

0.0.1版

なぜなにトレ  
ント  
はっじまる  
よ!!

Lorem Ipsum Dolor Facilisis

# はじめに

1

# はじめに

- 
- 1. Torrent は悪いイメージもあるが貢献している
  - 2. P2Pの学習に最適

## はじめに

Torrentクライアントを開発しています。この本は、その時にP2Pについて学習した事をまとめたものです。

今でこそ、仕様書に書かれている内容に満足していますが、当初は上手く理解できない所が、かなりありました。

実際に作成していくうちに、理解できる範囲が広がっていきました。そして、そろそろ完成というところまで進みました。

この本では、この経験にならい、実際に Torrent クローンを作成しながら、P2Pについて解説して行きたいと思います。

## ネガティブなイメージ

ちまたでは、Torrent は違法なファイル共有アプリという認識が強いように思います。Twitter の検索機能で、Torrent と検索してみてください。Torrentについてネガティブなイメージを持つことでしょう。違法な利用を助長するようなツイートを発見する事ができるからです。

---

Google で、Torrentと検索してみてください。 「アメリカ合衆国のデジタルミレニアム著作権法に基づいたクレームに応じ、このページからxx件の検索結果を除外しました。」 といった文言が表示されます。また、Twitterで検索した結果と同じように違法な利用を助長するようなサイトを発見することができるでしょう。

## 世の中に貢献している

しかし、こういったネガティブなイメージは Torrent の一面でしかありません。まっとうな使い方も多くされています。

例えば、大規模なネットワークシステムのデプロイといった事があげられます。デプロイとは、ユーザーへサービスを提供するための、準備作業の事をいいます。

Google であれば、データを即座に検索できる用にサーバーを立ち上げる。Twitter ならば、ツイートの送信、表示、などをできるようにするといった事です。

大手のネットワークでは数千、数万のコンピュータが動作しています。

これらのコンピュータへ変更を加える必要が出てきた場合には、この変更を数千、数万のコンピュータへ反映する必要があります。こういったデータの配信の Torrent の技術が利用されています。

また、大量にデータを配信する環境が必要なのは企業だけではありません。個人でも活用されています。例えば、OSのイメージの配信に利用されています。

OSというと、WindowsやMACなどメーカーが CD や DVD といった記憶媒体を通して配布される事をイメージするかも知れません。

しかし、独自にOSをパッケージングする事は、大手の企業だけがする仕事ではありません。今では個人が趣味でOSをパッケージングして配信する事も可能です。可能なだけではなく、ありふれた行為になりつつあります。

---

しかし、個人で配信する場合、当然ながら数千、数万のサーバーを用意する事はできません。また、CD や DVD の記憶媒体を大量に配るにしても限界があります。そういった、個人がデータを配布するのに Torrent の技術が利用されています。

## 学習に最適

また、Torrent を学ぶことは、P2P を学ぶ上で最適な教材でないかと考えられます。

まず Torrent は、もっとも普及した P2P の通信方法であります。それだけではなく、最新の技術を取り入れ進化し続いている技術であります。

Torrent を学習すると、ネットワークアプリの作成のノウハウ、分散ハッシュテーブル、ゲーム理論を応用した柔軟なネットワークなどなど、基本から応用まで扱う事になります。

なによりも、おおくのアプリや仕様がオープンに公開されており、P2P を実例をもって学ぶことができるのです。

Torrent 以外でメジャーな P2P アプリはあります。しかし Torrent ほど、オッピロゲなはないでしょう。日本で流行した Winny のソースは公開されていません。Winny について知りたければ、ハックする必要があります。海外で流行した Winmx もそうです。

Torrent はその仕様が公開されています。どのような通信プロトコルが利用されているのか文章化されています。おおくの実装例がオープンソースとして公開されています。公開されているだけではありません。さまざまな言語で書かれています。Python、Ruby、Java、JavaScript、C++ などなど、ありとあらゆる言語で書かれています。

もしも、煮詰まった時は、これらの実装を読む事で保管する事も可能でしょう。あなたの得意な言語で読む事ができるのです。

## さいごに

Torrent のネガティブなイメージは一面にすぎない事。多くの史実用的なプロダクトを利用されていること。また、Torrent が

---

P2P の教材として優れている事は理解して頂けたでしょうか？  
少しでも興味を持たれた方、本書を通して Torrent の理解の助け  
になれば幸いです。

# Torrentって何

1. データの配信はコストが高い
2. Torrentは、低成本でデータは配信を可能にした

## Torrentって何

Torrentを違法なファイル共有ソフトという認識をもたれてい  
るかも知れません。しかし、それは間違います。Torrentは違  
法な利用を助長するような機能は含まれていません。たとえ  
ば、Torrentには匿名性はほとんどありません。また、Torrent  
クライアントはネットワークから特定のファイルを探し出し共  
有する機能を持っていません。

つまり、Torrent クライアントを利用してデータをダウンロー  
ドする事は、Webブラウザーを利用して、Webサーバーからデ  
ータをダウンロードするのと差がありません。

Googleなどの検索エンジンを使用して欲しいデータを探し、  
データをダウンロードします。ただ違うのは、Torrentはどの  
通信方法よりも効率良くデータを配信することができることで  
す。

データの配信はとてもコストが高い

---

インターネットでデータ配信するとしましょう。例えば生放送で動画を配信したい場合、どのくらいのコストがかかるでしょうか？

例えば、320×240 の画面サイズの動画だと、1秒間に50kb程度の帯域を使用します。任期のある生放送などは、一度に2000人以上の人人が視聴します。この場合、 $2000 \times 50\text{kb} = 100\text{mb}$ のデータ転送が必要になります。

ご家庭にある通信回線は10MB程度ですから、100MBという値は既に個人で配信できる量ではありません。

## 低成本でデータ配信が可能な優れもの

Torrentはこれらのコストをきわめて最小化することができます。Torrentの仕組みを利用すれば、こういったサービスを個人が提供する事ができます。

Torrent はデータをダウンロードするユーザーもデータの配信に加わるようにする事でこの問題を解決しました。

当然のことですが、配信に加わるコンピュータが増えれば増えるほど配信できるデータ量は増えます。

例えば、2.5MBのデータ配信を配信するとしましょう。これは、同時に50人くらいに配信できます。しかし、これが限界です。もしも、この50人が2.5MBの帯域をデータの配信に利用してくれたら、 $50 \times 2.5\text{mb} = 150\text{mb}$ の配信が可能になります。3000人程度にデータを配信できます。

さらに、この3000人がデータの2.5MBの帯域を利用してくれたら、 $3000 \times 2.5\text{mb} = 7500\text{mb}$ の配信が可能です。これは、15万人に配信が可能な規模です。

Torrentはこのような仕組みを実際に形にしました。Torrentによって、データを欲しがっている全ての人に素早くデータを配信する事ができるようになったのです。

# 序文

**Lorem Ipsum**

---

## 1. 本書の目的は、P2Pが難解な問題であるという誤解をとく事

### 序文

本書は、Torrentプロトコルについてまとめたものです。しかし、作者(kyorohiro)はBittorrentの作成者ではないので、その仕組みや、その仕組みにした意図をすべて組み取ることはできません。本書に記載されている内容は、kyorohiroかせ解釈し解決した事が書かれています。オリジナルではありません。オリジナルのように思案して、思考できるようになるべく、Torrentクローンを作成しました。

その仕組みを理解して実現しました。そして、独自のP2Pネットワークを思案して検討するに至ります。しかし偽物です。

色々書いていますが、偽物のの発言ですから「そのように解釈したのか？」 「それは違うのではないか？」 「この人もしかして理解できていないのでは？」 的な視点で見てもらうのが、丁度良いと思います。

### 本書が目指すゴール

本書を読む事で

---

誰もがP2Pアプリを作る事ができるようになります。P2Pで動画配信、P2Pを応用した柔軟なネットワークについて思案し、実現できる事ができるようになります。

P2Pと聞いて「何か難しい事をしているのではないか?」「しDHTと聞いて、高度な数学的難解な問題?」といった誤解を解きます。

本書を読んだ後もP2Pはなんら難しい事ではなくて、小学生が算数の問題を解くような簡単な問題なのだと理解してしまう。そして、あなたは、最新の論文を読み。それも足らず、新たな仕組みを提案する側にたっているに違い有りません。

# おおまかな仕組み

---

**Lorem ipsum dolor rutur  
amet. Integer id dui sed  
odio imperd feugiat et nec  
ipsum. Ut rutrum massa  
non ligula facilisis in  
ullamcorper purus dapibus.**



# おおまかな仕組み

## **Lorem Ipsum**

---

1. トラッカーサーバーでPeerを管理
2. 協力してデータを配信

## おおまかな仕組み

本章では、Torrentクライアントを利用してデータをダウンロードする時のおおまかな処理の流れを解説します。

- ・ データを配信している端末をさがす
- ・ ネットワークに加わり、データをダウンロードする

最初に宣言した通り、本書はTorrentクライアントを実そうする仮定を通して、P2Pの仕組みを解説していきます。なので、直にTorrentの実装に入りたいのですが、ワンクッションおくことにしました。

この章では、これから対峙する相手を紹介します。対峙する相手の姿、形を明らかにします。

実装を行っていくと「行き詰まるポイント」があります。そのほとんどは、Torrentで扱われている技術がシンプルな事に由来します。シンプルな問題と理解しつつも、そのシンプルさゆえに、確信がもってづ行き詰ります。

---

本章ではTorrentが難解でない事を解説します。本章を読み終えた頃には、得た確信を疑うことなく実装できるようになっていることでしょう。

# 配信している端末を 探す

## **Lorem Ipsum**

---

1. サーバークライアント方式がある
2. DHTを利用する方式がある

## 配信している端末を探す

Torrent ネットワークでは、ひとつのデータに対して配信する端末は多数存在します。以前に説明した通り、データをダウンロードする端末がデータの配信に加わる訳ですから当然の事でしょう。

データをダウンロードするには、広大なネットワークからデータを配信している端末を探しだす必要があります。本パートでは、データを持っている端末を探し出す方法について解説します。

## 配信している端末を探すのは大変

インターネット上には数えられないほどの端末が存在しています。そして、どの端末がどのデータを持っているか知るすべはありません。

例えば、インターネットは、IPv4アドレスで管理されています。IPv4の数だけインターネット上に端末が存在できます。IPv4は24bitの数字で表現されていますから、 $2^{24} = 42,949,672$  億以上の端末を扱う事ができます。

---

さらに、現時点では、このアドレスは足りなくなりつつあり。IPv6というよりたくさん端末を扱う事ができる方法が利用されるようになりました。つまり、42億で収まらない端末が既にネットワーク上に存在しているのです。

この42億以上の端末から任意のデータを配信している端末を探すのは困難です。例えば、ひとつの端末へ確認するのに、10ミリ秒必要だとしましょう。すべての端末へ確認するとしたら、420億秒必要になります。これは、一年以上かかる事を意味します。

## Torrent でお互いに位置を知る方法は2つある

Torrent では2つの方法が提供されています。ひとつは、サーバー・クライアントの仕組みを利用したものです。データを配信したい端末を管理するサーバーを用意する方法です。そのサーバーに聞く事で、配信している端末を発見することができます。

もうひとつは、DHTを利用したものです。データの配信/ダウンロードをしたい端末が協力してデータを配信したい端末を管理します。DHTに参加している端末に、欲しいデータの在処を聞く事で、配信しているデータを発見することができます。

### サーバー・クライアント方式

データを配信したい場合、「データを配信する端末を管理するサーバー」を用意します。このサーバーの事を Tracker と呼びます。Tracker はGoogleやYahooといったサーバーと同じ仕組みで動作しており、「[www.exsample.com](http://www.exsample.com)」や「xx.xx.xx.xx」といったアドレスを指定する事でアクセスできます。

「xxxx.torrent」と拡張子をもつデータTorrentファイルに、このアドレスを記載しておいて、データをダウンロードしたい端末に渡します。データをダウンロードしたい端末は、Tracker にアクセスして、データを配信している端末を知ることができます。

---

Tracker を用意する必要があります。すこし敷居が高いです。公開されたサーバーをレンタルするなりして公開されたIPを取得する必要があります。

しかし、その瞬間データを配信している端末のアドレスを管理するだけの、超軽量なサーバーアプリを用意すめだけで、数千、数万の端末へデータを配信できるようになる訳ですから、メリットは大きいです。

また、自分で用意しなくても、公開されている Tracker サービスを利用する事もできます。

## DHT(kademlia)を利用する方式

サーバー・クライアント方式と同じように、データをダウンロードしたい端末へTorrentファイルを渡します。

サーバー・クライアント方式と異なり、データを配信している端末を管理しているサーバーはTorrentファイルに記載されていません。

Torrentファイルから機械的に生成されるID (Hash値) を、元に、DHTネットワーク上から、データを配信している端末を管理している端末を探し出します。

## 六次の隔たりを利用する

では、どのようにして、IDから配信している端末を探すのでしょうか？。その仕組みは簡単です。周知のTorrentクライアントへ、そのデータを配信していそうな端末を紹介してもらいます。これを繰り返す事で、データを配信している端末へたどりつくことができます。

例えば、自分は欲しいデータを配信している端末の事は知らなくても、友人は欲しいデータを配信している端末を知っているかも知れません。

友人は知らなくとも、友人の友人は知っているかもしれません。現実世界では、「六次の隔たり」と言われており、6人もたどれば、世界中の誰とでもつながっているそうです。

---

Torrent ネットワークも同様の仕組みを利用することで、データを配信している端末を探し当てます。

## 距離を定義する

「六次の隔たり」は6回聞くだけで欲しいデータを持っている端末を探すというものです。これを実現するには、「誰に聞くと良いのか？」を知らなくてはなりません。聞く人を間違えると、6人で到達する事はないでしょう。聞くだけではだめなのです。

誰に聞けば良いのかを知る尺度としてDHTでは距離を定義します。欲しいデータと距離が近い端末に聞くようにする事で、まったく関係の無い人に聞くリスクを軽減するのです。

全てのデジタルデータに値を割り振るアルゴリズムが存在します。DHTでは、このアルゴリズムを使用してデータに値を振ります。端末にも、同様に値を振ります。

値どうしの差分が距離です。「へちまたん」と「ありすたん」にこのアルゴリズムをかけた結果、「100」、「123」と値

を振られたとしましょう。この場合、「へちまたん」と「ありますたん」の距離は123-100で23と定義できます。

## 自分に近い距離の事はより詳しい

距離が定義できましたが、聞いた人が自分よりもそのデータについて詳しくないとダメです。

DHTネットワークでは、「自分と距離が近い人の情報はたくさん知っている。遠い距離の人もすこしあはっている」といった状態を維持するようにします。

これで、聞く相手は自分よりも多くの情報を知っている可能性があがりましたね。無事「六次の隔たり」は実現できそうです。

# データをダウンロードする

**Lorem Ipsum**

---

1. データ配信する
2. ブロックに分割する
3. 常に速度が早い回線を探す

## ネットワークに加わりデータをダウンロードする

欲しいデータをもっている端末が見つかると、Torrent クライアントはデータのダウンロード処理を開始します。前章で説明した通り、Torrent クライアントの特徴のひとつとして、データのダウンロードすると同時に、ダウンロードしたデータを配信します。

単純に配信するサーバーが増える訳ですから、配信できるデータ量は増えます。しかし、注意深くその仕組みを考えると、非効率になる恐れもあります。例えば、極度にデータの配信効率の悪い端末からデータをダウンロードしてしまう。一部の端末に負荷が集中してしまう。データの配信に加わらなくても、データのダウンロードが出来てしまう。といった事があげられます。

Torrentは、これらの問題について考慮しています、多数の端末から、効率的にデータを配信するダウンロードする仕組みを提供しています。

## ブロック単位で、複数からダウンロード

Torrent クライアントはデータを任意のブロックに分割して管理しています。データをダウンロードするのも、配信するのも、このブロック単位で配信します。

このブロック単位のデータを複数の端末から、戦略的にデータを配信してもらいます。これによって、効率的にデータを配信する事に成功しています。

- 転送速度が安定する

負荷が集中した場合、データの配信する量は減ります。このような場合、負荷が高い状態の端末からデータを配信してもらうべきではありません。データの配信を一度止めて、別の端末からデータを配信してもらう事を検討すべきです。

Torrentでは、常にデータを配信する速度は計測されており、その瞬間最も安定してデータを配信できると思われる端末へデータの配信を依頼します。

- 上がり速度、下がり速度の差を平均かできる

我々が利用している回線はダウンロードする速度の方が、アップロードする速度よりも圧倒的に早い場合が多いです。2013年のWimaxはダウンロード速度が10Mbps、アップロード速度は2Mbpsとなっており。5倍くら差があります。

ひとつの端末からダウンロードするだけだと、相手がアップロードす速度分しかデータを配信してもらえません。つまり、Wimax上だと2Mbpsです。しかし、複数の端末からデータをダウンロードすれば、10Mbpsを一杯を利用する事ができます。

- フリーライドを許さない

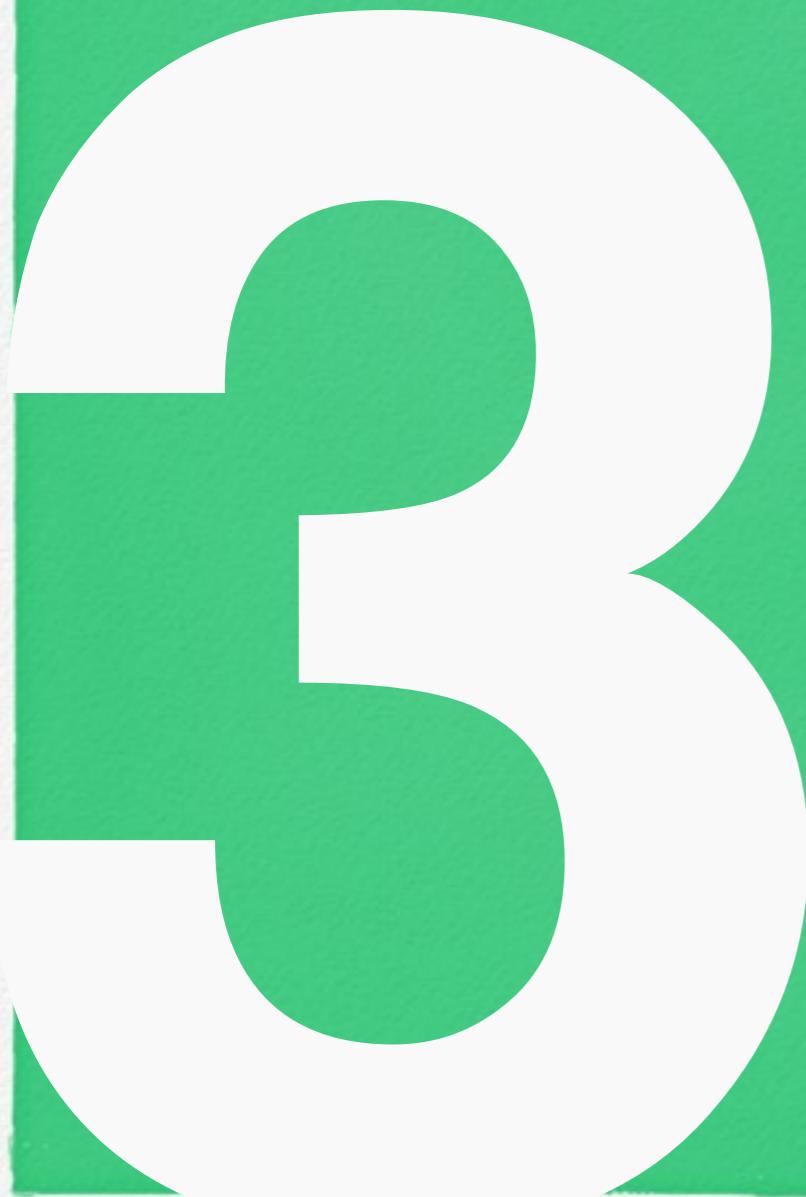
また、データを配信してくれた端末へ、優先的にデータを配信します。データをよりおおく配信してくれた端末とは通信が良好なはずです。また、Torrentネットワークに貢献していない、まったくデータを配信しない端末は冷遇される事を意味しています。つまり、他にデータを配信要求がある協力的な端末ほどにはデータが配信されないのでです。

---

これは、データの配信に加わった方が、より早くデータをダウンロードできる事を意味します。合理的に考えれば、データのダウンロードが完了するまでは、データの配信に協力する選択を選ぶことでしょう。それがもっとも早く、データをダウンロードする方法だからです。

# Torrentファイルを 読み込む

---



**Lorem ipsum dolor rutur  
amet. Integer id dui sed  
odio imperd feugiat et nec  
ipsum. Ut rutrum massa  
non ligula facilisis in  
ullamcorper purus dapibus.**

# Torrent ファイル

## Torrent ファイルを読みこんいでみる

Torrent でのデータのダウンロード処理は、Torrent ファイルを読み込む所からはじまります。

それなら、実際に Torrent ファイルを読み込み、必要な必要な情報を取得するところからはじめて見ましょう。

# Bencoding

### Lorem Ipsum

1. Torrentファイルはbencodingで書かれている。
2. Bencodingは、Integer、String、List、Dictionaryを扱える。

## Torrentファイルは Bencoding

Torrentファイルは、bencoding という形式で書かれています。Torrentファイルに記載されている事を読み解くためには、bencoding/bencode を解釈できるようにならなくてはなりません。

まずは、Bencodingのパーサーを書いていきましょう。

Bencodingは、文字列、整数、辞書、リストの4つのデータを扱うことができます。

beninteger : “i” [0-9]\* “e”

benstring : <string length> “.” <bytes array string>

string length : [0-9]\*

bendiction : “d” <dictelements> “e”

benlist : “l” <listelements> “e”

benobject : beninteger | benstring | bendiction | benlist

listelements : benobject ( benobject)\*

dictelements : benstring benobject (benstring benobject)\*

そして、上記のようなフォーマットで書かれています。

## • 文字列

Bencode で文字列は、「<文字の長さ> ":" <文字>」という形式で書かれています。例えば、「torrent」という文字列は、bencodeでは、「7:torrent」と書く事ができます。

もうひとつ、bencode の文字列は、バイトデータとして扱われる事もあります。IPアドレスのバイト表示や、Hash値などの非アスキーな範囲のデータなども、本形式で扱います。

例えば、日本語で「アイ」はSJISで表現すると「0x83, 0x41, 0x83, 0x43」の4バイトで表現できます。この場合、Bencode では、「4:アイ」と表記できます。

oden

4:oden

オデン SJIS

6:オデン

SHA1Hashデータ

20:<SHA1 Hashデータ>

-----

## • 整数

整数は0より大きな値を表現するデータです。「“i” \*[0-9] “e”」という形式で表すことができます。例えば、1024は「i1024e」と書くことができます。

ファイルのサイズ、ポート番号、といった、数字で表現できるものに利用します。

2

i2e

1024

i1024e

65526

i65526e

---

## • リスト

リストはデータ構造のひとつです。順序ありのデータを保持する事ができます。例えば、IPアドレスの一覧、ファイルの一覧、といったものを表現するのに使用します。

あいうえお、かきくけこ

l12:あいうえお12かきくけこe

128、100、500

l128ei100ei500ee

- 辞書

辞書はデータ構造のひとつです。キーワードとデータを関連づけて保持することができます。

例えば、RPGゲームの主人公のパラメータとして、名前、性別、レベル、得意な魔法、といったものが設定されているとしましょう。辞書はこの、パラメータ名とその値を関連づけます。「名前」というキーワードで、主人公の名前を記録したりできます。

例えばlevelが13で、nameが山田、を辞書で表現すると、

di5:leveli13e4:name4山田e

- 今後の表記について

今後、データ構造を表す場合は、以下の表記を利用します。

- リスト

「li512e4:teste」は、[512,test]

- 辞書

「d5:leveli13e5magic6halitoe」は {level:13,magic:halito}

# Bencoding実装

### Lorem Ipsum

---

1. Bencodingはパースしやすい構造
2. BNFから機械的にコードを抽出

## 見慣れたデータ構造に落とす

Dart言語には、Bencodingのデータ構造をもっています。今回は、Bencodingのデータを読み込み、Dart言語のデータ構造に落とこみます。

Bencodeの文字列は、Dart言語ではcore.Stringで表せます。

Bencodeの数字は、Dart言語では、core.intで表せます。Bencodeのリストは Dart言語のcore.List、Bencodeの辞書は、Dart言語ではMapで表現できます。

## Bencodeはパースしやすい構造

Bencodeはパースが容易な構造になっています。なぜならば、どのデータなのかが、最初の一文字目で判別する事ができるからです。

”i”ならば、整数。 ”0-9”ならば、文字列、 ”l”ならばリスト、 ”d”ならば辞書といった感じです。

これを、コードに直すと、以下のような感じになります。

```

Object decodeBenObject(data.Uint8List buffer) {
    if( 0x30 <= buffer[index] && buffer[index]<=0x39 ){//0-9
        return decodeBytes(buffer);
    } else if(0x69 == buffer[index]){// i
        return decodeNumber(buffer);
    } else if(0x6c == buffer[index]){// l
        return decodeList(buffer);
    } else if(0x64 == buffer[index]){// d
        return decodeDiction(buffer);
    }
    throw new ParseError("benobject", buffer, index);
}

```

書見ての通り、先頭の値に応じて処理が分岐しているだけです。後は、おののおのデータ構造とみなして、変換してあげれば完成です。

## BNF から機械的にパーサーを書く

BNFで書かれた構文は機械的にパーサーを書く事ができます。

- 規則名をルソッドにする。
- ルールに文字列がでてきた場合、一致するかチェックする
- ルールに規則がでてきた場合、そのメソッドを呼び出す

- ルールに違反かる場合は、Exception をスローする。
- プログラム言語、自然言語といったものは、もう少し工夫が必要ですが、今回は、基本的なルールを守るだけで実装できます。もう少し詳細な情報が欲しい場合は、「Language Implementation Patterns」という本を読む事をお勧めです。

具体的に、Bencode 用のパーサーを作成しながら、見ていきましょう。

例えば、Dictionary は以下のように書けます。

```

Map decodeDiction(data.Uint8List buffer){
    Map ret = new Map();
    if(buffer[index++] != 0x64){
        throw new ParseError("bendiction", buffer, index);
    }

    ret = decodeDictionElements(buffer);

    if(buffer[index++] != 0x65){
        throw new ParseError("bendiction", buffer, index);
    }
    return ret;
}

```

BNFと一対一の関係がある事が解ると思います。Dictionary は

---

「“d” bendictionelements “e”」と文法で表現されます。ですから、”d”という文字か確認。dictionelements のメソッドを呼び出す。“e”という文字か確認する。ルール通りです。

## テストを書く

テストを書きながら、パーサーを書いていきましょう。まずは整数からです。

```
unit.test("bencode: number", () {
    num ret = hetima.Bencode.decode(toBuffer("i1024e"));
    unit.expect(1024, ret);
});
```

このテストを満たすように、パーサーを書きます。bencodingで整数は、「i\*(0-9) e」と書けます。なので、これもルールに従って以下のように書けます。

```
num decodeNumber(data.Uint8List buffer) {
    if(buffer[index++] != 0x69) {
        throw new ParseError("benumber", buffer, index);
    }
    int returnValue = 0;
    while(index<buffer.length && buffer[index] != 0x65) {
        if(!(0x30 <= buffer[index] && buffer[index]<=0x39)) {
            throw new ParseError("benumber", buffer, index);
        }
        returnValue = returnValue*10+(buffer[index++]-0x30);
    }
    if(buffer[index++] != 0x65) {
        throw new ParseError("benumber", buffer, index);
    }
    return returnValue;
}
```

数字を取り出す部分が少し複雑ですが、無事テストが通るコードがかけました。この調子で、文字列、リスト、と同じようにテストしながら、作成すれば完成です。kyorohiroが作成した物は、以下にあります。[https://github.com/kyorohiro/dart\\_hetimalib](https://github.com/kyorohiro/dart_hetimalib) 事の顛末を知りたい方は参照してください。

## Bencodeデータを作成する

;;;;Dart言語のオブジェクトを Bencode に変換してみましょう。パーサーを作成した時と同様に、Dart言語の「Map, List, string, int」をBencodeの「Dictionary, List, String, Number」へ変換していきます。

パーサーを書いた時と同じように、BNFで書かれた構文は機械的に書けます。

- ・ 規則名をメソッドにする。
- ・ ルールに文字列がでてきた場合、文字列を追加する
- ・ ルールに規則がでてきた場合、そのメソッドを呼び出す

まずは、各データの種類に応じて、各規則を処理するメソッドへ処理を渡します。

```
void encodeObject(Object obj) {  
    if(obj is num){  
        encodeNumber(obj);  
    } else if(obj is String){  
        encodeString(obj);  
    } else if(obj is List){  
        encodeList(obj);  
    } else if(obj is Map){  
        encodeDictionary(obj);  
    }  
}
```

といった感じで書けます。後は、型に応じて、文法に従って変換していくべき完成です。

例えば、Dart言語のint型を Bencode の Number へ変換する方法は以下のようになります。

```
void encodeNumber(num num){  
    builder.appendString("i"+num.toString()+"e");  
}
```

"i"、値、"e" の順でデータを結合します。ルール通りです。例えば、Dart言語のList型を、Bencode のListへ変換する方法は以下の通りです。

```
void encodeList(List list) {
    builder.appendString("l");
    for(int i=0;i<list.length;i++) {
        encodeObject(list[i]);
    }
    builder.appendString("e");
}
```

“l”、Objectを変換するメソッド、“e”の順でデータを結合します。このように、BNFで記載した文法の通り、規則名をメソッドを渡することで実現できます。

## テストを書く

後は、テストを書きながら、完成させましょう。例えば、Numberのテストは以下のような感じでかけます。

```
unit.test("bencode: number", () {
    Uint8List ret = hetima.Bencode.encode(1024);
    unit.expect(UTF.encode(ret), "i1024e");
});
```

テストを書いている時に、仕様の考慮漏れが見つかることがあります。今回の場合でも、実際にテストしたことでの考慮漏れがあることがわかりました。以下のようないいコードを書くと、テストがFailedします。

```
unit.test("bencode: uint8list", () {
    Uint8List input = new Uint8List(1);
    input[0] = 0x61;//a
    Uint8List ret = hetima.Bencode.encode(1024);
    unit.expect(UTF.encode(ret), "2:a");
});
```

Bencodeでは、byte配列もStringで扱うのでした。今回、Byte配列は考慮できていませんでした。

この調子で、String, List, Dictionary の順で作成していくべきです。

# Torrent ファイルの 中身

**Lorem Ipsum**

---

1. TorrentファイルはBencodingで記載されている
2. Trackerのアドレスが記載されている
3. ファイル名が記載されている
4. ブロックデータごとのSHA1 Hashが記載されている

## Torrentファイルを読み込んでみよう

無事、Bencodeのパーサーを書く事ができました。これで、Torrentファイルの中身を解析できます。Torrentファイルの中身を確認してみましょう。

さっそく、Torrentファイルを読み込んでみましょう。Parserを作成していない方は、「[https://github.com/kyorohiro/dart\\_hetimalib\\_test/tree/master/hetimalib\\_sample/TorrentFileParser](https://github.com/kyorohiro/dart_hetimalib_test/tree/master/hetimalib_sample/TorrentFileParser)」を利用してください。

読み込んで見ると以下のようなデータ構成である事がわかります。

```
{  
    "announce":http://example.com/tracker,  
    "created by":torrent generator,  
    "creation date":1364723642,  
    "encoding":utf-8,  
    "info":{  
        "length":1024,  
        "name":xxx  
        "piece length":16384,  
        "pieces":<.....20バイト単位のバイナリデータ>  
    }  
}
```

---

- announce

Tracker サーバーのアドレスが記載されています。本アドレスのサーバー

にからデータを配信してくれる端末を紹介してもらえます。

- created by

本ファイル生成したツールを示す名前のようにです。

- creation date

本ファイルが生成された日にちを表しているようです。

- info.length

配信されているデータのサイズです。1024byte のデータである事がわかります。

- info.name

配信されているデータのファイル名です。xxx という名前である事が解ります。

- piece length

データを配信する際に、分割するサイズです。16kb単位で分割する事がわかります。

- pieces

分割されたデータごとのHash値です。データの正当性を判定するのに使用します。

## ファイルが複数の場合

ひとつのファイルを配信するデータは説明した通りです。複数のファイルをパッケージする場合は、もう少し構造が複雑になります。

```
{  
    "announce": http://example.com/tracker,  
    "created by": torrent generator,  
    "creation date": 1364723642,  
    "encoding": utf-8,  
    "info": {  
        "files": [  
            {  
                "length": 512,  
                "path": ["aaa", "bbb.mp3"]  
            },  
            {  
                "length": 1024,  
                "path": ["ccc", "ddd.mp3"]  
            },  
            ...  
        ]  
    },  
    "name": "xxx",  
    "piece length": 16384,  
    "pieces": <.....20バイト単位のバイナリデータ>  
}
```

さきほどのと比較すると、info辞書の中に、files リストが増えています。今回の場合だと、「xxx/aaa/bbb.mp3」、「xxx/ccc/ddd.mp3」という2つのデータが含まれている事が読み取れます。

## Torrentファイルを作成してみよう

これらの理解できた事を整理して、理解出来ていない事を発見しながら、Torrentファイルを作成するツールを作成してみましょう。無事作成できたならば、だいたいは理解できたという事になります。既に、Bencoding のパーサーはありますから、ゴールは目の前です。

```
{  
    Map file = {};  
    Map info = {};  
    file[TorrentFile.KEY_ANNOUNCE] = announce;  
    file[TorrentFile.KEY_INFO] = info;  
    info[TorrentFile.KEY_NAME] = name;  
    info[TorrentFile.KEY_PIECE_LENGTH] = piececSize;  
    info[TorrentFile.KEY_LENGTH] = targetLength;  
    info[TorrentFile.KEY_PIECE] = pieceBuffer;  
    Bencode.encode(file);  
}
```

といった感じで、必要な情報をMapに配置して、Bencode にエンコードすれば完成です。

---

実際に作成してみると、明らかでなかった点が現れてきます。例えば、ファイルが複数ある場合は、pieceデータをどのように算出するのでしょうか？まだ、明らかになっていませんでした。2つ生成する方法を思いつきました。どちらかが、Torrentで採用されている方法だと良いのですが..。

1. ファイル単位で、pieceデータを作成する。
2. 複数ファイルは結合してひとつのファイルと見なしてから、pieceデータを作成する。

実際に試してみたところ、2の方法が採用されているようです。具体的には、

```
{  
    Blob fileA =...  
    Blob fileB =...  
    Blob image = new Blob([fileA, fileB]);  
    FileReader reader = new FileReader();  
    reader.readAsArrayBuffer(image).then((e){  
        int start = 0;  
        do {  
            int end = start+pieceLength;  
            if(end<image.size()){ end = image.size();}  
            crypto.SHA1 sha1 = new crypto.SHA1();  
            sha1.add(reader.sublist(start,pieceLength));  
            print(sha1.close().toList().toString());  
            start = end;  
        } while(end < image.size());  
    });  
}
```

といったコードで生成できます。

# Trackerへアクセス してみる

---

**Lorem ipsum dolor rutur  
amet. Integer id dui sed  
odio imperd feugiat et nec  
ipsum. Ut rutrum massa  
non ligula facilisis in  
ullamcorper purus dapibus.**

# Trackerへアクセス

**Lorem Ipsum**

---

## 1. Peerの一覧を取得してみよう

### Trackerへアクセスしてみる

これまでの成果で、Torrentファイルから必要な情報を取出す事が出来るようになりました。Torrentクライアントは、Torrentファイルを解析が終わると、次にTrackerにアクセスします。そして、データを配信しているPeerの一覧を取得するのです。

本章では、実際に簡易のTracker サーバーも実際に作成しながら、Tracker から Peer の一覧を取得する方法について解説します。

# TrackerはHttpサーバ

**Lorem Ipsum**

1. TrackerはHttpサーバ
2. Getリクエストを用いる

## TrackerはHttpサーバー

Tracker は Httpサーバーです。皆さんのがいつも利用しているインターネットのからサイトを表示するのと同じルールで動作しています。

例えば、インターネットで調べ物をしたい時に、Googleを利用するとと思います。ChromeなりFirefoxなり、IEなどを利用して、「<http://www.google.com>」にアクセスします。すると、文字を入力するためのページが表示されます。

Trackerもそれと同様の仕組みで動作しています。異なるのは、人が見やすいように加工されたHtml形式のページを渡す変わりに、Bencodingでエンコードされたバイナリーデータが渡しています。

## Getリクエストで依頼をだす

Tracker では、Getリクエストを利用して、データを配信しているPeerの一覧を取得はます。Getリクエストは、URLの末端に、「?xx=yyy&mm=nnn」といった文字列を付与したもので、Googleなどの検索エンジンで検索した後、アドレスを確認

---

してみてください。例えば、androidと検索した場合、「?  
q=android&oq=android」といった文字列が追加されていると思  
います。Trackerも同様の仕組みで、依頼をだします。

## Httpサーバーを作成しよう

TrackerがHttpサーバーである事がわかったところで、Http サ  
ーバーを作成してみましょう。Dart言語では、簡単にHttpサー  
バを作成する事ができます。

まずは、ブラウザからGetリクエストを受け取った時にHel  
loと表示してみます。

```
{  
    HttpServer.bind(address, port).then((io.HttpServer server) {  
        server.listen((HttpRequest request) {  
            request.response.write("hello");  
            request.response.close();  
        });  
    });  
}
```

といった感じで書けます。Getリクエストで渡された値を知り  
たい場合には、「request.uri.queryParameters」として、確認  
できます。例えば、「

test」というキーで渡された値を相手に返すコードは以下のよ  
うに書けます。

```
{  
    HttpServer.bind(address, port).then((io.HttpServer server) {  
        server.listen((HttpRequest request) {  
            Map<String, String> parameter = request.uri.queryParameters;  
            request.response.write("hello"+ parameter["test"]);  
            request.response.close();  
        });  
    });  
}
```

これで、Getリクエストを扱う事ができるようになりました  
た。後は、このリクエストに応じて、Peerの一覧、つまりは、  
今までリクエストしてきた端末のアドレス等を渡せば、簡易の  
Trackerサーバーが完成します。

# リクエストの中身

**Lorem Ipsum**

---

1. peer id を生成する
2. info 辞書 の Hashを生成する
3. portを用意する
4. アドレスを生成する

## リクエストの中身

実際に、TrackerサーバーとTrackerクライアントの間でやり取りされるデータの形式について解析していきましょう。

Trackerはデータを配信しているPeerの一覧を管理しています。以前解説した通り、Torrentではデータをダウンロードする側もデータの配信にまわるようになりました。

Torrentではこの仕組みを実現するために、Trackerへ自身を登録する対価として、Peerの一覧を取得できるような仕組みになっています。

## データ配信に必要なものが入っている

データ配信に必要なものは、配信するデータ、配信するサーバーのIPアドレス、配信するサーバーのポート番号、配信する端末のID、どのデータを配信かるかの情報です。

まず一つ目、配信するデータは、Trackerに初めてアクセスする時には持っていないものなので、ここでは除外しましょう。

---

次に必要と思われるのが、IPアドレスです。このアドレスも用意する必要ありません。TrackerがあなたのIPアドレスを知っているからです。具体的には、Trackerにアクセスする際に、自身のIPアドレスをTrackerは知る事ができます。

例えば、Httpサーバーで通信相手のIPアドレスを返すようなプログラムは以下のような感じで書けます。

```
{  
    HttpServer.bind(address, port).then((io.HttpServer server) {  
        server.listen((HttpRequest request) {  
            request.response.write( request.connectionInfo.remoteAddress.toString());  
            request.response.close();  
        });  
    });  
}
```

このように、Httpサーバーは、相手のIPアドレスを知っています。なので、リクエストに含める必要はなさそうです。

その次に必要と思われる的是ポート番号です。他のPeerからの接続を待ち受けるには、IPアドレスとPort番号が必要です。

Trackerは待ち受けしているPort番号をIPアドレスのように知るすべはありません。リクエストとして通知してあげる必要がありそうです。

最後に、IDが必要です。Peerを識別する情報としてIPアドレスを利用する方法が考えられます。しかし、IPアドレスは複数の端末で共有している場合や、通信環境によっては同一のIPアドレスを長時間維持するができないかもしれません。

そこで、Peerごとに識別子を自身で生成して、それを利用します。

もう一つ、必要なものがありました。どのデータをダウンロードしたいかといった情報です。Trackerは同一のアドレスで多数のデータを配信するPeerを管理することができます。どのデータを配信してほしいのかをTrackerに通知する必要があります。

だいたい、リクエストら必要な情報が見えてきました。具体的にGetリクエストで使用する値について見てきましょう。

## Info辞書からIDを生成

まずは、どのデータをダウンロードしたいかを指定するための IDを用意しましょう。 Torrent が扱う全てのデータは衝突しない固有の識別子を持っています。これは、ひとつの配信データを複数の Tracker で管理したい場合に有効な方法です。 唯一の IDで有るならば、このTrackerで管理しても、関係のないデータが混ざる事がないからです。 このような方法を用いる事で、 Tracker への負荷を分散する事が可能になります。

では、どのように唯一IDを生成するのでしょうか。 特定の管理団体などにIDをもらう必要があるのでしょうか？ Torrentでは SHA1 が用いられています。

SHA1 は、全てのデジタルデータに 160bit(20byte) の ID をふるアルゴリズムです。 2の160乗のIDの振る事ができますから、異なるデータなのに、同じIDを振られる事はありません。

Torrentでは、 Info辞書のSHA1をとりIDとして利用します。

Dartでは以下のように書けます。

```
{  
    data.Uint8List list = Bencode.encode(file.mMetadata[TorrentFile.KEY_INFO]);  
    crypto.SHA1 sha1 = new crypto.SHA1();  
    sha1.add(list.toList());  
    sha1.close(); //return value is id  
}
```

見ての通り Torrent ファイルのなかの Info 辞書の部分のバイナリでデータのSHA1をとるだけです。

## 自身のIDを用意する

Info辞書の識別子と同様に、自身のIDも20バイトのバイナリデータを利用します。 そして、IDの生成は各Torrentクライアント行います。 IDの生成は、 Info辞書のSHA1をとった時のように明確に生成するルールは存在していません。

他の Torrent クライアントが生成したIDと衝突しない用に注意し生成する必要があります。

ここでは、「乱数を使う方法」を紹介します。

```
class PeerIdCreator {  
    static math.Random _random = new math.Random(new DateTime.now().millisecond);  
    static List<int> createPeerid(String id) {  
        List<int> output = new List<int>(20);  
        for (int i = 0; i < 20; i++) {  
            output[i] = _random.nextInt(0xFF);  
        }  
        List<int> idAsCode = id.codeUnits;  
        for(int i=0;i<5&&i<idAsCode.length;i++) {  
            output[i+1] = idAsCode[i];  
        }  
        return output;  
    }  
}
```

1byte づつ乱数を計算して代入しています。追加機能として、どのクライアントで生成されたを判別できるように、idの一部に文字列を追加できるようにしました。

## Portを用意する

Torrent クライアントは、サーバーの機能とクライアントの機能をもっています。他のTorrentクライアントからの接続を待ち

受けるには、サーバーの機能を立ち上げる必要があります。この時のサーバーのポート番号を用意します。

```
{  
    HttpServer.bind(address, port).then((io.HttpServer server) {  
        server.listen((HttpRequest request) {  
            request.response.write( request.connectionInfo.remoteAddress.toString());  
            request.response.close();  
        });  
    });  
}
```

以前、書いたHttpサーバーのコードで指定したPort番号で良いです。

実際には、ご家庭ネットワーク環境では、ここで指定したポート番号と、実際に外部から見えているポート番号が異なる場合があります。このような場合に対処するには、もうひと工夫必要です。

付録の「なぜなにNat Traversal」を参照してください。

# アドレスを生成する

だいたい必要な情報は整いました。さっそくアドレスを生成してみましょう。

実際には、以下のような情報も追加してアドレスを作成する事になります。

- "info\_hash"

TorrentファイルのInfo辞書のSHA1

- "peer\_id"

生成した20byteのバイナリデータ

- "event"

Torrentクライアントの状態を表します。“started”, “stopped”, “completed”の3つ。それぞれ、データーのダウンロード中である場合は、“started”、データーのダウンロードと配信から抜ける場合は、“stopped”、データーのダウンロードを完了した場合は、“completed”が指定されます。

- "downloaded"

今までダウンロードが完了したバイト数

- "uploaded"

今までに配信したバイト数

- "left"

ダウンロードが済んでいないデータのバイト数

これらのデータを結合して、URLを生成するば完成です。

```
String toString() {
    return scheme + "://" + trackerHost + ":"
        + trackerPort.toString() + ""
        + path + toHeader();
}

String toHeader() {
    return "?" + KEY_INFO_HASH + "=" + infoHashValue
        + "&" + KEY_PORT + "=" + port.toString()
        + "&" + KEY_PEER_ID + "=" + peerID
        + "&" + KEY_EVENT + "=" + event
        + "&" + KEY_UPLOADED + "=" + uploaded.toString()
        + "&" + KEY_DOWNLOADED + "=" + downloaded.toString()
        + "&" + KEY_LEFT + "=" + left.toString();
}
```

といった感じで書けます。無事URLを生成する事ができましたね。

# レスポンスの中身

### Lorem Ipsum

---

1.  **Lorem ipsum dolor sit amet, consectetur.**
2.  **Nulla et urna convallis nec quis blandit odio mollis.**

### Trackerを作成しよう

本章では、 Tracker へのレスポンス について解説していきます。リクエストとレスポンスの両方について理解ができた訳ですから、あなたは、 Tracker サーバーを作る事がでくるようになります。

実際に、 Tracker サーバーを作成してみましょう。そして、 実際動作させてみて、 想定した通りのリクエスト、 レスポンスが飛び交い、 データの配信が開始される事を確認ていきます。

### レスポンスの中身はPeerの一覧

Tracker はデータを配信している Peer の一覧を返します。 レスポンスとして最低限必要なのは、 アドレスとポート番号の一覧です。

この2つのデータさえあればも Torrent クライアント 同時でデータを配信は合う事が可能です。

実際に渡されるデータの形式を見ていきましょう。

## レスポンスはBencode形式

Peer の一覧も Torrent ファイルと同様に Bencoding 形式で渡されます。

```
{  
    peers:  
    [  
        {  
            peer_id:<benstring>  
            ip:<benstring 127.0.0.1の形式>  
            port:<beninteger>  
        }  
        {  
            peer_id:<benstring>  
            ip:<benstring 127.0.0.1の形式>  
            port:<beninteger>  
        }  
        ...  
    ]  
  
    interval:<beninteger 次にTrackerへアクセスする時間 単位は秒>  
}
```

intervalキーを除けば、ipアドレス、port番号、peer\_id いっ

た、リクエスト時にTracker へ渡した情報が含まれている事データが含まれている事が読み取れるでしょう。

“interval”は次に Tracker へアクセスすべき目安の時間です。

どの程度の間隔でアクセスすべきなのは、Torrent クライアントからは判断できません。どの程度のPeerが配信に参加していて、どの程度のPeerが新規に参加しているを知らないためです。

Tracker サーバーは自身の状態を加味して、この時間を決めます。

例えば、頻繁にTrackerへアクセスされるとTrackerサーバーへの負荷があがります。Peerのリストは更新されていない。更新されても対象にあなたのアドレスが含まれるので更新する必要性が弱い。といった時は、特別、アクセス時間を長く取るといった対応が考えられます。

## 圧縮して送信する事もできる

データを圧縮して相手へ送る事もできます。リクエストに "&compact=1" を含めると、対応しているTrackerであれば、データを圧縮して送ってくれます。

```
{  
    interval:<beninteger 次にTrackerへアクセスする時間 単位は秒>  
    peers:<benstring byte配列 4byte address, 2byte port>  
}
```

peer のアドレスと port 番号を、バイト配列として扱うことでデータを圧縮します。BIG\_ENDIANで記録されます。

たとえば、IP4のアドレスは、「127.0.0.1」ならば、[0x7F, 0, 0, 1] と配列になります。ポートが8080ならば、[0x1F, 0x90]といった並びで記録します。

Torrentでは、これせのデータを繋げて、アドレスが「127.0.0.1」で、ポートが8008ならば、[0x7F, 0x00, 0x00, 0x01, 0x1F, 0x90] といった並びのバイト配列として扱います。

## 不正な値を受け取った時もBencode

Trackerはデ不正な値を受け取った時も、Bencode形式のデータを返します。人がよんで解るようなメッセージを付けて以下のような形式で返します。

```
{  
    failure reason:<benstring 失敗した理由>  
}
```

これで、レスポンスされデータの形式については、済みました。では、実際にTrackerサーバー、Trackerクライアントほ実装してみましょう。