末尾呼び出し最適化とJavaScript

2021/04/23 - @kota_yata

すごくなりたい
がくせい
ぐるーぷ

おれ

Kota Yatagai (@kota_yata)

- 趣味でフロントエンド開発をしてます
- 暗号系結構好きです
- ブロックチェーンの勉強はじめました
- Svelte / Nuxt

今日のはなし

再帰関数の最適化を行う「末尾呼び出し最適化」の説明をして、今のJavaScript環境での実装状況とかの話もします。

末尾呼び出しってなに？

Tail Call

関数の末尾に返り値として関数を呼び出している関数

その中で自分自身を返り値として呼び出している=末尾で再帰している関数を末尾再帰関数という

```
// 階乗を求める末尾再帰関数の例
const factorial = (number, result) => {
  if (number === 0) return result;
  return factorial(number - 1, result * number);
}
```

末尾呼び出し最適化ってなに？

Tail Call Optimization（以下TCO）

コンパイラなどの言語処理系で行う関数の最適化処理（速度改善の最適化ではない）

末尾呼び出し関数においてスタックフレームを使い回すことでメモリ使用量を抑える手法

末尾再帰関数の場合は実質的に末尾の再帰を除去、ループ処理に変換している➡️末尾呼び出し除去（Tail Call Elimination）

TCOをすればスタックオーバーフローを起こさないことが保証される

もう少し詳しく

末尾再帰関数（さっきの階乗の関数の例）の場合、スタックを追うと

```
Trace
  at factorial (/Users/hoge/test.js:2:11)
  at factorial (/Users/hoge/test.js:4:10)
  at factorial (/Users/hoge/test.js:4:10)
  at factorial (/Users/hgoe/test.js:4:10)
```

末尾で同じ処理を繰り返してなんらかの条件で処理を終了する...

ループと同じでは？

実質的な処理はループと同じなので、機械的にループ文に変換することができる

逆に末尾再帰でない再帰関数を機械的にループ文に変換するのはかなり難しい

すごくなりたい
がくせい
ぐるーぷ

TCOが実装されている言語

- Kotlin
- Haskell
- Erlang
- Python...サードパーティーのライブラリで実装されている（baruchel/tco）
- Scala...末尾再帰の場合のみ（通常の末尾呼び出しでは最適化されない）

  ...

  JavaScriptさん？？

JavaScriptエンジンにおけるTCO

現在JavaScriptCoreエンジン（Safari）でのみ末尾最適化が実装されている

ES6でTCOを行うように定められたがV8（Chrome）, SpiderMonkey（Firefox）などの主要ブラウザでは実装されていない

なぜなのか。

経緯

2011年頃 : TC39内で同意をとり、TCOの実装をES6に盛り込んだ
2015年 : ES6がリリース。この時点ではまだどのブラウザもTCOを実装していなかった
2016年初頭 : Safariが実装。ChromeはExperimental Feature Flag（Origin Trials）として実装した。**ここでMozillaチームとMicrosoftチームが文句を言い始める**

Mozillaさんの主張

TCOがブラウザに実装された場合、特定の条件でのみ暗黙的にTCOが実行されることになり、どの関数がTCOの対象なのかわかりにくくなるじゃないか。

つまり...

TCOが適用される条件

その呼び出しが確実に最後の処理になる場合

- ブロック直下での返り値

```
const hoge = () => {
  ...
  return hoge() // 確実に最後の処理になるためTCOが適用される
}
```


TCOが適用される条件

その呼び出しが確実に最後の処理になる場合

- if文の場合if内とelse内の両方

```
const hoge = () => {
  if (...) {
    return hoge();
  } else {
    return hoge();
  }
}
```

どっちに転んでも `hoge()` が実行されるためTCOの対象

TCOが適用される条件

その呼び出しが確実に最後の処理になる場合

- try-catchの場合catch内

```
const hoge = () => {
  try {
    return hoge(); // この後にエラーハンドリングがあるのでTCOの対象外
  } catch (err) {
    return hoge(); // このエラーハンドリングが最後の処理になるのでTCOの対象
  }
}
```

などなどTCOが適用される条件は単純ではない

すごくなりたい
がくせい
ぐるーぷ

Mozillaさんの主張

TCOがブラウザに実装された場合、特定の条件でのみ暗黙的にTCOが実行されることになり、どの関数がTCOの対象なのかわかりにくくなるじゃないか。

つまり...

開発者がTCOされると思って実装した再帰関数が実はTCOの対象ではなく、スタックオーバーフローを起こすことが増えてしまう。TCOはされないと分かっていて実装するよりも危険な状態である

他にもデバッグが難しくなるなどの問題点も指摘した（スタックトレース時にはすでにTCOされているため何も出力されない）。

スタックオーバーフローが起こした事件

2013年、米国でトヨタ車が急加速する事故が多発。

原因はファームウェアの欠陥

CPUでスタックオーバーフローを回避する処理がなされていなかった

死傷者が出たことや当初トヨタが欠陥を隠蔽したこともあり、最終的に1100億円の賠償金を支払って和解

Microsoftさんの主張

WindowsABIとの兼ね合いで効率的に実装できません。

そうですか、頑張ってください。

経緯

2011年頃 : TC39内で同意をとり、TCOの実装をES6に盛り込んだ
2015年 : ES6がリリース。この時点ではまだどのブラウザもTCOを実装していなかった
2016年初頭 : Safariが実装。ChromeはExperimental Feature Flag（Origin Trials）として実装した。**ここでMozillaチームとMicrosoftチームが文句を言い始める**
2016年中頃 : Mozillaの指摘を受けてChromeチームがSyntactic Tail Callsというプロポーザルを提出する

Syntactic Tail Calls（以下STC）

その関数をTCOの対象にするかどうか明示的に指定するような文法を盛り込んだプロポーザル

```
const formalHoge = () => {
  return formalHoge(); // TCOされない従来の書き方
}
const syntacticHoge = () => {
  return continue syntacticHoge(); // STCで盛り込まれたcontinueを用いた明示的なTCOの指定
}
```

こうすることで、開発者が主体的にTCO対象にするか否かを選択できる。
デバッグが容易になり、スタックオーバーフローも正しく回避できる。

最高じゃん！

Safari 「ダメです」

STCプロポーザルが提出された時点での立ち位置

Microsoft : TCOは実装すべきでない
Mozilla : TCOは実装すべきでない。なんならTCOの仕様をES6から削除すべき
Chrome : TCOの代替策としてSTCを提示した
Safari : TCOをES6の仕様通りに実装すべきである

そもそもSTCとES6のTCO仕様は共存できない（明示的に指定するか暗黙的に対象にするか）

その上でSafariは唯一TCOの実装を推し進めているブラウザだったため、STCを承諾するわけには行かなかった。
Mozillaも反対し、**結局STCのプロポーザルは承諾されなかった。**

そんなこんなで現状整理

- Safari以外のブラウザはTCOを実装していない。STCは標準化されず。
- BabelではV5までTCOが実装されていたがV6で消滅（最新版はV7）

2021年のJavaScript環境で末尾呼び出し最適化は諦めた方が良さそう...

じゃあどうすれば...

自分で最適化しましょう。

```
const factorial = (number, result) => {
  if (number === 0) return result;
  return factorial(number - 1, result * number);
}
// ⬇ 自分でwhile文に書き直す
const factorial = (number) => {
  let result = 1;
  let count = number;
  while(count > 0) {
    result *= count;
    count--;
  }
  return result;
} // やってることはコンパイラが機械的に行っているTCOと同じ
```

感想

でもやっぱり再帰の方がスマートに書けることは多いからTCO欲しい。
当時の議論からすでに5年近く経ってもSafariのTCOに関して開発者から文句が出ているわけではないということは、TCOを実装してもある程度うまく回るのではないかという感想です。

TC39のIssues見ているとたまに「そろそろTCO再検討しても良いのでは？」みたいな提案も見受けられるので、今後の動向に注目したい

参考文献

https://tc39.es/proposal-ptc-syntax/
https://v8.dev/blog/modern-javascript
https://stackoverflow.com/questions/54719548/tail-call-optimization-implementation-in-javascript-engines
https://kangax.github.io/compat-table/es6/
https://stackoverflow.com/questions/25228871/how-to-understand-trampoline-in-javascript/27704484#27704484
http://js-next.hatenablog.com/entry/2016/01/28/232111
http://js-next.hatenablog.com/entry/2016/04/27/194215
https://2ality.com/2015/06/tail-call-optimization.html#checking-whether-a-function-call-is-in-a-tail-position
https://github.com/tc39/proposal-ptc-syntax
https://github.com/tc39/proposal-ptc-syntax/issues/23

すごくなりたい
がくせい
ぐるーぷ