扱うethereum.orgのページによる2026年11月27日時点のものです。

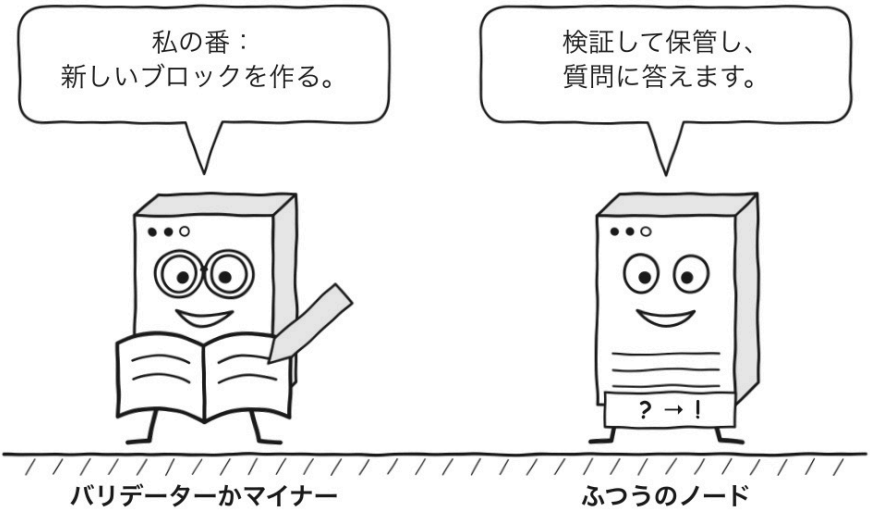
2行目は自分で確かめられます。Bitcoinのノードは、自分の大きさを正直に教えてくれます。

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getblockchaininfo","params":[]}'
```

長い答えの中から2か所を探してください。この章を書いていたときは、`"size_on_disk":896027659764`（補助ファイルを含めて約896 GB）と`"pruned":false`（プルーニングしていない）でした。

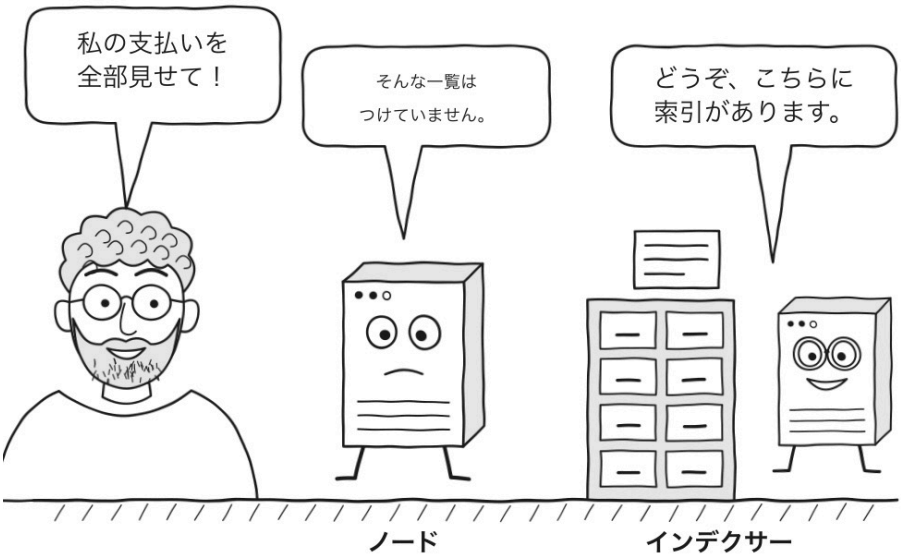
では、ブロックを書くのは誰？

それは別の役割です。バリデーターやマイナーがブロック作りに参加します（第8章）。支払いを確かめるだけなら、その役になる必要はありません。読んで、検証して、答えるノードがあれば十分です。



落とし穴：「私の支払いを全部、ノードに聞こう」

Ethereumノードへの標準の質問の中に、「このアドレスのすべての送金を見せて」というものはありません。アーカイブノードでもできません。`eth_getTransactionCount`というメソッドが教えるのは、そのアドレスから送ったトランザクションの数だけです。

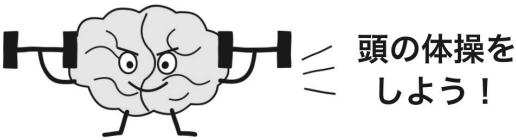


そういう一覧にはインデクサーが必要です。ブロックを読み、記録を索引カードのように整理するプログラムです。ブロックチェーンエクスプローラーのサイトもこの仕組みです。お店は、最初から自分の支払いを記録して、新しい支払いを見張るほうが簡単です。これは第26章で。

目的に合うのはどれか

やりたいこと	向いているもの
最新の支払いを確かめる	ふつうのフルノード
過去のある日の残高を知る	アーカイブノード
アドレスの全送金の一覧を得る	インデクサー
ブロック作りに参加する	バリデーターまたはマイナー

考えてみよう



1. アルチョムは、1日1点で、お店の1年分の残高グラフを作りたいと考えています。どのノードが必要でしょう。
2. Bitcoinのプルーニングノードは古いブロックを削除しました。新しいブロックの検証は任せられるでしょうか。
3. お客さんに過去の注文と支払いをすべて見せるには、何が必要でしょう。アーカイブノードですか、お店自身のデータベースですか。

答え：1) アーカイブノード。2) はい。削除する前にすべて検証しています。3) 自分のデータベース。ノードはそういう一覧をつけていません。

まとめ

- 履歴は記録、状態はある瞬間の残高です。
- フルノードは新しい状態を、アーカイブノードはどの時点の状態も覚えています。
- アドレスの送金一覧はノードからは得られません。インデクサーが必要です。

次へ：第11章。ここまでノードに10ほどの質問をしてきました。そろそろ、どんな言葉で聞いているのかを理解しましょう。JSON-RPC、REST、gRPC、WebSocketです。

第11章　ノードとの話し方

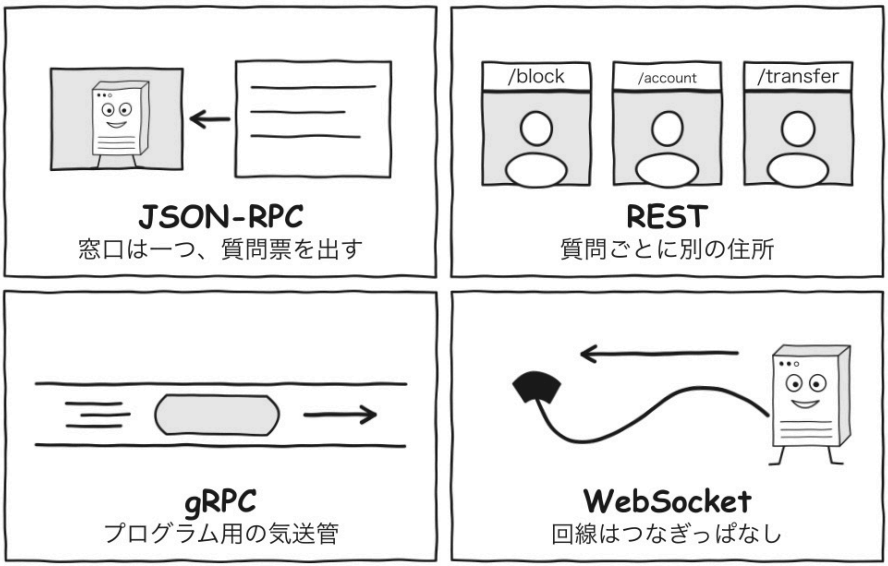
リクエストに並んでいるこの文字は何？

アルチョムはこの本のコマンドをもう10回ほどコピーし、どれも動きました。そろそろ中身を理解する番です。RPCとは何か、なぜこんなに括弧が多いのか、ネットワークによってドキュメントのリクエストの形が違うのはなぜか。

解説：APIとRPC

APIは、あるプログラムが別のプログラムに、あらかじめ決めたルールで用件を伝える窓口です。RPC（遠隔手続き呼び出し）は、別のコンピューターにあるプログラムに、まるで隣にいるかのように仕事を頼む方法です。ノードはこうした依頼を受け付けます。

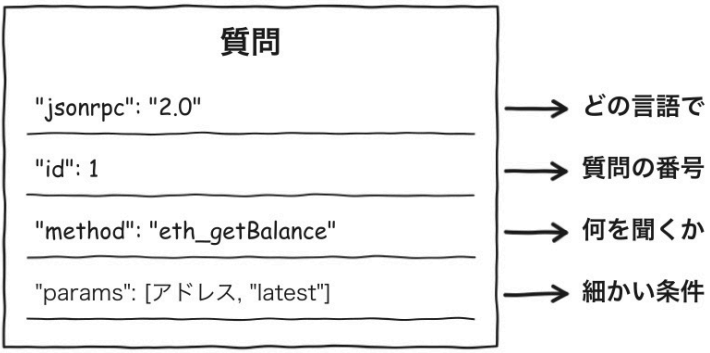
話し方はいくつかあります。



方法	どんな形か	どこで出会うか
JSON-RPC	アドレスは一つ。リクエストにメソッド名とパラメーターを書く	Ethereumとその仲間、Solana、Bitcoin
REST	質問ごとにアドレス内のパスが違う	TRON、TON、Aptos
gRPC	厳密な定義に沿った、小さなバイナリのメッセージ	プログラム同士のやり取り。たとえばSui
WebSocket	閉じない接続	ノードのほうから新しいことを知らせてほしいとき

WebSocketは言葉ではなく通信回線です。Ethereumでは、その上を同じJSON-RPCのメッセージが流れます。複数の方法を同時に持つネットワークもあります。

JSON-RPCを分解する



JSONは、データを書く簡単な方法です。波括弧、引用符で囲んだ名前、そして値。第3章のコマンドを1行ずつ見てみます。

コマンドの部分	意味
`curl`	ターミナルからリクエストを送るプログラム
`https://…`	エンドポイントのアドレス
`-H '…'`	「中身はJSON」という目印
`-d '{…}'`	質問そのもの

レスポンスでは、`result`（得られた結果）と`error`（うまくいかなかったこと）を見ます。どちらか片方だけを返すノードもあれば、両方返すノードもあり、その場合は空のほうが`null`です。レスポンスの`id`は質問と同じ値で、これで対応づけます。

一度にいくつもの質問

質問は角括弧で囲んだ一覧にまとめられます。これをバッチ（batch）と呼びます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '[{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]},
       {"jsonrpc":"2.0","id":2,"method":"eth_chainId","params":[]}]'
```

```
[{"jsonrpc":"2.0","id":1,"result":"0x1946420"},
 {"jsonrpc":"2.0","id":2,"result":"0x1"}]
```

1つ目の答えはブロック番号26,502,176、2つ目はネットワーク番号で、Ethereumは1です。JSON-RPCのルールでは、バッチの答えはどんな順番で返ってもかまわないので、`id`を見てください。そしてバッチは往復の回数は節約しても、利用枠は節約しないことを覚えておきましょう。RpcNodeでは、質問10個のバッチは10リクエストと数えられ、まるごと受け付けられるか、まるごと拒否されます。

こちらはREST

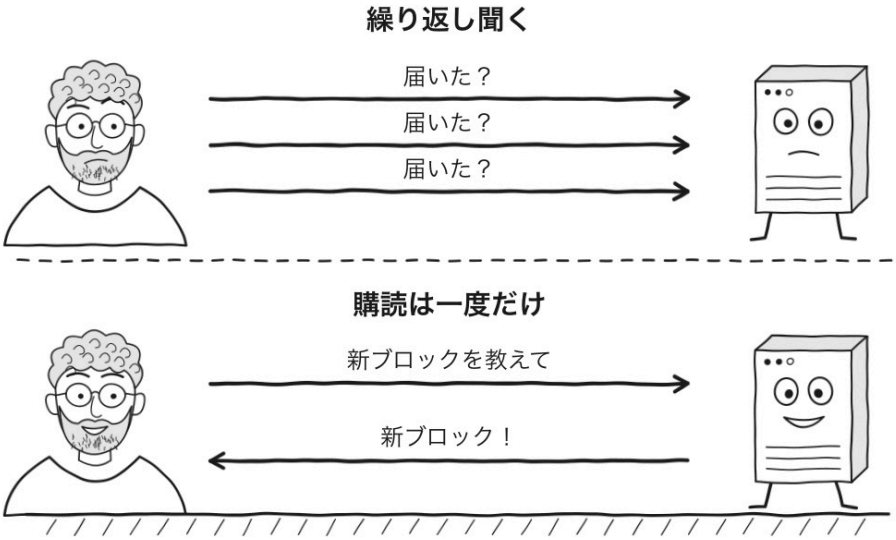
RESTでは、質問がアドレスそのものに埋め込まれています。TRONに最新のブロックを聞いてみます。

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

送金の入ったブロックについての長い答えが返ります。この章を書いていたときは88,351,889番のブロックで、トランザクションが194件入っていました。質問が変われば、アドレスの後ろのパスも変わります。

WebSocket：聞くのではなく、耳を澄ます

「届いた？」と毎秒聞くのは無駄遣いです。開いたままの回線なら、一度購読すれば、ノードのほうから知らせを送ってきます。



RpcNodeの公開ノードはWebSocketには応答しないので、個人用エンドポイントが必要です。そのアドレスは `wss://` で始まります。JavaScriptでの簡単な例です。

```
const ws = new WebSocket('wss://あなたのエンドポイント')

ws.onopen = () => ws.send(JSON.stringify({
  jsonrpc: '2.0', id: 1,
  method: 'eth_subscribe', params: ['newHeads'],
}))

ws.onmessage = (event) => console.log(event.data)
ws.onclose = () => console.log('接続が切れました')
```

回線は切れるものです。プログラムは、接続し直し、もう一度購読し、取りこぼしがないか確かめられなければなりません。購読でもクレジットを消費します。

gRPC

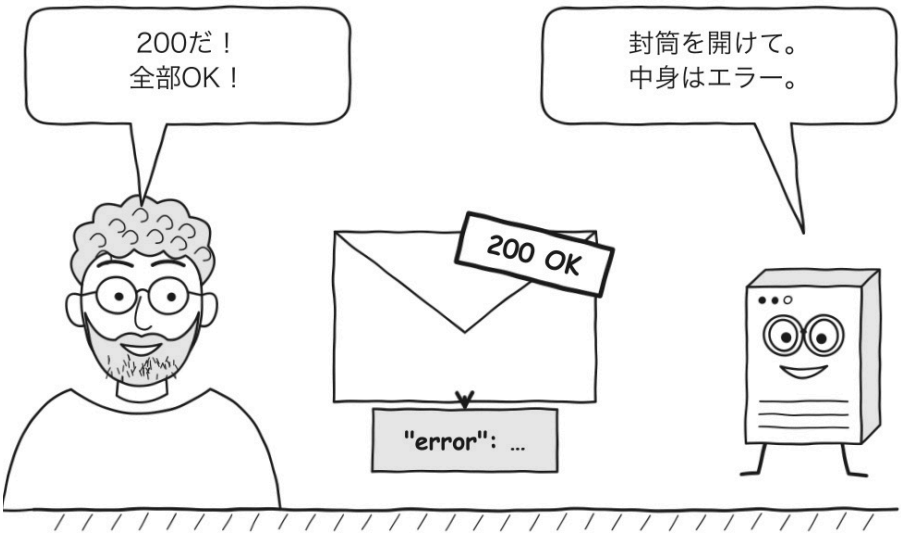
速さと厳密な形式を重視するプログラム向けの方法です。メッセージは人の目では読めず、フィールドはあらかじめ別のファイルに定義されています。お店を始めたばかりなら、ふつうは必要ありません。RpcNodeではgRPCがすべてのネットワークにあるわけではないので、エンドポイントの対応トランスポート一覧を確認してください。

落とし穴：「200なら全部うまくいった」

200は配達の記号で、封筒が届いたという意味です。中には工ラーが入っているかもしれません。「hello」というアドレスの残高を聞いてみます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["hello","latest"]}'
```

ステータスコードは200なのに、中身は `-32602` 番の `error` 、つまりパラメーターの誤りです。



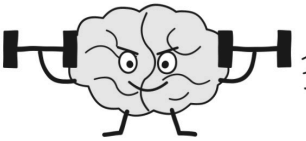
だからプログラムは、いつもステータスコードと `error` フィールドの両方を確かめます。

よくあるエラー

コード	何が起きたか	どうするか
`-32700`	リクエストの文面が解析できない	括弧と引用符を確かめる
`-32600`	リクエストがルールどおりでない	フィールドを確かめる
`-32601`	メソッドがないか、使えない	名前とプランを確かめる
`-32602`	パラメーターが誤り	値を確かめる
`-32005` （HTTP 429）	RpcNode：頻度が高すぎる	待って、回数を減らす
`-32006` （HTTP 402）	RpcNode：クレジット不足	使用量とプランを確かめる

最初の4つはJSON-RPC 2.0の標準のコード、最後の2つはRpcNodeのコードです。

考えてみよう



頭の体操をしよう！

1. バッチの1つ目で、`eth_blockNumber`の代わりに`eth_blockNumbr`と書いてください。どのコードが返りますか。正しい2つ目の質問には答えてもらえるでしょうか。
2. 自動で更新したい「最新ブロック」の表示板には、毎秒聞くのと購読と、どちらがよいでしょう。
3. プログラムがHTTP 200を受け取り、すぐに注文を支払い済みにしました。何が問題でしょうか。

答え：1）`-32601`が全体で1つ返ります。RpcNodeはバッチをまるごと拒否するので、2つ目の質問にも答えはありません。2）購読。3）`error`フィールドも、送金そのものも確かめていません（第7章）。

まとめ

- JSON-RPCはメソッドをリクエストの中で、RESTはアドレスで指定します。
- WebSocketは回線を開いたままにして、ノードのほうから知らせてもらいます。
- 成功とはコード200のことではなく、`error`がなく、期待した中身が入っている答えのことです。

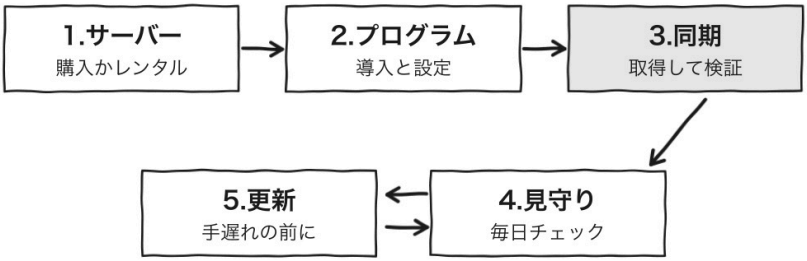
次へ：第12章。窓口の向こう側をのぞきます。自前のノードの暮らし、同期とは何か、なぜ壊れるのか。

第12章　自前ノードの内側：インストール、同期、故障

「そんなの一晩で終わるよ！」

知り合いのプログラマーがアルチョムを説得します。「ダウンロードして起動すれば、もう自分のノードだよ」。たしかに起動はすぐです。長いのは、そこから先です。

解説：ノードの一生

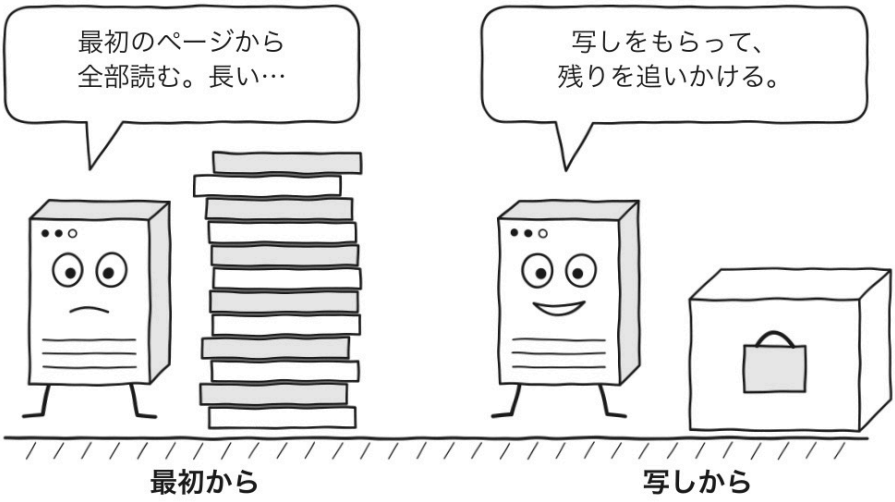


ノードが動く限り4と5は繰り返し

1. サーバー。要件は第2章で見ました。
2. プログラム。ネットワークごとに専用のものがあり、複数から選べることもよくあります。Ethereumでは2つが同時に必要です。実行クライアント（送金とコントラクトを実行する）とコンセンサスクライアント（どの鎖が正しいかを見張る）です。
3. 同期。ノードがネットワークに追いつきます。
4. 見守り。遅れていないか、容量は足りるか。
5. 更新。ネットワークがルールを変えるので、プログラムは前もって入れ替えます。

同期：ネットワークに追いつく

新しいノードは何も知りません。ほかに追いつくまで、残高を聞いても意味がありません。過去のことを答えてしまいます。



- 最初から。ノードがすべてのブロックを自分で検証します。確実ですが時間がかかります。Bitcoinの最初のダウンロードは約740 GBです。
- スナップショットから。できあいのデータベースの写しを使い、残りを追いかけます。速い代わりに、写しを作った人を信用することになります。
- 中間の方法。Ethereumのプログラムは、ふつう新しい状態をダウンロードし、ブロックヘッダーと照らし合わせます。

かかる時間は数時間から何日にも及びます。ネットワーク、方法、ディスク、回線しだいです。

やってみよう：2行の設定

設定ファイル`bitcoin.conf`の冒頭は、たとえばこうなります。

```
server=1
prune=5000
```

行	意味
`server=1`	RPCの質問を受け付ける。別に許可しないかぎり、同じコンピューターからだけ
`prune=5000`	プルーニングノード（第10章）。古いブロックは削除し、約5 GBぶんを残す。管理用のデータはその上に場所を取る

これは完全な手順ではありません。インストールの前に、使うプログラムのドキュメントを読んでください。

ノードが追いついたかを知る方法

ノードに直接聞けます。Ethereumではこうです。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_syncing","params":[]}'
```

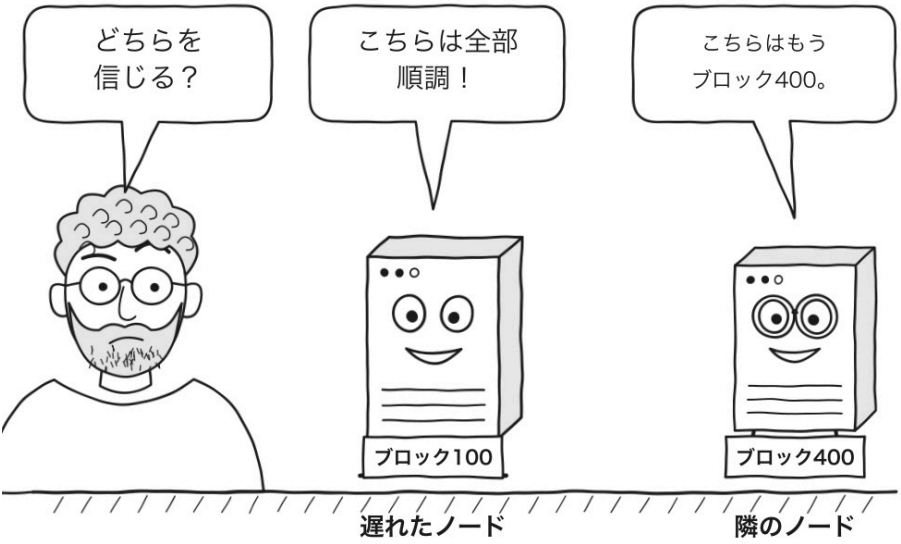
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false`は「今は追いかけている最中ではない」という意味です。追いかけている間は、代わりに「今どこにいて、どこを目指しているか」の番号が返ります。Bitcoinでは、第10章の`getblockchaininfo`の答えを見ます。

フィールド	確認時の値	意味
`blocks`	977,715	検証したブロックの数
`headers`	977,715	存在を知っているブロックの数
`initialblockdownload`	`false`	最初のダウンロードは終わった

落とし穴：「本人が大丈夫と言った」

ノードは、自分の知っていることで判断します。ほかのノードとの接続を失うと、自分が遅れていることに気づかない場合があります。



だから2つを確かめます。最後のブロックの新しさと、別の情報源との比較です。新しさは簡単にわかります。最後のブロックの時刻を聞いてください。

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["latest",false]}' | grep -o '"timestamp":"0x[0-9a-f]*"'
```

```
"timestamp":"0x6b08aacb"
```

これは1970年からの経過秒数です。現在の時刻からこれを引きます（自分の値に置き換えてください）。

```
echo $(( $(date +%s) - 0x6b08aacb ))
```

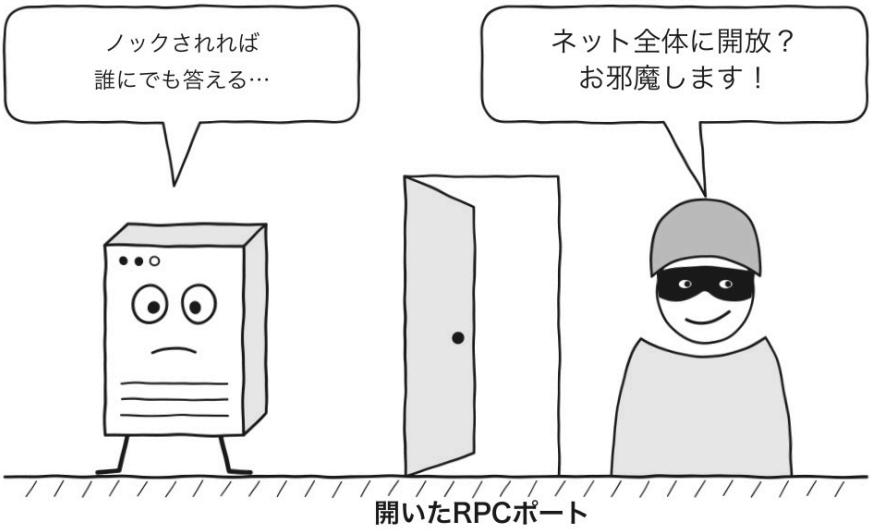
ブロックの古さが秒で出ます。この章を書いていたときは7秒でした。12秒に1ブロックのEthereumなら正常です。あなたのノードが何千秒にもなるなら、遅れています。第1章の、1時間遅れたノードの問題を覚えていますか。

何が壊れるのか

見える症状	よくある原因	対処
ノードが止まった	ディスクの空きがなくなった	空き容量を見張り、早めに増やす
ずっと遅れている	ディスクや回線が弱い、接続先が少ない	機材とネットワークを確かめる
ネットワークの更新後に「固まった」	プログラムのバージョンが古い	告知された期日までに更新する
停電のあと起動しない	データベースが壊れた	バックアップから復元するか、同期し直す
理由もなく遅い	他人のリクエストで負荷がかかっている	部外者の接続を閉じる

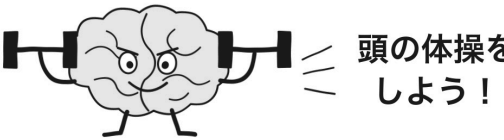
もう一つの落とし穴：開けっ放しのドア

Bitcoin CoreとGethは、初期状態では自分のコンピューターからのRPCの質問だけを受け付けます。これには理由があります。ポートをインターネット全体に開けば、誰でも入れることになり、ノードにウォレットがあれば、お金が危険にさらされます。



- 接続は自分のアドレスからだけ許可する（ファイアウォール）。
- ノードの手前でキーの確認をする。
- サイトに応答するノードに、お金の入ったウォレットを置かない。

考えてみよう



1. `eth_syncing` は `false` と答えますが、最後のブロックは40分前のものです。このノードは健康でしょうか。
2. 2000 GBのディスクのうち1900 GBを使用中で、1日に5 GB増えます。何日で止まりますか。
3. サイトは別のサーバーにあります。「サイトから聞けるように」、RPCポートを全員に開けてよいでしょうか。

答え：1）いいえ。2）遅くとも20日後。対処は今日から始めます。3）いいえ。サイトのアドレスだけを許可します。

まとめ

- ノードの起動は、仕事の終わりではなく始まりです。
- 健康状態は、ブロックの新しさと、別の情報源との比較で確かめます。
- RPCポートをインターネット全体に開けてはいけません。

次へ：第13章。こうしたことがあなたの趣味でないなら。プロバイダーをどう選び、何を聞くか。

第13章　プロバイダーの選び方

どこも「最高」をうたう

アルチョムは決めました。自前のノードは自分向きではない。プロバイダーのサイトを開くと、どこも感嘆符だらけです。



解説：まず自分のことから

自分に何が必要かを知らずにプロバイダーを比べても、意味がありません。まず次の質問に答えます。

- どのネットワークが必要か。テストネットも必要か。
- 1秒あたり、1か月あたりのリクエスト数は？
- 古い履歴（アーカイブ）は必要か。
- 出来事をすぐに知る必要はあるか（WebSocket、通知）。

自分のリクエスト数を数える

アルチョムの店では20件の注文が支払いを待っていて、サイトは10秒に1回、それぞれを確認します。

注文20件 × 毎分6回 = 毎分120リクエスト、毎秒にすると2回。
30日では5,184,000リクエスト。

次に、メソッドのクレジット単価を掛けます。単価はメソッドによって違います。そして、そんなに頻繁に聞く必要があるかを考えてください。30秒に1回にすれば、リクエストは3分の1になります。

質問リスト



聞くこと	注意すべき兆し
自分のネットワークはあるか。必要なメソッドも使えるか	ネットワークの長い一覧はあるのに、メソッドの一覧がない
上限：毎秒と合計のリクエスト数。超えたらどうなるか	数字のない「無制限」。わかりにくい追加料金
アーカイブ、WebSocket、gRPC：どのプランで、どのネットワークで使えるか	「全部あります」なのに詳しい説明がない
ノードの故障：別のノードに切り替わるか。サーバーはどこにあるか	答えがない
稼働率：何を約束し、どう測り、破ったらどうなるか	期間も条件もないパーセント表示
保護：アドレスやサイトを制限できるか。リクエストの何を記録するか	キーは誰が使っても動く
サポート：どこに連絡し、何語で返事が来るか	名前もわからない入力フォームだけ
乗り換え：メソッドは標準のものか。月払いはできるか	長期契約が必須

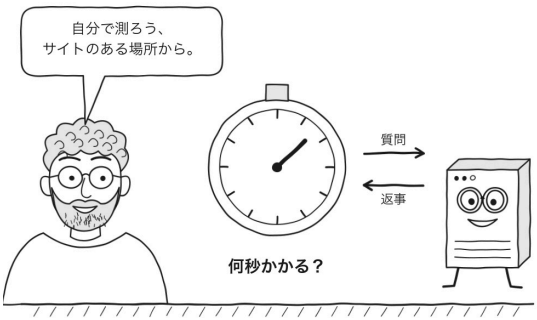
パーセントが隠すもの

「稼働率99%」は、ほとんど「いつでも」に聞こえます。30日でどれだけの停止が許されるか、計算してみましょう。

約束	止まってよい時間
99%	7時間12分
99.9%	約43分
99.99%	約4分

そして、何が動いていれば「稼働」と数えるのかを聞いてください。プロバイダーのサイトなのか、あなたのネットワークの、あなたのエンドポイントなのか。

やってみよう：速度は自分で測る



他人の測定は、他人の場所から行われたものです。あなたのサイトが動くことになるサーバーから、5回リクエストしてください。

```
for i in 1 2 3 4 5; do
  curl -s -o /dev/null -w '%{time_total} s\n' \
    https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
  sleep 1
done
```

5つの数字が出ます。秒単位の応答時間です。いちばん良い値だけを見ないでください。お客さんはいちばん悪い値にも当たります。この例では毎回接続を開き直していますが、接続を保ち続けるプログラムはふつうもっと速くなります。そして速さだけでなく、ブロックの新しさも比べてください（第12章）。

私たちにも同じ質問を

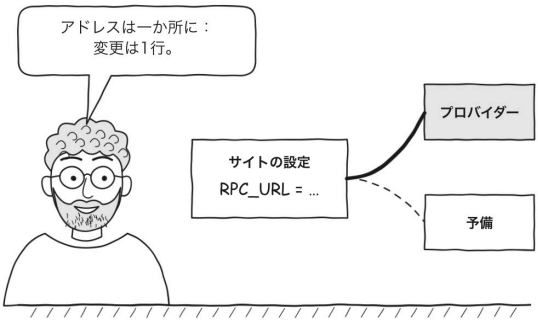
これはRpcNodeの本なので、RpcNodeも調べましょう。2026年11月27日時点の情報です。

質問	RpcNode
ネットワーク	サイトの一覧に25
無料で試す	Freeプランと、キー不要の公開ノード
上限	Free：毎秒15リクエストと1500万クレジット。Starter：毎秒40と5000万
超えたとき	超過の追加料金はない。リクエストは拒否される（429または402、第11章）
アーカイブ	有料プランで利用可
WebSocketとgRPC	個人用エンドポイント経由。すべてのネットワークではない
サーバーの場所	フランクフルト、アメリカ、日本
保護	エンドポイントはIPアドレスとサイトで制限できる
プライバシー	プロバイダーにはあなたのリクエストが見える（第3章）
稼働率とサポート	rpcnode.devで現行の条件を確認してください

落とし穴：いちばん大きな数字で選ぶ

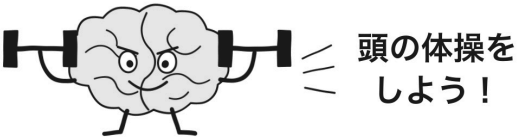
ネットワークが1000あっても、あなたの2つがうまく動かなければ役に立ちません。いちばん安くても、必要なメソッドが高かったり使えなかったりすれば役に立ちません。無料か最安のプランで、実際のリクエストを数日流してみてください。

根を張りすぎない



- エンドポイントのアドレスは、コードではなく一つの設定にまとめる。
- できるだけネットワークの標準メソッドを使う。
- 予備の選択肢を用意し、一度は試しに切り替えてみる。

考えてみよう



1. 注文20件を30秒に1回確認すると、30日で何リクエストになりますか。
2. 99.9%が約束されていたのに、1か月でエンドポイントが2時間止まりました。約束は守られたでしょうか。
3. エンドポイントのアドレスが、コードの14か所に書かれています。何がまずいのでしょう。

答え：1）1,728,000。2）いいえ。許されるのは約43分です。3）変更するとき直し忘れやすく、コードに書いたキーは漏れやすくなります。

まとめ

- まず自分に必要な量を数え、それから比べます。
- 約束は、自分の測定と自分のリクエストで確かめます。
- いつでも乗り換えられる道を残しておきます。

第3部はここまでです。アルチョムは、ノードとは何か、どう話すか、どこから借りるかを知りました。でも、ネットワークはそれぞれ違うのです！　次へ：第4部、第14章。Bitcoinとその親戚たち。

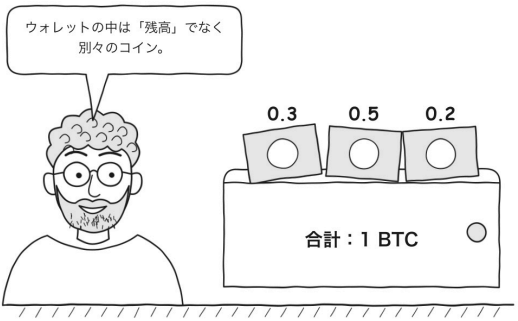
第14章　Bitcoinとその親戚たち

「残高はどこにあるの？」

お客さんがビットコインで払いたいと言います。アルチョムはもう慣れたものです。Ethereumのように、ノードにアドレスの残高を聞けばいい。ところがBitcoinのノードには、そんな単純な質問がありません。なぜでしょう。

解説：口座ではなく、コインのかけら

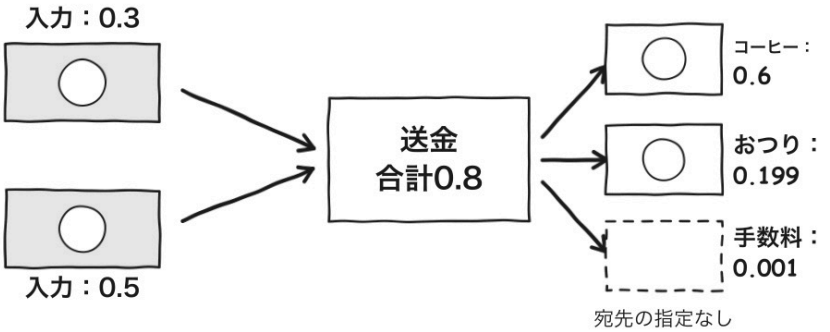
Ethereumでは、アドレスに銀行口座のような残高があります。Bitcoinにはありません。あるのは、額面の違う紙幣のような、ばらばらの「お金のかけら」です。



このかけら一つひとつをUTXO（未使用の出力）と呼びます。残高とは、あなたの未使用のかけらを全部足した額です。ウォレットかインデクサー（第10章）が計算します。

送金：入力、出力、おつり

紙幣は少しだけちぎって使えません。まるごと渡して、おつりをもらいます。ここも同じで、送金は古いかけらを丸ごと使い切り（入力）、新しいかけらを作ります（出力）。



- 出力の一つは受取人へ。
- もう一つは自分へのおつり。
- どの出力にも書かれなかった分は、マイナーのものになります。これが手数料で、入力から出力を引いた額です。

> 0.8 − 0.6 − 0.199 = 手数料0.001 BTC

手数料の値段は金額ではなく送金データの大きさで決まり、1仮想バイトあたりのサトシで表します。サトシはビットコインの1億分の1です。入力のかけらが多いほど送金データは大きくなり、高くつきます。

人から人への最初の送金

2009年1月12日、ブロック170で、50 BTCのかけら1つが、10 BTCと40 BTCの2つになりました。10のほうを受け取ったプログラマーのハル・フィニーは、サトシが試しに送ってくれたのだと自ら語っています。おつりの40 BTCは送った側に戻りました。手数料はありませんでした。

ブロックチェーンに名前は書かれません。フィニーのことは本人の話からわかっています（bitcointalk.org、2013年）。

この送金は今でも呼び出せます。

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getrawtransaction","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",true]}'
```

答えの中の `vin`（入力、ここでは1つ）と `vout`（出力、ここでは2つで `10.0` と `40.0`）を探してください。

やってみよう：そのコインは使われた？

`gettxout` メソッドは「このかけらはまだ残っている？」と聞きます。かけらは、送金の番号と出力の番号（0から数える）で指定します。あの10 BTCについて聞いてみます。

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",0]}'
```

```
{"jsonrpc":"2.0","result":null,"id":1}
```

`null` は、そんな未使用のかけらはない、つまりとっくに使われたという意味です。次は、いちばん最初のブロックの次のブロックの報酬です。

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098",0]}'
```

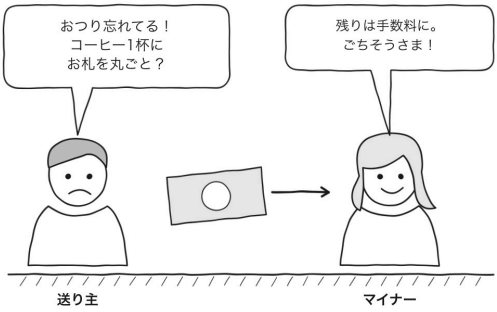
こちらは `"value":50.00000000` と、100万近い承認数が返ります。この章を書いていた時点で、この50 BTCにはまだ誰も手をつけていません。

Bitcoinについて、あと3つ

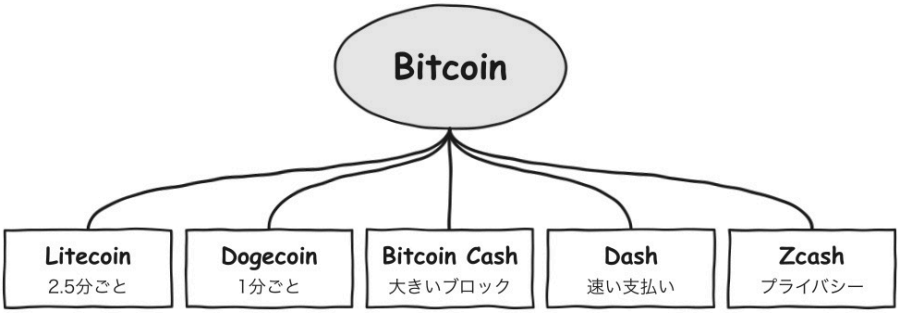
- コインは全部で2100万枚を超えません。
- マイナーの報酬は、上に100ブロック積まれるまで使えません。
- 大きな金額では6承認、約1時間待ちます（第8章）。

落とし穴：忘れられたおつり

ふつうのウォレットは、おつりを自動で付け足します。しかし、プログラムが手作業で送金を組み立てて、おつりの出力を忘れると、残り全部が手数料になります。



親戚たち



作りは親戚でも、ネットワークもコインも別々

ネットワーク	ブロック間隔の目安	違うところ	アドレスの先頭
Bitcoin	10分	一家の長男	`1`、`3`、`bc1`
Litecoin	2.5分	別のマイニングアルゴリズム、Scrypt	`L`、`M`、`ltc1`
Dogecoin	1分	発行量に上限がない。Litecoinと同時に掘れる	`D`
Bitcoin Cash	10分	ブロックがずっと大きい	`bitcoincash:q…`
Dash	2.5分	高速支払い（InstantSend）と、ブロックの置き換えを防ぐ仕組み（ChainLocks）	`X`
Zcash	75秒	送金を隠せる	公開型は `t1`、秘匿型は `zs`、`u1`

作りが似ているので、質問もほとんど同じです。5つのネットワークのチェーンの高さを、まとめて聞いてみます。

```
for net in bitcoin litecoin doge bch dash; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"getblockcount","params":[]}'
  echo; sleep 1
done
```

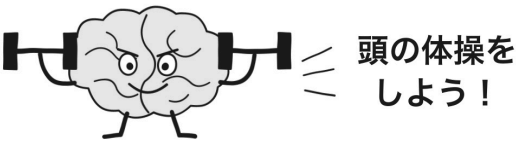
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

もう一つの落とし穴：「errorがあるからエラーだ」

答えをよく見てください。4つのネットワークには、`error`フィールドがいつも付いています。値の`null`は「問題なし」という意味です。第11章でも勧めたように、フィールドがあるかどうかではなく、中に何が入っているかを確かめてください。

そしてネットワークを取り違えないでください。親戚同士でも、お互いのコインは受け付けません。Bitcoin Cashの古いアドレス形式はBitcoinと同じ見た目だったため、`bitcoincash:`という接頭辞の付く新しい形式が導入されました。

考えてみよう



1. ウォレットに0.3、0.5、0.2 BTCのかけらがあります。0.4を支払い、手数料は0.001です。かけら1つで足りますか。おつりはいくらでしょう。
2. 100個の小さなかけらを集めて1 BTCを送るほうが、丸ごと1つで送るより高いのはなぜでしょう。
3. 入力は1.0 BTC、出力は0.4 BTCが1つだけ。残りの0.6はどこへ行きましたか。

答え：1）はい、0.5のかけらで足ります。おつりは0.099。2）入力が多いほど、送金データが大きくなるからです。3）手数料です。おそらくおつりを忘れています。

まとめ

- Bitcoinには口座の残高はなく、未使用のかけら（UTXO）があります。
- 手数料は入力から出力を引いた額。おつりを忘れないこと。
- 親戚たちは作りが似ていますが、別々のネットワークです。

次へ：第15章。Ethereum：アカウント、ガス、イベント。

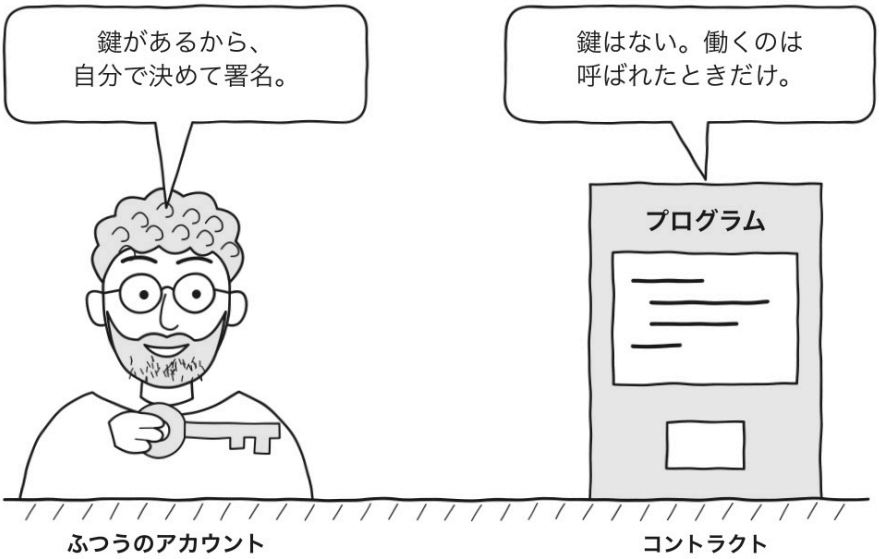
第15章　Ethereum：アカウント、ガス、イベント

「ドルを送るほうが高いのはなぜ？」

アルチョムは仕入れ先にETHを少し送り、次にUSDTを送りました。金額は似たようなものなのに、USDTでは「ガス」を2倍以上取られました。このガスとは何で、なぜそうなるのでしょう。

解説：2種類のアカウント

Ethereumでは、アドレスに銀行のような残高があります。Bitcoinのかけら（第14章）とは違います。そしてアカウントは2種類あります。



	ふつうのアカウント	コントラクト
誰が操るか	鍵の持ち主	そのプログラム
自分から送金を始められるか	はい	いいえ。呼び出されたときだけ
コードがあるか	ふつうはない	ある

プログラムを実行するのはEVM、Ethereumの仮想マシンです。どのノードの中にもゲーム機が一台あるようなもので、同じプログラムはどこでも同じ結果を出します。

やってみよう：プログラムはどこにあるか

`eth_getCode`メソッドは、アドレスにあるコードを見せます。USDTのコントラクトについて聞き、答えの先頭だけを残します。

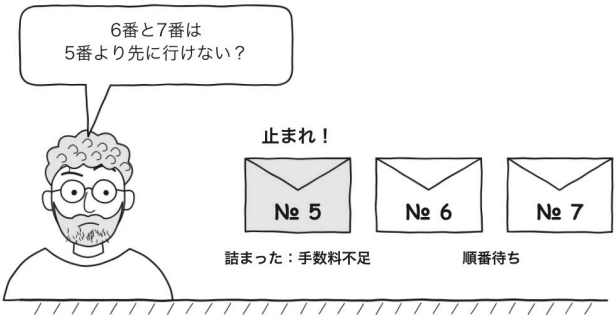
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

長い文字の連なりの冒頭が見えます。これがプログラムで、11,000バイト以上あります。アドレスを`0x0000000000000000000000000000000000000dEaD`に替えると`"0x"`が返ります。コードはないという意味です。

2025年から、ふつうのアカウントも自分の操作をプログラムに任せられるようになりました（Pectraアップグレード）。その場合、答えは`0xef0100`で始まります。つまり「コードがある」からといって、必ずしも「コントラクトだ」とは言えなくなりました。

通し番号：nonce

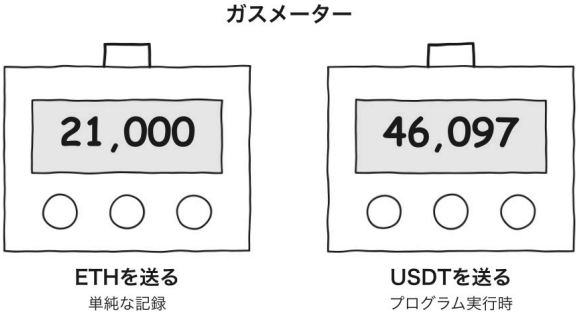
アカウントのトランザクションには、0、1、2と順に番号が付きます。これをnonce（ノンス）と呼びます。ネットワークは厳密に順番どおりに実行し、同じ番号を二度は受け付けません。



手数料の小さい送金が詰まると、後ろの送金は待たされます。ウォレットの「スピードアップ」は、同じ番号で手数料を増やした新しい送金を送り直す機能です。

ガス：仕事量のメーター

ガスは、トランザクションがどれだけの仕事を必要とするかを測る単位です。「ETHを送る」という単純な記録は、現在21,000です。トークンの送金はプログラムを動かすので、ガスがもっと必要です。



手数料 = 使ったガス × (基本料金 + チップ)

- 基本料金はネットワーク自身が決めます。このお金は燃やされます。
- チップはバリデーターが受け取ります。高いほど、喜んで取り上げてもらえます。
- ガスの値段はgweiで数えます。ETHの10億分の1です。

ブロックに入るガスの量には限りがあります。ブロックが半分より多く埋まると基本料金は上がり、少なければ下がります。たとえばブロック26,502,250は容量が6000万ガスで、93%埋まっていました。

ウォレットはガスの上限を設定します。引かれるのは使った分だけです。ただし上限が足りなければ送金は実行されず、使ったガスは戻りません。

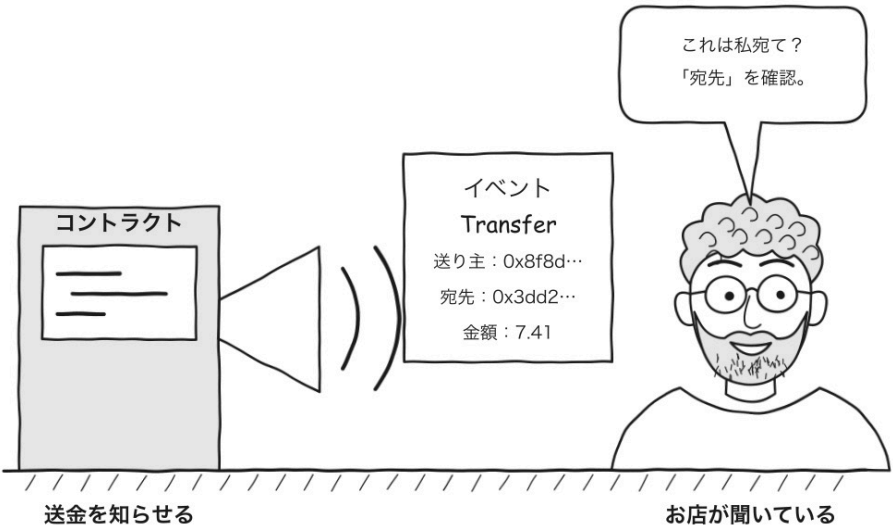
本物のUSDT送金

同じブロックにあった7.41 USDTの送金です。送り主は、この時点ですでに423,464件のトランザクションを行っていました。

項目	値
使ったガス	46,097
基本料金	0.058 gwei
チップ	2 gwei
手数料	約0.000095 ETH
うち燃やされた分	約0.000003 ETH

イベント：コントラクトが送金を知らせる方法

トークンの残高は、コントラクトの中にあります。送金があったことを世間に知らせるために、コントラクトはイベント (log) を発します。トークンでは `Transfer` という名前で、誰から、誰へ、いくら、を伝えます。



イベントは、第7章に出てきたトランザクションのレシートに入っています。この送金のレシートを聞いてみます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016"]}'
```

答えの中の`logs`を探してください。

フィールド	先頭と末尾	意味
`address`	`0xdac1…1ec7`	発した者：USDTのコントラクト
`topics[0]`	`0xddf2…b3ef`	イベントの種類：Transfer
`topics[1]`	`…8f8d…884e`	誰から
`topics[2]`	`…3dd2…a9c0`	誰へ
`data`	`…711150`	7,410,000、つまり7.41 USDT

`eth_getLogs`メソッドを使うと、イベントを検索できます。最新のブロックにあるUSDTの送金を数えてみます。

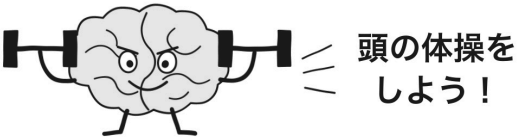
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],
  "fromBlock":"latest","toBlock":"latest"}]}' | grep -o '"logIndex"' | wc -l
```

1ブロック内の送金の件数が出ます。過去のブロックについては、公開ノードは答えません。それはもうアーカイブです（第10章）。

落とし穴：「イベントだけ聞いている」

- ETHそのものの送金は、Transferイベントを作りません。イベントだけを待つお店は、そういう支払いを見逃します。今後のネットワーク更新で変わる可能性はあります。
- ETHはコントラクトから届くこともあります。たとえば一部のウォレットからです。その場合、トランザクションそのものには、あなたのアドレスが現れません。
- イベントはどのコントラクトでも発せます。`address`を確かめ、本物のトークンかを見てください（第9章）。

考えてみよう



1. ETHの送金で21,000ガス、チップ込みの値段が2 gweiです。手数料はgweiとETHでいくらですか。
2. 数ブロック続けて30%しか埋まっていません。基本料金はどうなりますか。
3. お店はUSDTコントラクトのTransferイベントだけを見ています。お客さんはETHで払いました。お店は気づくでしょうか。

答え：1）42,000 gwei、つまり0.000042 ETH。2）下がっていきます。3）気づきません。

まとめ

- アカウントを操るのは、鍵かプログラムのどちらかです。
- 手数料は、ガスの量に値段（基本料金とチップ）を掛けたものです。
- トークンの送金はイベントで知り、ETHは別に捕まえる必要があります。

次へ：第16章。Ethereumには親戚がたくさんいます。ほかのEVMネットワークと「2階」の話です。

第16章　Ethereumの親戚と「2階」

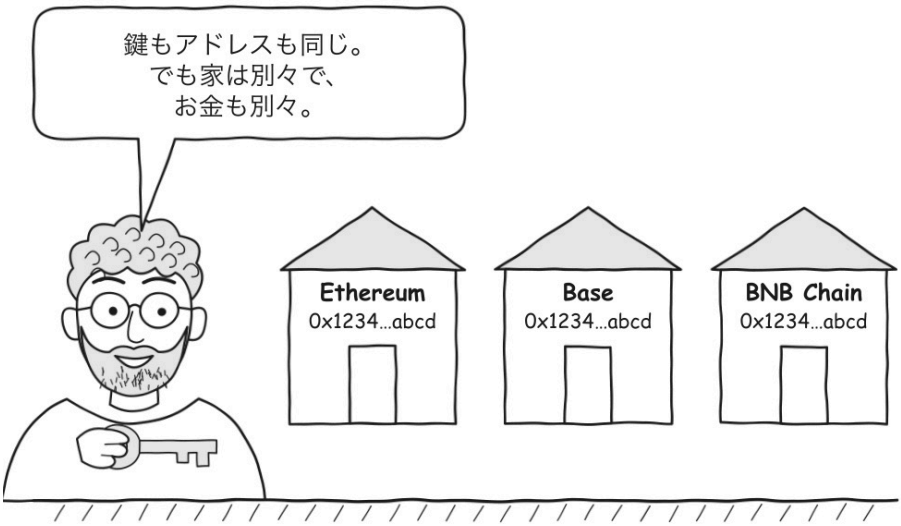
「Baseで送りました」

お客さんから「BaseでUSDCを送りました、アドレスは同じです」と連絡が来ます。アルチョムは戸惑います。別のトークン？　別のネットワーク？　なぜアドレスが同じなのでしょう。

解説：同じ機械、別々の家

第15章のEVMは、多くのネットワークに採用されました。そのためアドレスは同じ形で、`eth_…`の同じメソッドがあり、一つのコントラクトのプログラムをどこでも動かせます。こうしたネットワークをEVM互換と呼びます。

しかし、ノートは別々です。ブロックも、残高も、手数料もそれぞれです。



ネットワークを取り違えないように、それぞれに番号があります。chain IDです。これは署名に含まれるので、あるネットワーク向けに署名した送金は、別のネットワークでは受け付けられません。

やってみよう：6つのネットワークに番号を聞く

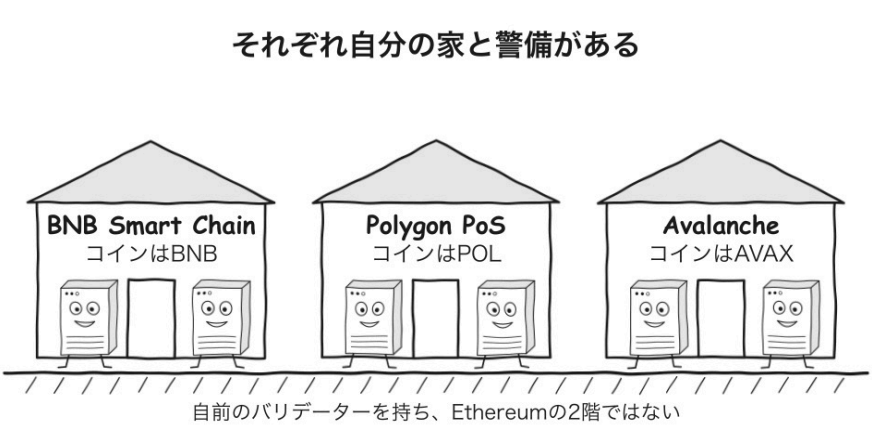
```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

16進数の数が返ってきます。ふつうの数に直したものを、確認時のブロックの間隔と合わせて示します。

ネットワーク	Chain ID	ブロック間隔の目安	手数料に使うもの
Ethereum	1	12秒	ETH
Base	8453	0.2秒	ETH
Arbitrum One	42161	0.3秒	ETH
BNB Smart Chain	56	0.45秒	BNB
Mantle	5000	2秒	MNT
Monad	143	0.3秒	MON

親戚その1：独立型

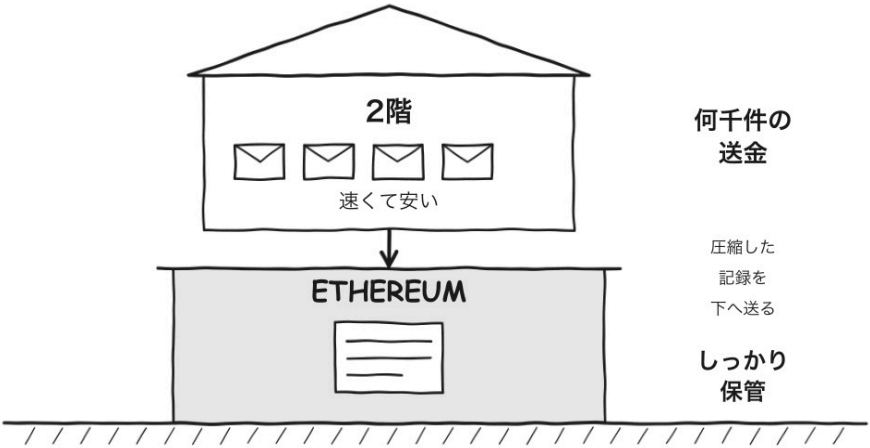
自前のバリデーターを持ち、ブロックについて自分たちで合意します。



ただしPolygon PoSは、定期的にEthereumへ自分のブロックの目印を残します。2025年11月に始まったMonadも独立型です。

親戚その2：2階（L2）

L2は送金を自分のところで速く安く実行し、その圧縮した記録をEthereumに送ります。いちばん多い型はrollup（ロールアップ、「巻き上げ」）と呼ばれます。



- 送金の順番は、ふつう一つの運営者、シーケンサーが決めます。だから何でも速いのです。そしてだから、その一点では運営者を信用することになります。
- オプティミスティックロールアップ（Arbitrum One、OP Mainnet）は記録を正しいものとみなしますが、異議を唱える期間を設けます。
- ZKロールアップ（たとえばMantle）は、数学的な証明を添えます。
- Baseは2026年に独自のコードへ移り、複数の種類の証明を組み合わせています。

確認に時間がかかるため、公式ブリッジでEthereumにお金を戻すのは、すぐにはできません。

どのネットワークから	Ethereumへの引き出しで待つ時間
Arbitrum One	7日以上
OP Mainnet	7日
Base	5日、場合によっては1日

2026年11月27日時点の各ネットワークのドキュメントによります。これらの期間は変わります。OP Mainnetは2027年1月に作り替えを予定しています。

「すぐ見えた」は「確定した」ではない

第8章の`latest`、`safe`、`finalized`を覚えていますか。Baseのノードはこう答えました。

聞いたもの	最新ブロックからの遅れ
`latest`	—
`safe`	約17秒
`finalized`	約19分

ブロックは1秒もかからずに現れますが、確定するのは、その記録がEthereum自体で確定したときです。

2階の手数料

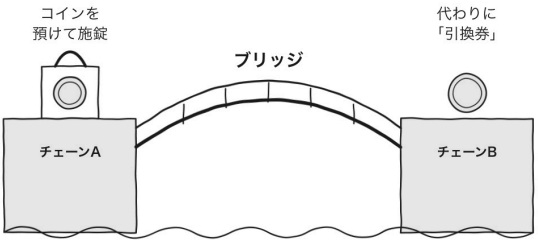
主に2つの部分からできています。L2自体での仕事の代金と、Ethereumへの書き込み代です。細かい追加料金があることもあります。Baseのレシートでは、2つ目は `l1Fee` フィールドに入っています。

```
curl https://public-base-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xa1d4398ce62b570a2810ad2b2ed347f388f9ad85aee4b62ba3215917cf3ac7e5"]}'
```

この例では、Baseでの仕事に約0.0000035 ETH、Ethereumへの書き込みに約0.0000000006 ETHが取られました。いつもこれほど安いとは限りません。

ブリッジ

ネットワークから別のネットワークへ「引っ越す」には、ブリッジを使います。多くの場合、ブリッジは一方のネットワークでコインを預かって鍵をかけ、もう一方で代わりのコインを発行します。



引換券の価値はブリッジの信頼しだい

そのため、USDCとUSDC.eは、コントラクトアドレスの違う別のトークンです。前者はCircle社が自ら発行し、後者はブリッジを通じて生まれます。

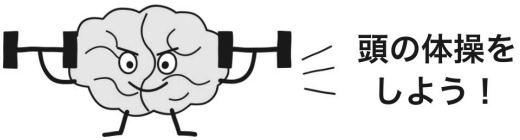
落とし穴：違うネットワークで送ってしまった

第6章で、お店にはそういう支払いが見えないと言いました。それは本当です。では、お金は失われたのでしょうか。

- アドレスの鍵を自分で持っている：ふつうは取り出せます。ウォレットをそのネットワークに切り替えてください。送り直すには、手数料としてそのネットワークのコインが必要です。
- 取引所のアドレスだった：鍵はあなたにありません。取引所がそのネットワークに対応していなければ、おそらく戻りません。
- コントラクト型のウォレットだった：別のネットワークには存在しないかもしれません。詳しい人の助けが必要です。

ルール：シークレットフレーズは誰にも教えない。送金を「取り戻してあげる」と言う人にも（第6章）。

考えてみよう



1. お店はEthereumでUSDTを待っていましたが、自分のアドレスにBNB Smart Chainで届きました。鍵はアルチョムが持っています。どうしますか。
2. Baseのブロックは0.2秒で現れました。大きな支払いは確定したと考えてよいでしょうか。
3. お客さんがUSDCの代わりにUSDC.eを送ってきました。同じものでしょうか。

答え：1）ウォレットをBNB Smart Chainに切り替えます。送り直すにはBNBが必要です。2）いいえ、finalizedを待ちます。3）いいえ、別のトークンです。受け付けるかはお店が決めます。

まとめ

- EVMネットワークは似ていますが、別々のノートです。chain IDを確かめます。
- 独立型のネットワークと、Ethereumの上の2階があります。
- 速いブロックは、まだ確定したブロックではありません。

次へ：第17章。TRON：ガスの代わりのエネルギーと、ステーブルコインのお気に入りの道。

第17章　TRON：ガスの代わりのエネルギーと、ステーブルコインのお気に入りの道

「TRC-20でいいですか？」

お客さんが「TRC-20のUSDTは受け付けていますか」と聞いてきます。アルチョムがこれを聞くのは、今週もう3度目です。TRC-20とは何で、なぜみんなこれを頼むのでしょう。

解説：独立したネットワーク

TRONは、独自のコインTRXを持つ別のブロックチェーンです。いちばん小さな単位をsunと呼び、1 TRXは100万sunです。

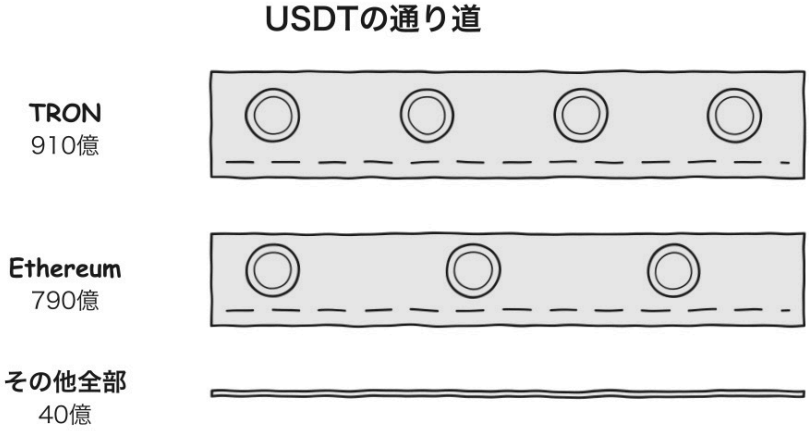
TRONの仕組みはEthereumによく似ています。アカウントの残高、スマートコントラクト、イベントがあります。プログラムを実行するのは独自の仮想マシンで、第15章のEVMの近い親戚です。

TRC-20は、TRONでトークンを作るときの決まりで、EthereumのERC-20規格の兄弟です。「TRC-20のUSDT」とは、TRONネットワーク上のUSDTトークンという意味です。

ブロックは3秒ごとに出ます。書くのは、選挙で選ばれた27人の代表です（第8章）。

ステーブルコインのお気に入りの道

USDTは多くのネットワークにありますが、その分布はとても偏っています。これは、このコインを発行するTether社自身の数字で、2026年11月27日に閲覧しました。



全USDTの半分以上がTRONにあります。お客さんがこのネットワークをよく選ぶのは、そのためです。

出典：app.tether.to/transparency.json。発行されたコインから、まだ流通していない在庫を引いて数えています。

アドレスはTで始まる

内部では、TRONのアドレスはEthereumと同じ20バイトで、先頭に `41` が付きます。人に見せるときは別の形になり、`T` の文字で始まります。

人が目にする形	内部の形（USDTのコントラクトアドレス）
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

`T` で始まる表記にはチェックサムが隠れていて、打ち間違いに気づけます。本物のアドレスについてノードに聞き、次に最後の文字を変えてみます。

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t","visible":true}'

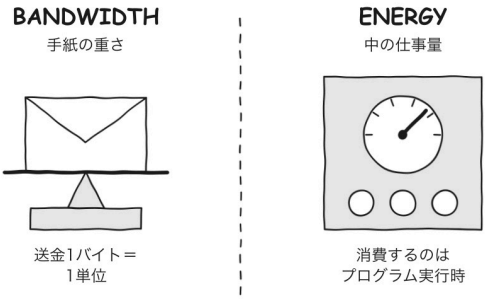
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

```
{"result":true,"message":"Base58check format"}
{"result":false,"message":"Invalid address"}
```

`"visible":true` は「アドレスはTの付く形で書きます」という意味です。

ガスの代わりに2つの支払い

第7章で、TRONが送金の「重さ」をどう測るかを話すと約束しました。TRONは2つの物差しで測ります。



- Bandwidth（帯域幅）は場所代です。消費量は、送金データのバイト数とほぼ同じです。どんな送金も払います。

- Energy（エネルギー）はプログラムの仕事代です。コントラクトを呼ぶとき、つまりUSDTの送金でも必要です。

一度でもTRXを受け取ったことのあるアドレスには、おまけがあります。1日600単位のbandwidthです。使った分は24時間かけて少しずつ回復します。無料のエネルギーはありません。

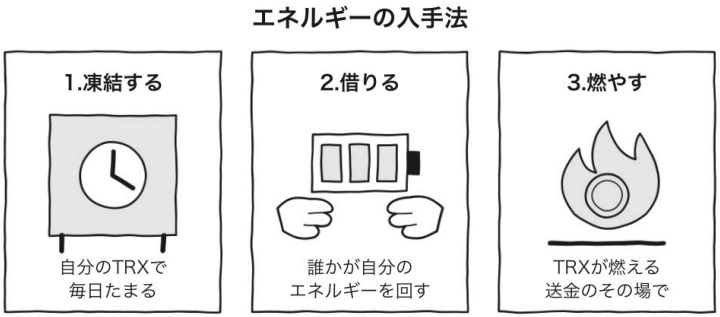
何かが足りないと、ネットワークはTRXを取って燃やします。いくらでしょう。ノードに自分で聞いてみます。

```
curl -s https://public-tron-mainnet.rpcnode.dev/wallet/getchainparameters |
  python3 -c "import json,sys
for p in json.load(sys.stdin)['chainParameter']:
    if p['key'] in ('getFreeNetLimit','getTransactionFee','getEnergyFee'):
        print(p['key'], p['value'])"
```

```
getTransactionFee 1000
getEnergyFee 100
getFreeNetLimit 600
```

つまりbandwidth 1単位は1000 sun、エネルギー1単位は100 sunです。この値段は代表の投票で変わります。

エネルギーはどこから手に入れるか

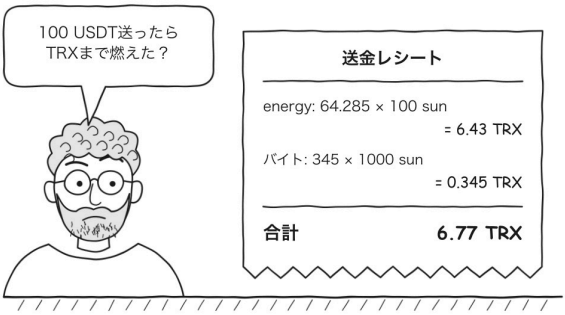


- TRXを凍結する（英語でstaking）。コインはあなたのもののままで、アドレスは毎日、全体のエネルギーの分け前を受け取ります。取り戻すには、凍結解除のあと14日待ちます。
- 借りる：凍結したTRXの持ち主が、自分のエネルギーを別のアドレスに回します。
- 何もしない：足りない分は、燃やされるTRXで支払われます。

どれくらい凍結すればよいでしょう。それは、ほかのみんながどれだけ凍結しているかによります。確認した日は、1日1件のUSDT送金に、凍結TRX約6600枚で足りる計算でした。

本物の送金のレシート

ブロック88,352,467の100 USDTの送金です。送り主には、エネルギーも無料のbandwidthもありませんでした。



レシートは `gettransactioninfobyid` メソッドで取ります。公開ノードでは使えず、個人用エンドポイントが必要なので、これは雛形です。

```
curl https://あなたのエンドポイント/walletsolidity/gettransactioninfobyid \
  -d '{"value":"0eb7b372f5a268ce56430392de8db2fd970455bb0970c67b47a7b2673abb513c"}'
```

答えのフィールド	値	意味
`energy_usage_total`	64,285	使ったエネルギーの合計
`energy_penalty_total`	49,635	うち、人気の高さへの割増
`energy_fee`	6,428,500	エネルギー代として燃やされた分（sun）
`net_fee`	345,000	バイト代として燃やされた分（sun）
`fee`	6,773,500	合計。6.7735 TRX
`result`	`SUCCESS`	プログラムは正常に終わった

割増とは何でしょう。一つのコントラクトが使われすぎると、ネットワークはその呼び出しを高くします。この日のUSDTは、割増がない場合のほぼ4.4倍のエネルギーを使っていました。

さらに、受取人がそれまでUSDTを持っていなかった場合、エネルギーは2倍の130,285が必要です。コントラクトが、その人のために新しい行を作らなければならないからです。

新しいブロックと確定したブロック

TRONのノードには2組のアドレスがあります。`/wallet/…` はいちばん新しいデータで答えます。`/walletsolidity/…` は確定したデータだけで答えます。ブロックは、その高さ以上で、27人の代表のうち19人が自分のブロックを出すと確定します。ふつうは約1分です。

確認した日、最新はブロック88,352,763、確定していたのは88,352,746でした。

実験用にはテストネットのNileがあり、専用の公開ノード `public-tron-nile.rpcnode.dev` があります。

落とし穴：「ブロックにあるなら届いた」

確定した最新ブロックにある、USDTの直接の送金を見てみます。

```
curl -s https://public-tron-mainnet.rpcnode.dev/walletsolidity/getnowblock > block.json
python3 <<'EOF'
import json
usdt = '41a614f803b6fd780986a42c78ec9c7f77e6ded13c'
rows = []
for t in json.load(open('block.json'))['transactions']:
    v = t['raw_data']['contract'][0]['parameter']['value']
    data = v.get('data', '')
    if v.get('contract_address') == usdt and data.startswith('a9059cbb'):
        amount = int(data[72:136], 16) / 1_000_000
        rows.append((t['txID'][:8], t['ret'][0]['contractRet'], amount))
print('USDTの送金:', len(rows))
print('不成立:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

数字は人によって違います。ブロック88,352,467では、そういう送金41件のうち2件が成立しませんでした。その一つは100,000 USDTです。

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

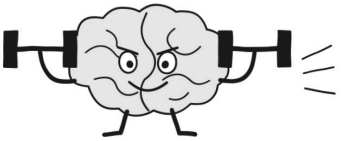
`OUT_OF_ENERGY` はエネルギー不足でプログラムが止まり、USDTはどこへも動かなかったという意味です。それでもブロックには記録があり、画面の写真では立派な金額に見えます。

使ってよいエネルギーの量は、送り主が `fee_limit` フィールド（sunでの上限）で決めます。上限が足りなければ送金は成立せず、燃やされた分は戻りません。

お店が覚えておくこと

- 確定したブロックを待ち、レシートの `result` を確かめる。
- レシートの中で、コントラクト `TR7NHq…Lj6t` の `Transfer` イベントを探す。そうすれば、ほかのプログラムから行われた送金も見えます。USDTは小数点以下6桁です。
- 自分のアドレスからUSDTを送り出すには、そこにTRXかエネルギーが必要です。注文ごとのアドレスは便利ですが、100のアドレスからお金を集めるのは、手数料のかかる送金100回です。
- 新しいアドレスへの最初のTRX送金は、送り主にとって少なくとも1 TRX高くつきます。

考えてみよう



頭の体操をしよう！

1. アドレスに無料のbandwidthが600と、たくさんのエネルギーがあります。345の送金を続けて2件送ると、2件目はどうなりますか。
2. エネルギーはなく、受取人はまだUSDTを持ったことがありません。1単位100 sunで130,285単位が必要です。エネルギー代に何TRX燃えますか。
3. お客さんがハッシュを送ってきました。トランザクションはブロックにありますが、`OUT_OF_ENERGY`です。コーヒーを発送しますか。

答え：1）無料の残りは255で、345には足りません。345の全部に0.345 TRXが燃えます。無料分と分けて足し合わせることはできません。2）13.0285 TRX。3）いいえ、送金は行われていません。

まとめ

- TRONには2つの物差しがあります。バイトのbandwidthと、プログラムの仕事のenergyです。
- エネルギーがなければTRXが燃えます。USDTは、単純な送金よりかなり多くかかります。
- 支払いとは、本物のUSDTが、確定したブロックで、送金に成功したことです。

次へ：第18章。Solana：すべてがアカウントに分けて置かれている。

第18章　Solana：すべてはアカウントに分けて置く

「私のUSDCはどこ？」

お客さんがUSDCで支払いました。アルチョムが自分のウォレットの残高をノードに聞くと、わずかなSOLしか見えません。USDCはどこにもありません。だまされたのでしょうか。

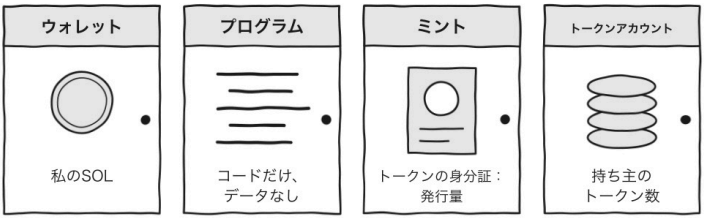
解説：アカウントでできたネットワーク

Solanaは、コインSOLを持つ独立したネットワークです。いちばん小さな単位はラムポートで、1 SOLは10億ラムポートです。

Solanaのアドレスは、32バイトを文字と数字で書いたものです。決まった先頭文字はなく（第6章）、内部にチェックサムもありません。

ネットワークが保存するものは、すべてアカウント（accounts）に入っています。ここでのアカウントはお金の口座に限らず、アドレスを持つあらゆる「入れ物」です。Ethereumではコントラクトがデータを自分の中に持ちます。Solanaではプログラムとデータが別々に置かれます。

4つのアカウント、全部別物



英語：wallet、program、mint account、token account

どのアカウントにも、持ち主のプログラム（`owner`フィールド）があります。アカウントのデータを変えられるのは、そのプログラムだけです。

やってみよう：トークンの身分証

どのトークンにも、身分証にあたるアカウントがあり、mintと呼びます。USDCの身分証を聞いてみます。

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getAccountInfo","params":
  ["EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",{"encoding":"jsonParsed"}]}'
```

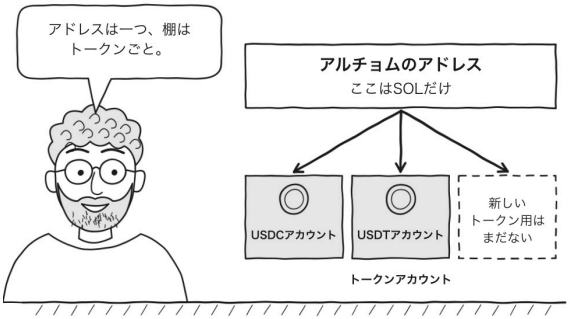
答えの要点はこうです。

フィールド	値	意味
`owner`	`Tokenkeg…Q5DA`	このアカウントはトークンプログラムが管理している
`supply`	`8513455604386747`	発行量（最小単位で）
`decimals`	6	つまり約85億USDC
`mintAuthority`	`BJE5…5ruG`	新しく発行できる者
`freezeAuthority`	`7dGb…Crar`	凍結できる者（第9章）

USDCには自分専用のプログラムがありません。一つの共通のトークンプログラムが何千ものトークンを受け持ち、トークンは身分証で区別されます。この共通プログラムは2つあります。ふつうのToken（USDCとUSDTはこちら）と、追加機能を持つ新しいToken-2022です。

トークンアカウント：専用の棚

アルチョムがUSDCをいくら持っているかは、ウォレットにも身分証にも書かれず、トークンアカウントに書かれています。トークンごとに専用のものがあります。



基本となるトークンアカウントをATA（associated token account）と呼びます。そのアドレスは3つから計算されます。ウォレットのアドレス、トークンの身分証、トークンプログラムです。だからお客さんにはウォレットのアドレスを渡せばよく、棚はお客さんのプログラムが自分で見つけます。

本物のトークンアカウントを見てみます。まずトークンがいくらあるかを、次にSOLがいくらあるかを聞きます。

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTokenAccountBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'

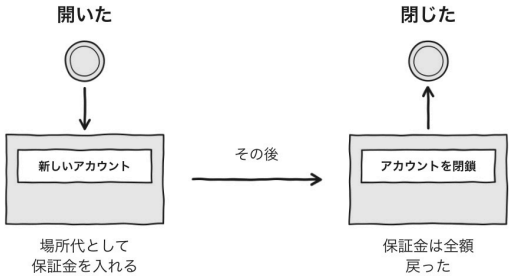
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

1つ目の答えの`"uiAmountString":"207637.336362"`はUSDTです。2つ目の`"value":2039280`はラムポートで、わずか0.00203928 SOLです。`getBalance`メソッドにはトークンが見えません。アルチョムのUSDCが「消えた」先はここでした。

ちなみに、この棚の持ち主はUSDTの身分証そのもののアドレスです。第9章のように、誤って「USDTのアドレス」へUSDTを送った人たちがいたようです。

場所の保証金

どのアカウントも、すべてのノードのメモリーを使います。そのためアカウントには、SOLの保証金が置かれます（英語ではrent、「家賃」ですが、家賃ではありません）。



手数料ではなく保証金：トークンアカウント1つで約0.0015 SOL

トークンアカウントの大きさは165バイトです。それだけの場所にはいくらの保証金が必要でしょう。

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
  "method":"getMinimumBalanceForRentExemption","params":[165]}'
```

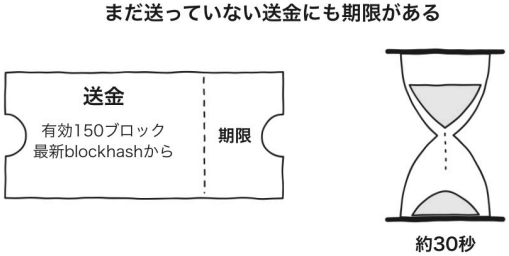
```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

約0.0015 SOLです。2026年には保証金が段階的に引き下げられているので、プログラムに数字を書き込まず、そのつど聞きます。受取人にまだそのトークンの棚がない場合、ふつうは送り主が同じ送金の中で棚を作り、保証金も自分で入れます。

手数料と有効期限

基本手数料は署名1つにつき5000ラムポートです。署名が1つのふつうの送金なら0.000005 SOLです。半分は燃やされます。急ぎの追加料金を上乗せでき、こちらは全額バリデーターが受け取ります。手数料は、送金が失敗しても取られます。

どの送金にもblockhash、つまり最近のブロックの指紋が入ります。これが有効期限も決めます。



ブロックに間に合わなければ永久に入らない

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getLatestBlockhash",
  "params":[{"commitment":"finalized"}]}'

curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockHeight",
  "params":[{"commitment":"finalized"}]}'
```

1つ目の答えは`"lastValidBlockHeight":452330099`、1秒後の2つ目の答えの高さは452,329,960でした。残りは139ブロックです。期限が切れてもブロックに送金がなければ、もう通りません。そのときは新しい送金に署名できます。例外は特別な遅延送金（durable nonce）で、こうした期限がありません。

2026年：さらに速く

変わったこと	以前	現在
スロット（1ブロック分の順番）の長さ（10月）	0.4秒	0.2秒
ファイナリティ。新しい投票手順Alpenglow（10月）	約13秒	約0.15秒
トランザクションの最大サイズ。新形式v1（9月）	1232バイト	4096バイト

速さを自分で確かめます。次のメソッドは、1分間に何スロット進んだかを返します。

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getRecentPerformanceSamples","params":[1]}'
```

私たちの場合は60秒で `numSlots` = 277、1スロットおよそ0.22秒でした。スロットは空のこともあるので、スロット番号とブロックの高さは別の数です。混同しないでください。

ノードは、いくつかの厳しさで答えられます。`processed` はブロックを確認したが、まだ飛ばされる可能性がある状態、`finalized` は確定です。中間の `confirmed` という言葉もありますが、Alpenglow以後は `finalized` と同じ意味です。注文の発送は `finalized` を基準にします。

落とし穴：「トランザクションが読めない」

新形式v1のため、`getTransaction` と `getBlock` メソッドには、値を1にした `maxSupportedTransactionVersion` を加える必要があります。これがないと、新しいトランザクションではエラーが返ります。`getTransaction` は公開ノードでは使えないので、これは個人用エンドポイント向けの雛形です。

```
curl https://あなたのエンドポイント \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTransaction","params":
  ["送金の署名",{"encoding":"jsonParsed","commitment":"finalized",
  "maxSupportedTransactionVersion":1}]}'
```

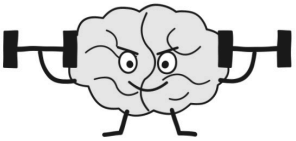
Solanaでは「送金の署名」が、トランザクションハッシュの代わりに送金の番号の役目をします。

お店が確かめること

- トランザクションが成功した：`meta.err` が `null`。
- 目的のトークンの身分証を持つ、あなたのウォレットのトークンアカウントが、正しい金額だけ増えた。`preTokenBalances` と `postTokenBalances` を比べます。
- それがこの注文の支払いである。そのために送金にreference（注文用の使い捨ての目印アドレス）を加えます。
- この署名をまだ支払いに数えていない。

ふつうのSOLには特に注意してください。間違ったアドレスへの送金でも実行されてしまいます。そのお金を返してくれる相手は、おそらくいません。

考えてみよう



頭の体操をしよう！

1. ウォレットが送金に署名しましたが、ネットワークがそれを受け取ったのは2分後でした。どうなりますか。
2. お店はUSDCの支払いを捕まえようと、自分のウォレットに`getBalance`を聞いています。何が間違っているでしょう。
3. アルチョムは、まだその棚を持たない仕入れ先にUSDCを送ります。手数料のほかに何を入れる必要がありますか。

答え：1) 期限切れで、送金は通りません。2) `getBalance`が見せるのはSOLだけで、USDCはトークンアカウントにあります。3) 新しいトークンアカウントの保証金、約0.0015 SOLです。

まとめ

- Solanaではすべてがアカウントで、プログラムとそのデータは別々に置かれます。
- トークンはトークンアカウントにあります。トークンごとに一つで、保証金が必要です。
- トランザクションは`maxSupportedTransactionVersion`を付けて読み、`finalized`を待ちます。

次へ：第19章。TON：電話ではなく手紙。

第19章　TON：電話ではなく手紙

「送りました。で、このあとは？」

お客さんがTONネットワークでUSDTを送りました。エクスプローラーを見ると、記録は一つではなく長い連なりです。5つのトランザクション、見知らぬアドレス、小さなおつり。これは支払い1件でしょうか、5件でしょうか。

解説：全員がコントラクトで、全員が手紙を書く

TON（The Open Network）は独立したネットワークです。そのコインはGram（GRAM）といいます。以前はToncoinという名前で、今もこの名をよく見かけます。1 Gramは10億ナノグラムです。

2026年4月から、ブロックはおよそ0.4秒に1回出て、記録は約1秒で確定します。

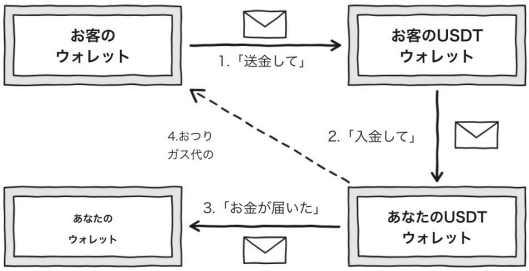
いちばんの違いはこれです。ここではどのアカウントもスマートコントラクトで、ウォレットもそうです。コントラクトは互いを直接呼び出さず、メッセージを送り合います。

Ethereumでは、一つの送金が引き起こすことはすべて一つのトランザクションの中で起き、全部成功するか、全部なかったことになるかです。これは電話の会話のようなものです。TONは文通です。手紙を出したら、電話口で返事を待ったりしません。受け取った側は、一通ごとに後で、別のトランザクションとして処理します。こうしたやり方を非同期と呼びます。

一つの依頼から生まれた文通の全体をtrace（トレース、足跡）と呼びます。その中の一つのトランザクションが成功しても、用件全体がうまくいったとはまだ言えません。

USDTの運ばれ方

TONのトークンはjetton（ジェットン）と呼びます。それぞれに親となるコントラクト（master）があり、持ち主ごとに専用のjettonウォレットがあります。第18章のトークンアカウントに似ていますが、こちらは独立したコントラクトです。USDTの親コントラクトは`EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`です。



手紙は一通ずつ別のトランザクションで処理

これは作り話ではなく、2026年11月27日の夜に行われた11.99 USDTの本物の送金です。

誰のトランザクションか	受け取ったもの	次に送ったもの
送り主のウォレット	署名された依頼	「送金して」と、経費の0.05 Gram
送り主のUSDTウォレット	「送金して」	「入金して」
受取人のUSDTウォレット	「入金して」	通知とおつり
受取人のウォレット	通知と1ナノグラム	なし
送り主のウォレット	おつり、約0.0494 Gram	なし

すべては1秒で終わりました。経費のお金は多めに添え、余りは戻ります。

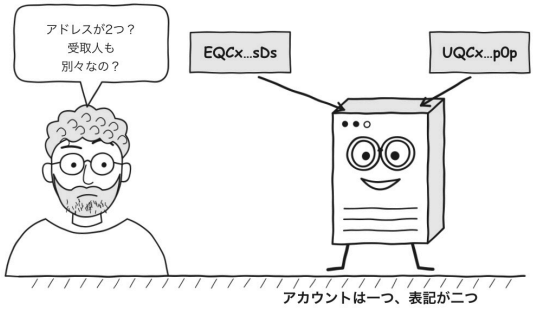
面白い細部があります。4番目のトランザクションには、中断（`aborted`）の印が付いています。届いたのがたった1ナノグラムだったので、受取人側でプログラムを実行するガスが足りなかったのです。それでもUSDTは、3番目のトランザクションですでに入金されています。記録一つだけで判断することは、成功にも失敗にもできません。

連なり全体は、インデクサーへのリクエストで見られます。公開ノードでは使えないので、これは個人用エンドポイント向けの雛形です。

```
curl -G https://あなたのエンドポイント/api/v3/traces \
  --data-urlencode "trace_id=7rg6DNoAZeR1KPVJHd1FWIc/dobTITAsXqK1Y2kuQlM="
```

アカウントは一つ、アドレスは二つ

アカウントの本当の名前は「生の形式」で、ネットワークの区画番号と64文字です。人には、チェックサム付きの48文字の別の表記を見せます。しかも一つのアカウントに、そういう表記が複数あります。



USDTのアドレスについてノードに聞いてみます。

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs" |
  python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['raw_form'])
print(r['bounceable']['b64url'])
print(r['non_bounceable']['b64url'])"
```

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…` は戻り型（bounceable）のアドレスで、ふつうのコントラクトはこう書きます。
- `UQ…` は非戻り型で、ウォレットはこう書くことが勧められています。
- テストネットの表記は `kQ` と `0Q` で始まります。

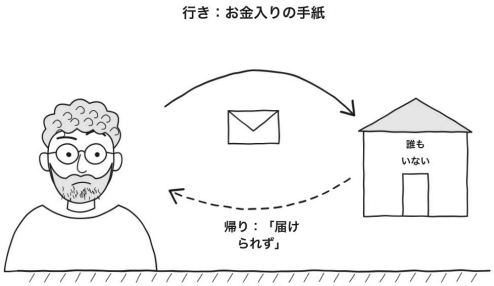
そのためお店のデータベースでは生の形式を比べ、`EQ` や `UQ` の付いた文字列は比べません。

チェックサムは打ち間違いを捕まえます。末尾の `sDs` を `sDt` と書くと、ノードはコード422で答えます。

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDt"
```

空き家への手紙

受取人が手紙を処理できなかったり、コントラクトがまだ存在しなかったりしたら、どうなるでしょう。それはメッセージの印で決まります。



戻り型（bounceable）メッセージの動き

戻り型のメッセージは、帰り道の費用が足りれば、残りのお金とともに送り主へ戻ります。非戻り型は受取人のもとに残ります。新しいウォレットは、まだ「展開」されていない、つまりプログラムがネットワークに書き込まれていないことがあります。だからウォレットへの支払いには、非戻り型のアドレスを使います。

やってみよう：速さを確かめる

最新のメインブロックの番号と、そのブロックと1000個前のブロックが出た時刻を調べます。

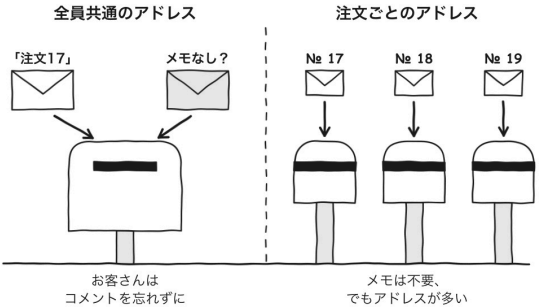
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s" |
    python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['id']['seqno'], r['gen_utime'])"
done
```

```
108011078 1795732480
108010078 1795732076
```

2列目の数は秒で表した時刻です。1000ブロックで404秒の差、つまり1ブロックおよそ0.4秒です。

別々のアドレスの代わりにコメント

TONの送金には、文字のコメントを添えられます。そこから、誰の支払いかを知る方法が2つ生まれます。



共通のアドレスは手間が少ない代わりに、お客さんがコメントを忘れたり、書き写し間違えたりします。しかも、コメントを付けられないウォレットもあります。そういう支払いは手作業で整理します。

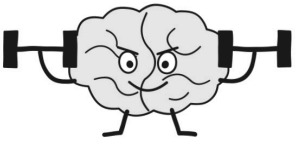
落とし穴：「通知が来たから発送する」

あなたのウォレットに同じような通知を送るコントラクトは、誰でも書けます。「USDT」という名前の自前のjettonも発行できます。だから次を確かめます。

- 通知があなたのUSDTウォレットから来たこと。そのアドレスは、あらかじめ本物の親コントラクトに聞いておきます（`get_wallet_address`）。
- 金額が一致すること（USDTは小数点以下6桁）。
- コメントかアドレスがこの注文のもので、この支払いをまだ数えていないこと。
- 記録がメインチェーン（masterchain）に入った、つまり確定していること。

通知が届くのは、送り主がそのために少しでもGramを添えた場合だけです。そうするのが一般的ですが、自作のプログラムは忘れることもあります。その場合、トークンは通知なしで届きます。

考えてみよう



1. データベースには `UQ` で始まるアドレスがあり、エクスプローラーには中間部分が同じ `EQ` のアドレスが出ています。同じアカウントかどうか、どう確かめますか。
2. 連なりの最初のトランザクションは成功しました。では、USDTは届いたのでしょうか。
3. お客さんは共通のアドレスに支払いましたが、コメントを忘れました。お店はどうすべきでしょう。

答え：1) 両方を生の形式に直して (`detectAddress`) 比べます。2) まだです。受取人のUSDTウォレットが入金する必要があります。3) 自動では数えず、金額、送り主、送金番号を手作業で調べます。

まとめ

- TONではコントラクトが文通し、一つの依頼がトランザクションの連なりを生みます。
- 一つのアカウントにアドレスの表記が複数あるので、生の形式を比べます。
- jettonは、親コントラクトと自分のjettonウォレットで確かめます。

次へ：第20章。そのほかのやり方：SuiとAptos、XRP LedgerとStellar、Cardano、Polkadot、Cosmos。

第20章　別のやり方：SuiとAptos、XRP Ledgerと Stellar、Cardano、Polkadot、Cosmos

「XLMは受け付けますか？」

あるお客さんは、Stellarで払えるかと聞きます。別のお客さんは、SuiでUSDCを送りたいと言います。アルチョムはもうBitcoinとEthereumを理解しました。その知識で足りるでしょうか。十分とは言えません。ここには独自の流儀があります。

解説：新しいネットワークへの5つの質問

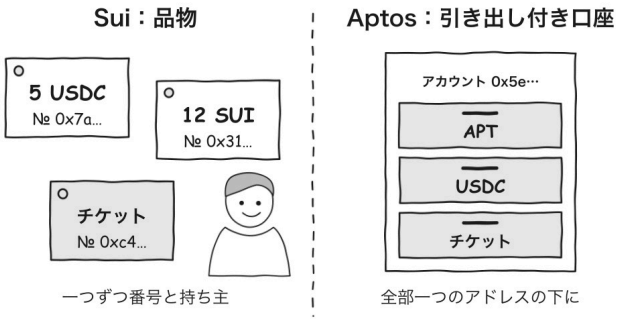
知らないネットワークを丸ごと勉強する必要はありません。まずは5つの答えがあれば十分です。

1. お金はどこにあるか。口座の残高か、ばらばらのかけらか。
2. そもそも送金を受け取るために、前もって何をしておくか。
3. どの注文の支払いか、どうやって見分けるか。
4. いつ確定するか。
5. お金をさらに送り出すとき、手数料は何で払うか。

SuiとAptos：言葉は共通、仕組みは別

どちらのネットワークも、プログラムはMove言語で書きます。この言語は、大事なものが簡単にコピーされないように見張ります。ただし、お金の持ち方は違います。

- Suiはすべてをオブジェクトとして持ちます。一つひとつ番号のある「品物」です。お金は別々のコインオブジェクトとして置かれることがあり、Bitcoinのかけらにかなり似ています。2026年からは、アドレスに合算の残高があることもあります。全残高は、この2つの合計です。
- Aptosは、なじみのある口座に近い形です。アドレスの下に持ち物が積まれていて、これをリソースと呼びます。事前に口座を開く必要はありません。



どちらも、送金はふつう1秒かからずに確定します。手数料は自分のコイン、SUIかAPTで払います。Suiには例外があり、2026年から、USDCを含むいくつかのステーブルコインは、手数料なしでそのまま送れます（0.01以上の場合）。

どちらのアドレスも `0x` と、ふつう64文字です。先頭はEthereumと同じでも長さが違い、あちらは `0x` のあとが40文字です。

やってみよう：2つのネットワークでUSDCを探す

この章のネットワークには、RpcNodeの無料公開ノードはありません。そこで、ネットワークを開発している側が公開しているアドレスに聞きます。みんなで使うもので制限があり、お試し向きで、お店には向きません。

SuiはGraphQLという方式です（必要なフィールドを自分で並べます）。

```
curl -s https://graphql.mainnet.sui.io/graphql \
  -H 'Content-Type: application/json' \
  -d '{"query":"{ coinMetadata(coinType: \"0xdba34672e30cb065b1f93e3ab55318768fd6fef66c15942c9f7cb846e2f900e7::usdc::USDC\") { symbol decimals } }"}'
```

```
{"data":{"coinMetadata":{"symbol":"USDC","decimals":6}}}
```

Aptosは、第11章のふつうのRESTです。

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00aac12edcdda169e591cd41c94180b46f3b/resource/0x1::fungible_asset::Metadata
```

```
{"type":"0x1::fungible_asset::Metadata","data":{"decimals":6,"icon_uri":"https://circle.com/usdc-icon","name":"USDC","project_uri":"https://circle.com/usdc","symbol":"USDC"}}
```

どちらでも、トークンは「USDC」という言葉ではなく、発行元であるCircle社の一覧にある長いアドレスで探します。第9章のルールです。小数点以下は、どちらも6桁です。

古い例には注意してください。以前、SuiにはJSON-RPCで聞いていました。2026年7月、Sui財団のノードではそれが停止され、今はgRPCとGraphQLです。記事の例をコピーする前に、日付を確かめてください。

XRP LedgerとStellar：送金のネットワーク

どちらも昔から、自分のコインを送るだけでなく、ほかのトークンも扱えます。ここではチェーンの1ページをledger（レジャー）と呼びます。XRP Ledgerは3〜5秒に1回、Stellarはおよそ5秒に1回、新しいページを承認します。承認されたものは書き換えられません。

ただし、なじみのないルールが2つあります。

1つ目：場所の保証金。最低限の金額、つまり準備金が置かれるまで、口座は存在しません。そしてUSDCのようなトークンを受け取るには、口座の持ち主があらかじめトラストライン（trust line）を開きます。「はい、これを持つことに同意します」という宣言です。こうした記録が一つ増えるごとに、準備金も増えます。

	XRP Ledger	Stellar
アドレスの先頭	`r`	`G`
口座の最低額	1 XRP	1 XLM
trust line 1つあたりの追加	0.2 XRP（最初の2つは不要な場合がある）	0.5 XLM
最低手数料	0.00001 XRP	操作1件につき0.00001 XLM
注文番号	destination tag	memo、または `M` で始まるアドレス

これらの数はネットワークの投票で変わることがあるので、ノードに聞きます。

```
curl -s https://s1.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"server_info","params":[{}]}' | grep -o '"validated_ledger":{[^}]*}'
```

```
"validated_ledger":{"age":10,"hash":"AD65FEE0E665A2CA5D08D3757A9E5A65047ECC2391B5E118F3912F25C76AB64F","seq":10859
3026,"base_fee_xrp":1E-5,"reserve_base_xrp":1E0,"reserve_inc_xrp":2E-1}
```

`1E-5` は0.00001、`1E0` は1、`2E-1` は0.2のことです。次はStellarです。

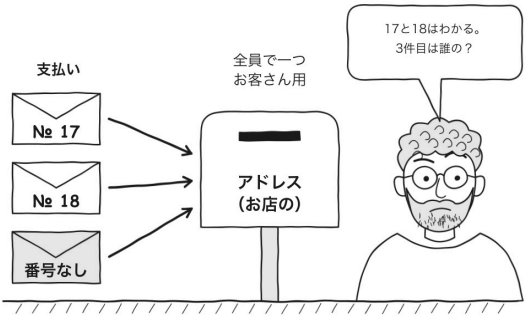
```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

```
        "base_fee_in_stroops": 100,
        "base_reserve_in_stroops": 5000000,
```

1 XLMは1000万stroops（ストループ）です。つまり手数料は0.00001 XLM、準備金の1単位は0.5 XLMです（口座には最低2単位必要です）。

全員で一つのアドレス

2つ目のルール。ここでは注文ごとに別のアドレスを作るのは高くつきます。それぞれに準備金を置くことになるからです。そのためお店は一つのアドレスを示し、番号を添えてもらいます。XRP Ledgerではdestination tag、Stellarではmemoです。

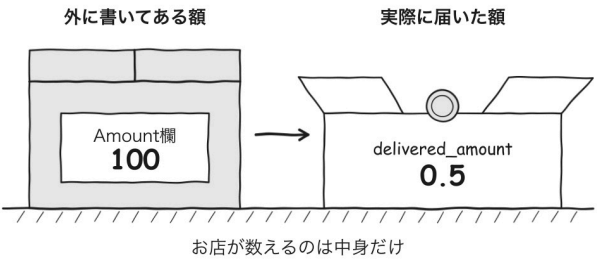


お客さんが番号を忘れると、お金は届いても、何の支払いかわかりません。役立つ対策はこうです。

- XRP Ledgerでは`RequireDest`という設定を有効にできます。番号のない送金は、ネットワークが通しません。
- 番号はアドレスそのものに埋め込めます。XRP LedgerのXアドレス、Stellarの`M`で始まるアドレスです。お客さんのウォレットが対応していればの話です。

落とし穴：ラベルは100、中身は0.5

XRP Ledgerでは、送る側が部分支払い（partial payment）を許可できます。届いた額が少なくても、送金は成功とみなされます。そのとき`Amount`フィールドは「届いた額」ではなく「最大の額」を意味します。実際に届いた額は`delivered_amount`に書かれます。



これはネットワークにある本物の記録です。ハッシュを指定して聞いてみます。

```
curl -s https://s2.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"tx","params":[{"transaction":"18F797F64D0255522223B1A404BA9CFF7EAE28E2ABB74CC91DD986E162D42DE
C"}]}' \
  | grep -o '"Amount":{[^}]*}\|"delivered_amount":{[^}]*}\|"TransactionResult":"[^"]*"'
```

```
"Amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"9.552474017069"}
"TransactionResult":"tesSUCCESS"
"delivered_amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"4.5344917829"}
```

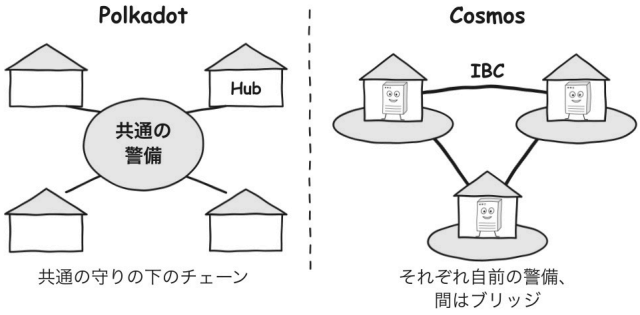
結果は`tesSUCCESS`で、`Amount`は約9.55なのに、届いたのは約4.53です。この記録そのものは不正ではありません。しかし詐欺師は、大きな金額を書いてごくわずかだけ送り、商品を待つことができます。数えるのは`delivered_amount`だけにして、どのトークンか、どの発行元かを確かめてください。

Cardano：同封物のあるかけら

CardanoはBitcoin（第14章）に近い作りです。残高ではなく、未使用のかけらです。ただし、かけらにできることが多く、この方式は拡張UTXO（eUTXO）と呼ばれます。

- かけらの中には、ADAコインと並んで別のトークンも入ります。トークン用の別のコントラクトはいりません。
- トークンは単独では旅をしません。同じかけらに少しのADAが必ず入ります。だからトークンと一緒に、ほんの少しのコインも届きます。
- 手数料は、送金データのバイト数で決まります。
- 確かさは少しずつ増します。上のブロックが多いほど信頼できます。

PolkadotとCosmos：一つではなく、たくさんのネットワーク



Polkadotは、接続されたネットワークを守る中央のチェーンです。大事な変化があります。2025年11月から、利用者の残高は中央のチェーンではなく、Polkadot Hub（Asset Hubとも呼ばれます）というネットワークにあります。

- USDT（資産番号1984）とUSDC（番号1337）もそこにあります。

- 口座には生存の下限、existential depositがあります。0.01 DOTです。DOTも、必要な量のUSDTやUSDCも足りないと、口座は消されます。
- アドレスは、Bitcoinの古い形式と同じく `1` で始まります。混同しないでください。

Cosmosは、共通の部品で作られた独立したネットワーク群です。それぞれに自前の警備、コイン、アドレスの接頭辞があります（Cosmos Hubは `cosmos1…`）。ブロックはすぐに確定します。送金には注文番号用のmemoフィールドがあります。

ネットワーク同士はIBCプロトコルでつながります。隣のネットワークから来たトークンは、なじみの名前ではなく、`ibc/` と長いはんこで呼ばれます。たとえば `ibc/7F1D3FCF…` です。はんこは、どの道を通って来たかを示します。

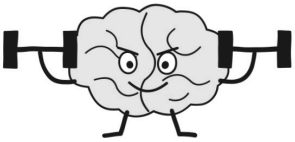
ステーブルコインが出ていくこともあります。Circle社は、Cosmos一族のネットワークNobleでのUSDC発行を終了すると発表しました。新規発行は2026年10月13日から止まり、2027年1月12日にはそこでのUSDCが停止されます。保有者には、事前に移すことが勧められています。

まとめて比べる

ネットワーク	アドレス	事前に必要なこと	確定まで
Sui	`0x…`	なし	1秒未満
Aptos	`0x…`	なし	1秒未満
XRP Ledger	`r…`	準備金、trust line	3〜5秒
Stellar	`G…`	準備金、trust line	約5秒
Cardano	`addr1…`	なし	少しずつ
Polkadot Hub	`1…`	下限を上回る残高	確定の印で
Cosmos Hub	`cosmos1…`	なし	すぐ

trust lineが必要なのはトークンだけです。2026年11月27日に確認しました。ルールは変わります。

考えてみよう



頭の体操をしよう！

1. アルチョムはStellarで新しい空のアドレスを作り、すぐにUSDCで請求しました。何がまずく、XLMは最低いくら必要ですか。
2. 送金が届きました。`Amount`は100 USDC、`delivered_amount`は0.5、結果は`tesSUCCESS`。100ぶんのコーヒーを発送しますか。
3. XRP Ledgerのお客さんがdestination tagを忘れます。何を有効にしますか。

答え：1）口座はまだ存在せず、trust lineも開いていません。準備金が最低1.5 XLMと、手数料ぶんが少し必要です。2）いいえ、届いたのは0.5です。3）RequireDest設定。

まとめ

- 知らないネットワークでは、お客さんにアドレスを見せる前に5つの質問をします。
- XRP LedgerとStellarには準備金、trust line、注文番号が必要です。XRP Ledgerでは`delivered_amount`だけを数えます。
- Ethereumの習慣や古い例は、別のネットワークには持ち込めません。

次へ：第21章。どのネットワークを選ぶか。

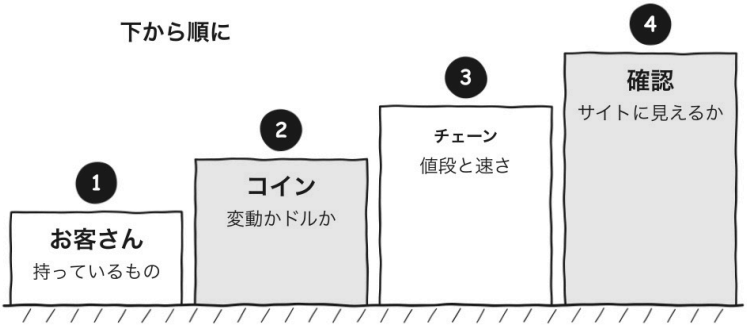
第21章　どのネットワークを選ぶか

「で、結局どれにすればいいの？」

7章のあいだに名前がたくさん出てきました。Bitcoin、Ethereum、TRON、Solana、TON、ほかに10ほど。アルチョムにあるのは、ネットショップ一軒とコーヒーの焙煎所一つです。全部つなぐ？　一つに絞る？　何を基準に？

解説：4つのステップ

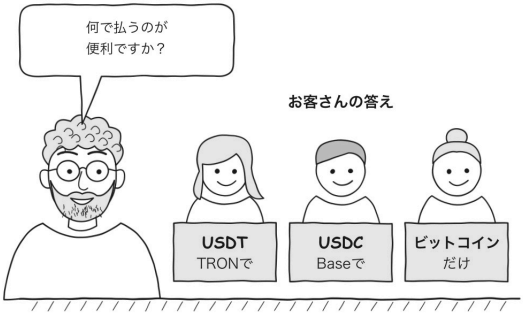
「最高のネットワーク」はありません。あるのは、あなたのお客さんと、あなたの注文に合うネットワークです。順番に進むと楽です。



ここで話すのは、何で支払いを受けると便利かです。何に投資すべきかの助言ではありません。

ステップ1：お客さんに聞く

支払うのは、すでにコインを持っている人です。その人のコインがあるネットワークと、あなたが受け付けるネットワークが違えば、おそらく支払いは来ません。



メールマガジンでも、チャットでも、注文のときでも聞けます。質問は2つで十分です。「何を持っていますか」と「どのネットワークですか」。

ステップ2：どのコインか

	ネットワーク固有のコイン	ステーブルコイン（USDT、USDC）
ドル建ての値段	変動する。ときに急激に	1ドル付近を保とうとする
主導権を持つのは	ネットワークのルール	発行元がいて、凍結できる（第9章）
手数料の支払いに使うもの	そのコイン自身	ふつうはステーブルコインではなく、ネットワークのコイン

アルチョムのコーヒーの値段がドル建てか、ドルに連動しているなら、ステーブルコインのほうが驚くことが少なくて済みます。ただしそれも信頼の問題です。発行する会社を信じることになります。

ステップ3：どのネットワークか

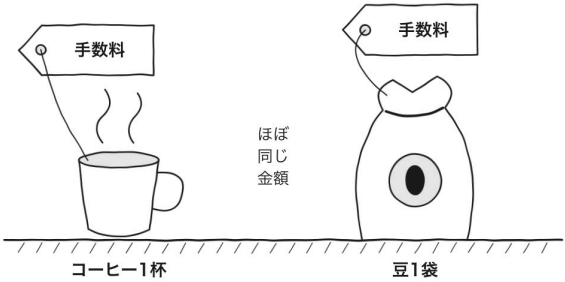
同じステーブルコインがどこにでもあるわけではありません。第14〜20章のネットワークを並べます。「あり」は、発行元のサイトに、そのネットワークでのまさにそのコインのアドレスが載っているという意味です。

ネットワーク	手数料	Tether社のUSDT	Circle社のUSDC	忘れないこと
Bitcoin	BTC	なし	なし	承認数
Litecoin、Dogeなど	自分のコイン	なし	なし	別々のネットワーク
Ethereum	ETH	あり	あり	ガス
Base、Arbitrum	ETH	なし	あり	finalized
BNB Smart Chain	BNB	なし	なし	発行元の確認
TRON	TRX	あり	なし	エネルギー
Solana	SOL	あり	あり	トークンアカウント
TON	Gram	あり	なし	メッセージ
Aptos	APT	あり	あり	長いアドレス
Sui	SUI	なし	あり	gRPC、GraphQL
Polkadot Hub	DOT	あり	あり	口座の下限
XRP Ledger	XRP	なし	あり	準備金、タグ
Stellar	XLM	なし	あり	準備金、タグ
Cardano	ADA	なし	なし	トークンに添えるADA

2026年11月27日時点の tether.to/en/supported-protocols と developers.circle.com によります。「なし」は、その名前のトークンが存在しないという意味ではありません。誰が発行しているかを確かめてください（第9章）。Cosmosは、個々のネットワークによって違います。

送金の費用と注文の大きさ

手数料は金額ではなく、送金の「重さ」とネットワークの混み具合で決まります（第7章）。



豆1袋なら小さな額です。コーヒー1杯だと、手数料が代金より高くなることもあります。そのため少額の注文では、どこで送ると安いかを見ます。値段は変わるので、古い表ではなく、送る直前にウォレットで確かめます。

帰り道も忘れないでください。受け取ったものを送り出すには、そのアドレスに手数料用のコインが必要です。アドレスが100個なら、手数料のかかる送金が100回です。

どれだけ待つか

確定までの時間	ネットワーク
数秒以内	BNB Smart Chain、Solana、TON、Aptos、Sui、XRP Ledger、Stellar
約1分	TRON
数十分	Ethereum、Base、Arbitrum One
約1時間	Bitcoin（6承認）

第8章と第17〜20章にもとづく目安です。

やってみよう：確定ブロックはどれくらい古いか

EVMネットワークに`finalized`ブロック（第8章）を聞き、何秒前に出たものかを計算します。

```
for net in ethereum base arb bsc; do
  t=$(curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber","params":["finalized",false]}' \
    | grep -o '"timestamp":"0x[0-9a-f]*"' | cut -d'"' -f4)
  echo "$net: $(( $(date +%s) - t )) 秒"
  sleep 1.5
done
```

```
ethereum: 929 秒
base: 1343 秒
arb: 990 秒
bsc: 1 秒
```

Ethereumでは約15分、Baseでは約22分です。BNB Smart Chainは1秒です。あなたの数字は違うはずです。確定の印はまとめて飛び飛びに進むからです。少額の注文なら、いちばん確かな印まで待たなくてもかまいませんが、そのときの危険はあなたが引き受けます。

ステップ4：あなたのサイトに見えるか

サイトにアドレスを載せる前に、次を確かめます。

- プロバイダーに、そのネットワークのエンドポイントがあるか（第13章）。
- そのネットワーク向けの支払い通知があるか。これは別のことです。RpcNodeでは、イベントのために解析しているネットワークより、エンドポイントを取得できるネットワークのほうが多くあります（第26章と第27章）。
- 本物のお金を使わずに試せるテストネットがあるか。
- あなたのウォレットと、コインを交換する場所が、そのコインをまさにそのネットワークで扱えるか。

プランも確かめてください。RpcNodeのFreeでは、エンドポイントが使えるのはEthereumとSolanaだけです（第22章）。

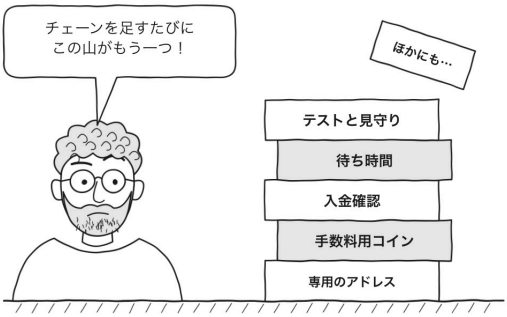
例：アルチョムはこう決める

20人のお客さんに聞いたとします。数字は説明用で、あなたの店では違う数字になります。

何で払いたいか	人数	判断
TRONのUSDT	9	すぐに対応する
BaseのUSDC	5	すぐに対応する
BTC	3	要望が続けば後で
ETH	2	後で
XLM	1	今のところ見送り

2つのネットワークで20人中14人をカバーできます。残りの人には、まだ全部は受け付けていないと正直に伝えます。

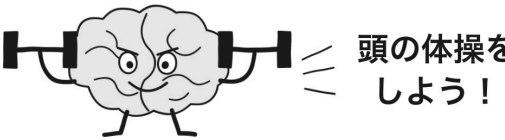
落とし穴：「全部まとめてつなごう」



支払いページに1行増やすのは、ただのように見えます。けれどその裏には、専用のアドレス、手数料用の専用コイン、独自の確認ルール、更新のあとの独自の驚きがあります。1つか2つから始め、要望に応じて足しましょう。

もう一つの落とし穴：支払いページに「USDT」とだけ書くことです。お客さんは自分でネットワークを選び、間違ったものを選ぶかもしれません（第16章）。コインとネットワークを一緒に、大きく書きます。「TRONネットワークのUSDT」。

考えてみよう



1. 注文は5ドルなのに、選んだネットワークの今日の手数料は2ドルです。誰が困り、どうすればよいでしょう。
2. お客さんがBaseのUSDTを受け付けてほしいと言います。Tether社のサイトの一覧はどうなっていますか。
3. アルチョムは初日から8つのネットワークを受け付けたいと言います。何と言って反対しますか。

答え：1）困るのはお客さんです。手数料は注文の40%の上乗せになります。今、送金がもっと安いネットワークを提案します。2）Baseは載っていません。その名前のトークンを受け付ける前に、誰が発行しているかを調べます。3）手間の山が8つになります。お客さんにいちばん必要な1つか2つから始めます。

まとめ

- 選ぶ順番は、お客さん、コイン、ネットワーク、確認です。
- 手数料を注文の大きさと比べ、待ち時間も考えます。
- 1つか2つのネットワークから始め、コインは必ずネットワークと一緒に示します。

第4部はここまでです。アルチョムはネットワークの違いを知り、何から始めるかを決めました。次へ：第5部、第22章。RpcNodeでの最初の5分。

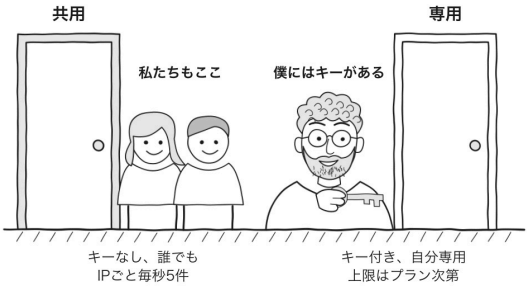
第22章　RpcNodeでの最初の5分

「自分専用のドアがほしい」

この本を通して、アルチョムは公開ノードのドアを叩いてきました。登録なしで答えてくれましたが、このドアはみんなの共用です。お店には専用のドアが必要です。作るのに時間がかかるのでしょうか。何を入力することになるのでしょう。

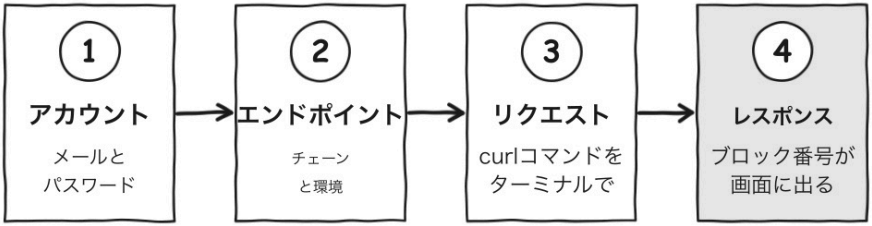
解説：共用のドアと専用のドア

プログラムがノードに質問を送るアドレスをエンドポイントと呼びます（第3章）。RpcNodeには2種類あります。



	公開アドレス	個人用エンドポイント
登録	不要	必要
キー	なし	アドレスに埋め込まれている
上限	IPごとに毎秒5リクエスト	プランによる
重いメソッド	使えない	使える
アーカイブ（第24章）	使えない	有料プランで
WebSocketとgRPC	なし	対応するネットワークで

専用のエンドポイントは、4つのステップで手に入ります。



ステップ1：登録

`cabinet.rpcnode.dev/register` を開くか、rpcnode.devのサイトで「今すぐ始める」を押します。マイページはサービス内のあなたのページで、エンドポイント、使用量、プランがここにまとまっています。

1. メールアドレスを入力し、パスワードを決めて、もう一度入力します。名前は省略できます。
2. 利用規約を読み、「次の内容に同意します: 利用規約」にチェックを入れます。「ロボットではない」ことの確認が出たら、それも済ませます。
3. 「続ける」を押します。
4. メールに一度だけ使えるコードが届きます。入力して「確認して続ける」を押します。

近道もあります。GoogleかGitHubでログインする方法です。

新しいアカウントには無料のFreeプランが付き、カードは不要です。月1500万クレジット、毎秒15リクエスト、エンドポイント3つ、ネットワークはEthereumとSolanaの2つです。クレジットとは何かは第25章で説明します。

ステップ2：エンドポイント

マイページのメニューで「エンドポイント」を選び、「新規エンドポイント」を押します。3段階の画面が開きます。

画面の段階	すること
ネットワークを選択	たとえばEthereum
環境を選択	本番のネットワークかテスト用か（後で説明します）
確認	内容を確かめて「作成」を押す

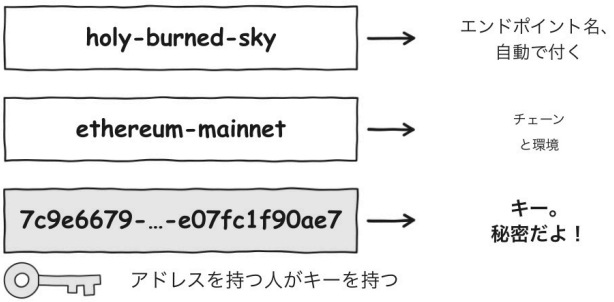
環境とは、ネットワークの種類、つまり本物か練習用かです。エンドポイントの名前は、マイページが英単語3つで自動的に付けます。

Freeプランでは、作成の前にアーカイブノードについての注意が表示されます。エラーではありません。第24章で説明します。

アドレスの中身

RpcNodeのドキュメントにある例です。本来は1行ですが、ここではスラッシュの前で改行しています。

```
https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
/7c9e6679-7425-40de-944b-e07fc1f90ae7
```



スラッシュのあとの長いしっぽはUUIDと呼ばれ、当てることが事実上不可能な番号です。これがキーです。リクエストに別のパスワードを添える必要はありません。

アドレスは手で打たず、マイページのボタンでコピーします。画面ではキーの一部が隠れていますが、コピーされるのは全体です。

メニューには「APIキー」もあります。これは別のもので、「支払いを作成して」のようなサービス自体への命令に署名するためのキーです。ノードへのふつうの質問には必要ありません。

ステップ3：やってみよう、最初のリクエスト

アドレスを設定に入れて、最新ブロックの番号を聞きます。これは雛形なので、文字の部分を自分のアドレスに置き換えてください。

```
export RPC_URL='マイページでコピーしたアドレス'
curl -s "$RPC_URL" \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

個人用アドレスを持っているのはあなただけなので、本では同じ行を公開アドレスで実行した答えを示します：`RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`。

```
{"jsonrpc":"2.0","id":1,"result":"0x19464e4"}
```

`0x19464e4` は26,502,372番のブロックです。第3章でノードが告げたのは26,502,035でした。ネットワークはその間に337ブロック進みました。

Solanaでは、`eth_blockNumber` の代わりに `getSlot` と書きます。

マイページの「概要」には「はじめに」という一覧があります。そこには自分のアドレスが入ったできあいのコマンドと、「最初のリクエストを待っています…」という表示があります。エンドポイントのページの「ライブ」タブでは、届いたリクエストそのもの、つまり時刻、メソッド、結果、遅延が見られます。

うまくいかなかったら

わざと、ゼロだけで作ったでたらめなキーで聞いてみます。

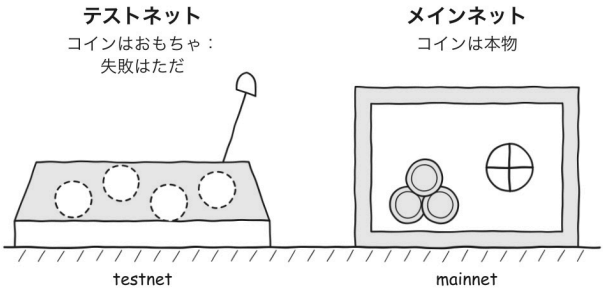
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"endpoint not found"}}  HTTP 404
```

`endpoint not found` は「そのエンドポイントはありません」という意味です。アドレスを丸ごとコピーしたか確かめてください。間違ったアドレスをループで繰り返してはいけません。しつこく試すと、しばらく相手にしてもらえなくなります。

テストネット：おもちゃのお金の練習場

テストネット（testnet）は本物と同じルールで動きますが、そのコインには価値がありません。本物のほうはmainnet（メインネット）と呼びます。テストネットでは、ただで失敗できます。



テスト用コインは「蛇口」（faucet）からもらいます。指定したアドレスにコインを送ってくれるページです。

ネットワーク	RpcNodeのテスト環境の例
Ethereum	Sepolia、Hoodi
Solana	testnet、devnet
Bitcoin	testnet4、signet
TRON	Nile、Shasta

Ethereum上のアプリには、ethereum.orgのサイトはSepoliaを勧めています。Hoodiは、ネットワーク自体やバリデーターを試す人向けです。

登録なしで使えるテストネットもあります。2026年11月27日時点の公開ノード一覧では、Bitcoin testnet4、TRON Nile、TON testnetです。TRONの本物とテスト用に、ネットワーク番号を聞いてみます（第16章）。

```
for host in public-tron-mainnet public-tron-nile; do
  echo -n "$host: "
  curl -s https://$host.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 2
done
```

```
public-tron-mainnet: {"jsonrpc":"2.0","id":1,"result":"0x2b6653dc"}
public-tron-nile: {"jsonrpc":"2.0","id":1,"result":"0xcd8690dc"}
```

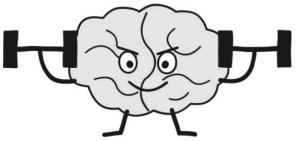
番号が違います。別々の2冊のノートだからです。エンドポイントも、一つのネットワークの一つの環境に結び付いています。テスト用から本番に移るには、新しく作ります。

落とし穴：「アドレスをサイトのページに直接貼ろう」

ページのコードに入ったものは、訪問者なら誰でも見られます。あなたのアドレスを持っていって、あなたの利用枠を使います。

- ノードへの質問は、お客さんのブラウザーからではなく、自分のサーバーから送る。
- アドレスは、コードや共有のコード置き場ではなく、サーバーの設定に置く。
- エンドポイントのページの「設定」で、「IP許可リスト」を指定する。そうすれば、よそのコンピューターは拒否されます。
- 「許可するOrigin」は、よそのサイトからブラウザー経由でエンドポイントを呼ぶのを防ぎます。ただし、ブラウザー以外のプログラムからは守れません。
- アドレスが漏れた？　新しいものを作り、サイトを切り替え、古いものを削除します。削除したものは元に戻せません。

考えてみよう



頭の体操をしよう！

1. アルチョムは、ブロック番号を表示するために、お店のページのスクリプトにエンドポイントのアドレスを書き込みました。キーは誰に見えますか。
2. Sepoliaですべて確認できました。同じエンドポイントで本物の支払いを受け付けられるでしょうか。
3. `endpoint not found`が返りました。どこから調べますか。

答え：1) 訪問者全員。2) いいえ。メインネットのエンドポイントが必要です。3) アドレスをマイページのものと照らし合わせ、丸ごとコピーし直します。

まとめ

- 個人用エンドポイントは、キーを含んだアドレスです。パスワードのように大切にします。
- エンドポイントは、一つのネットワークの一つの環境に結び付いています。
- まずテストネット、それから本番です。

次へ：第23章。リクエストに何が起きるのか：ゲートウェイ、ノードの選択、3つの地域。

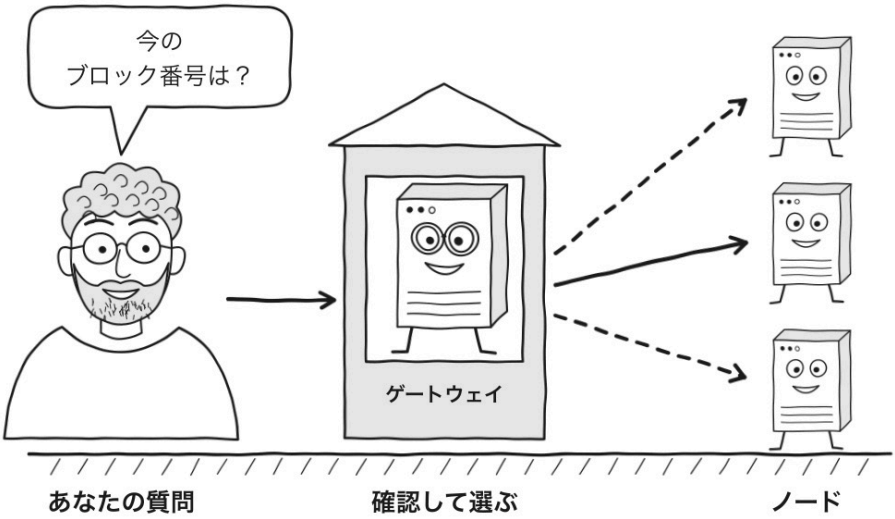
第23章　リクエストに何が起きるのか

「答えてくれたのは誰？」

アルチョムはリクエストを送り、ブロック番号を受け取りました。でも質問はどこへ行ったのでしょう。どのノードが答えたのでしょう。セールの真っ最中にそのノードが壊れたら、どうなるのでしょう。

解説：入口にはゲートウェイがいる

あなたのリクエストは、ノードに直接届くのではありません。まずゲートウェイに入ります。これは配車係のようなプログラムで、リクエストを受け取り、確認し、適したノードに渡します。



ゲートウェイが道中で確かめること

順番は、入館ゲートに似ています。一つの確認に通らなければ、リクエストはその先へ進めません。

確認内容	通らなかった場合
そのエンドポイントはあるか、アドレスは正しいか	HTTP 404、`-32000`
無効化されていないか。あなたのIPとサイトは許可されているか	HTTP 403、`-32000`
聞く頻度が高すぎないか	HTTP 429、`-32005`
このネットワークにそのメソッドはあるか	HTTP 404、`-32601`
クレジットは足りるか	HTTP 402、`-32006`

ネットワークごとにメソッドの一覧があります。一覧にないものを聞いてみます。

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_noSuchMethod","params":[]}'
```

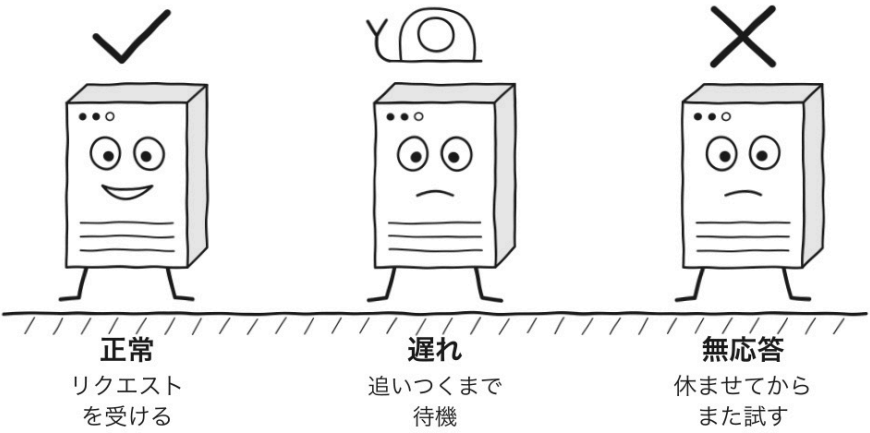
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

この質問はノードまで届いていません。ゲートウェイが自分で答えました。

ノードの選択：負荷分散

一つのネットワークに、ノードは複数ある場合があります。リクエストをそれらに振り分けることを負荷分散（ロードバランシング）といいます。お店のレジのようなもので、次の会計は空いているほうへ回されます。

ゲートウェイは、まずノードの調子を見ます。

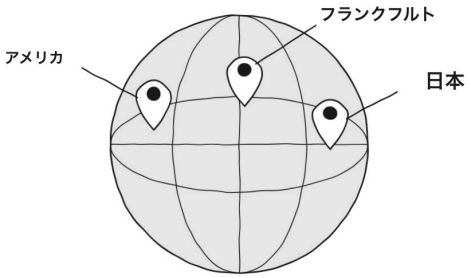


- 遅れているノードは元気よく答えますが、中身は昨日のものです（第12章）。新しいノードがあるうちは、リクエストはそちらへ行きます。
- 健康なノードのうち、忙しいものには少なく、手の空いたものには多く回ります。
- 接続が切れたり、ノードが内部エラーを返したりすると、同じリクエストが次のノードに送られます。あなたは気づきません。
- 何度か続けて失敗したノードは、しばらく休ませてから、また試します。

ただし注意があります。健康なノードがまったく残っていない場合でも、ゲートウェイは遅れたノードを試します。何も返さないより、古い答えのほうがましなこともあるからです。新しさが大事な場面では、自分でブロックの時刻を確かめてください（第12章）。

3つの地域

RpcNodeは3つの地域で動いています。フランクフルト、アメリカ、日本です。ノードが近いほど、往復の道のりは短くなります。ゲートウェイは、自分にいちばん近い健康なノードを使い、遠いノードは予備にします。



RpcNodeの3地域（地図ではなく絵）

すべてのネットワークが3か所全部にノードを持つわけではありません。必要なネットワークがどこでどう応答しているかは、`rpcnode.dev/status`のページで確認できます。

やってみよう：封筒の印を読む

レスポンスと一緒にヘッダーが届きます。封筒の消印のような事務的な目印です。そのうち2つは、ゲートウェイがどの国にあり、ノードがどの国にあったかを教えます。

```
for net in ethereum base arb; do
  echo "== $net"
  curl -s -D - -o /dev/null https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
    grep -iE 'x-gateway-region|x-upstream-country'
  sleep 2
done
```

```
== ethereum
x-gateway-region: DE
x-upstream-country: DE
== base
x-gateway-region: DE
x-upstream-country: JP
== arb
x-gateway-region: DE
x-upstream-country: JP
```

`DE`はドイツ、`JP`は日本です。このときは、すべてのリクエストをドイツのゲートウェイが受け、BaseとArbitrumについては日本まで聞きに行きました。あなたの結果は違うかもしれません。これは事務的な目印なので、サイトの動作をこれに頼らせないでください。

ノードのエラーも答えのうち

USDTのコントラクトに聞いてみます。空のアドレスから巨額を送ろうとしたら、どうなるでしょう。`eth_call`メソッドは試すだけで、何も送りません。`data`フィールドには、送金の関数番号、受取人（`TO`）、金額（`SUM`）がつながっています。

```
TO=0000000000000000000000000000000000000000000000000000000000000dead
SUM=00000000000000000000000000000000000000000000000fffffffffffffff
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[{
  "from":"0x000000000000000000000000000000000000dEaD",
  "to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0xa9059cbb'$TO$SUM'"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"invalid opcode: INVALID"}}  HTTP 200
```

この古いコントラクトは、こうやって「できません」と言います。ノードは役目を果たしました。プログラムが停止したことを正直に伝えたのです。別のノードも同じ答えを返すので、ゲートウェイはこうした答えをHTTP 200で、そのままあなたに渡します。

誰も答えなかったとき

もっと悪い場合もあります。ゲートウェイがノードを順に試しても、答えがない。そのときはコード`-32002`、本文`upstream unavailable`（「ノードが利用できません」）、500番台のHTTPが返ります。この呼び出しのクレジットは返却されます。

お金の送信だったら？

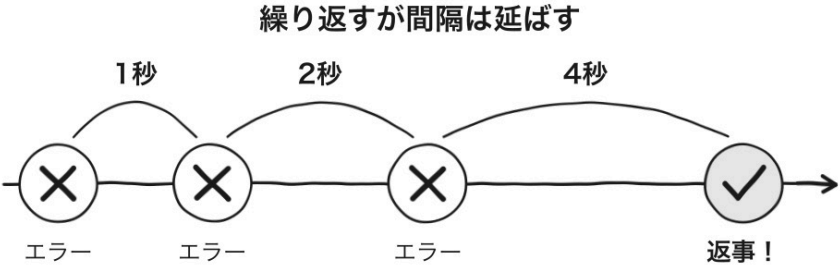
ノードがトランザクションの送信を断ったなら、それが答えです。ノードがまったく反応しなかった場合、ゲートウェイは同じ署名済みの送金を別のノードに渡すことがあります。

それでお金が二重に出ていくことはありません。署名済みトランザクションのハッシュは同じで、ネットワークは一度だけ受け付けます。ただし、あなたが答えを受け取れなかった場合

は、当てずっぽうで新しいトランザクションに署名しないでください。まず古いものをハッシュで探します（第7章）。

やってみよう：賢くやり直す

ゲートウェイはノードを順に試しますが、あなたとゲートウェイの間の接続が失敗することもあります。自分でも、間隔を延ばしながらやり直しましょう。1、2、4、8秒です。英語ではこれをbackoff（バックオフ）といいます。



ファイル `ask.js` を `node ask.js` で実行します（Node.jsが必要です）。

```
const url = 'https://public-ethereum-mainnet.rpcnode.dev'
const sleep = (ms) => new Promise((ok) => setTimeout(ok, ms))

async function ask(method, params = []) {
  for (const pause of [1, 2, 4, 8]) {
    try {
      const res = await fetch(url, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ jsonrpc: '2.0', id: 1, method, params }),
        signal: AbortSignal.timeout(10000),
      })
      if (res.status !== 429 && res.status < 500) return await res.json()
      console.log('HTTP', res.status)
    } catch {
      console.log('接続できません')
    }
    await sleep(pause * 1000)
  }
  throw new Error('問い合わせに失敗しました')
}

ask('eth_blockNumber').then(console.log)
```

```
{ jsonrpc: '2.0', id: 1, result: '0x19464ee' }
```

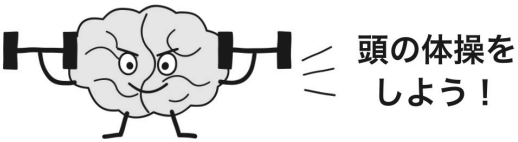
やり直すのは、2回目に通る可能性があるものだけです。「頻度が高すぎる」、接続の切断、500番台です。リクエスト自体の誤りは、繰り返しても無駄です。そして答えを受け取ったら、`error` フィールドの確認を忘れずに（第11章）。

落とし穴：「答えがない」は「支払いがない」ではない

聞くのに失敗したなら、サイトはお客さんが払ったかどうかを知りません。「未払いです」ではなく「支払いを確認中です」と表示して、後でもう一度聞きます。

そして覚えておいてください。ノードを複数持つプロバイダーは、ノード1台よりは落ちにくくても、「絶対に落ちない」わけではありません（第3章）。

考えてみよう



1. HTTP 200と`execution reverted`が返りました。リクエストを5回繰り返せば解決するでしょうか。
2. プログラムが`eth_blokNumber`と聞きました。リクエストはノードに届いたでしょうか。
3. アルチョムは送金を送りましたが、答えが返ってきません。まず何をすべきでしょう。

答え：1）いいえ。故障ではなく、本物の答えです。2）いいえ、ゲートウェイが知らないメソッドを拒否しました。3）ハッシュでトランザクションを探します。

まとめ

- ゲートウェイはリクエストを確認して健康なノードを選び、一つが答えなければ別を試します。
- あなたのリクエストについてのノード自身のエラーは、そのまま届きます。
- それでも、あなたのプログラムは間隔をあけて再試行する必要があります。

次へ：第24章。履歴とアーカイブデータ。

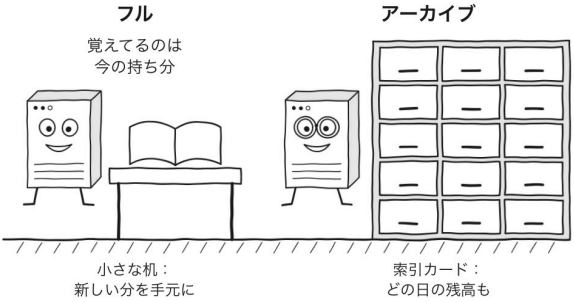
第24章　履歴とアーカイブデータ

「1月1日にはいくらあった？」

会計担当がアルチョムに頼みます。「年の初めに、お店のアドレスにいくらあったか教えてください」。今日のことならノードは即答します。では、あのときのことは？

解説：記録と残高

第10章を思い出しましょう。履歴はブロックと送金です。状態は、ある瞬間に誰が何を持っていたかです。



フルノードは、新しい残高を手元に置いています。アーカイブノードはどのブロック時点の残高も覚えていますが、その代わりディスクがはるかに多く必要です。

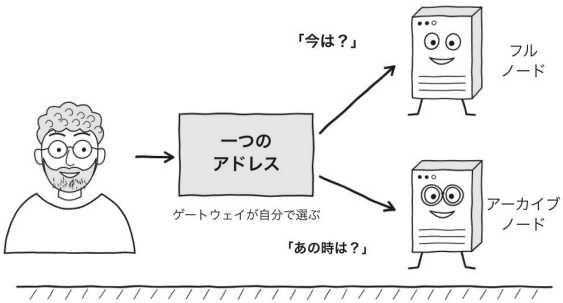
アーカイブが必要になる質問

質問	例
古いブロック時点の残高や、コントラクトの答え	ブロック番号を指定した `eth_getBalance`、`eth_call`
過去のブロックのイベント	`eth_getLogs`
トランザクションを一歩ずつ「巻き戻して」見る（トレース）	`trace_…`、`debug_…`
状態についての証明	古いブロックでの `eth_getProof`
Solanaの昔のスロット	古い番号を指定した `getBlock`

一方、「今、口座にいくらあるか」や、ふつうの新しい支払いには、アーカイブはいりません。お店の日常業務では、たいてい不要です。

RpcNodeではどうなっているか

エンドポイントのアドレスは一つです。ノードを自分で選ぶ必要はありません。ゲートウェイが質問を見て、誰に渡すかを決めます。



アーカイブは有料プランに含まれ、Freeにはありません。Freeでエンドポイントを作ると、マイページにこう表示されます。

> 「無料プランではアーカイブノードを利用できません。リクエストは通常ノードが処理し、ノードが保持しているデータはすべて返します。アーカイブノードだけが持つデータ（古い状態、過去の深い履歴、一部のトレース）は利用できず、ノード自身のエラーが返ります。」

- 「古い」質問に別料金はありません。支払うのはメソッドの代金です（第25章）。
- アーカイブの深さはネットワークごとに違います。何か大事なものをその上に作る前に、自分のネットワークで、いちばん古い質問を試してください。
- 公開アドレスは、アーカイブの質問を受け付けません。

やってみよう：公開ノードは厳しい

残高を「今」ではなく、わずか10ブロック前、約2分前のブロック番号で聞いてみます。

```
E=https://public-ethereum-mainnet.rpcnode.dev
H='Content-Type: application/json'
N=$(curl -s $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
  grep -o '0x[0-9a-f]*')
OLD=$(printf '0x%x' $((N - 10)))
sleep 2
curl -s -w '  HTTP %{http_code}\n' $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","'$OLD'"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"archive data is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

公開アドレスは、こんな最近の過去でもアーカイブとみなします。聞くときは第8章の言葉、`latest`、`safe`、`finalized`を使います。

2つ目の実験：ノードが捨てているかもしれない

ブロックは状態ではなく履歴です。しかし履歴も、全部のノードが持っているわけではありません。2025年7月から、Ethereumのノードプログラムは、ステーキング移行前のブロックの中身を保管しないことができます（第10章）。2015年8月のブロック46,147（`0xb443`）を頼んでみます。

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["0xb443",false]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":4444,
"message":"pruned history unavailable"}}  HTTP 200
```

「切り落とされた履歴は利用できません」。ノードは、持っていないと正直に言っています。有料プランでは、ふつうのノードから「それは持っていません」と返ってきた場合、そのネットワークにアーカイブノードがあれば、ゲートウェイがそちらに聞き直せます。

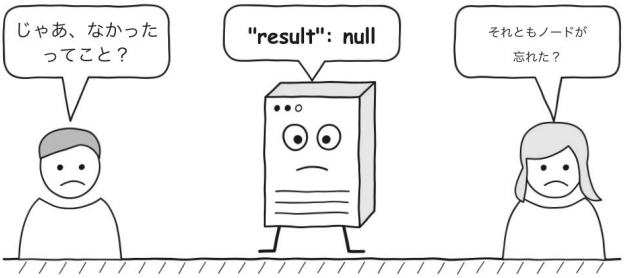
落とし穴：「nullなら、そんなものはなかった」

そのブロックには送金が一つあります。それをハッシュで聞き、続けて、私たちがでっち上げたハッシュも聞いてみます。

```
for hash in \
  0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060 \
  0x0000000000000000000000000000000000000000000000000000000000000001
do
  curl -s https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash",
    "params":["'$hash'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":null}
{"jsonrpc":"2.0","id":1,"result":null}
```

答えは同じです！　1つ目の送金は実際にあったもので、2つ目は一度も存在していません。



答えだけでは区別できない

`null` 一つだけで結論を出さないでください。古い記録ではなおさらです。「なかった」という意味のときもあれば、「保管していない」のときも、新しい送金なら「まだブロックに入っていない」のときもあります。

Solanaのノードはどれだけ覚えているか

Solanaは1秒に何回もブロックを作るので、記録は山のように積み上がります。最新スロットの番号と、ノードがまだ持っているいちばん古いブロックを聞いてみます。

```
for method in getSlot getFirstAvailableBlock; do
  echo -n "$method: "
  curl -s https://public-solana-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$method'"}'
  echo; sleep 2
done
```

```
getSlot: {"jsonrpc":"2.0","result":474316453,"id":1}
getFirstAvailableBlock: {"jsonrpc":"2.0","result":473916707,"id":1}
```

差はおよそ40万スロットです。確認時には、これはたった1日分でした。そしてブロック1000は、とうに片づけられています。

```
curl -s https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockTime","params":[1000]}'
```

```
{"jsonrpc":"2.0","error":{"code":-32001,"message":"Block 1000 cleaned up,
does not exist on node. First available block: 473916707"},"id":1}
```

会計担当の話に戻る

「年初の残高」とは、前の年の最後のブロックのあとの残高です（時刻は協定世界時UTCで数えます）。日付からブロック番号を、ノードはすぐには教えてくれません。その代わり、ブロ

ックは順番に並んでいるので、二分探索ができます。真ん中のブロックを取って時刻を見て、いらないほうの半分を捨てるのです。

2026年の最初のブロックを探します。スクリプトの実行には約1分かかります。

```
E=https://public-ethereum-mainnet.rpcnode.dev
ask() {  # method, params
  curl -s $E -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$1'","params":'$2'}'
  sleep 2
}
WHEN=1767225600   # 2026年1月1日 00:00 UTC
low=20000000
high=$(( $(ask eth_blockNumber '[]' | grep -o '0x[0-9a-f]*') ))
steps=0
while [ $low -lt $high ]; do
  mid=$(( (low + high) / 2 ))
  hex=$(printf '0x%x' $mid)
  t=$(ask eth_getBlockByNumber '["'$hex'",false]' |
    grep -o '"timestamp":"0x[0-9a-f]*"' | grep -o '0x[0-9a-f]*')
  if [ -z "$t" ]; then echo "応答なし"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "ブロック ${low}、質問回数: $steps"
```

```
ブロック 24136053、質問回数: 23
```

何百万ものブロックから、目当てのものが23回の質問で見つかりました（あなたの回数は1つ違うかもしれません）。つまり2025年の最後のブロックは24,136,052（`0x1704974`）です。そのブロック時点の残高は、1つ目の実験と同じように、この番号を指定して聞きます。公開アドレスは断るので、有料プランの個人用エンドポイントが必要です。トークンの場合は、`eth_getBalance`の代わりに、コントラクトに`balanceOf`を聞く`eth_call`を使います。

長い区間は小分けにする

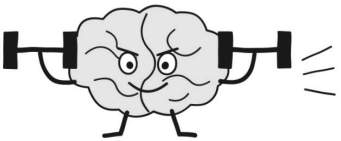
「1年分の全イベント」というリクエストは、おそらくノードの手に負えません。答えが大きすぎます。ブロックの区間を分割し、順に聞いて、答えをつなぎ合わせます。

これは雛形です。`rpc`は、第23章の`ask`のように、個人用エンドポイントに聞くあなたの関数です。

```
async function logsByChunks(rpc, filter, from, to, size = 1000) {
  const all = []
  for (let a = from; a <= to; a += size) {
    const b = Math.min(a + size - 1, to)
    const answer = await rpc('eth_getLogs', [{
      ...filter,
      fromBlock: '0x' + a.toString(16),
      toBlock: '0x' + b.toString(16),
    }])
    if (answer.error) throw new Error(answer.error.message)
    all.push(...answer.result)
  }
  return all
}
```

通らなければ、区間を小さくします。区間ごとが、それぞれ有料の呼び出し1回です。そして、お店にとっていちばん安いアーカイブは自分のものです。支払いを見つけたら、すぐにデータベースに記録しておきます（第26章）。

考えてみよう



頭の体操をしよう！

1. 昔の送金のハッシュに、ノードが`null`を返しました。お客さんが払わなかった証拠になるでしょうか。
2. 7月1日時点のUSDC残高が必要です。Freeプランで足りますか。
3. 5000ブロック分のイベントを、1000ずつに分けて調べます。呼び出しは何回ですか。

答え：1）いいえ。ノードがその記録を保管していないのかもしれません。2）アーカイブはFreeに含まれないので、当てにできません。3）5回。

まとめ

- 古い残高、古いイベント、トレースは、アーカイブノードの仕事です。
- アーカイブは有料プランに含まれ、エンドポイントのアドレスは変わりません。
- `null`や空の答えは、何もなかったことの証明にはなりません。

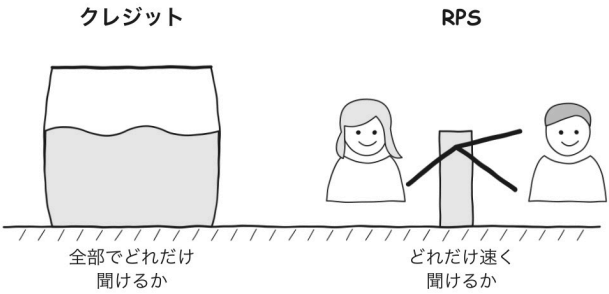
次へ：第25章。プラン、呼び出しの値段、上限。

第25章　プラン、呼び出しの値段、上限

「で、月いくらになる？」

エンドポイントは動いています。次にアルチョムが知りたいのは、お店の費用です。料金ページには主な数字が2つあります。クレジットと、毎秒のリクエスト数です。それぞれ何でしょう。

解説：2つの別の物差し



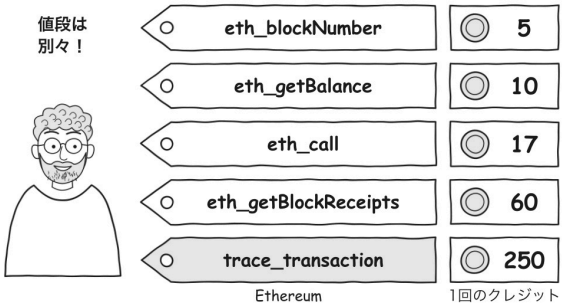
クレジットはサービス内の「お金」で、1か月に合計でどれだけ聞けるかを決めます。RPSは毎秒のリクエスト数で、どれだけ速く聞けるかを決めます。両方とも足りていなければなりません。

やってみよう：呼び出しの値段を知る

質問によって値段は違います。ノードが記憶からすぐ答えられる質問もあれば、ブロックをまるごと調べる質問もあります。値段は公開されていて、登録なしで聞けます。Ethereumのメソッドをいくつか取り上げます（`python3`が必要です）。

```
curl -s https://rpcnode.dev/api/v1/public/networks/ethereum/methods |
python3 -c 'import json, sys
want = """eth_blockNumber eth_getBalance eth_call
eth_getLogs eth_getBlockReceipts trace_transaction""".split()
for m in json.load(sys.stdin)["methods"]:
    if m["method"] in want and m["variant"] == "default":
        print(m["method"], m["credits"])'
```

```
eth_blockNumber 5
eth_getBalance 10
eth_call 17
eth_getLogs 17
eth_getBlockReceipts 60
trace_transaction 250
```



- `trace_transaction` 1回は、ブロック番号の質問50回分です。
- 料金表はネットワークごとに違います。アドレス内の `ethereum` を `solana` や `tron` に替え、メソッド名も変えてください。
- バッチ（第11章）の値段は、中の呼び出しの合計です。
- WebSocketの購読も有料です。Ethereumでは、新しいブロックの購読を開くのが10クレジット、届くイベント1件ごとに2クレジットです。

マイページでは、値段は「ネットワーク」のセクションにまとまっています。

プラン

2026年11月27日時点の情報です。最新は `rpcnode.dev/pricing` で確認してください。

プラン	月額（$）	年額（$）	月のクレジット	RPS	エンドポイント数
Free	0	—	1500万	15	3
Starter	19	190	5000万	40	5
Build	36	360	1億	75	12
Grow	99	990	3億	150	25
Scale	219	2,190	7億	300	50
Business	449	4,490	15億	500	100

プランはRPCだけではありません。ほかに次のものも含まれます（第26章と第27章）。

プラン	監視アドレス数	お支払い：未完了 / 月間	トランザクションの監視数	アーカイブと全ネットワーク
Free	—	—	—	なし：EthereumとSolanaのみ
Starter	15	10 / 200	15	あり
Build	50	25 / 1,000	40	あり
Grow	150	50 / 5,000	80	あり
Scale	400	100 / 20,000	150	あり
Business	1,000	250 / 100,000	400	あり

- 新しいアカウントは、一度だけ、カードなしで、Starterを7日間無料で試せます。
- 用意されたプランが合わない場合、料金ページに「自分でプランを作成」があります。
- プランを買うには「購入」を押して確認します。15分間有効な請求書が表示され、暗号資産で支払えます。
- 期限が切れて新しい支払いがなければ、アカウントはFreeに移ります。自分でリマインダーを設定しておきましょう。

アルチョムの店を計算する

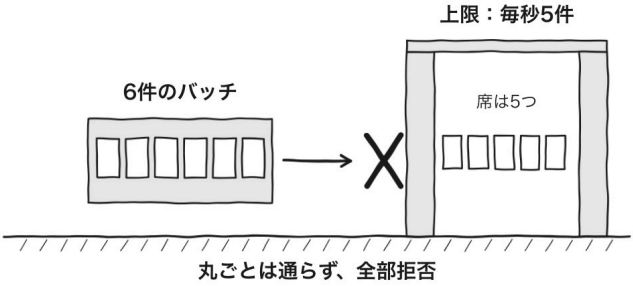
支払い待ちの注文が20件あり、それぞれにEthereumの別のアドレスがあります。30日間で、注文はつねに待っていると仮定します。これは多めに見た見積もりです。

サイトの聞き方	リクエスト数	クレジット	足りるプラン
各アドレスの残高を10秒ごと	5,184,000	51,840,000	Build
同じことを30秒ごと	1,728,000	17,280,000	Starter
ブロック番号と `eth_getLogs` 1回を12秒ごと	432,000	4,752,000	Free

3番目の方法は、1番目より10倍以上安くなります。全アドレス分を一つの質問で聞くからです。ただし、この比較には注意があります。最初の2つはETHの着金に気づきますが、3番目はトークンの送金に気づく方法です（第15章）。それにFreeのネットワークは、EthereumとSolanaの2つだけです。

毎秒の上限

RPSのカウンターは毎秒リセットされ、RpcNodeのドキュメントによれば、エンドポイントごとに別々です。超えたリクエストにはHTTP 429とコード `-32005` が返ります。N個の呼び出しが入ったバッチはNリクエストで、全部通るか、全部通らないかです。



上限が5の公開アドレスで確かめます。質問6個のバッチ1つで十分です。

```
q='{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
curl -s -i https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d "[$q,$q,$q,$q,$q,$q]" | grep -iE '^(HTTP|retry-after)|error'
echo
```

```
HTTP/2 429
retry-after: 1
{"jsonrpc":"2.0","id":null,"error":{"code":-32005,"message":
"rps limit exceeded: a batch of 6 calls counts as 6 requests
(the limit is 5 a second)"}}
```

`retry-after: 1` は「1秒後にやり直してください」という意味です。公開アドレスは、高価なメソッドも制限しています。

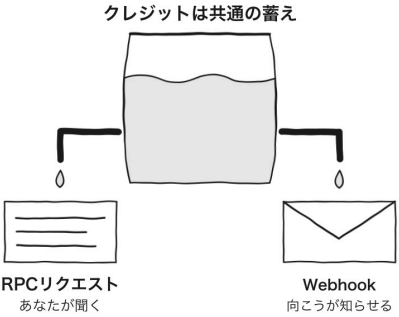
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"method eth_getBlockReceipts is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

何が引かれ、何が引かれないか

起きたこと	クレジット
ノードが結果を返した	引かれる
ノードがあなたのリクエストについてのエラーを返した（たとえば `execution reverted`）	引かれる。これも答えだから
毎秒の上限で拒否された	引かれない
そのメソッドがネットワークの一覧にない	引かれない
どのノードも答えなかった	返却される
公開アドレスへのリクエスト	無料

蓄えは全部で一つ



Webhookの代金も、同じ蓄えから支払われます。Webhookは、サービスのほうからあなたのサイトに送る知らせです（第26章）。Ethereumのメインネットでは、監視アドレスのイベント1件が450クレジット、お支払いが750クレジットです。Ethereumのテストネットでは半額です。

残りはマイページの「使用量」で確認できます。よく使うメソッドもそこに出ます。

クレジットがなくなったら

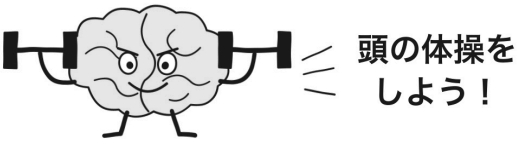
超過分の請求はありません。パックを超える追加料金はないのです。ただし、サービスがそのまま何事もなく動き続けるとは思わないでください。第11章の拒否（`-32006`）をプログラムに組み込み、残りは前もって見張ります。

落とし穴：値札でプランを選ぶ

まず回数×メソッドの値段を計算してください。重い呼び出しがいくつかあるだけで、軽い呼び出し何千回より多く消費することがあります。

メソッドの値段と条件は変わることがあります。サービスの利用規約では、法律で別に定められていない限りプランの代金は返金されず、中断のない稼働も保証されません。有料プランで大きな障害があった場合は、支払い済みの期間が延長されます。支払う前に読んでください。

考えてみよう



1. サイトが`eth_call`（17クレジット）を10秒に1回、30日間呼び続けます。何クレジットになりますか。
2. Freeプランで20個の呼び出しのバッチを送りました。どうなりますか。
3. リクエストが429を受けました。クレジットは引かれましたか。

答え：1）259,200×17 = 4,406,400。2）まるごと拒否されます。20は15より大きいからです。3）いいえ。

まとめ

- クレジットは1か月の量を、RPSは速さを制限します。
- メソッドごとに値段があり、バッチは呼び出しの数で数えます。
- 大きなプランを買うより、賢く聞くほうが得なことがよくあります。

次へ：第26章。アドレスのイベント：ひっきりなしに聞く代わりに、Webhookとソケット。

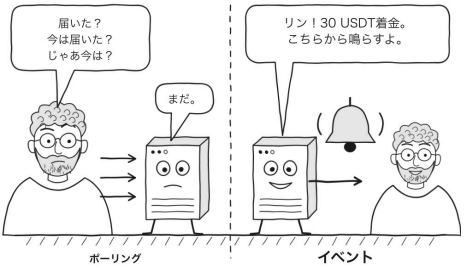
第26章　アドレスのイベント：ひっきりなしに聞く代わりに、Webhookとソケット

「注文20件なら、毎秒20回の質問？」

第3章で計算しました。支払い待ちの注文20件を毎秒1回ずつ確認すると、それだけでFreeプランが許す毎秒15リクエストを超えます。しかも答えはほとんど全部同じで、「まだ届いていません」です。

解説：聞くか、呼び鈴を待つか

ポーリング（聞き取り）は、サイトのほうから「届いた？」と何度も聞くことです。イベントは、何かが起きた瞬間に、向こうから知らせてもらうことです。



やってみよう：全アドレスを一つの質問で

まず、もっと賢く聞きます。注文ごとに別々に聞く必要はありません。第15章の `eth_getLogs` メソッドには、受取人の一覧をまとめて指定できます。

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
  null,
  ["0x0000000000000000000000000000000000000000000000000000000000000dead",
   "0x0000000000000000000000000000000000000000000000000000000000000001",
   "0x000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"]],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

質問の内容はこうです。「最新ブロックに、この3つのアドレスのどれかへのUSDT送金はあるか。送り主は誰でもよい」。アドレスは左側をゼロで埋めて64文字にしてあります。返ってきた答えはこうです。

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

空の一覧は、このブロックでは誰も支払っていないという意味です。アドレスが20個でも、リクエストはやはり1回です。ただし、新しいブロックの一つひとつについて聞き、承認数を自分で数え、ETHそのものの送金は別に捕まえる必要があります（第15章）。

Address Watch：見張りは任せる

Address Watch（アドレス監視）は、この仕事を引き受けます。RpcNodeがブロックを解析し、トランザクションがあなたのアドレスに関係していれば、あなたのサーバーに知らせます。

アドレスはマイページで追加します。「通知」→「アドレス」→「アドレスを追加」です。プログラムからは、本文 `{"network_slug": "tron", "address": "あなたのアドレス"}` を付けた `POST /v1/address-watches` リクエストで同じことができます。

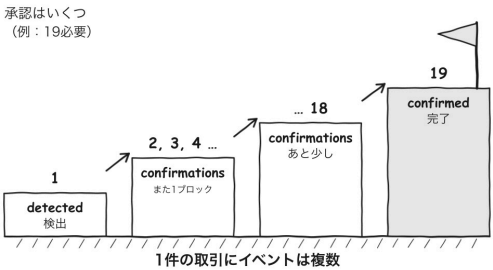
レスポンスで監視の番号 `watch_id` が返ります。1つのネットワークの1つのアドレスが、プランのスロットを1つ使います。監視を削除するとスロットは空きますが、追加と削除の回数にも上限があります。Starterのスロットは15です。Freeにはメインネットのスロットはなく、「サンドボックス」（テストネットの数個のアドレス）だけがあります。

エンドポイントを取得できるネットワークのほうが、イベントのために解析しているネットワークより多くあります。2026年11月27日時点では、Ethereum、Bitcoin、Solana、TRON、BNB Smart Chain、Base、Arbitrum One、TON、Litecoin、Dogecoin、Bitcoin Cash、Dash、Mantle、Monadで、テストネットはTRON Nileだけです。

お金の着金は `payment` という種類のイベント、出金は `payout`、そのほか（たとえばapproveの許可）は `address` です。新しいブロックは `block` という種類で届きます。`tx` と `payment_order` は第27章で説明します。

一つのトランザクションで呼び鈴は何度も鳴る

送金の知らせは1回ではなく、送金が「固まって」いくにつれて届きます（第8章）。何が起きたかは `event` フィールドが伝えます。



event	意味	お店がすること
`detected`	トランザクションがブロックで見つかった	「送金を確認しています」と表示する
`confirmations`	承認が一つ増えた	カウンターを更新する
`confirmed`	承認が十分で、成功	お金が届いたとみなす
`failed`	承認は十分だが、トランザクションは失敗	発送しない

待つ承認数は「設定」→「承認数」で指定します。指定しなければ、ネットワークの標準値が使われます。

Bitcoin、Litecoin、Dashでは、どれよりも早く`pending`が届くことがあります。送金はまだmempoolにあります（第7章）。この種類はつねに`payment`ではなく`address`で、まだお金ではありません。こうした送金は置き換えられることがあります。

Webhook：あなたの郵便受けへの手紙

Webhookとは、サービスがインターネット上のあなたのアドレスに自分から送るリクエストです。アドレスとネットワークごとの購読は、「通知」→「配信設定」で指定します。アドレスはインターネットからアクセスできる必要があります。`localhost`は使えません。

何も選ばずにWebhookを有効にすると、全ネットワークの入金と出金の送金が届きます。ブロックは届きません。

ドキュメントの例の一部です。TRONの30 USDTについての最初の呼び鈴です。

```
{ "delivery_id": "3f9c…6b3d90",
  "type": "payment", "event": "detected", "network": "tron",
  "tx_hash": "bb08…c320c", "address": "TToL…QRzRG",
  "direction": "in", "currency": "USDT",
  "amount": "30", "amount_raw": "30000000",
  "confirmations": 1, "required_confirmations": 19,
  "confirmed": false }
```

金額は、丸められないように文字列で届きます。`amount_raw`は小数点なしの同じ値です（第9章）。サービスのカタログにないトークンには、`amount_raw`だけが付きます。ヘッダーには`X-RpcNode-Delivery`（配信番号）、`X-RpcNode-Event`（種類）、`X-RpcNode-Signature`（署名）が入ります。

「本当にあなたですか？」

Webhookのアドレスは公開されています。誰でもノックして「支払いがありました」と書けます。そのため、配信には毎回署名が付きます。時刻、ピリオド、本文をつなげ、Webhookのシークレットを使って指紋を計算します。この方式をHMAC-SHA256といいます。シークレットは「シークレットとIP」の欄にあります。APIキーのシークレットとは別のものです。

やってみよう：自分で署名する

ここのシークレットと本文は、でっち上げたものです。

```
printf '%s' '1787347206.{"type":"test"}' |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}'
```

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

本文かシークレットを1文字でも変えると、まったく違う文字列になります。あなたのサーバーは同じ計算をして、`t=…,v1=…`の形のヘッダーと比べます。

Node.jsでの受信側の雛形です。自作のリクエストでは確認しましたが、本物の配信では試していません。それには個人のマイページが必要です。300秒の確認は、昨日の配信をもう一度送りつけられるのを防ぎます。

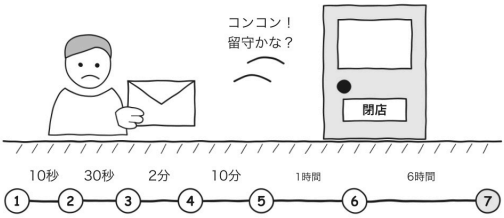
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // 本物のお店ではデータベース
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // 届いたままのバイト列
  const [, t, v1] = /^t=(\d+),v1=([0-9a-f]{64})$/
    .exec(req.headers['x-rpcnode-signature'] ?? '') ?? []
  const ours = createHmac('sha256', SECRET)
    .update(`${t}.`).update(body).digest()
  const ok = v1 && Math.abs(Date.now() / 1000 - t) < 300 &&
    timingSafeEqual(ours, Buffer.from(v1, 'hex'))
  if (!ok) return res.writeHead(401).end()
  const id = req.headers['x-rpcnode-delivery']
  if (!seen.has(id)) {
    seen.add(id)
    console.log(id, body.toString())  // ここに自分の処理
  }
  res.writeHead(200).end()
}).listen(3000)
```

留守だったら

サーバーが10秒以内に200番台のコードで応答すれば、配信は成功です。そうでなければ再送され、7回目の失敗で打ち切られます。だから次のことが大事です。

- 一つのイベントが2回届くことがあります。配信番号は同じなので、それで重複を除きます。
- 順番は保証されません。`confirmations`が`detected`を追い越すこともあります。
- 時間のかかる処理は、応答したあとで行います。



間隔は延びる。全7回、約7時間

自分のアドレスは、テスト配信で前もって確かめます。無料です。マイページの配信ログには、1日分の記録が残ります。

ソケット：開いたままの回線

2つ目の方法はWebSocketです。あなたのプログラムが自分で接続を開いて耳を澄ますので、インターネット上の自分のアドレスはいりません。これは第11章の `eth_subscribe` の購読とは違います。あちらはノードがネットワークについて話し、こちらはサービスがあなたのアドレスと注文について話します。本文はWebhookと同じですが、あなたが不在の間に逃した分は再送されません。

プロトコルはPusherと互換性があり（ `pusher-js` ライブラリーが使えます）、キーはあなたのAPIキーの番号です。ソケットは「WebSocketに配信する」のスイッチで別に有効にします。両方の方法を有効にすると、知らせは1件ごとに2回配信され、2回分の料金がかかります。

費用はいくら

イベントは、リクエストと同じクレジットを使います。2026年11月27日時点で、TRONかEthereumのメインネットのアドレスイベントは450クレジットで、トランザクションにつき1回だけです。承認の知らせと結果は無料です。サーバーが後で受け取れなくても、配信した分の料金はかかります。

Ethereumで30日間の1か月を見積もります。表の各リクエストは17クレジットです。

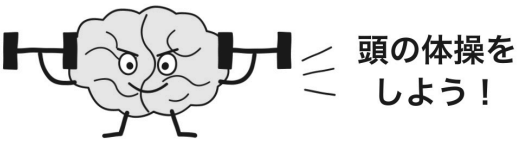
支払いの知り方	リクエスト数またはイベント数	クレジット
注文20件、 `eth_call` を10秒ごと（第13章）	5,184,000	88,128,000
ブロックごとに `eth_getLogs` 1回、12秒ごと	216,000	3,672,000
Address Watch、1か月に送金200件	200	90,000

Starterのクレジットは5000万なので、最初の方法は収まりません。これは一例です。送金が多いほどイベントの費用は増え、他人があなたのアドレスに送った少額の送金も数えられます。

落とし穴：「イベントが来たから発送する」

- どのイベントも支払いとは限りません。`detected`はまだ確定ではなく、`payout`はあなたの出費です。
- トークンはコントラクトアドレスで見分けます。名前ではなく、`token`フィールドのアドレスです（第9章）。
- 特殊な送金は事前に試してください。たとえば、コントラクトの内部から送られたETHです（第15章）。
- アドレスのイベントはアーカイブではありません。APIで取れるのは1日分です。

考えてみよう



1. お店のサーバーを1時間止めました。Webhookの配信はどうなり、ソケットの知らせはどうなりますか。
2. 同じ`X-RpcNode-Delivery`の配信が2つ届きました。お金は何回入金処理しますか。
3. 1か月に、監視アドレスへのTRONの送金が1000件、配信はWebhookだけです。何クレジットですか。

答え：1）Webhookは7回の試行を使い切るまで再送されます。ソケットは逃した分を再送しません。2）1回。3）450,000。

まとめ

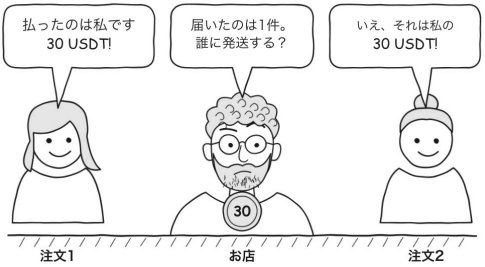
- ひっきりなしに聞くのはやめて、送金は知らせてもらいます。
- 署名を確かめ、重複を除き、すばやく応答します。
- お金が届いたと言えるのは`confirmed`で、最初の呼び鈴ではありません。

次へ：第27章。お支払いとトランザクションの追跡。

第27章　お支払いとトランザクションの追跡

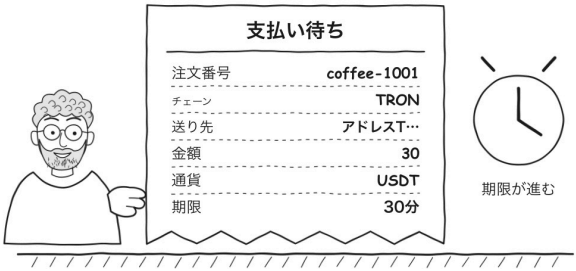
「お金は届いた。で、誰の？」

アルチョムにイベントが届きました。アドレスに30 USDTが入ったのです。注文一覧を開くと、30の注文が2件あります。イベントはアドレスと金額は知っていますが、お客さんの買い物かごのことは何も知りません。



解説：お支払いとは「待ち受け」

お支払い（マイページの「お支払い」セクション）は、あらかじめ書き留めた待ち受けです。この注文は、これだけの額を、このアドレスで、この時刻まで待つ、というものです。条件に合う送金が届くと、サービスが自分で注文番号を教えてくれます。



アドレスは自分のものを指定します。お金はお客さんから直接あなたに届き、サービスにシークレットフレーズは必要ありません（第6章）。

お支払いは「新規お支払い」ボタンか、`POST /v1/payments`リクエストで作成します。

フィールド	内容
`order_id`	あなたの注文番号。重複しないこと
`network_slug`、`address`	ネットワークと、そこでのあなたのアドレス
`amount`	文字列の金額：`"30"`、`"12.5"`
`token_contract`	トークンのコントラクト。省略するとネットワークのコインを待つ
`ttl_seconds`	期限（秒）。省略すると1時間、最長30日

やってみよう：キーなしでノックする

```
curl https://api.rpcnode.dev/v1/payments
```

```
{"ok":false,"error_code":"public_api.key_missing",
 "error":"Missing X-Api-Key header."}
```

これはコード401の答えで、「どなたですか？」という意味です。このAPIへのリクエストはすべて、APIキーのシークレットで署名します（「APIキー」セクション）。署名するのは4行です。時刻、メソッド、パス、本文の指紋です。

シェルでの雛形です。構文と署名の計算は確かめましたが、送信はしていません。個人のキーが必要だからです。

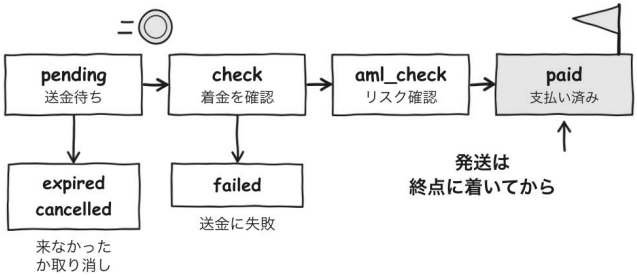
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"あなたのアドレス","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$HASH" |
  openssl dgst -sha256 -hmac "$SECRET" | awk '{print $NF}')

curl https://api.rpcnode.dev/v1/payments \
  -H 'Content-Type: application/json' \
  -H "X-Api-Key: $KEY_ID" -H "X-Timestamp: $TS" \
  -H "X-Signature: $SIG" -d "$BODY"
```

レスポンスで`payment_id`、ステータス`pending`、期限`expires_at`が返ります。同じ`order_id`で、ネットワーク、アドレス、トークン、金額も同じリクエストを繰り返すと、同じお支払いが返ります。違う内容だと`payment.order_exists`エラーになります。

お支払いの道のり

各ステップは、`payment_order`イベントとして、Webhookかソケットで届きます（第26章）。



event	何が起きたか
`check`	お金を確認した。承認を待っている
`confirmations`	承認数が増えた
`aml_check`	承認は足りた。リスク確認中
`paid`	支払い済み。発送してよい
`expired`、`cancelled`	期限が切れたか、待機中のお支払いをあなたが取り消した
`failed`	ネットワーク上のトランザクションが失敗した（このステータスだけを頼りにしないこと。お支払いは単に期限切れになる場合もある）

AMLは、送られてきたお金が不審な経歴とつながっていないかを調べる確認です。この機能があるところでは、`paid`イベントの`aml`オブジェクトに、0から100の評価`score`（大きいほど危険）と、レベル`risk_level`が入ります。別料金はありません。すべてのネットワークにあるわけではなく、ない場合、お支払いはすぐ`paid`になります。

これは参考情報であって、判決でも法的な見解でもありません。危険度の高い送金をどう扱うかは、あなたが決めます。お支払いが`aml_check`で止まったら、それはまだ支払い完了ではありません。サポートに連絡してください。

ドキュメントの例の一部です。

```
{ "type": "payment_order", "event": "paid", "order_id": "ord_123",
  "network": "tron", "status": "paid", "tx_hash": "bb08…c320c",
  "received_amount_raw": "30000000",
  "confirmations": 19, "required_confirmations": 19,
  "aml": { "score": 0, "risk_level": "low" } }
```

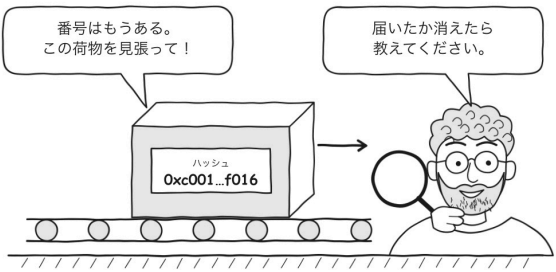
小さな文字の注意書き

- 不足した額は数えられません。お支払いは待ち続け、期限切れになります。ドキュメントには `underpaid` というステータスがありますが、お店の仕組みをこれに頼らせないでください。不足はアドレスのイベントで気づくようにします。
- 多すぎる額は数えられます。実際に届いた額は `received_amount_raw` が教えます。
- 遅れた送金では支払いになりません。期限内に確認された送金は、期限後も問題なく承認を積み重ねます。
- 注文ごとに専用のアドレス。そうすれば「30」の2件を取り違えません（第28章）。
- アドレスのスロットは自動では空きません。`paid` や `expired` のあとも同じです。監視を削除してください。お支払いを取り消した場合は、そのためだけに追加したアドレスは自動で外れます。

お支払いは有料プランに含まれます。Starterでは、同時に未完了10件、月200件です。TRONかEthereumのメインネットのお支払い1件は750クレジットで、お金が確認されたときだけ引かれます。Freeにはサンドボックスがあり、テストネット（現在はTRON Nile）で未完了のお支払いを3件まで持てます。そこのリスク確認は本物ではなく、`"sample": true` の印が付きます。

トランザクションの追跡

逆の場合もあります。ハッシュがすでに手元にあるときです。アルチョムがお客さんに返金したり、仕入れ先に支払ったりして、届いたかを知りたい場合です。



「通知」→「トランザクション」→「ハッシュを追加」、または本文 `{"network_slug": "tron", "tx_hash": "ハッシュ"}` を付けた `POST /v1/tx-watches` です。ハッシュは、ネットワークに現れる前でも追加できます。

届くのは `tx` イベントで、`detected`、次に `confirmations`、最後に `confirmed` か `failed` です。ハッシュが最後まで見つからなければ `cancelled` です。イベントが届くのは、そのネットワークで入金か出金の送金を有効にしている場合です。

実行中の監視はスロットを使い、終わると空けます。Starterは15スロットで、Freeにはこの機能がありません。料金はトランザクション1件につき1回で、TRONとEthereumでは150クレジ

ットです。

やってみよう：承認数を自分で数える

こうした監視の裏側で行われているのは、これです。第15章のUSDT送金を取り上げます。

```
URL=https://public-ethereum-mainnet.rpcnode.dev
H=0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt",
  "params":["'$H'"]}' | grep -oE '"(status|blockNumber)":"0x[0-9a-f]+"' | sort -u
sleep 1
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":2,"method":"eth_blockNumber","params":[]}'
```

```
"blockNumber":"0x194646a"
"status":"0x1"
{"jsonrpc":"2.0","id":2,"result":"0x19464c5"}
```

ステータス `0x1` は成功です。送金の入ったブロックが1つ目の承認と数えられるので、`echo $((0x19464c5 - 0x194646a + 1))` の結果は92です。あなたの数はもっと大きくなります。

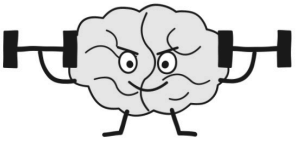
落とし穴：同じお金で呼び鈴が2回

お支払いはアドレスを監視対象にします。そのため一つの送金について、`payment`（アドレスのイベント）と `payment_order`（お支払いのイベント）の両方が届くことがあります。料金はそれぞれの値段でかかります。

> ルール：注文を完了にするのは、`event` = `paid` の `payment_order` だけ。そして注文番号、`payment_id`、トークン、金額を照らし合わせます。

お支払いの状態を延々と聞くのもやめましょう。APIには独自の上限があり、アカウントごとに毎分100リクエストです。聞くのは停止のあとの突き合わせのためで、`GET /v1/payments/orders/{orderId}` を使います。

考えてみよう



頭の体操をしよう！

1. 30 USDTのお支払いに、29.5が届きました。お支払いはどうなりますか。
2. 期限が切れ、その1分後に送金が確認されました。お金はどこにあり、ステータスはどうなりますか。
3. `check` イベントが届きました。コーヒーを発送してよいでしょうか。

答え：1) 支払い済みにはならず、期限が来れば切れます。2) お金はあなたのアドレスにあり、お支払いは `expired` のままです。手作業で対応します。3) いいえ、 `paid` を待ちます。

まとめ

- お支払いは送金を注文に、ハッシュの監視はトランザクションの行方に結び付けます。
- 発送の合図は `paid` で、最初の呼び鈴ではありません。
- 不足、遅れ、ふさがったスロットは、あなたが面倒を見ます。

第5部はここまでです。アルチョムは、エンドポイントを手に入れ、リクエストの道のりとプランを理解し、ポーリングせずにお金のことを知れるようになりました。

次へ：第6部、第28章。お店を組み立てます。請求、アドレス、支払い待ち、承認。

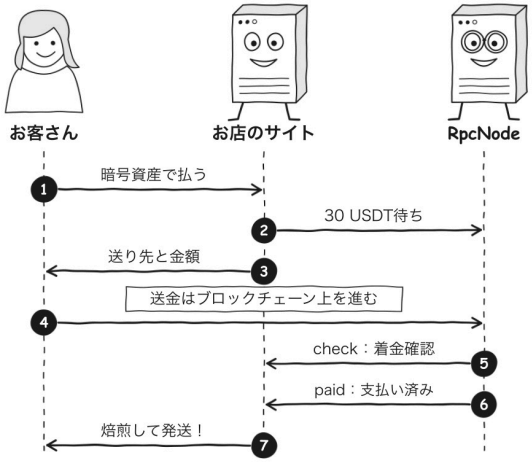
第28章　お店を組み立てる：請求、アドレス、支払い待ち、承認

「一つずつならわかる。全部合わせると？」

アルチョムは、アドレスも、承認も、Webhookも、お支払いも知っています。さて、お客さんが「暗号資産で支払う」を押しました。このあとサイトは、順を追って何をすればよいのでしょう。

解説：登場人物

登場するのは4者です。自分のウォレットを持つお客さん。注文データベースを持つお店のサイト。ネットワークを見張るRpcNode。そして鍵が入っているアルチョムのウォレット。サイトとサービスの会話に、ウォレットは加わりません。

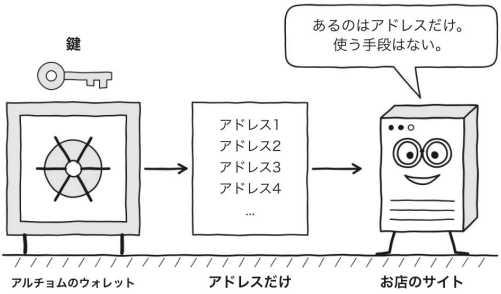


ステップ4を見てください。お金はサイトにもRpcNodeにも行きません。アルチョムのアドレスに直接入り、RpcNodeはブロックの中でそれに気づくだけです。

サーバーに鍵を置かずにアドレスを用意するには

第6章で、注文ごとに専用のアドレスを、と約束しました。同じ章で、秘密鍵はお店のサーバーに置くものではない、とも言いました。どう両立させるのでしょう。

アドレスは、鍵とは別に用意できます。



方法は2つあります。

	アドレスの一覧	拡張公開鍵
サーバーに置くもの	できあいのアドレス。注文ごとに1つ	鍵1つ（多くは `xpub` で始まる）。プログラムがそこからアドレスを計算する
使える場所	どのネットワークでも	Bitcoin、Ethereum、TRONとその仲間。SolanaとTONではこの方法は使えない
サーバーが破られたら	一覧のアドレスが見える	あなたの全アドレスと、そこへの全入金が見える
手間	切らさないように補充する	一度設定するだけ

この鍵でお金を使うことはできません。ただしBIP-32規格は警告しています。この鍵と、同じ枝の秘密鍵が一つでもそろうと、枝のすべての秘密鍵を計算できてしまいます。ですから、そういうウォレットの鍵を個別に書き出さないでください。初心者は、一覧から始めるほうが簡単です。

一覧は、鍵がインターネット上のサーバーに置かれていないウォレットから取ります。ハードウェアウォレットが便利です。鍵を内部に保管する専用の機器です。読み込む前に、いくつかのアドレスを機器の画面と照らし合わせてください。またTONでは、お客さんに `UQ` 表記のアドレスを示します（第19章）。

やってみよう：データベースは取り違えを許さない

アドレスを配るのはデータベースの仕事です。おもちゃの例を作ります。空きアドレス2つと、注文3件です。`sqlite3` というプログラムが必要ですが、多くのシステムに最初から入っています。

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('アドレス-1'), ('アドレス-2');

UPDATE addresses SET order_id = 'coffee-1001' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1002' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1003' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
SELECT '3件目に渡すアドレスがない' WHERE changes() = 0;

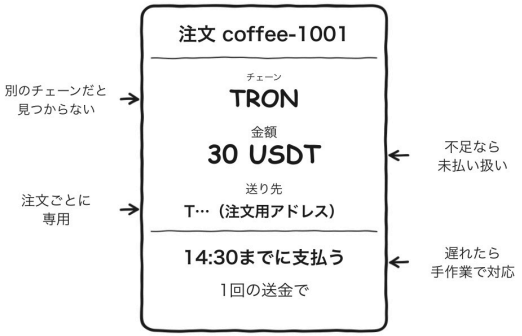
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'アドレス-2';
SQL
```

```
coffee-1001|アドレス-1
coffee-1002|アドレス-2
3件目に渡すアドレスがない
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

3件目の請求は出せません。一覧を補充する時期です。最後の命令は注文1001に2つ目のアドレスを与えようとしましたが、データベースが断りました。見張っているのは`PRIMARY KEY`と`UNIQUE`という言葉です。アドレスは一度しか登録されず、注文が持てるアドレスは1つまでです。

そして古いアドレスを二巡目に配らないでください。前の注文への遅れた送金が、新しい注文の支払いになってしまいます。

請求を出す



お客さんに必要なものは5つです。ネットワーク、コイン、正確な金額、アドレス、期限。そしてお願いを一つ。全額を1回の送金で送ってもらいます。半分ずつの2回は合算されません。また、ウォレットや取引所が送金額そのものから手数料を差し引くと、届く額が少なくなることも伝えておきます。

ルーブルやドルの価格は、注文の時点でコインに換算し、期限まではそのレートを保ちます。そのため期限は短くします。たとえば30分です。

次に必要なのは、第27章のお支払いです。以下はPythonの雛形です。中の `rpcnode` 関数は、署名付きリクエストを送ります。第27章を参考に自分で書くか、RpcNodeの公式クライアント（Python、TypeScript、PHP、Go、Kotlin向けがあります）を使ってください。データベースは同じく2つの表です。

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # TRONのUSDTコントラクト
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — 最初は 'waiting')

def take_address(db, order_id):
    with db:                                    # データベースの1トランザクション
        db.execute('INSERT OR IGNORE INTO orders(id) VALUES (?)',
                   (order_id,))
        mine = db.execute('SELECT address FROM addresses'
                          ' WHERE order_id=?', (order_id,)).fetchone()
        if mine:
            return mine[0]                      # 2回目以降は同じもの
        free = db.execute(
            'UPDATE addresses SET order_id=? WHERE address='
            ' (SELECT address FROM addresses WHERE order_id IS NULL'
            '  ORDER BY address LIMIT 1) RETURNING address',
            (order_id,)).fetchall()
        if not free:
            raise RuntimeError('アドレスがなくなりました')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount は文字列
    address = take_address(db, order_id)
    reply = rpcnode('POST', '/v1/payments', {
        'order_id': order_id, 'network_slug': 'tron',
        'address': address, 'amount': amount,
        'token_contract': USDT, 'ttl_seconds': 1800})
    if not reply['ok']:
        raise RuntimeError(reply['error_code'])
    payment = reply['payment']
    with db:
        db.execute('UPDATE orders SET payment_id=?, amount_raw=?'
                   ' WHERE id=?', (payment['payment_id'],
                   payment['amount_raw'], order_id))
    return payment        # お客さんへ: address, amount, expires_at
```

ここで大事なのは順番です。サービスに問い合わせる前に、アドレスを注文に割り当てます。接続が切れても、呼び直せば同じアドレスと注文番号が使われ、サービスは同じお支払いを返します。この動きは、でっち上げの `rpcnode` 関数で確かめました。本物のサービスでは、この雛形を実行していません。

お支払いは、買い物かごができるたびではなく、お客さんが暗号資産払いを選んだときに作ります。プランの月間上限は、作成されたお支払いを、未払いのものも含めて数えます。

RpcNodeは、アドレス、コイン、金額でお支払いを見分けます。第19章のコメントや第18章の `reference` の目印は、ここでは使いません。お支払いにそういうフィールドはありません。第20章のXRP LedgerとStellarは、お支払いに対応するネットワークの一覧に入っていません（第26章）。

支払いを待つ

注文ページは数秒ごとにあなたのサーバーに聞き、サーバーは自分のデータベースから答えます。データベースはWebhookが更新します。ブラウザーから直接RpcNodeに聞いてはいけません。キーのシークレットがお客さんの手に渡りますし、APIの上限はアカウントごとに毎分100リクエストです。

ステータス	お客さんに見える表示	お店がすること
`pending`	「14:30まで送金をお待ちします」	何もしない
`check`	「送金を確認しています。承認数 3 / 19」	発送しない
`aml_check`	「送金を審査しています」	待つ
`paid`	「お支払いを確認しました。ご注文を準備します」	発送する
`expired`	「期限が切れました。この請求には送金しないでください」	請求を閉じる
`failed`	「送金は成立しませんでした」	最初からやり直すよう案内する
`cancelled`	「請求は取り消されました」	請求を閉じる

送金がブロックに入るまで、お支払いは何も言いません。だから「未払い」ではなく「お待ちしています」と表示します。

承認をいくつ待つか

「設定」→「承認数」で、ネットワークごとに指定します。指定しなければ、ネットワークの標準値が使われます。必要な数は、お支払いの `required_confirmations` フィールドで確認できます。

標準より少なくするのは危険です。新しいブロックは置き換わることがあります（第8章）。しかも数はネットワークごとに1つで、注文ごとではありません。コーヒー1杯より豆1袋で長く待ちたいなら、`paid` のあとさらに待って、第27章のように自分で確かめます。

場所を空ける

「場所」は2種類あります。お支払いのスロットは、お支払いが `paid`、`expired`、`failed`、`cancelled` になると自動で空きます。アドレスのスロットは自動では空きません（取り消しの場合は例外です。第27章）。

Starterでは、未完了のお支払い10件、監視アドレス15個、月200件のお支払いです。11件目の未完了のお支払いは、`payment.slot_limit` で拒否されます。アドレスが全部ふさがると、`watch.address_limit` が返ります。

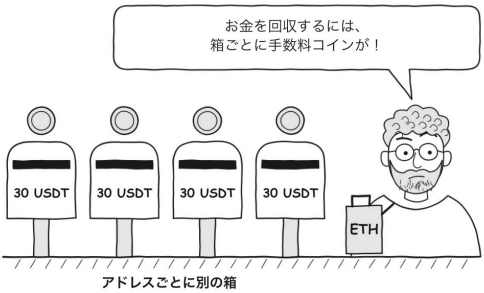
完了した注文は、送金が遅れて来る場合に備えて、少し寝かせておきます。そのあと監視を外します。`GET /v1/address-watches?network_slug=tron&address=…` で `watch_id` を探し、

`DELETE /v1/address-watches/{watch_id}` を送ります。以後そのアドレスの知らせは来ませんが、お金はまだ届くことがあります。

アドレスの追加と削除にも上限があります。1日、1週間、1か月の単位で、スロットが多いほど余裕があります。注文ごとに新しいアドレスを使うと、この余裕を消費します。セールの日には、スロットが空いていても `watch.quota` が返ることがあります。注文が多くなりそうなら、上限を前もって確かめてください。

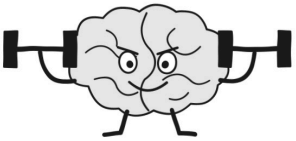
落とし穴：100個のアドレスに散らばったお金

注文ごとのアドレスは管理には便利ですが、お金はばらばらに置かれることになります。あるアドレスからUSDTを送り出すには、そのアドレスに手数料用のコインが必要です（第9章と第17章）。EthereumならETH、TRONならTRXかエネルギーです。注文100件なら、別々の送金が100回です。Bitcoinでも、小さな「かけら」が多いと送金が高くつきます（第14章）。少額の会計に使うネットワークを選ぶ前に、これを計算してください。



もう一つ。BIP-44規格のウォレットは、空のアドレスが20個続くと探すのをやめます。放置された注文が多いと、遠くのアドレスへの支払いがウォレットに表示されないことがあります。お金は無事ですが、その先まで探せる機能があるか確かめてください。

考えてみよう



頭の体操をしよう！

1. お客さんが支払いページを再読み込みしました。新しいアドレスと新しいお支払いは必要でしょうか。
2. Starterのアルチョムには、すでに未完了のお支払いが10件あります。11人目のお客さんが来たらどうなりますか。
3. 支払いページがブラウザーから直接RpcNodeに聞いています。困ることを2つ挙げてください。

答え：1）いいえ。同じ注文番号と同じアドレスを使います。2）`payment.slot_limit` で拒否されます。少し待ってもらうか、別の支払い方法を案内します。3）キーのシークレットがお客さんに見え、リクエストの上限もすぐに尽きます。

まとめ

- 鍵はウォレットに残し、サイトにはアドレスだけ。1注文に1アドレスで、使い回しません。
- お客さんはあなたのサイトに聞き、サイトはWebhookで知らせを受けます。
- スロットと、お金を集める費用は、前もって計算します。

次へ：第29章。何もかもうまくいかないとき：エラー、再試行、詰まったトランザクション、二重の入金処理。

第29章　何もかもうまくいかないとき：エラー、再試行、詰まったトランザクション、二重の入金処理

「コーヒーが2回発送された！」

朝、アルチョムは、1件の注文に2つの包みが送られたことに気づきました。別のお客さんは「払ったのに、そちらはずっと『お待ちしています』のままです」と書いてきます。奇跡ではありません。好天の日しか想定せずに書いたプログラムでは、こうなります。

解説：困りごとは3種類

困りごと	例	処方
あなたのリクエストに答えがなかった	お支払いの作成中に接続が切れた	害のない形でやり直す
知らせの届き方がおかしい	2回届く、順番が違う、まったく届かない	データベースの鍵と突き合わせ
送金がうまくいかなかった	詰まった、遅れた、不足した	ルールと手作業の対応

APIへのリクエストをやり直すとき

間隔を延ばす方法は第23章で見ました。ただし、安心してやり直せるのは冪等（べきとう）なもの、つまり何回行っても1回行ったのと結果が同じものだけです。お支払いの作成は、`order_id`のおかげで冪等です。同じ番号と同じデータでやり直します。署名には時刻が含まれるので、やり直すたびに署名し直します。

RpcNodeの公式クライアントは、接続の切断や500番台のあと、`POST`を自動ではやり直しません。ですからお支払い作成の再試行は、あなたの仕事です。

APIのさまざまな答えへの対処です。

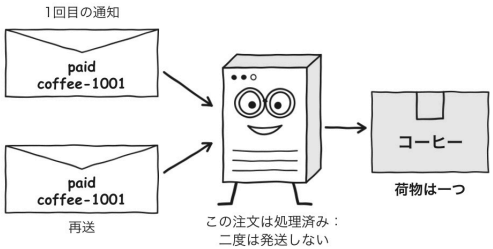
返ってきたもの	どうするか
何もない。502、503、504	間隔をあけてやり直す
`Retry-After` ヘッダー付きの429	指定された時間だけ待つ
429 `watch.quota`（第28章）	すぐにやり直しても無駄。余裕を使い切った
400	リクエストの誤り。`errors` に該当フィールドが書かれている
401	キー、署名、サーバーの時計を確かめる
403	プラン、スロットの上限、またはIP許可リスト
409 `payment.order_exists`	この番号は、別の内容のお支払いで使用済み。当てずっぽうに番号を変えず、原因を調べる

お支払いが期限切れになり、お客さんがもう一度試したい場合は話が別です。古い `order_id` では、期限切れの同じお支払いが返ります。新しい番号が必要です。たとえば `coffee-1001-2` です。

一つの知らせが2回

重複はあちこちから生まれます。

- あなたが10秒以内に応答しなかったので、Webhookが再送された。
- Webhookとソケットの両方を有効にしている。
- 同じお金について `payment` と `payment_order` が届いた（第27章）。
- Webhookと、あなたの定期確認が同時に動いた。
- 担当者が手で「発送」を押した。



配信番号（第26章）は、同じ配信の繰り返しを除きます。しかし、同じお金についての知らせが別の経路で届く場合は防げません。いちばんの鍵は注文の状態です。「待ち」から「支払い済み」への移行は、一度だけしかできないようにします。

やってみよう：同じWebhookを2回

これが鍵の全体です。知らせは私たちがでっち上げました。2つの部分を`once.py`という一つのファイルに保存し、`python3 once.py`で実行してください。

```
import sqlite3
db = sqlite3.connect(':memory:')
db.executescript("""
CREATE TABLE orders(id TEXT PRIMARY KEY, payment_id TEXT,
                    amount_raw TEXT, status TEXT);
CREATE TABLE to_ship(order_id TEXT PRIMARY KEY);
INSERT INTO orders VALUES('coffee-1001','p1','30000000','waiting');
""")

def on_news(n):
    if n['type'] != 'payment_order' or n['event'] != 'paid':
        return 'まだ支払いではない'
    with db:                # この中は全部でデータベースの1トランザクション
        cur = db.execute(
            "UPDATE orders SET status='paid' WHERE id=? AND"
            " payment_id=? AND amount_raw=? AND status='waiting'",
            (n['order_id'], n['payment_id'], n['amount_raw']))
        if cur.rowcount != 1:
            return '処理済みか、対象外'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return '発送する'
```

続いて、4つの知らせを立て続けに送り、プログラムの答えを見ます。

```
paid = {'type': 'payment_order', 'event': 'paid',
        'order_id': 'coffee-1001', 'payment_id': 'p1',
        'amount_raw': '30000000'}
print(1, on_news({**paid, 'event': 'check'}))
print(2, on_news(paid))
print(3, on_news(paid))                       # 同じWebhookをもう一度
print(4, on_news({**paid, 'event': 'check'}))  # 遅れて来た知らせ
print('荷物の数:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

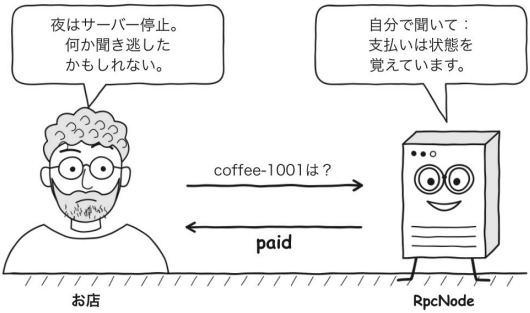
```
1 まだ支払いではない
2 発送する
3 処理済みか、対象外
4 まだ支払いではない
荷物の数: 1
```

荷物は1つです。すべてを支えているのは一つの命令です。「`paid`にするのは、注文がまだ待ちで、お支払いと金額が一致する場合にかぎる」。この印と発送の指示は一緒に書き込まれます。両方か、どちらもなしです。これをデータベースのトランザクションといいます（ブロックチェーンと同じ言葉ですが、別のものです）。同じ知らせを8つ同時に送る試験もしました。発送は1回でした。

第26章の雛形では、配信番号はメモリー上にあり、作業の前に記憶していました。お金を扱うときはそれではいけません。プログラムが途中で落ちると、すべてが混乱します。まずデータベースに書き込み、そのあとで200を返します。

知らせがまったく来なかったら

Webhookは7回の試行で打ち切られます。約7時間です。ソケットは逃した分をまったく再送しません。つまり、知らせの一部は失われる可能性があります。処方は突き合わせです。まだ開いたままの注文について、自分から聞きます。



`GET /v1/payments/orders/{orderId}` リクエストは、お支払いを返します。項目名が少し違い、`event` の代わりに `status` を見て、ネットワークは `network_slug`、トークンのコントラクトは `token_contract` という名前です。答えから同じ形の「知らせ」を組み立て、同じ鍵を通します。

突き合わせは、サーバーが停止したあと、ソケットを接続し直したあと、そして定期的に行います。未完了の注文10件を1分ごとに確認すると、許される100件のうち10リクエストです。

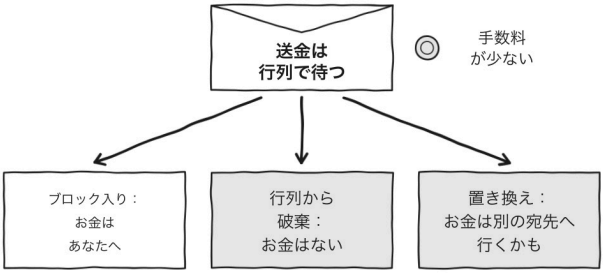
送金が詰まった

お客さんがハッシュを送ってきました。「1時間前に送りました」。お支払いは何も言いません。そのトランザクションを知っているか、ノードに聞いてみます。ここのハッシュはでっち上げです。

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash","params":
  ["0x0000000000000000000000000000000000000000000000000000000000000001"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":null}
```

`null` は「このノードはそれを知らない」という意味です。理由はさまざまです。打ち間違い、別のネットワーク、送金がまだ届いていない、あるいはもう捨てられた。トランザクションが見つかっても `blockNumber` が `null` なら、待ち行列で待っています。



送金がブロックに入るまで、お金はない

ネットワーク	長く待つ送金に起こること
Ethereumとその仲間	送り主の後続の送金をすべて止める（第15章）。同じ `nonce` で手数料を増やした送金に置き換えられる。ノードプログラムのgethは、初期設定では10%以上の上乗せを求める
Bitcoin	送り主は、手数料を増やした送金に置き換えられる。Bitcoin Coreは初期設定で、2週間残ったものを待ち行列から捨てる
TRON	トランザクションには有効期限があり、初期値は1分。ブロックに間に合わなければ、もう入らない
Solana	ふつうのトランザクションは、参照するブロックが新しい間だけ有効。150ブロック、ほんの数十秒

他人の送金をお店が速めることは、ふつうできません。送り主の鍵が必要だからです。ブロックを待ち、発送しないでください。その間にお支払いが期限切れになり、それでも送金が通った場合、お金はあなたのアドレスに入りますが、お支払いは `expired` のままです。

詰まったのがあなたの送金、たとえば返金だったら？　同じ署名済みの送金は、もう一度送れます。ハッシュは変わりません。一方、新しい送金は、最初の送金を置き換えるのでなければ、2回目の支払いになります。Ethereumでアドレスに待ち行列があるかは、こうして確かめられます。

```
URL=https://public-ethereum-mainnet.rpcnode.dev
A=0x8f8d1206d1bce12ff892731f8a140ed18b93884e
for tag in latest pending; do
  curl -s $URL -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0",
    "id":1,"method":"eth_getTransactionCount","params":["'$A'","'$tag'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
```

1つ目の数はそのアドレスの送金のうちブロックに入った数、2つ目は待機中も含めた数です。同じなら、このノードには待ち行列が見えていません。これは第15章の送り主のアドレスです。

ハッシュの監視（第27章）にも注意してください。`cancelled` という答えは、サービスがトランザクションを見つけられなかったという意味で、今後も現れないという意味ではありません。

お金は来たが、様子が違う

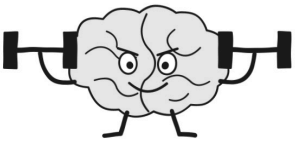
起きたこと	お支払いはどうなるか
不足していた	支払い済みにならず、期限で切れる。お金はもうあなたの手元にある
2回に分けて届いた	合算されないので、同じ結果になる
多く届いた	`paid` 。実際の額は `received_amount_raw` に
`paid` のあとにもう一度送られた	お支払いはもう待っていない。知る手段はアドレスのイベントくらい
期限のあと	`expired`
違うトークン、違うネットワーク	数えられない（第16章）
トランザクションが失敗した	お金はないので発送しない。お支払いは `failed` になるか、単に期限切れになる

それぞれの場合にどうするかを前もって決め、支払いページでお客さんに知らせておきましょう。

落とし穴：「来たところに返そう」

お客さんが払いすぎたので、アルチョムは余りを送り主のアドレスに返そうとします。しかしそれは、取引所の共用アドレスかもしれません。お金を受け取るのは、お客さんではなく取引所です。返金先のアドレスとネットワークは、お客さんに聞いてください。送るのは、サイトからではなく、自分の手でウォレットから行います。

考えてみよう



頭の体操をしよう！

1. お支払い作成の答えが返ってきませんでした。やり直せますか。どの `order_id` で？
2. 同じ `paid` がWebhookとソケットの両方で届きました。例のプログラムは荷物をいくつ発送しますか。
3. 30 USDTの代わりに、15ずつの送金が2回届きました。30のお支払いはどうなりますか。

答え：1) はい、同じ番号と同じデータで。2) 1つ。3) 支払い済みになりません。送金は合算されないからです。

まとめ

- やり直すのは、やり直しても安全なものだけです。
- 発送はデータベースの鍵が守り、失われた知らせは突き合わせで見つけます。
- 送金がブロックにない間、お金はありません。おかしな支払いへのルールは前もって決めます。

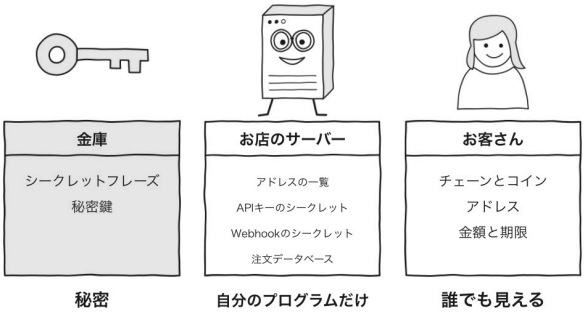
次へ：第30章。安全対策と、公開前のチェックリスト。

第30章　安全対策と、公開前のチェックリスト

「明日オープン。何を忘れている？」

テストネットでは全部動いています。アルチョムはもうお客さんに「暗号資産を受け付けます！」と知らせています。なのに、よく眠れません。お金が本物になったら、何がまずくなりうるのでしょう。

解説：何がどこにあるか



APIキーのシークレットがあっても、あなたのコインは使えません。RpcNodeに秘密鍵はないからです。それでも悪さはできます。キーに「許可IP」を設定しておけば、盗まれたシークレットは他人のコンピューターでは使えません。

2段階認証は、パスワードのほかにアプリのコードを必要にする仕組みです。マイページでは「設定」にあります。メールにも同じものを有効にしてください。ほかのすべてのアクセスは、メールを通じて取り戻せることが多いからです。

次のどれかが漏れたら、何が起きるか。

漏れたもの	他人にできること	対処
シークレットフレーズ、秘密鍵	お金を取り戻せない形で持ち去る	急いで残りを新しいウォレットに移す
APIキーのシークレット	お支払い、監視、エンドポイント、Webhookを操作し、そのシークレットを読む	キーを削除して新しく発行し、Webhookのシークレットを変え、設定を確認する
Webhookのシークレット	`paid` の知らせを偽造する	「新しいシークレットを生成」
エンドポイントのアドレス	あなたのクレジットを使う	新しく作り、古いものを削除する（第22章）
マイページへのログイン	この表のほぼ全部。あなたのコインは除く	パスワードを変え、2段階認証を有効にする

シークレットはコードに住まわせない

コードに書いたシークレットは、共有のコード置き場にも、外注先に渡すアーカイブにも、画面の写真にも付いていきます。シークレットはサーバーの設定（環境変数）に置き、そのファイルは部外者が読めないようにします。

やってみよう：自分のところを探す

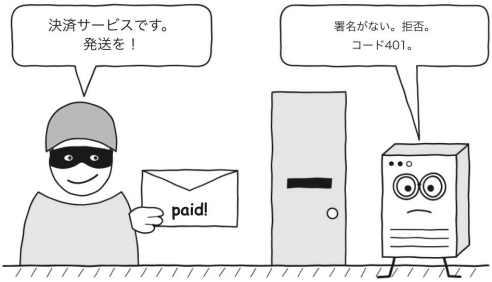
APIキーのシークレットは `sk_live_` で始まります。間違いが一つある練習用フォルダーを作り、見つけてみます。

```
mkdir demo
echo "const SECRET = 'sk_live_primer'" > demo/pay.js
echo "const SECRET = process.env.RPCNODE_API_SECRET" > demo/ok.js
grep -rnE 'sk_live_|PRIVATE KEY|xprv' demo
```

```
demo/pay.js:1:const SECRET = 'sk_live_primer'
```

検索は1つ目のファイルを見つけ、2つ目には反応しませんでした。`demo` を自分のプロジェクトのフォルダーに置き換えてください。`xprv` は拡張秘密鍵の先頭です。何か見つかったら、行を消すだけでは不十分です。変更履歴に残っているかもしれません。シークレットは漏れたものとみなして、新しく発行します。なお、シークレットフレーズはこの方法では見つかりません。ふつうの単語の並びだからです。そもそもコンピューターに置かないでください。

届くものを信じない



自分のドアを確かめます。第26章の受信側の雛形を `receiver.mjs` に保存して実行してください。

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

別のターミナルの窓から、他人のふりをして3通りの方法でノックします。

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) 署名なし
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) でたらめな署名
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$(date +%s),v1=$(printf 'a%.0s' {1..64})"
# 3) 本物の署名だが、1時間前のもの
T=$(( $(date +%s) - 3600 ))
SIG=$(printf '%s.%s' $T "$BODY" |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}')
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$T,v1=$SIG"
```

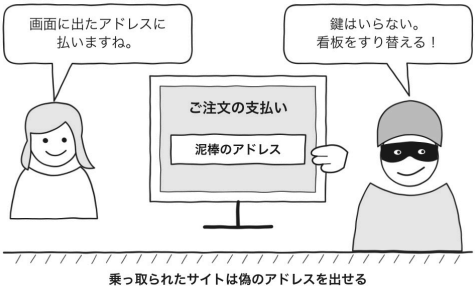
```
401
401
401
```

3回とも「通しません」です。どこかで200が返ったら、ドアは開いています。`- 3600` を `- 0` に替えると、3回目のノックは200になります。署名が新しくて正しいからです。

高額の注文には、2つ目の鍵を掛けます。`paid` を受け取ったら、自分からお支払いを聞き（`GET /v1/payments/orders/{orderId}`）、`payment_id`、アドレス、トークン、金額、ステータスを照らし合わせます。そうすれば、Webhookのシークレットが盗まれても、ただでコーヒーは手に入りません。

さらに、Webhookは `https` で受け、届くリクエストの大きさを制限してください。第26章の雛形はそれをしていません。加えて、「シークレットとIP」セクションのアドレスだけを通すこともできます。これは署名への上乗せであって、代わりにはなりません。

すり替えられた看板



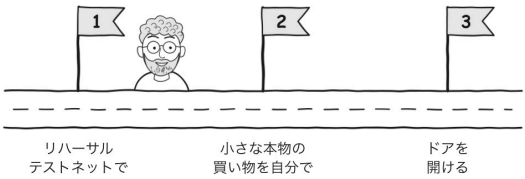
乗っ取られたサイトは偽のアドレスを出せる

泥棒は、鍵を手に入れる必要がありません。サイトに侵入して、お客さんに自分のアドレスを見せれば十分です。役立つ対策はこうです。

- サイトの管理画面から、アドレスの一覧をこっそり差し替えられないようにする。
- ときどき、別のコンピューターから、配ったアドレスを自分のウォレットと照らし合わせる。
- たまに自分の店で買い物をして、どこへ払えと言われるかを見る。
- サーバーのプログラムを更新し、管理画面には2段階認証でログインする。

泥棒はあなた個人にも連絡してきます。「サポートです」「止まった支払いを戻してください」、よく似たアドレスからの小さな送金（第6章）。本物のサポートはシークレットフレーズを聞きません。

リハーサル



まず全部をテストネットで。テストネットのお支払いとイベントは、TRON Nileで動きます（第26章）。Freeにはサンドボックスがあります。初期値は、未完了のお支払い3件とアドレス3個で、そこのリスク確認は本物ではありません。サンドボックスでも、アドレスは自動では空きません。

何を演じるか	どうなるべきか
ぴったり払う	荷物は1つ
少なく払う	何も発送されず、お支払いは期限切れ
同じ知らせを自分に2回送る	荷物は1つ
自分のサーバーを止め、支払い、また動かす	Webhookの再送か突き合わせで、支払いが見つかる
署名なしでノックする	401

それから、実際に商売するネットワークで、自分から小さな本物の買い物を一度します。ついでに、ウォレットにお金が見えるか、そして先へ送り出せるかを確かめます。手数料用のコインは足りていますか。

法律と税金

多くの国で、暗号資産の受け取りは制限されているか、課税されます。公開前に、自分の国の法律を確認してください。これは法的な助言ではありません。会計担当には、支払いのどんな記録が必要かも聞いて、初日から残しておきます。

落とし穴：「一度確認して、それきり」

お店は動いていても、周りは変わります。プランの期限が切れると、アカウントはFreeに移り、メインネットのお支払いはなくなります（第25章）。クレジットはイベントにも使われます。アドレスの一覧は残り少なくなります。自分でリマインダーを設定しておきましょう。

公開前のチェックリスト

鍵とアクセス

- □ シークレットフレーズは書き留めてあり、どのサーバーにもコンピューターにも置いていない。
- □ サイトにあるのはアドレスだけで、いくつかはウォレットと照合した。
- □ コードにシークレットがなく、プロジェクト内の検索結果は空。
- □ APIキーに許可IPを設定した。
- □ マイページ、メール、サイトの管理画面で2段階認証を有効にした。

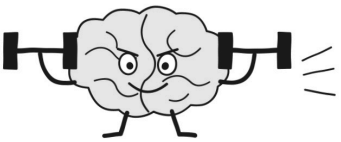
お金の受け取り

- □ 1注文に1アドレスで、使い回しがない。
- □ 請求には、ネットワーク、コイン、金額、アドレス、期限、「1回の送金で」が書いてある。
- □ トークンのコントラクトは`GET /v1/network-tokens`から取り、発行元のサイトと照合した。
- □ 承認数は、自分の扱う金額に合わせて選んだ。
- □ Webhookの署名を確かめ、発送はデータベースの鍵が守っている。
- □ 未完了の注文の突き合わせが定期的に動いている。

周辺の仕事

- □ 不足、払いすぎ、遅れ、返金のルールがある。
- □ スロット、クレジット、プランの期限、アドレスの残りを見張っている。
- □ お金を集めるときの手数料を何で払うか、わかっている。
- □ 注文データベースのバックアップを取り、一度は復元してみた。
- □ リハーサルを終え、自分の本物の買い物を済ませた。
- □ 自分の国の法律と税金を確認した。

考えてみよう



頭の体操をしよう！

1. APIキーのシークレットが漏れました。ウォレットのコインは盗まれるでしょうか。それでも直ちにすべきことは何ですか。
2. 受信側が、署名のないメッセージに200を返しました。何が危険ですか。
3. TRON Nileでの確認はすべて通りました。新たな確認なしで、Ethereumのお支払い受付を始めてよいでしょうか。

答え：1）いいえ、そこにウォレットの鍵はありません。ただし、APIキーを削除して新しく発行し、Webhookのシークレットを変え、設定を確認します。2）誰でもお金なしで注文を「支払い済み」にできます。3）いいえ。ネットワークも、アドレスも、手数料も、承認数も違います。

まとめ

- お金の鍵はサイトから遠くに、シークレットはコードから遠くに置きます。
- 届いたメッセージも、画面の写真も、メールも、うのみにしません。
- まずリハーサル、次に小さな本物の買い物、それからオープンです。

第6部はここまでです。アルチョムはお店を組み立て、壊れたときの対処と、守り方を知りました。

次へ：第31章。DeFi：仲介者のいない金融。

第31章　DeFi：仲介者のいない金融

「コインは寝かせておくしかないの？」

アルチョムの手元には、売上の一部がUSDCでたまりました。知人が勧めます。「DeFiに入れなよ。お金がお金を生むんだ」。窓口も出納係もないこの場所は、何なのでしょう。

> これは金融の助言ではなく、説明です。プロトコルの名前は例にすぎません。あなたの国では、こうしたサービスが制限されていたり、課税されたりすることがあります。法律を確認してください。

解説：銀行とプログラム

DeFi（分散型金融）は、交換、貸し借り、そのほかのお金のサービスを、スマートコントラクト（第9章）が実行するものです。共通のルールを持つコントラクトの一組をプロトコルと呼びます。

	銀行	DeFi
お金のある場所	銀行の中	アドレスかコントラクトの中
誰が実行するか	職員と銀行のシステム	ネットワーク上のプログラム
いつ動くか	銀行の営業時間	24時間
ルールのある場所	契約書	コード
間違えたら	銀行に相談できる	ふつう、取り消してくれる人はいない
アクセスを失ったら	身分証が助けになる	鍵がなければ戻らない

「仲介者がいない」は、「信用がいらない」という意味ではありません。コード、コードを変えられる人、価格の提供者、コインの発行元を信用することになります。

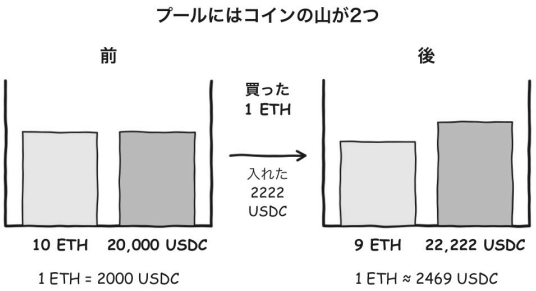
取引所なしの交換：流動性プール

ふつうの取引所では、買い手のために売り手を探します。DeFiでは、よく流動性プールを使います。2種類のコインを一緒に入れたコントラクトです。プールに一方を入れて、もう一方を取り出します。相手の人を待つ必要はありません。

いくら受け取れるかは、式で決まります。単純なプールの式は$x \times y = k$で、xとyは2つのコインの残高です。交換によって、この積が減ってはいけません。

ここでの価格は、残高の比です。プールに10 ETHと20,000 USDCがあるとします。つまり1 ETHは約2000 USDCで、$k = 200{,}000$です。アルチョムが1 ETHを取り出したいとします（手

数料はまず考えません）。残りは9 ETHなので、USDCは200,000÷9≒22,222にならなければなりません。



彼が払うのは2000ではなく2222 USDCです。自分の買いが価格を押し上げたのです。これを価格への影響（プライスインパクト）といいます。100倍大きいプール（1000 ETHと2,000,000 USDC）なら、同じ1 ETHは約2002 USDCで済みます。

プール内のETHがほかの場所より高くなると、誰かがそこへETHを持ち込んでUSDCを持ち出し、価格が近づくまで続けます。これが裁定取引（アービトラージ）です。

プールにコインを入れるのは流動性提供者で、手数料の分け前を受け取ります。しかし価格が開くと、プールは値上がりしたコインを手放し、もう一方を集めます。手数料を除くと、価格が2倍になった場合、預けた分の価値は、ただ持っていた場合より約5.7%少なくなります。これをインパーマネントロス（一時的な損失）といいます。そのときに引き出せば、損失はすっかり恒久的になります。

やってみよう：本物のプールに聞く

EthereumのUniswap v2プロトコルにある、USDCとWETHのプールを取り上げます。WETHは、ETHをトークンの形にしたもので、1対1です。

プールのアドレスは、適当な検索で拾ってはいけません。私たちは、プールの「工場」（ファクトリー）のアドレスをUniswapのドキュメントから、USDCのアドレスを発行元のCircle社から、WETHのアドレスをethereum.orgから取りました。そしてファクトリーに`getPair`で「この組のプールはどこ？」と聞きました。答えは`0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`です。

次は`getReserves`、「残高はいくら？」という質問です。第9章の`eth_call`なので、読むだけです。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
   "data":"0x0902f1ac"},"latest"]}'
```

2026年11月27日の夜、ノードはこう答えました。

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000000079ecc8266bc00000000000000
0000000000000000000000000000000010b762ae141ddef4f50000000000000000000000000000000000000000000000000000000006b08b4a
f"}
```

`0x`のあとに、64文字の数が3つ並んでいます。

数（左のゼロを除く）	何か
`79ecc8266bc`	USDCの残高、小数点以下6桁：8,378,617 USDC
`10b762ae141ddef4f50`	WETHの残高、18桁：4933.80 WETH
`6b08b4af`	残高が最後に変わった時刻（1970年からの秒数）

どちらのコインが先かは、`token0`と`token1`の質問でわかります。このプールの1番目はUSDCです。あとは割り算です。

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

`8378617.325244 4933.79553339896 1698.2092728662094`と表示されます。

このプールでは、1 ETHは約1698 USDCでした。あなたの数字は違うはずです。

大口ならいくらもらえるでしょう。ここの交換手数料は0.3%で、0.25%が提供者に、0.05%がプロトコルに入ります。同じ残高で計算すると、こうなります。

渡す量	受け取るUSDC	1 ETHあたり	価格1698.21より不利な幅
1 ETH	1692.77	1692.77	0.3%
100 ETH	165,957.86	1659.58	2.3%
1000 ETH	1,408,492.58	1408.49	17%

この計算は、Uniswapの補助コントラクト（router）に`getAmountsOut`で聞いた答えと、最後の桁まで一致しました。

重要：残高の割り算が使えるのは、v2型のプールです。Uniswap v3とv4では、預けられた資金が価格の帯ごとに分かれていて、価格の計算方法が違います。

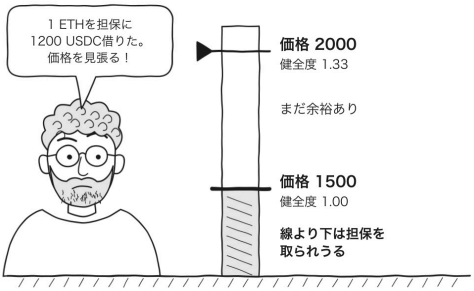
担保を取る貸し借りと清算

貸し借りのプロトコル（例はAave）では、担保を入れて、その価値より少ない額を借ります。何のためでしょう。たとえばドルが必要だが、ETHは売りたくない場合です。

借り入れには「健全度」（health factor）があります。

> 健全度 = 担保の価値 × 清算しきい値 ÷ 借入額

架空の例です。担保は1 ETH（1枚2000 USDC）、しきい値は80%、借入は1200 USDC。健全度は2000×0.8÷1200≒1.33です。価格が1500に下がると、ちょうど1になります。



健全度が1を下回ると、誰でも借入の一部、ときには全部を返済し、その代わりに担保を割増付きで受け取れます。これが清算です。「お金を追加してください」という電話は来ません。すべてプログラムが行います。真夜中でもです。借入は、利息で自然にも増えていきます。

ステーブルコイン：「1ドル」を支えるもの

第9章のとおり、ステーブルコインは約1ドルの価値を保とうとするだけです。支え方はさまざまです。発行元の準備資産：ドルなどの資産（USDT、USDC）。コントラクト内の余裕を持たせた暗号資産担保（DAI、USDS。担保にほかのステーブルコインが含まれることもあります）。担保なしでコインの数を増減させるアルゴリズム。

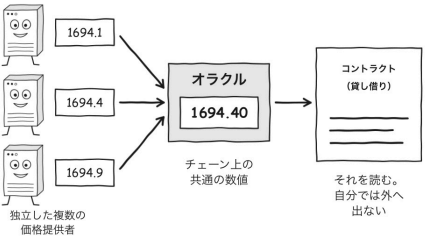
どの方法も保証ではありません。プール内でそうしたコインが値下がりすると、提供者の手元には、まさにそのコインがたまります。

ステーキングを手短に

第8章のバリデーターには32 ETHが必要です。それより少ない額では、共同ステーキングプールを通じて参加します（交換用のプールとは別物です）。引き換えに、引換券のトークンがもらえることがよくあります。危険は、バリデーターへの罰則、コントラクトの誤り、運営者への信用です。収益の大きさは、誰も約束しません。

オラクル：コントラクトはどこから価格を知るのか

貸し借りのコントラクトには、ETHの価格が必要です。しかし、自分でインターネットに出ていくことはしません。どのノードも同じ結果にならなければいけないからです（第15章）。価格をネットワークに書き込むのはオラクルです。



例はChainlinkのオラクルです。「ETH / USD」フィードのアドレスは、Chainlink自身の一覧から取りました。`latestRoundData`の質問は、最新の記録を求めます。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419",
   "data":"0xfeaf968c"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000000700000000000008e4f000000000000000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000006b08aefa000000000000000000000000000000000000000000000000000000006b08af0f000000000000000000000000000000000000000000000000000000000000000000008e4f"}
```

5つの数の2番目、`2773693800`が、小数点以下8桁の価格です。1694.40ドルです。プールは1698.21 USDCを示していて、差は約0.2%です。4番目の数は記録の時刻で、私たちが質問する25分前でした。このフィードは、価格が記録値から0.5%離れたときと、定期的に約1時間に1回、更新されます。

なぜプールから直接価格を取らないのでしょう。大きな取引で、一瞬だけ動かせるからです。ただし、オラクルも間違えたり遅れたりすることがあり、清算はその数値に従って行われます。

DeFiアプリはどうやってネットワークを読むか

交換や貸し借りのサイトは、店のショーウィンドウです。データはノードから取っています。

- `eth_call`は「今どうか」。プールの残高、各自の残高、オラクルの価格、借り入れの健全度。
- イベント（第15章）は「何が変わったか」。v2プールは、変化のたびに新しい残高を載せた`Sync`イベントを発します。`eth_getLogs`で探すか、WebSocketで聞きます（第11

章）。

- 送信。交換はノートを書き換えます。ウォレットが署名し、ノードが先へ伝え（第7章）、ガスがかかります。

これは、最新ブロックでの、このプールの `Sync` イベントについての質問です。

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
  "topics":["0x1c411e9a96e071241c2f21f7726b17ae89e3cab4c78be50e062b03a9fffbbad1"],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

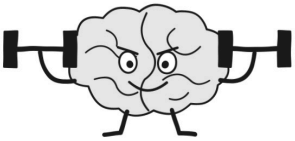
空の一覧も答えです。このブロックでは、プールは変化していません。`getReserves` の3番目の数がそれを裏付けます。最後の変化は、質問の約10分前でした。たとえば価格グラフ用の過去の変化となると、アーカイブか自分のデータベースの出番です（第10章）。

落とし穴：何がまずくなりうるか

危険	何が起きるか	危険を減らすもの
コードの誤り	お金が盗まれる、または動かせなくなる	コードの監査。ただし保証ではない
偽物のトークン	名前は知っているのに、アドレスは別	アドレスを照合する（第9章）
無制限の許可（`approve`）	後でトークンを持っていかれる	許可は少なく、不要なものはゼロに戻す
スリッページ	期待より少なく受け取る	最低受取額を指定する
清算	担保が割増付きで持っていかれる	大きな余裕と見守り
そっくりサイト	コピーサイトが署名を求める	アドレスを確かめる。シークレットフレーズは入力しない
「確実な利回り」	約束して、消える	参加しない

スリッページは、予想した価格と実際の価格の差です。交換が待ち行列で待つ間に、他人の取引が価格を動かします。ときには意図的にです。最低受取額が身を守ります。それより悪い条件では交換は成立しません。ただし、ガスは消費されます。

考えてみよう



頭の体操をしよう！

1. プールに100 ETHと200,000 USDCがあります。誰かが20 ETHを取り出しました。何USDC入れましたか（手数料なし）。
2. 担保は2 ETH（1枚2000 USDC）、しきい値80%、借入2400 USDCです。健全度はいくつですか。ETHがいくらになると1になりますか。
3. サイトが「1日3%、危険なし」と約束し、USDTを無制限に使う許可を求めています。どうしますか。

答え：1）残りは80 ETHなので、USDCは250,000になります。入れたのは50,000で、1 ETHあたり2500です。2）約1.33。価格1500で1になります。3）何にも署名せず、サイトを閉じます。

まとめ

- DeFiではコントラクトがサービスを実行し、危険と責任の多くはあなたが負います。
- v2型のプールでは、価格は2つの残高から決まり、大きな取引がそれを動かします。
- アプリは、あなたと同じリクエストでネットワークを読みます。`eth_call`とイベントです。

次へ：おわりに。お客さんの一通のメールから、自分のお店まで。

2026年11月27日に確認：developers.uniswap.org、developers.circle.com/stablecoins/usdc-contract-addresses、ethereum.org (wrapped-eth, defi, stablecoins, staking, developers/docs/oracles)、aave.com/help/borrowing/liquidations、docs.chain.link。

おわりに　お客さんのメールから、自分のお店まで

すべては、メールの中の一行から始まりました。「暗号資産で払えますか？」。アルチョムはそのとき、ノードが何かも知らず、ウォレットを手で確認していました。あなたが彼と一緒にたどった道のりを見てください。

アルチョムが今できること

部	学んだこと
1. なぜプロバイダーが必要なのか	自前サーバーの費用を計算し、ノードを自分で持つか、プロバイダーから借りるかを選ぶ
2. ゼロから学ぶブロックチェーン	ブロック、鍵、署名、手数料、承認、トークンを理解する
3. ノードとRPC	ノードに質問し、ノードの種類を区別し、遅れていないかを確かめる
4. ネットワークの仕組み	ネットワークの違いを見分け、お客さんに合うものを選ぶ
5. RpcNode	エンドポイントを手に入れ、呼び出しの値段を計算し、ポーリングせずに支払いを知る
6. 実践	支払いの受け付けを組み立て、障害を乗り越え、お店を守る
7. この先へ：DeFi	ネットワークからプールを直接読み、危険を正直に見積もる

覚えておきたい7つのルール

1. アドレスは誰に見せてもよく、秘密鍵とシークレットフレーズは誰にも見せない。
2. ネットワーク、トークンのコントラクトアドレス、受取人、金額を、まとめて全部確かめる。
3. お金が届いたと言えるのは、トランザクションが成功して承認がそろったときで、画面の写真が届いたときではない。
4. コード200はまだ成功ではない。答えの中身を読む。
5. 知らせが何回届いても、支払い1回につき発送は1回。
6. エンドポイントとシークレットは、パスワードのように大切にする。
7. うのみにしない。ネットワークに聞いて、答えを確かめる。

明日から始めること

1. 第3章の公開ノードへのリクエストを繰り返す。
2. 自分のエンドポイントとテストネットを用意する（第22章）。
3. おもちゃのお金で、自分自身への支払いを演じてみる（第28章と第30章）。

4. 第30章のチェックリストを通す。
5. 自分の国の法律と税金を確認する。

情報が古くなっていたら

この本は、2026年11月27日時点のネットワークとサービスを描いています。ネットワークは更新され、値段やプランは変わります。あなたのノードの答えが本と違っていたら、ノードと最新のドキュメントを信じてください。RpcNodeのプランと機能は、サイトrpcnode.devで確認してください。

さようなら

売上の一部をDeFiに入れるかどうか、アルチョムはまだ考え中です。その代わり、何を聞けばよいかは知っています。それがおそらく、この物語全体から彼が持ち帰った一番のものです。

この本の数字の一部は古くなりますが、確かめる習慣は古くなりません。あなたと、あなたの商いに幸運を！



付録A　用語集

かっこ内は、その言葉を詳しく説明した章の番号です。

AML：お金が不審な経歴とつながっていないかを調べる確認（27）。

API：あるプログラムが別のプログラムに用件を伝える窓口（11）。

Chain ID：EVMネットワークの番号（16）。

DeFi：スマートコントラクトが実行するお金のサービス（31）。

EVM：コントラクトを実行する、Ethereumの仮想マシン（15）。

gRPC：プログラム同士が、厳密な定義に沿った小さなメッセージでやり取りする方法（11）。

L2、「2階」：送金を自分のところで実行し、その記録をEthereumに送るネットワーク（16）。

nonce（ノンス）：アカウントのトランザクションの通し番号（15）。

REST：質問ごとにアドレス内のパスが違う聞き方（11）。

RPC、JSON-RPC：別のコンピューターのプログラムに仕事を頼む方法（11）。

UTXO：Bitcoinとその親戚にある、未使用の「お金のかけら」（14）。

Webhook：何かが起きたとき、サービスのほうからあなたのアドレスに送ってくるリクエスト（26）。

WebSocket：知らせを受け取るために、閉じずにおく接続（11）。

wei：ETHのいちばん小さな単位（1）。

アーカイブノード：過去のどのブロック時点の状態も覚えているノード（10）。

アドレス：お金を送ってもらう「郵便受けの番号」（6）。

イベント（log）：何が起きたかをコントラクトが知らせる発表（15）。

インデクサー：ブロックチェーンの記録を「索引カード」に整理するプログラム（10）。

ウォレット：鍵を保管するプログラムや機器（6）。

エンドポイント：プログラムがノードに質問を送るアドレス（3）。

オラクル：価格など、外の情報をネットワークに書き込む者（31）。

ガス：Ethereumとその仲間で、手数料のもとになる仕事量の単位（15）。

クレジット：サービスがリクエストの使用量を数える単位（3、25）。

コンセンサス：参加者が一つの鎖を選ぶためのルール（8）。

再編成（リオーグ）：新しいブロックの一部が、別の枝に置き換わること（8）。

シークレットフレーズ：鍵を復元できる12語または24語（6）。

承認数：あなたのトランザクションの上に積まれたブロックの数。入ったブロック自体も数える（5）。

署名：鍵の持ち主が送金を許可したことの証明（6）。

ステーキング：コインを担保にして、ネットワークの防衛に参加すること（8）。

ステーブルコイン：1ドル付近を保とうとするトークン（9）。

スマートコントラクト：ブロックチェーンに書き込まれたプログラム（9）。

清算：借り入れが担保で守りきれなくなったとき、担保を売ること（31）。

手数料：ブロック内の場所に対してネットワークに払う料金（7）。

テストネット：値段のないコインを使う練習場（22）。

同期：ノードがネットワークに追いつくこと（12）。

トークン：スマートコントラクトがつけている記録（9）。

二重支払い：一枚のコインで二度支払おうとすること（4）。

ノード：ネットワークのプログラムを動かすコンピューター。保管し、検証し、答える（1）。

ハッシュ、「はんこ」：どんなデータからも作れる短い指紋（5）。

バリデーター：担保を預けて、ブロック作りに加わる参加者（8）。

秘密鍵：お金を動かすためのもの。誰にも見せない（6）。

ファイナリティ：それを過ぎるとブロックが置き換わらなくなる時点（8）。

負荷分散：リクエストを複数のノードに振り分けること（23）。

ブリッジ：価値をあるネットワークから別のネットワークへ移す方法（16）。

プルーニングノード：すべてを検証したあと、古いブロックを削除したノード（10）。

ブロック：「ノートの1ページ」。トランザクション、番号、時刻、前のブロックのはんこがある（5）。

ブロックチェーン：共通のルールで、共同で管理する記録の共通一覧（4）。

プロバイダー：ノードを運用し、その利用手段を提供するサービス（1、3）。

冪等性（べきとうせい）：何回繰り返しても、結果が1回と同じになる性質（29）。

マイニング：ブロックを追加する権利をかけて、総当たりで問題を解くこと（8）。

メンプール（mempool）：ブロックを待つ送金の行列（7）。

リクエスト上限（RPS）：1秒に何回聞けるか（3、25）。

流動性プール：2種類のコインを入れ、式に従って交換するコントラクト（31）。

付録B　本書のネットワーク早見表

第6章、第8章、第14〜20章の情報で、2026年11月27日時点のものです。ネットワークの更新で変わります。

Bitcoinと親戚たち（第14章）

ネットワーク	コイン	ブロック間隔の目安	アドレスの先頭
Bitcoin	BTC	10分	`1`、`3`、`bc1`
Litecoin	LTC	2.5分	`L`、`M`、`ltc1`
Dogecoin	DOGE	1分	`D`
Bitcoin Cash	BCH	10分	`bitcoincash:q…`
Dash	DASH	2.5分	`X`
Zcash	ZEC	75秒	`t1`、`zs`、`u1`

EthereumとEVMネットワーク（第15章と第16章）

アドレスはすべて`0x`で始まります。

ネットワーク	Chain ID	手数料の通貨	ブロック間隔の目安
Ethereum	1	ETH	12秒
Base	8453	ETH	0.2秒
Arbitrum One	42161	ETH	0.3秒
BNB Smart Chain	56	BNB	0.45秒
Mantle	5000	MNT	2秒
Monad	143	MON	0.3秒

そのほか（第17〜20章）

ネットワーク	コイン	アドレスの先頭	確定まで
TRON	TRX	`T`	約1分
Solana	SOL	決まった先頭文字はない	1秒未満
TON	Gram	`EQ`、`UQ`	約1秒
Sui	SUI	`0x`	1秒未満
Aptos	APT	`0x`	1秒未満
XRP Ledger	XRP	`r`	3〜5秒
Stellar	XLM	`G`	約5秒
Cardano	ADA	`addr1`	少しずつ
Polkadot Hub	DOT	`1`	確定の印で
Cosmos Hub	ATOM	`cosmos1`	すぐ

付録C　リクエスト早見表

どの例もRpcNodeの公開ノードで動きます。登録もキーも不要ですが、頻度に制限があります。自分のプロジェクトでは、アドレスを個人用エンドポイントに置き換えてください。

JSON-RPC：すべてに使える一つのコマンド

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"メソッド","params":[パラメーター]}'
```

表のメソッドとパラメーターを当てはめます。別のネットワークでは、アドレス内の`ethereum`を変えます。

EthereumとEVMネットワーク

知りたいこと	メソッド	パラメーター
最新ブロックの番号	`eth_blockNumber`	なし
ネットワーク番号	`eth_chainId`	なし
コインの残高	`eth_getBalance`	`"0xアドレス","latest"`
トランザクションのレシート	`eth_getTransactionReceipt`	`"0xハッシュ"`
ガスの値段	`eth_gasPrice`	なし
コントラクトに聞く	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

BitcoinとSolana

アドレス	メソッド	知りたいこと
`public-bitcoin-mainnet`	`getblockcount`	チェーンの高さ
`public-bitcoin-mainnet`	`getblockchaininfo`	ノードの状態
`public-solana-mainnet`	`getSlot`	スロット番号
`public-solana-mainnet`	`getBalance`	ラムポート単位の残高。パラメーターはアドレス

TRONとTON：質問はアドレスの中に

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

エラーコード

コード	何が起きたか
`-32700`	リクエストが解析できない
`-32601`	メソッドがないか、使えない
`-32602`	パラメーターが誤り
`-32005`、HTTP 429	頻度が高すぎる
`-32006`、HTTP 402	クレジット不足

完全なドキュメントと最新の料金プラン

https://rpcnode.dev

Telegramはこちら

@rpcnode_dev

本書の著者：rpcnode.dev

おわり

やさしい言葉で

ブロックチェーンの霧が晴れる

この本で何が学べる？

お客さんが聞きます。「暗号資産で
払えますか？」何と答え、お金の着金を
どう確かめ、途中で失わないためには？

中身は、ブロック、鍵、手数料、トークン、
ノードへのリクエスト、Bitcoin、Ethereum、
TRON、Solana、TONなどのネットワーク。
続いて実践：RpcNodeのエンドポイント、支払い、
Webhook、お店の守り方。最後は
DeFiについての正直な話。

「僕のサイトは
かごの計算も
メール送信もできる。
でもブロックチェーンで
起きていることを
どうやって知る？」

— アルチョム、
本書の主人公



ブロックチェーン
の仕組み

ノードと
チェーン

サイトでの
支払い

DeFi
とそのリスク

この本の特長は？

予備知識は
いりません。
各章に主人公の
疑問、やさしい
答え、図と
練習問題。
リクエストは
自分で実行して
本物の答えを
確かめられます。

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
著者：rpcnode.dev