ソフトウェア開発
第 14 回目授業

平野 照比古

2018/1/18

良いコードとは

多くの公開されているコードを眺めているといくつかの共通点が見つかる。

- プログラムの構造がわかりやすくなるように字下げ (インデント) を付ける。
- 変数名や関数名は実体を表すようにする。
  いくつかの単語をつなげて変数名や関数名にすることが多くなる。
  これらが読みやすいように途中に出てくる単語の先頭は大文字にすることが最近の傾向である (キャメル形式と呼ばれる)。
- 不必要に長いプログラム単位を作らない。
  ある程度まとまった作業をする部分は関数にすると見通しの良いプログラムとなる。
- プログラム自体が説明書になっている。
  コメントを多用して必要な説明をすべて記述する。

プログラムとデータの分離

- ある処理をするときの分岐の条件をプログラム内に直接記述すると、分岐の条件が変わったときにはプログラムをコンパイルしなおす必要が生ずる。
- これを避けるためには外部からデータを読み込んで、それに基づいた処理を行うようにする。

成績評価

期末試験の結果成績をつける。

- 90 点以上ならば S
- 80 点以上ならば A
- 70 点以上ならば B
- 60 点以上ならば C
- それ以外は E

期末試験の点を与えて、評価の S などを返す関数 seiseki を作成する。

簡単な方法 (JavaScript)

```
function seiseki(val) {
  if(val >= 90) return "S";
  if(val >= 80) return "A";
  if(val >= 70) return "B";
  if(val >= 60) return "C";
  return "E";
}
```

配列の利用 (1)

```
function seiseki(val) {
  var Score = [90,80,70,60,0];
  var Results= ["S","A","B","C","E"];
  var i;
  for(i=0;i<Grade.length;$i++) {
    if(val>=Grade[i]) return Hyouka[i];
  }
  return "Error";
}
```

配列の利用 (2)

配列をまとめる。

```
var Results =[["S",90],["A",80],["B",70],["C",60],["E",0]];
```

連想配列の利用

```
var Results ={S:90,A:80,B:70,C:60,E:0};
function seiseki(val) {
  for(v in Results) {
    if(val >=Results[v]) return v;
  }
}
```

JavaScript では `for(..  in ..)` で連想配列の要素をすべて渡ることができ、かつわたる順序が定義された順なのでこのコードが可能

JavaScript における注意

- JavaScript 内から外部ファイルを自動で読み込むことができない
- 変数の形で値を用意
- この変数部分を別のプログラムから作成する方法も可能

複数でシステム開発をするときの問題点

- 複数の人間で開発している場合、開発者の間で最新のコードの共有
- ファイルを間違って削除したり、修正がうまくいかなかったときに過去のコードに戻す
- 安定しているコードに新規の機能を付け加えてテストをする場合、安定したコードに影響を与えないようにする

このようなコードの変更履歴の管理するソフトウェアをバージョン管理ソフトという。バージョン管理ソフトウェアの対象となるファイルはテキストベースのものを主としている。

バージョン管理ソフトとは

バージョン管理ソフトは各時点におけるファイルの状態を管理するデータベースである。このデータベースは一般にリポジトリと呼ばれる。

バージョン管理ソフトを用いた開発手順

- リポジトリからファイルの最新版を入手
- ローカルな環境でファイルを変更。
- ファイルをリポジトリに登録

バージョン管理ソフトを用いた開発手順の問題点

- 同じファイルを複数の開発者が変更した場合、競合が発生していないかをチェックする機能
- この機能がない場合には同じファイルの変更は同時に一人しか変更ができないようにファイルをロック

開発の流れ

- 開発の流れをブランチと呼ぶ
- あるブランチに対して新規の機能を付け加えたいときなどには元のブランチには手を付けずに別のブランチを作成して、そこで開発
- 機能が安定すれば元のブランチに統合 (merge)

CVS(Concurrent Versions System) と Subversion

- 管理するためにサーバーが必要
- 開発者はこのサーバーにアクセスしてファイルの更新などを行う

git の場合

- 各開発者のローカルな環境にサーバー上のリポジトリの複製を持つ
- ネットワーク環境がない状態でもバージョン管理が行える
- git ではネットワーク上に GitHub と呼ばれるサーバーを用意しており、ここにリポジトリを作成することで開発者間のデータの共有が可能
- データを公開すれば無料で利用できる
- 有料のサービスやサーバー自体を個別に持つサービスも提供
- 有名なオープンソースのプロジェクトが GitHub を利用

git の特徴

詳しい使い方は https://git-scm.com/book/ja/v2/を参照

- 分散型のバージョンシステムなので、ローカルでブランチの作成が可能
  集中型のバージョン管理システムではブランチの作成が一部の人しかできない。
- 今までのファイルの変更履歴などが見える。
- GitHub 上ではファイルの更新などの通知を受けることができる。
- 開発者に対して変更などの要求が可能 (pull request)

git のインストール

- git はローカルにリポジトリを持つのでそれを処理する環境をインストールする
- git for windows が良い
- 「Git Bash」、「GitCMD」と「Git GUI」の 3 つがインストールされる。
- Unix 風のコマンドプロンプトが起動する Git Bash がおすすめ

初期設定

- GitHub で使用するリポジトリのユーザ名を登録
  `git --config --global user.name "Foo"`
  ここでは Foo がユーザ名となる。
- GitHub からの通知を受けるメールアドレス
  `git --config --global user.email "Foo@example.com"`

SSH Key の登録

- GitHub との接続には SSH による通信で行う
- この通信はは公開鍵暗号方式で行う
- キーの生成は次のコマンドで行う
- `ssh-keygen -t rsa -C "Foo@example.com"`
  `-C` のオプションはコメント
- キーを保存するフォルダが聞かれるが、デフォルトでかまわない
- passphrase の入力が求められる
- 指定したフォルダの `.ssh` の下に `id_rsa`(秘密鍵) と `id_rsa.pub`(公開鍵) の 2 つのファイルが作成される。

GitHub のアカウントの取得

- ローカルだけで開発をするのであればアカウントは不要
- データを交換するのであればアカウントを取ることを勧める
- 作成した公開鍵の内容をアカウントで設定する
- アカウントを取ったら GitHub 上でテストのプロジェクトを中身が空で作成

リポジトリの作成と運用

リポジトリの初期のブランチ名は master

コマンド	パラメータ	説明
git init		プロジェクトの初期化
git add	ファイルリスト	プロジェクトへのファイルの登録
git commit	-m コメント	リポジトリへの変更の登録
git remote add	短縮名 リポジトリ	リポジトリの短縮名の登録
git push	-u 短縮名 ブランチ名	ブランチへの変更の登録
git clone	リポジトリ名	リモートのリポジトリのコピーを作成
git pull	リポジトリ名 ブランチ名	リモートのリポジトリのコピーを作成

新しいプロジェクトの構成

1. git init で新たにリポジトリを作成、または、git clone で GitHub からリポジトリのコピーを取得する。
2. ファイルを編集
3. 編集したファイルを git add でステージ
4. git commit でその時点での変更を登録
5. -m オプションで簡単なコメントをつけることを忘れないこと。
6. このオプションをつけないと vim というテキストエディタが立ち上がり、コメントの編集を行う
7. git push リモート名 ブランチ名 で変更をリモートのリポジトリに反映される。
8. これが拒否された場合には誰かが push しているのでその変更を pull してから調整して再度 push する。

ファイルの状態の注意

- リポジトリ内のファイルは追跡されている (tracked) ものと追跡されていない (untracked) に分けられる。
- ファイルの状態として変更されていない (unmodified)、変更されている (modified) とステージされている (staged) の 3 つの状態がある。
- modified から staged にするコマンドが `git add` である。このコマンドを新規にリポジトリに追加するという意味にとらえないことが必要である。

分散処理とは

- あるデータを処理するためにはいくつかのプログラムが必要
- そのプログラムの実行単位をプロセスと呼ぶ。
- プロセスは他のプロセスと環境が独立しており互いに直接、データの参照ができない。
- プロセス間でデータを交換するための機能としてプロセス間通信がある。

スレッド

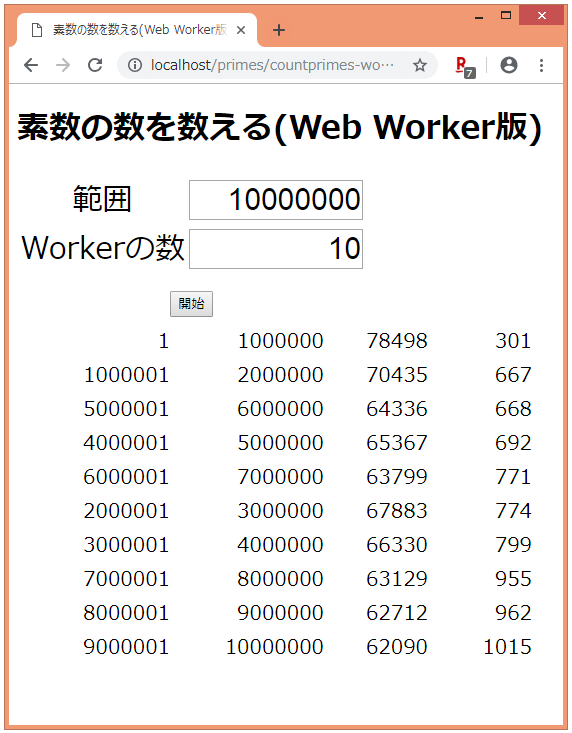
- 最近の OS ではプロセスの中で並行処理が可能となるスレッドが提供されている。
- スレッドはプロセスとは異なり、スレッド間でプロセス内のデータを共有することが可能
- 共有されるデータの書き込みに関しては複数のスレッドから同時に書き込みが起きないようにするなどの注意が必要
- それができる機能を OS は提供

Web Workers

- JavaScript で並行処理を行う機能を提供するものとして Web Worker オブジェクトが提案
- Worker オブジェクトは呼び出されるプログラムとは別のメモリー空間で実行
- 元のプログラムとのデータ交換はメッセージの通知というイベントを通じて行う。
- Web Workers はプロセス間通信に似た環境を提供している

素数の数を求める

$10,000,000$ 以下の素数を $1,000,000$ ずつの区間に分けてその区間内にある素数の数を Web Workers を使用して求めるプログラム



項目は左から区間の下限と上限、その区間に含まれる素数の数、開始時間からその区間の実行終了までの時間 (ミリ秒単位)

素数の数を求める–HTML ファイル

```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <meta charset="UTF-8"/>
 5     <script type="text/ecmascript" src="countPrimes-workers.js"></script>
 6         <link href="primes.css" rel="stylesheet" type="text/css"></link>
 7     <title>素数の数を数える (Web Worker 版)</title>
 8   </head>
 9   <body>
10     <h1 class="head">素数の数を数える (Web Worker 版)</h1>
11     <form id="form">
12       <table>
13         <tr>
14           <td align="center">範囲</td>
15           <td><input type="text" size="7" id="limit"/></td>
16         </tr>
17         <tr>
18           <td align="center" >Worker の数</td>
19           <td><input type="text" size="7" id="No"/></td>
20         </tr>
21         <tr>
22           <td colspan="2", align="center">
23             <input type="button" id="Go" value="開始"></input>
24           </td>
25         </tr>
26       </table>
27       <table id="result"></table>
28   </body>
29 </html>
```

素数の数を求める–HTML ファイル (解説)

- 5 行目で Worker を起動し、その結果をを処理する JavaScriptっファイルを読み込む。
- 6 行目でこのページに適用されるスタイルシートの読み込む。
- 13 行目から 16 行目で素数の個数を求める範囲を指定するテキストボックスを配置
- 17 行目から 20 行目で同時に発行する Worker の個数を指定するテキストボックスを配置
- 17 行目から 20 行目で計算開始するボタンを配置

素数の数を求める–CSS ファイル

```
 1 input[type=text]{
 2     text-align:right;
 3     font-size:30px;
 4 }
 5 td{
 6     font-size:30px;
 7 }
 8 #result td{
 9     text-align:right;
10     font-size:20px;
11     width:150px;
12 }
13 #result td:nth-child(n+3){
14     width:100px;
15 }
```

素数の数を求める–CSS ファイル (解説)

- 1 行目から 4 行目はテキストボックスに適用される。このページのテキストボックスは数が入力されるので、文字列を右寄せに (2 行目)
- 5 行目から 7 行目では table 要素内の td 要素の文字の大きさを指定
- 8 行目から 12 行目では素数の個数を表示する table 要素ないの td 要素の文字の位置 (9 行目)、文字の大きさ (10 行目) と表示幅 (11 行目) を指定
- 13 行目から 15 行目では右 2 つの td 要素の幅を前の 2 つと異なる値に設定 (100px)

素数の数を求める–JavaSCript ファイル (1)

```
1 window.onload = function(){
2   let form = document.getElementById("form");
3   let table = document.getElementById("result");
```

2 行目と 3 行目で form 要素と結果を表示するための table 要素を得ている。

素数の数を求める–JavaSCript ファイル (2)

4 行目から 39 行目は「開始」ボタンが押されたときの処理

```
 4   document.getElementById("Go").addEventListener("click",function(E){
 5     E.target.setAttribute("disabled", "disabled");
 6     form.removeChild(table);
 7     table = table.cloneNode(false);
 8     form.appendChild(table);
 9     let limit = document.getElementById("limit").value-0;
10     if(limit <= 10**6) limit = 10**6;
11     let no = document.getElementById("No").value-0;
12     if(no <= 0) no = 1;
13     let step = Math.floor(limit/no);
14     let start = new Date();
```

素数の数を求める–JavaSCript ファイル (解説)

- 5 行目ではボタンの操作ができないようにしている。
- 6 行目から 8 行目は結果を表示している内容を消去するために、表示する要素である `table` 要素を取り除き (6 行目)、その要素だけのコピー (子要素はなし) を作成 (7 行目) てそれを再び、子要素として登録 (8 行目) している。
- 9 行目から 10 行目では素数を求める範囲を最低で 10^6 になるようにしている。
- 11 行目から 12 行目では区間を分ける数を設定している。
- 13 行目では個々の処理で求める範囲の幅を求めている。
- 14 行目ではその時点での時間を求めている。

素数の数を求める–JavaSCript ファイル (3)

```
15      let workersNo = 0;
16      let worker = [];
17      for(let i = 0; i<no; i++){
18        worker[i] = new Worker("primes.js");
19        workersNo++;
20      }
```

- 15 行目では発行した Workers の数を管理する変数を初期化
- 17 行目から 20 行目で必要な個数の Web Worker オブジェクトを作成
- Web Worker オブジェクトは `Worker` コンストラクタで作成
- この引数には処理をする JavaScript ファイルを指定

素数の数を求める–JavaSCript ファイル (4)

```
21      for(let i = 0; i<no; i++){
22        worker[i].postMessage({from:1+i*step, step:step, time:start.getTime()});
23        worker[i].onmessage = function(M){
24          let tr = document.createElement("tr");
25          table.appendChild(tr);
26          Object.keys(M.data).forEach((key)=>{
27            let td = document.createElement("td");
28            tr.appendChild(td);
29            td.innerText=M.data[key];
30          });
31          workersNo--;
32          if(workersNo == 0){
33            E.target.removeAttribute("disabled");
34          }
35          worker[i].terminate();
36          delete worker[i];
37        };
38      }
39    });
40  }
```

素数の数を求める–JavaSCript ファイル (解説)

- Worker からのメッセージは onmessage イベントを通じて受け取る
- この引数は message オブジェクト
- data プロパティが送られてきたデータ (26 行目)
- 送られてきたデータのキーを得て (Object.keys)、それぞれを td 要素内に記述 (29 行目)
- データが送られてきたので worker の動作を停止し (35 行目の terminate メソッド)、オブジェクトを消去 (36 行目)。

素数の数を求める–JavaSCript ファイル (Workers)

```
 1 let Status = false;
 2 let limit = 10010;
 3 let primes = [];
 4 self.onmessage = function(M){
 5   if(!Status) {
 6     console.log(‘initialized worker ${M.data.No}‘);
 7     primes =[2,3];
 8     for(let i=5;i<limit;i+=2) {
 9       for(let j=1; j<primes.length; j++){
10         if(i%primes[j] ==0) break;
11         if(i<(primes[j]+2)*primes[j]) {
12           primes.push(i);//console.log(i);
13           break;
14         }
15       }
16     }
17     Status = true;
18   }
```

素数の数を求める–JavaSCript ファイル (解説)

- 1 行目の `self` は `Worker` のもととなるオブジェクトを指す。これに `onmessage` イベントの処理のプログラムを登録
- worker が 2 回目以降呼び出されたかどうかを記憶する変数 (2 行目の変数 `Status`) を利用して 2 回目以降の呼び出し時には素数のリストを再度作成しないようにするためにいくつかの変数をイベント処理関数の外で定義している (1 行目から 3 行目)。
- 最後に `postMessage` メソッドを用いて呼び出し元にデータを送信 (39 行目から 40 行目)。
- 2 行目では保存する素数の上限の値を設定している。変数 `primes` は小さいほうの素数を配列として格納する。この変数は 7 行目で初期化
- 8 行目から 16 行目で残りの素数を求めている。ある数 `i` がその数の平方根以下の素数で割り切れないのならば素数と判定できる (11 行目)。

素数の数を求める–JavaSCript ファイル (Workers)

```
19   let cnt = 0;
20   let from = M.data.from-0;
21   if(from < limit) {
22     cnt = primes.length;
23     from = limit;
24   }
25   from = from | 1;
26   let step = M.data.step;
27   let to = (from < step)?(step+1): (from + step);
28   let pNo = primes.length;
29   for(let i=from; i<to; i+=2) {
30     for(let j=0; j<pNo; j++){
31       let p = primes[j];
32       if(i%p == 0) break;
33       if(i < (p+2)*p) {
34         cnt++;//console.log(i);
35         break;
36       }
37     }
38   }
39   postMessage({from:M.data.from, to:to-1, count:cnt, No:M.data.No,
40                time:new Date().getTime()-M.data.time});
41 //  close();
42 };
```

素数の数を求める–JavaSCript ファイル (解説)

- 19 行目以下が与えられた区間における素数の個数を求める部分である。
- 開始の値が小さい場合には `limit` 以降から探し、素数の個数を `primes` の長さに初期化する (18 行目から 21 行目)。
- 開始の値を奇数に設定し (25 行目)、最後の値を求める (27 行目)
- 29 行目から 37 行目で与えられた区間の奇数が素数であるかの判定を行っている。この部分は小さい素数を求める部分と同じアルゴリズムである。
- 39 行目から 40 行目で結果を送信している。