第三世代ブロックチェーン

EOSによる分散型アプリケーション開発

魚　振江　[著]

EOS による分散型アプリケーション開発

魚振江

本書はこちらで販売中です http://leanpub.com/eos

この版は 2019-01-14 に発行されました。

本書は Leanpub の電子書籍です。Leanpub はリーンパブリッシングプロセスで著者や出版社を支援します。リーンパブリッシング は新しい出版スタイルです。軽量なツールを使って執筆中の電子書籍を出版し、読者のフィードバックをもらいながら魅力的な本に仕上がるまでピボットを繰り返すことができます。

Twitter でシェアしませんか？

本書に関するコメントをTwitter でシェアして魚振江を応援してください！

本書のハッシュタグは #eosbookjp です。

本書に関するコメントを検索する場合は、次のリンクをクリックして下さい。
Twitter のハッシュタグを使って検索できます。

#eosbookjp

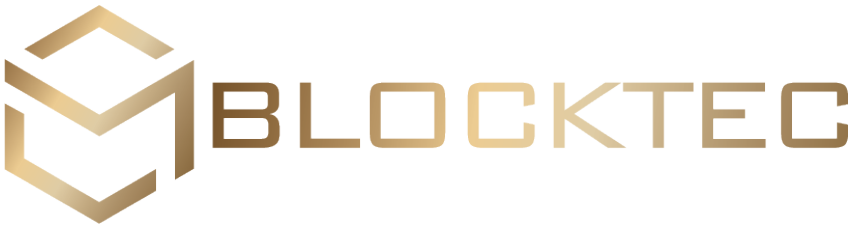
Contents

著者について

二人の子供を持つ４人の家族で横浜で暮らしています。派遣社員から Ruby on Rails 開発者になって、昨年から仮想通貨取引所システムやイーサリアム上の PoC プロジェクトに参加し、現在は、フルスタック・エンジニアとしてブロックチェーン事業を推進しています。

Twitter : @blueplanet42

Qiita : @blueplanet



BlockTec CTO

CHAINBOW.IO

ChainBow CTO

まえがき

本書を手にとってくださり、どうもありがとうございます。本書は第三世代ブロックチェーンと呼ばれている EOS の仕組みと EOS 上で分散型アプリケーションを開発する仕方を説明した本です。

この数年間、ビットコインからはじめ、仮想通貨とブロックチェーンという言葉は熱いキーワードになっています。特に昨年 2017 年は、仮想通貨の数がすごく増えて、価格変動もものすごく激しく状況でした。2018 年は、市場が冷静になってきて、仮想通貨ではなく仮想通貨の基盤であるブロックチェーン技術自体がだんだん注目されているようになってきています。

実は、仮想通貨はブロックチェーン上に構築されている一つのアプリケーションに過ぎません。スマートコントラクトのおかげでブロックチェーン上で様々なアプリケーションを構築できます。今の時点は、App Store が出たばかりの時と同じ時期であり、ブロックチェーン上でどんな素晴らしいアプリケーションを作れるかは、まだアイデア勝負です。

数多くブロックチェーンの中で、本書は **ブロックチェーン 3.0** と呼ばれている EOS 上でどうやって分散型アプリケーションを構築するかを紹介します。

本書はどんな人に向いているか？

本書は、初心者ではなく、ブロックチェーンの基礎知識をある程度理解している方を対象にしています。

- ブロックチェーンとは何か、スマートコントラクトとは何か、について、本書の中で特に解説してません。
- 本書の中で、EOS を紹介することにあたって、既に広く知られているイーサリアムと比較する箇所がしばし出ているので、イーサリアムの知識があると理解しやすいと思います。
- また、現状 EOS のスマートコントラクトは C++ でしか実装できませんが、スマートコントラクトの制限があるため、C++ の深い知識の必要がなく、ほかの任意のプログラミング言語の経験があれば、理解できると思いますので、ご安心ください。
- フロント側のほうは Nuxt.js を使っている為、HTML / CSS / JavaScript / Vuejs の基本知識があると理解しやすいと思います。

手と頭を動かして体感してみましょう

新しいプログラミング言語を勉強する際、写経しながら勉強するのがよくオススメされていると思います。ブロックチェーンの場合それがよりオススメです。

その理由は、現時点の各ブロックチェーンはまだまだ発展中のため、設計と実装の変化が結構激しいです。そのせいで、先週こう実装すればうまく行けたのに、今週同じような実装はうまく行かないということは、よくあることです。そのため、ちゃんと自分で動かして、最新バージョンはどうなっているかを確認しながら進めるほうが良いかと思います。

本書のソースと使ってるライブラリのバージョン

本書で実装したソースは下記リポジトリで公開しています。

・フロント側 blueplanet/eos_addressbook
・スマートコントラクト blueplanet/eos_contracts

本書使っているライブラリは以下になっています。

ライブラリ	バージョン
Docker Community	18.09.0
EOSIO	1.4.2
eosio.cdt	1.3.2
Nuxt.js	2.0.0
eosjs	16.0.9
scatter-js	2.5.2
Scatter Desktop App	9.6.0

謝辞

まず、林陽氏に感謝します。彼のおかげで、仮想通貨とブロックチェーンに出会うことが出来ました。仮想通貨取引所システムの開発からイーサリアムの勉強、そして一緒にハッカソンのチャレンジ、思い出せば、最初のはじめは既に一年半前のお話でした。

次は、家族に感謝します。私がどんなものを書いているかはほとんど分からないかもしれませんが、新しいチャレンジで家族との時間を奪ってしまったことを我慢してくれた妻に感謝します。一緒に遊ぶ時間が奪われても理解してくれた息子と娘にも感謝します。

そして、本書をレビューして頂いた以下の方にも感謝します。(敬称略)

- アットコイン株式会社 CTO 琴畑尚哉
- @shwld

2018 年 12 月魚　振江

1. EOS とは

EOS は、**Enterprise Operating System** の略語です。言葉どおり、エンタープライズ向けのオペレーティングシステムを提供しようとし、既存のブロックチェーンで起きている課題を意識しながら新しく設計・開発したパブリックブロックチェーンです。

ソフトウェア (EOSIO) はスタートアップ企業 Block.One が主導で開発されていますが、リリースと運営はコミュニティによって実施しています。

最終的に稼働しているパブリックチェーンが **EOS** です。

- EOSIO 公式サイト
- block.one 公式サイト

EOS パブリックチェーンのノードを運営する組織は、BP (Block Producer) と呼ばれています。全世界の BP のノードが繋がって EOS のネットワークを構成しています。BP の情報は、以下のリンクから確認できます。

EOS Go BP

本書では **EOS はブロックチェーン 3.0 である**と述べています。その理由をブロックチェーンの歴史から振り返しながら説明します。

1.1 ブロックチェーンの歴史

ブロックチェーン 1.0

ブロックチェーンの歴史は、ビットコインから始めたものです。

ビットコインは、2009 年頭リリースされ、今まで（2018 年 12 月現在）9 年間経ちました。パブリックネットワークの中で、ブロックチェーンを使うことで、参加者同士がお互いの信頼がなくても合意を形成でき第三者に依存せずに価値を移転できることを証明できたと思われます。

そのため、ビットコインは、**ブロックチェーン 1.0** と言われています。

ブロックチェーン 2.0

その次に **ブロックチェーン 2.0** になるものは、**イーサリアム**です。

ビットコインは、パブリックネットワークの中で、参加者同士が信用なしで合意形成できるかどうかの社会実験として成功できたと思います。しかし、ビットコインにはスマートコントラクト仕組みである **Bitcoin Script** がありますが、設計上は敢えてチューリング不完全にしています。プログラミング言語のチューリング完全は厳密な定義がありますが、ここでは理解しやすい例をあげますと、**Bitcoin Script** にはプログラミングの中で不可欠な繰り返し仕組みすら提供されていません。このように設計されているのは勿論理由がありますが、スマートコントラクトの開発には大きな壁を作ってしまったと言えるでしょう。

Bitcoin Script がチューリング不完全にされている理由

ビットコインは、パブリックネットワーク上で稼働しています。そして、スマートコントラクトは、任意の開発者が実装でき、任意のタイミングで実行できるものです。

そのため、想像してみればお分かりと思いますが、仮に **Bitcoin Script** がチューリング完全である場合、もし悪意がある開発者から、無限ループの処理を持っているスマートコントラクトをブロックチェーンにデプロイしてしまうと、ビットコインネットワークは **DDoS** 攻撃が受けられているように、全体が落ちてしまう恐れがあります。

このようなことを防ぐ為に、ビットコインの **Bitcoin Script** は、敢えてチューリング不完全にしています。

この問題を解決しスマートコントラクトをより開発しやすくしたのが、イーサリアムです。イーサリアムのミッションはスマートコントラクトのプラットフォームであるため、スマートコントラクトの開発言語をチューリング完全にしています。そのおかげで、スマートコントラクトを開発するハードルが大分低くなってきています。

2017 年仮想通貨が爆発的に増えてきたのも、このおかげだと思います。

イーサリアムは、ビットコインで証明できたブロックチェーンを発展させ、スマートコントラクトの普及を大きく進めさせたので、**ブロックチェーン 2.0** と言われています。

ブロックチェーン 3.0

ブロックチェーン 2.o と呼ばれているイーサリアムは、ビットコインの課題を解決して、スマートコントラクトのプラットフォームを提供しています。2013 年リリースされた以来、数多いアプリケーションがイーサリアム上で構築されています。これらのアプリケーションを構築している中で、いくつかの課題が出てきました。

1. エンドユーザーがイーサリアム上で稼働するアプリケーションを使う時は、手数料が発生すること
2. スケーラビリティ問題
3. スマートコントラクトはアップデートできないこと

一つずつ詳細を見てみましょう。

1. 手数料問題

ビットコインのスマートコントラクトがチューリング不完全であると説明しました。イーサリアムは、それを改善する為、チューリング完全の開発言語を提供しています。

チューリング完全について例をあげた時、無限ループの問題を説明しました。イーサリアムでは、この問題を解決する為に、**GAS** という概念を導入されています。

イーサリアムは、スマートコントラクトの処理ステップ毎に手数料を払うようになっています。この手数料は、**GAS** と言います。トランザクションを発行しコントラクトのある処理を呼び出す時、その処理に必要な手数料を前払いでトランザクションに含める必要があります。処理が終わると、前払いの分から実際発生する手数料を控除し残りの分を返してくれます。ただし、処理にかかる手数料が前払いの分を超えた場合は、その時点で処理が失敗になって更に発生する手数料が返されないような仕様になっています。

この仕組を導入することで、チューリング完全のスマートコントラクト開発言語を提供出来ました。悪意の無限ループ処理があっても、その処理をさせたエンドユーザーが払った手数料がなくなるまで実行するだけのため、イーサリアムネットワーク全体に対する攻撃が成り立たなくなります。

素晴らしい仕組みではあると思いますが、残念なことに、イーサリアム上のサービスを使いたいエンドユーザーから見ると、大きいハードルになってしまいます。今の時代は、個人顧客向けのサービスの場合は、フリーの基本利用＋課金のオプション機能のモデルが圧倒的に多いです。それに慣れているエンドユーザーからみると、イーサリアム上のアプリケーションを使うと最初から課金しないと行けないことは、おそらく受け入れられないでしょう。このせいで、イーサリアム上で展開するサービスは、最初のエンドユーザーを作るのに結構難航しています。

この問題は、イーサリアムの根本的な設計になっているため、基本的には変わらないと思われます。

サービス側が GAS を負担することで、エンドユーザー側が無料でトランザクションを実行できるようなサービスも出ています。

BUIDL EOS-like DApps on Ethereum

2. スケーラビリティ問題

イーサリアムの処理スピードは、平均的には大体秒間 20 トランザクションを処理できる程度 (2018 年 12 月現在) です。

この問題は、アーキテクチャと合意形成アルゴリズムなどイーサリアムの根本的な部分と関連しています。

- イーサリアムの合意形成アルゴリズムは、2018 年 12 月現在まだビットコインと同じく PoW になっています。理論上、１つのブロック時間の中は、１つのノードしか勝ちません。そのためネットワーク全体の処理能力は１つのノードと変わりません。
- アーキテクチャのほうは、合意形成アルゴリズムと関係して、並列実行などは出来ません。

イーサリアムの開発チームは新しいアーキテクチャの採用（シャーディング、Plasma ）や合意形成アルゴリズムの変更（PoS）など、いろいろな改善を計画しています。

3. スマートコントラクトがアップデートできない問題

イーサリアムは分散型アプリケーションのプラットフォームを目指しています。イーサリアムにデプロイされるスマートコントラクトは、設計上一回デプロイされると後から変更できないようになっています。スマートコントラクトは現実世界の契約と同じなので勝手に変更してはならないという考えです。

この考えに関する議論はさておき、実際起きた問題としては、スマートコントラクトにバグがあっても修正版にアップデートできないため、ハッカーに悪用され大きい金額の被害が出た事例が既にあります。

この問題については、イーサリアムの開発チームが設計を変更する予定がないようなので、分散型アプリケーションの開発者達がスマートコントラクト側の実装で頑張っているようになっています。

EOS は、これらの問題を解決し、もっとエンタープライズ向けの、商用システムレベルに耐えられるような分散型アプリケーションのプラットフォームを目指しているため、**ブロックチェーン 2.0 であるイーサリアム**を超えて、**ブロックチェーン 3.0** になっています。

1.2 なぜ EOS ？

EOS は、資金調達が始まった 5 日間で 1 億 8,500 ドル調達でき、１年間かけて最終的に 40 憶ドル調達できました。なぜそこまで注目されているのでしょうか。

ブロックチェーン 2.0 であるイーサリアムはいくつの課題があると述べました。EOS はこれらを解決する為に新しく設計され、主に下記の特徴があります。

基本利用手数料が無料

EOS は、アプリケーションを利用する手数料があるかどうかは、プラットフォーム側で強制的に決められるのではなく、アプリケーション開発者が柔軟に決められるように設計されています。そのため、EOS の送金処理や EOS 上で稼働するアプリケーションを利用する場合は、基本的に手数料なく、無料で行えます。

そのかわりに、アプリケーション開発者、すなわちサービス提供者側が手数料を払う必要があります。ウェブサービスと同じく、サービス側が事業内容によって柔軟に決められる一方、エンドユーザー側が基本無料で利用できるので、より現実的ではないでしょうか。

処理スピードが速い

2018 年 12 月現在、イーサリアムの秒間処理できるトランザクション数は、15 - 20 程度になっています。EOS の場合は、2018 年 6 月リリースされた以来、実測値として記録された最大値は、3,994 個でした。ブロック #14487862 に、1,997 個のトランザクションが記録されています。EOS のブロック生成時間が 0.5 秒なので、秒間は 1,997 * 2 = 3,994 個のトランザクションになります。更に、理論上は秒間何百万トランザクションに耐えられるような設計になっています。

何故これほど速くできているのかは、第二章のコンセンサスアルゴリズムの節で詳しく説明します。

スマートコントラクトがアップデートできる

EOS のスマートコントラクトは、アカウントに紐付いていて、必要な権限を持っていれば、新しいバージョンをデプロイし、スマートコントラクトの処理をアップデートできるようになっています。

ガバナンス仕組みでトラブルを解決する

ガバナンスとは、ある組織を管理するプロセスのことです。

いままでのブロックチェーンでは、明確なガバナンスプロセスがなかったため、トラブルが発生する都度その場で論争し対応方針を決めてしまう流れでした。EOS は、明確なガバナンスが設けられています。

- まず、EOS トークンを保有している者（トークンホルダーと呼ばれている）は、権限を持っています。
- 次に、トークンホルダーは権限をブロック生成者に委任します。
- 最終的に、ブロック生成者は、付与されている権限を利用して、アカウントの凍結やバグあるアプリケーションの更新、さらに、基本プロトコルのハードフォーク変更提案を行うなども行えます。

アカウントとロールベースで権限管理できる

EOS には、アカウントと権限が設けられています。

- アカウントは最大 12 桁までの文字で構成されているユニークな名前で識別されます。アカウント毎にプライベートなデータベースを持つことができ、そのアカウントからしかアクセスできません。他のアカウントに対して **アクション**を送信でき、他のアカウントからアクションを受けたあとの処理も定義できます。アクションと処理内容を定義することは、EOS でのスマートコントラクトそのものを定義することになります。
- 権限は、ロールベースで管理する仕組みになっています。どのロールがどのアクションを実行できるかを定義でき、その権限を１つまたは複数のキーペアと紐づけできます。複数のキーペアと紐付いている場合は、マルチサインになり、更に重み係数も定義できます。

　このようなアカウントと権限の仕組みをプラットフォーム側で標準化し提供することで、アプリケーションのビジネスロジックに権限管理仕組みを用意する必要がなくなります。

　上記だけではなく、他にもいろいろ優れている仕組みがあるため、EOS は **ブロックチェーン 3.0** と呼ばれ、世界中に注目されています。

1.3 この章のまとめ

本章では、ブロックチェーンの歴史を振り返しながら、EOS の特徴を紹介しました。EOS は、上記以外でもスケーラビリティやセキュリティなど、いろいろな優れている仕組みがあるため、イーサリアムキラーと呼ばれているぐらい、ブロックチェーン業界で注目されています。

次章は、詳しく EOS の仕組みを紹介します。

2. EOS の仕組み

本章は EOS の仕組みを紹介していきます。

まず EOS の合意形成アルゴリズムから、基本的なブロックやトランザクションのデータ構造を説明します。その後、開発環境を構築して、実際触りながら開発する際使うツールチェーンを紹介します。

合意形成は、英語の単語 **consensus** からきた **コンセンサス**という単語がよく使われるので、以降 **コンセンサス**を使います。

2.1 コンセンサスアルゴリズム

EOS のコンセンサスアルゴリズムは、厳密的には **DPoS + 非同期 BFT** になっています。

DPoS とは

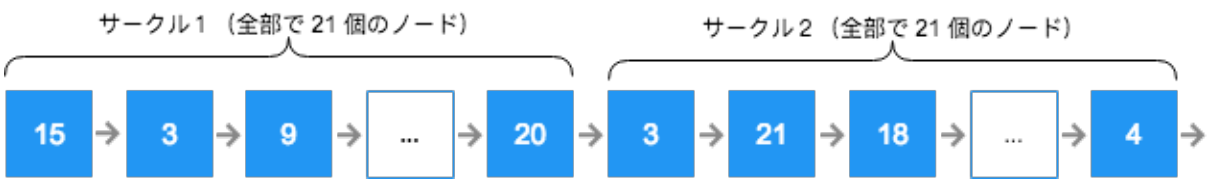
DPoS は、**Delegated Proof of Stake** の略語です。通貨を保有している人たちが投票を行うことによって、ブロックの承認者になるノードを選出します。それらの承認ノードたちがお互いに協力しながらブロックを生成して行くアルゴリズムです。

ビットコインやイーサリアムに使われている **PoW** の場合、新しいブロックを生成する権利を競争するため、ネットワークに参加しているノード全員がナンスを計算しますが、最終的にはその中の１つのノードだけ新しいブロックの生成する権利を貰えるようになっています。そのノード以外のすべてのノードの計算が無駄になってしまうので、よく批判されています。

DPoS の場合は、投票によって選出されたノードから、投票数が上位になっている特定の数（EOS の場合 21 個）のノードをブロック生成者としています。これらのブロック生成者ノードたちは、サイクル毎に下記の流れでブロックを生成します。

- 新しいサイクルが始まると、まず投票数が上位になっているノードから、上位 20 個のノードと、それ以外のランダムで 1 個のノードを選択します
- この 21 個のノードをランダムで順番を決めます
- あるノードが自分の番になると、その時のブロック生成者になり、トランザクションを収集しブロックを生成します
- 終わったら次の番のノードが同じ処理を続きます
- １つのサイクルの 21 個のノードが全部終わったら、新しいサイクルが始まります

図で表現すると、下記のようになります。



DPoS のブロック生成

サイクル１では、ノードが *15 -> 3 -> 9 -> ... -> 20* の順番でブロック生成します。次のサイクル２では、もう一回ランダムで順番を決めるので、*3 -> 21 -> 18 -> ... -> 4* の順番で生成します。

ブロック生成者たちはお互いに競争するのではなく、協力しながら進めるので、競争するための計算が要りません。そのおかげで、ブロック生成者が全力でブロックを生成することができるので、ブロック生成するスピードが PoW と比べて桁違いぐらい速くなっています。

EOS は DPoS のパフォーマンスを満足していないため、更に下記の対応を取り込んでいます。

1. ブロック生成時間を 0.5 秒にする

EOS は、ブロック生成時間を 0.5 秒にすることで、もっと短い時間単位でトランザクションを確認できるようになります。

もちろん、こうすることで他の課題が出てきていますが、合わせて他の対策も取っています。

2. ブロック生成者の順番を特定の順番にする

EOS は、選出されたブロック生成ノードたちの順番をランダムから、物理的な距離が近いノードを隣にするような順番に変更しました。

ブロック生成時間を 0.5 秒にすると、ネットワークの通信遅延が無視できないレベルになります。ランダム順番の場合、たとえ中国のノードの次がアメリカノードになると、中国ノードが生成したブロックがまだアメリカノードまでに伝えてなかった可能性は大分高くなってしまいます。そのため、物理的に近いノードを隣にすることで、できるだけ隣になっているノード間のデータ不整合が起きる可能性を低く抑えられます。

3. ブロック生成者は連続で 12 個のブロックを生成する

ブロック生成ノードが自分の番になる時、１つのブロックではなく、連続で 12 個のブロックを生成する。

隣にいるノードの物理的な距離が近いとはいえ、0.5 秒単位でノードが交代するとやはり不安なので、連続で 12 個のブロックを生成することで、交代する時間を 6 秒にしています。

連続で 12 個のブロックを生成する場合は、一個前のブロックが自分が生成したブロックなので、データ不整合が起きる可能性を更に低く抑えられます。できるだけネットワーク通信遅延の影響を抑えることで、0.5 秒のブロック生成時間を保証しています。

ホワイトペーパーには連続生成する数は 6 個と書かれていますが、2018 年 12 月現在、実際稼働している実装は、12 個になっています。

4. 非同期 BFT

BFT とは、**Byzantine Fault Tolerance** の略語です。P2P ネットワーク上でビザンティン将軍問題を解決でき正常に稼働でいるシステムが **BFT** を導入していると言われています。

ビザンティン将軍問題とは、ネットワークでの正しい合意形成を問う問題「ビザンチン将軍問題」 の記事より引用しますが、

> 相互に通信し合う P2P ネットワーク上で、通信そのものや個々のノードが故障、または故意に偽の情報を伝達する可能性がある場合に、全体として正しい合意が形成できるかを問う問題

になります。P2P ネットワーク上で稼働するシステムの場合は、この問題を解決しないとシステム自体が成り立たなくなります。まだ詳細を把握出来てない方は、上記記事をご確認ください。

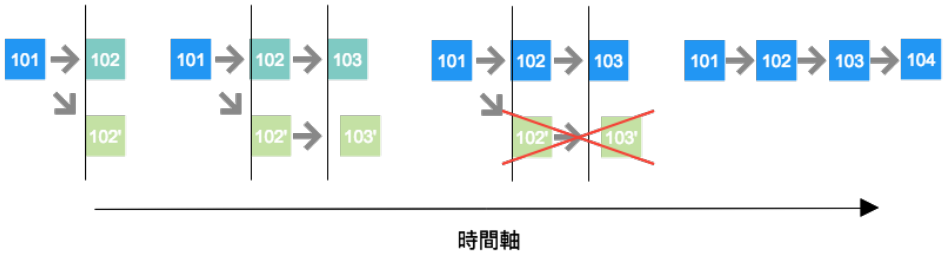
BFT の対応内容はいろいろありますが、ここではトランザクションの確定時間で例をあげます。

トランザクションの確定時間とは

トランザクションがブロックに含まれ、更にそのブロックがブロックチェーンのネットワークにブロードキャストされれば、トランザクションが承認されたと言います。

ただし、ビットコインやイーサリアムの場合、ネットワーク上で通信遅延などによって、異なるところから同時に新しいブロックがブロードキャストされる可能性があります。この現象はフォークと呼ばれています。

現状、ビットコインやイーサリアムは、フォークが発生する際、一旦出てきた両方とも保留とし、次のブロック生成を進めます。**同時新しいブロックがブロードキャストされた**可能性はそもそも十分小さいので、その2つのチェーンがそれぞれの次のブロックまた同時にブロードキャストされる可能性が極めて小さくなります。そのため、フォークが発生したとしても、大体 2, 3 個のブロック時間で、ブロックのブロードキャスト時間差が出てくるので、その時点で、どちらのチェーンが長いかを判別できるため、長いほうが正しいとして残されます。



上記の図の場合、102 番目のブロックで水色と緑色のブロックが同時に出てきましたが、103 番目のブロックで既に水色のほうが先にでききたので、最終的に、上のほうが残され、下の緑色の 102’ と 103’ が削除されます。

ビットコインのデータによりますと、フォークができてしまった場合、ほぼ次のブロックか、3 個目のブロックで解決できるようになっています。

ここで注目してほしいのは、緑色のブロックです。103 番目のブロックで水色より後で生成され、結果的に緑色の 102’ ブロックと 103’ ブロックが捨てられ、この2つのブロックに含まれているトランザクションがキャンセルされてしまう可能性がでているところです。

そのため、トランザクションがブロックにに含まれていても、まだ取り消される可能性があるため、その時点ではまだ **確定**とは言えないです。よって、ビットコインの場合は、6 個のブロックを待つことで、ほぼ取り消される可能性がゼロに近いところで、トランザクションが確定されたと言えるので、このことを **トランザクションの確定**と言い、トランザクションが確定されるまでの時間を **トランザクションの確定時間**と言います。

DPoS の場合、同じ時間帯は１つのブロック生成者しかないため、悪意のブロック生成者がいない限りそもそもフォークが発生しません。そのため、トランザクションがブロードキャストされていれば、99.99% 確定されていると言われています。そのうえ、ネットワーク上にある 21 個ノードのうち、2/3 (14 個) のノードに確認して貰えば、100% 確定になります。

DPoS の **ブロック確認（確定ではなく）**の定義は、**別のブロック生成者がそのブロックの後に新しいブロックを生成したこと**になります。その為、ブロック生成しブロードキャストしてから、次のブロック生成者がそのブロックの後にブロックを生成すること待つ必要があります。更に後のブロック生成者の確認を待たないといけません。

このように、ブロックの確定時間は、

・今のブロック生成者がブロック生成する時間
・他の 14 個のノードがそのブロックの後ろにブロックを生成する時間

結局 **3 秒 x (1 + 14) = 45 秒**を待つことで、トランザクションが確定されるようになります。

これを改善するために、EOS は、ブロック確認の仕組みを非同期にしました。

・まず、ブロックの確認処理を変更し、ノードがブロックデータを検証し問題なければサインすることに変更しました
・次に、EOS のブロック生成者は、ブロックを生成した後、ブロックにサインしネットワークにブロードキャストします
・他のブロック生成者は、新しいブロックを受信すると、すぐに検証し問題なければサイン（確認）して、その結果を送信元のブロック生成者に返します
・送信元のブロック生成者は 14 個のサインを入手できれば（自分を含めると 15 個になる)、そのブロックが確定され、トランザクションが確定されるようになります

こうすることで、トランザクションの確定時間は、ブロック生成時間 0.5 秒 + 別のブロック生成者による確認時間になります。後者も大体 0.5 秒以内で完了できるので、EOS のトランザクション確定時間は、1 秒程度までに短縮できます。

このように、EOS は既存の DPoS を更に改善することで、商用レベルの分散型アプリケーションプラットフォームを目指しています。

この非同期確認機能について、ホワイトペーパーに記載されていますが、2018 年 12 月現在、まだ実装されていません

2.2 EOS のブロックの構成

レイテンシーは、あるアカウントが別のアカウントにアクションを送信し、レスポンスを受け取るまで要する時間を指しています。EOS は、この値を最小限に抑え、2つアカウントの間、ブロック生成時間を待たずに、１つブロック以内でアクションをお互いに交換できるようにするために、基本的なブロックチェーン構成（ブロックとトランザクション）に手を加えています。

具体的には、下記のようにブロックを更に **リージョン・サイクル・シャード・アクション**という概念を導入しています。

ブロック構成

縦の方向は順番実行、横の方向は並列実行の意味になります。

1つずつ詳細を見て行きましょう。

リージョン

リージョンは、スマートコントラクトのハイレベルのグループであり、お互いに独立しているブロックチェーンのように動きます。

サイクル

ブロック生成者は、アクションがなくなるまたは最大実行時間になるまで、ブロックにサイクルを追加し続けます。１つのアクションで生成したトランザクションは、その後のサイクルまたはブロックに配信されます。

こうすることで、あるアカウントが別のアカウントに送信し、後者が受け取ったアクションを処理する際生成したアクションは、次のサイクルで処理されることができるため、１つのブロック生成時間を待たずに、0.5 秒以内で完結できるようになります。

また、異なるサイクルにあるトランザクションが、上記のように順番を保証する必要があるため、サイクルは順番実行になっています。

シャード

サイクルにあるトランザクションは、静的分析によってどのアカウントを変更するかを検証できるため、異なるアカウントに対するトランザクションをシャード単位に分けておき、シャード同士は並列で実行できます。

このような、アクション処理が関連しているアカウントで分けている単位は、シャードです。

トランザクション

EOS のトランザクションは、データベースのトランザクション定義と似ています。トランザクションは、アトミック性を持ち、１つまたは複数のアクションを含めていて、これらのアクションは１つの単位として、全部成功か全部失敗かしかありません。

アクション

アクションは、EOS の処理の最小単位であり、送信先とアクション名とアクションに必要なパラメータを持っています。

ブロック構成のまとめ

このように、EOS は基本的なブロックとトランザクション構造の上に、更にリージョン・サイクル・シャードを導入することで、異なるアカウント間の相互通信を更に速くできるようにしています。

2018 年 6 月リリースされた 1.x バージョンには、シングルスレッドバージョンであり、並列実行処理はまだありません

2.3 Docker で開発環境を構築する

百聞は一見にしかず、実際に EOS が動ける環境を構築し、触って見てみましょう。

本書は Docker を使って環境を構築するので、まだインストールされてない場合は、先に Docker をインストールしてから進めましょう。

EOS の node を構築する

まず作業用のディレクトリを作成します。本書の以降は、このディレクトリを作業ディレクトリとします。

作業ディレクトリ作成

```
mkdir -p ~/dev/eos_contracts
cd ~/dev/eos_contracts
```

docker の image を取得します。

Docker イメージ取得

```
docker pull eosio/eos:v1.4.2
```

EOS のコンテナーを作成します。

EOSIO コンテナ作成

```
docker run --name eosio \
  --publish 7777:7777 \
  --publish 127.0.0.1:5555:5555 \
  --volume $(pwd)/:/eos_contracts \
  --detach \
  eosio/eos:v1.4.2 \
  /bin/bash -c \
  "keosd --http-server-address=0.0.0.0:5555 & exec nodeos -e -p eosio --plugin eo\
sio::producer_plugin --plugin eosio::chain_api_plugin --plugin eosio::history_plu\
gin --plugin eosio::history_api_plugin --plugin eosio::http_plugin -d /mnt/dev/da\
ta --config-dir /mnt/dev/config --http-server-address=0.0.0.0:7777 --access-contr\
ol-allow-origin=* --contracts-console --http-validate-host=false --filter-on='*'"
```

コンテナーが動いているのかを確認します。

コンテナ確認

```
docker logs --tail 10 eosio
```

下記のようなログが表示されていれば、ノードが正しく動作されていることになります。

コンテナログ

```
2018-10-27T06:05:26.005 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001eef5d1ed68... #494 @ 2018-10-27T06:05:26.000 signed b\
y eosio [trxs: 0, lib: 493, confirmed: 0]
2018-10-27T06:05:26.501 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001ef9fdf5b1a... #495 @ 2018-10-27T06:05:26.500 signed b\
y eosio [trxs: 0, lib: 494, confirmed: 0]
2018-10-27T06:05:27.002 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f0a8f5d428... #496 @ 2018-10-27T06:05:27.000 signed b\
y eosio [trxs: 0, lib: 495, confirmed: 0]
2018-10-27T06:05:27.502 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f10c8edbad... #497 @ 2018-10-27T06:05:27.500 signed b\
y eosio [trxs: 0, lib: 496, confirmed: 0]
2018-10-27T06:05:28.004 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f2cef0814c... #498 @ 2018-10-27T06:05:28.000 signed b\
y eosio [trxs: 0, lib: 497, confirmed: 0]
2018-10-27T06:05:28.501 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f3bf17a190... #499 @ 2018-10-27T06:05:28.500 signed b\
y eosio [trxs: 0, lib: 498, confirmed: 0]
2018-10-27T06:05:29.004 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f436b25e49... #500 @ 2018-10-27T06:05:29.000 signed b\
y eosio [trxs: 0, lib: 499, confirmed: 0]
2018-10-27T06:05:29.500 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f51771b68e... #501 @ 2018-10-27T06:05:29.500 signed b\
y eosio [trxs: 0, lib: 500, confirmed: 0]
2018-10-27T06:05:30.001 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f67066b5f1... #502 @ 2018-10-27T06:05:30.000 signed b\
y eosio [trxs: 0, lib: 501, confirmed: 0]
2018-10-27T06:05:30.501 thread-0   producer_plugin.cpp:1302      produce_block   \
     ] Produced block 000001f752c39181... #503 @ 2018-10-27T06:05:30.500 signed b\
y eosio [trxs: 0, lib: 502, confirmed: 0]
```

表示されているブロック番号などの情報は実行環境とタイミングによって異なります

wallet の動作を確認する

ウォレット確認

```
$ docker exec -it eosio bash
root@5b77933bb052:/#
root@5b77933bb052:/# cleos --wallet-url http://127.0.0.1:5555 wallet list keys
Wallets:
[]
Error 3120006: No available wallet
Ensure that you have created a wallet and have it open
root@5b77933bb052:/#
```

初回実行する場合は、上記通りに **Error 3120006: No available wallet** エラーが表示されますが、内容通りに、まだ wallet 作成していないだけなので、**Wallets: []** の内容が表示されていれば、ノードが正しく動作しています、安心して次に進んで下さい。

動作確認できたら、`exit` コマンドでコンテナーから出ましょう。

RPC API エントリポイントを確認する

コンテナを起動した時、ポートを設定したので、ローカルマシンから以下のように http の RPC エントリポイントでも確認出来ます。

API エントリポイント確認

```
$ curl http://localhost:7777/v1/chain/get_info
{"server_version":"60c8bace","chain_id":"cf057bbfb72640471fd910bcb67639c22df9f924\
70936cddc1ade0e2f2e7dc4f","head_block_num":886460,"last_irreversible_block_num":8\
86459,"last_irreversible_block_id":"000d86bbda3b33558502a41c1a46502a23147dcaac747\
61cdf4194e6373ce89b","head_block_id":"000d86bcaccd8fbb477465d8b6b8ac1b4018b7d8c58\
9b010febe190099e7f493","head_block_time":"2018-11-24T04:30:24.000","head_block_pr\
oducer":"eosio","virtual_block_cpu_limit":200000000,"virtual_block_net_limit":104\
8576000,"block_cpu_limit":199900,"block_net_limit":1048576,"server_version_string\
":"v1.4.2"}
```

コマンドのエイリアスを定義しておく

今後都度コンテナーに入らずにローカルから直接に実行できるように、コマンドラインのエイリアスを定義しておきます。

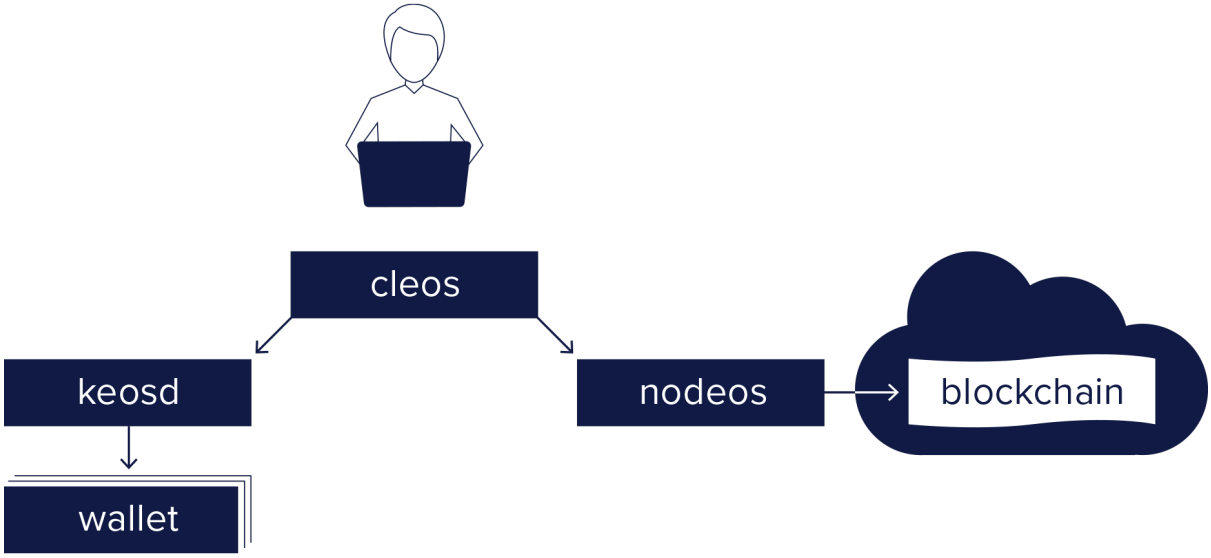
コマンドエイリアス定義

```
// ~/.bashrc や ~/.zshrc に追記する
$ alias cleos='docker exec -it eosio /opt/eosio/bin/cleos --url http://127.0.0.1:\
7777 --wallet-url http://127.0.0.1:5555'

// 上記内容を適用する
$ source ~/.bashrc

// エイリアスを使って動作確認する
$ cleos wallet list keys
Wallets:
[]
Error 3120006: No available wallet
Ensure that you have created a wallet and have it open

// 先程と同じ内容が表示されていればエイリアスが正しく設定されていることになります
```

これで EOS が動いている環境を構築できました。

2.4 ツールチェーン構成を知る

ここで、少し構築した環境と EOS のアーキテクチャを説明します。
今回構築した環境は、下記の図通りになっています。



EOS アーキテクチャ

参照元：公式ドキュメント

一番右側にある **blockchain** は、今回の場合、ローカルのマシンに１つだけのノードで構成しているようになっています。

nodeos (node + eos = nodeos)

EOS ノードのデーモンアプリケーションです。これを起動することで、EOS ネットワークにある他のノードと通信し、データを同期してくれます。プラグイン仕組みがあり、設定によって各種機能たとえチェーン API や履歴 API などを追加でき、エントリポイント経由しアクセスできるようになります。

cleos (cli + eos = cleos)

ブロックチェーンとウォレットにアクセスするコマンドラインインタフェースです。裏側でノードと通信し各種 API を呼び出すことができます。

keosd (key + eos = keosd)

安全にキーペアを保管できるウォレットコンポーネントです。名前無しのデフォルトウォレット以外に、名前を指定して作成することもできます。

ウォレットアプリケーションを言うと、トークンを保管しているイメージになると思いますが、keosd はウォレットアプリケーションではなく、単にプライベートキーとパブリックキーのペアを安全に保管するだけで、トークンの保管機能はありません。

これらのツールを使って EOS とやり取りできます。

2.5 ウォレット・キーペア・アカウント・権限の関係を理解する

前の節でウォレットはキーペアを安全に保管できると説明しました。また、前章で EOS のアカウントと権限について少し説明しました。ここでは、実際作りながら、この 4 者の関係を整理して説明します。

ウォレットを作成する

```
$ cleos wallet create --to-console
Creating wallet: default
Save password to use in the future to unlock this wallet.
Without password imported keys will not be retrievable.
"PW5KGq44bFVtS5epwgMZPvF52WpZ3ffTzwCHPSBh3FQ3FKxwi8Nyx"

// list サブコマンドでウォレットリストを確認できる
$ cleos wallet list
Wallets:
[
  "default *"
]
```

上記コマンドは、ウォレットの名前を指定していないので、**default** という名前のウォレットが作成されます。パスワードはランダムで生成された値であり、ここ 1 回しか表示されないので、必ずメモしておいてください。後でウォレットをアンロックする時に必要になります。再度実行すると、**default** という名前のウォレットが既に存在しているので、エラーになります

重複作成

```
$ cleos wallet create --to-console
Error 3120001: Wallet already exists
Try to use different wallet name.
```

名前を指定してウォレットを作成したい場合は、*-n second* のようにパラメータを指定する必要があります。

名前を指定してウォレットを作成

```
$ cleos wallet create -n second --to-console
Creating wallet: second
Save password to use in the future to unlock this wallet.
Without password imported keys will not be retrievable.
"PW5JCUR9cXgdTYe3fUhyZ5PeXTy9pAKCwZbxHjqJGxgmeTQizzEDu"

// 指定した名前で作成されたことを確認できる
$ cleos wallet list
Wallets:
[
  "default *",
  "second *"
]
```

“*” が付いているウォレットは、今現在アンロックされているウォレットになっています。

しばらくは `default` ウォレットだけ使うため、一旦ウォレットを全部ロックし *default* だけアンロックします。

default ウォレット使う

```
$ cleos wallet lock_all
Locked All Wallets

// 上記でメモしておいたパスワードを入力しアンロックする
$ cleos wallet unlock
password: Unlocked: default

$ cleos wallet list
Wallets:
[
  "default *",
  "second"
]
```

キーペアを理解する

まず新しいキーペアを作成します。

キーペアを作成する

```
cleos wallet create_key
Created new private key with a public key of: "EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8P\
iskXW3HnvG8XYbihtKQ"
```

新しいランダムなプライベートキーが生成され、そこからパブリックキーが算出されましたが、EOS のアカウントはキーペアと関係なく独立しているため、アカウントはまだ出来ておらず、単にキーペアが作成されウォレットに保管されるだけです。

ウォレットに保管されているキーペアは下記コマンドで確認出来ます。

キーペアを確認する

```
$ cleos wallet keys
[
  "EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8PiskXW3HnvG8XYbihtKQ"
]
```

アカウントを作ってみる

EOS のアカウントは、アカウント名とデフォルトの **Owner / Active** 2つのロールを持っています。ロールは、キーペアと紐付いて、そのキーペアで制御する形になります。そのため、アカウントを作成する際は、**Owner / Active** に紐付くキーペアのパブリックキーを指定する必要があります。

また、EOS のアカウントを作るには、別のアカウントから操作する必要があります。EOS のチェーンが初期化された時、デフォルトのシステムユーザ **eosio** が作成されます。このアカウントは、EOS のガバナンスとコンセンサスのコントラクトをロードします。最初のアカウントを作るには **eosio** アカウントを使います。

Linux や Windows を使ったことがあれば理解できると思いますが、最初から自分が使うユーザを作る必要があって、その時裏側には Linux が root ユーザ、Windows が administrator ユーザがあるのと同じです。

開発環境の場合、**eosio** に紐付いているキーペアのプライベートキーは、下記の固定値になっています。

5KQwrPbwdL6PhXujxW37FSSQZ1JiwsST4cqQzDeyXtP79zkvFD3

これをウォレットにインポートします。

インポートする

```
$ cleos wallet import
private key: imported private key for: EOS6MRyAjQq8ud7hVNYcfnVPJqcVpscN5So8BhtHuG\
YqET5GDW5CV
```

では、アカウントを作りましょう。

アカウント作成

```
$ cleos create account eosio bob EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8PiskXW3HnvG8XYb\
ihtKQ
executed transaction: a9984ddaaf88858b8892656b2e70c425c0f164d054663d1fdad2900524f\
97d7e  200 bytes  316 us
#         eosio <= eosio::newaccount            {"creator":"eosio","name":"bob","\
owner":{"threshold":1,"keys":[{"key":"EOS7bKBnxtQMXWkffYesa7Fh6dY5Q...
warning: transaction executed locally, but may not be confirmed by the network ye\
t         ]
$ cleos create account eosio alice EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8PiskXW3HnvG8X\
YbihtKQ
executed transaction: a7ede25b698cc5566ec29cf742b227f6a5ae5a8fd1a4014f7ef5e590562\
5ca4d  200 bytes  224 us
#         eosio <= eosio::newaccount            {"creator":"eosio","name":"alice"\
,"owner":{"threshold":1,"keys":[{"key":"EOS7bKBnxtQMXWkffYesa7Fh6dY...
warning: transaction executed locally, but may not be confirmed by the network ye\
t         ]
```

ここでは２つのアカウントを作成しました。アカウントには１つのキーペアだけ指定しましたが、この場合、**Owner** と **Active** ロールとも指定されたキーペアで制御出来ます。別々に指定したい場合は、もう1つキーペアを作成して、下記のように２つキーペアを指定する必要があります。

個別指定

```
$ cleos wallet create_key
Created new private key with a public key of: "EOS8PmYu42wo7wZGb4eD97SyFNE7wL2b64\
1npX1yDQHhhfMCQKKuy"

// ２つのキーペア指定した場合、1つ目は Owner、2つ目は Active ロールに紐づけます
$ cleos create account eosio jim EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8PiskXW3HnvG8XYb\
ihtKQ EOS8PmYu42wo7wZGb4eD97SyFNE7wL2b641npX1yDQHhhfMCQKKuy
executed transaction: 87386a2e60da9f0255894ac72f0b038748ae0370466d85a2f57edfe3ef0\
b9add  200 bytes  413 us
#         eosio <= eosio::newaccount            {"creator":"eosio","name":"jim","\
owner":{"threshold":1,"keys":[{"key":"EOS7bKBnxtQMXWkffYesa7Fh6dY5Q...
warning: transaction executed locally, but may not be confirmed by the network ye\
t         ]
```

アカウントを作成する前にプライベートキーをウォレットにインポートしたように、あるアカウントで何かの操作を実行する時は、その操作に必要なロールに紐付いているキーペアがウォレットに存在している必要があります。

操作する時、ウォレットから必要なロールに紐付いているキーペアを検索し、見つかったプライベートキーで操作にサインすることで、その操作を承認することを表明します。EOS は、その操作とサインをチェックし、必要な権限が持っていれば処理を実行します。

アカウントを作成する時特にロールなどを指定してなかったが、デフォルトは、操作対象アカウントの **Active** ロールの権限が求められます。そのため、上記のアカウント作成のコマンドは、下記と同じ意味になります。

アカウント作成

```
$ cleos create account eosio jim EOS7bKBnxtQMXWkffYesa7Fh6dY5QHFY8PiskXW3HnvG8XYb\
ihtKQ -p eosio@active
```

これでウォレット・キーペア・アカウント・権限の４者を触ってみて、お互いの関係を理解できると思います。

2.6 この章のまとめ

本書では、EOS のコンセンサスアルゴリズムである **DPoS + 非同期 BFT** 、ブロックのデータ構造などの基本知識を紹介することで、EOS の秒間トランザクション数が何故これほど速く出来ているかを説明しました。これらの内容を把握しておくと、その上に稼働するアプリケーションを構築する際は、実際裏側がどう動いているかを理解しやすいと思います。

あとがき

EOS は 2018 年 6 月リリースされ、まだ 6 ヶ月も経っていないので、本書を書いている期間の中でも、EOS 本体や周りの開発ライブラリがいくつのバージョンがリリースされているくらい、まだまだ速いスピードでどんどん進化しています。

EOS でよく批判されるのは、21 ノードしかないので分散型ではないと言うところですが、リアルタイムの地図で BP のノードを確認すると、中国に集中していることもなく、世界中で分散されているので、中央集権ではないと、筆者は考えています。もちろん、根本的には DPoS と PoW の論争になるので、ここでは割愛します。

EOS Go BP より

日本の EOS 分散型アプリケーション開発者はまだまだ少ないので、本章の内容を通じて読者の皆さんが本書を通じて自分の分散型アプリケーションを開発・公開できるようになると幸いです。

履歴

日付	内容
2018/12/01	初版リリース
2018/12/27	誤字修正