モダン開発をかじってみる

AL16030 Nobuhiro Kasai

目次

この講座の目的

- モダンなソフトウェア開発におけるキーワードを知る
- その中で自分の中で役立てるものを探す
- モダンって最初に書いてあるが、別に古くからあるものも紹介していく

テスト

テストとは

- ある機能が正しく実装出来てるかどうかを検証するためのもの
- ざっくりわけて単体テストと結合テストがある
- 今回は単体テストの話

いかにしてテストをかくか

- 一般的には**テストフレームワーク**と言われる物を使う
  - テストを簡単に記述することができるもの
  - 統一した方法でテストを記述することが可能
- それぞれの言語において大抵誰かが作ってくれている
  - C++ : CppUnit
  - JavaScript : Mocha
  - Java : JUnit

どんなふうにテストを書くか

- 関数レベルの場合
  - ある関数にある引数を渡した場合、その結果の返り値がどうなるかということは本来わかるはずである
  - それが実際に正しいかどうかをテストする
  - 幾つかのパターンに対してその関数の結果が予期したものになるかどうかを調べて、プログラムが実装出来ているかどうかを考える

例

```
int add(int a,int b){
  return a + b;
}
```

- このadd関数が正しい動作をしているかを調べたい
  - この関数には足し算をしてほしいわけなので、例えばaに2、bに3を代入したときの結果は5となはずである
  - これを確かめたい

こんなふうなtest関数を作る

```
void test(void){
  assert(add(2,3) == 5);
  assert(add(3,4) == 7);
  assert(add(-1,-3) == -4);
}
```

- assertマクロは引数が真であれば何もせず、偽出会った場合は終了する関数
- この関数を実行すると、add関数が自分の予期したとおりに実装出来てるかがわかる

テストはなぜ書くか

- ソフトウェアのバグを減らすため
  - ある関数やクラスなどが想定通りに実行できるかどうかを把握できる
  - 動作が複雑な所に関してテストケースを網羅的に記述することができれば、そのコードの信頼性は高まる

注意

- プログラムの**正しさが保証できるわけではない**
  - 定理証明系などで調べるとよいかも
- ある実装が正常に動作していたとしても、そのソフトウェアが要求に対して正しいとは限らない
- テストが間違っていると、どうしようもない

テストはどこに書くか

- **すべてにテストを書く**ことは諦めたほうがいい
  - 場合によります。個人開発の場合は特に
- 例 : UIテスト
  - UIのテストを書くのは難しい
    - 如何にして操作を関数にするか？
    - どういった場合をテストするのか？

- ゲームを自作している場合
  - UIは割と変更が激しくなりがち
  - 新しくするたびにテストを記述するのは大変
  - 捨ててしまったコードのテストはゴミになる
- 物によっては最低限でいい
  - CPU操作が複雑な時、CPUの判定のテストを記述しておく
  - 配列の扱いが発生するところだけテストを記述しておく

大切なこと

- 関数はテスト可能にしておく
  - 関数への切り分けが適切にされておらず、グローバルリソースへのアクセスが発生する場合はテストが書きづらい
  - **testable** にしておくことが大切

バージョン管理

バージョン管理とは

- コンピューター上で作られるファイルの変更履歴を管理すること
- これをよりソフトウェア開発に特化し使いやすくしたものを「バージョン管理システム」と呼ぶことがある

バージョン管理をどうやるか

- zipで固めて日付事にファイルを保存しておく
  - 古典的な方法
  - つらい
- バージョン管理システムを使う

バージョン管理システム

- プログラムに関するファイルの変更履歴を保存することができるプログラム
- 有名な例 : **git**
- 変更を保存するだけでなく、その変更に対してコメントをつけたりすることができる
- Visual Studioにも元から入ってる（はず）

なぜバージョン管理をするのか

- どういった変更をどのような目的で行ったかを保存しておきたいということがある
  - 「あれ？俺どうしてこの関数変更したんだっけ」
  - 「あれ？俺どうしてこの値にしたんだっけ」
  - 「あれ？俺どうしてこんなコード書いたんだっけ」
  - 「なんで俺はこんなことしてるんだ？」

なぜバージョン管理をするのか

- ある時点での実装に戻すことができる
- ある時点での実装を参照することができる

バージョン管理をはじめるために

- 最初は気張らずとりあえずなんとなく使ってみる
- コミットメッセージ等をこだわるのは後でOK

CI(continuous integration)

CIとは

- continuous integrationの略
- 広義にはソフトウェア開発における品質改善・速度向上のための対策
- 狭義にはソフトウェアのビルド・テストなどを継続的に**自動的に**行うことを指す

例

- あるWebサービスを運用しているとする
  - 新しい機能を実装するたびに
    - サーバー側にファイルを配置し直し
    - ソフトウェアを本番環境でビルドし
    - プロセスを動かす
  - この手の作業を自動化したい

なぜCIが必要なのか

- 基本的にプログラマはコードを書く時間に集中するべき
- デプロイ作業、ビルド作業は大切ではあるが自動的に行えること
- これを自動化していくことで**簡単**かつ**ミスを減らす**ことができる

CIのためのツール

- Travis Ci
  - Gitのpushをホックにしてビルド・テストの実行等を行うことができる
  - 実はデジクリのWebサイトでも使ってたりする
- Jenkins
  - 自動テスト環境を構築してテストを行ったりできる

CIをどう使うか

- 結局CIの本質は「作業の自動化」(笠井の考え)
- 最初からゲーム開発にCIツールを使う必要はない
- けど、自動化できる部分を自動化するという意識は忘れないようにすること

Docker(インフラ)

Dockerとは

- シンプルな仮想環境
  - 仮想環境とはコンピュータの動作を再現するソフトのこと
  - エミュレータを想像してくれればいい
- UbuntuやWindowsなどのOSを再現してその中でプログラムを実行することができる

Dockerの使い方

- DockerFileというファイルを作って、その中にどういった環境を作るかを記述する

```
FROM ubuntu # Ubuntuのイメージを持ってくる
# Ubuntu内で環境を構築する
RUN apt-get install -y software-properties-common python
RUN add-apt-repository ppa:chris-lea/node.js
RUN echo "deb http://us.archive.ubuntu.com/ubuntu/ precise
RUN apt-get update
RUN apt-get install -y nodejs
RUN mkdir /var/www
# ファイルを追加
ADD app.js /var/www/app.js
# 実行
CMD ["/usr/bin/node", "/var/www/app.js"]
```

Dockerをなぜ使うのか

- ハードウェアを用意するのがそもそも面倒
  - たとえばUbuntuの環境を用意したい場合、新しくPCを購入しその中にUbuntuをインストールしないといけない
  - ハードウェアは壊れる
- ソフトウェアをデプロイするのは大変
  - サーバー側に接続し、ファイルを置き、実行し……
  - 大変

Dockerを何故使うのか

- Dockerを使えば仮想環境を **使い捨てる**ことが可能
- ハードウェアを用意する必要がない
- これをクラウド上で実行することができればもっと簡単になる

みんながDockerを使うべきか？

- 正直言うと**おすすめできない**
  - 結局普段Ubuntuで実際に行っていたものを「**仮想的に行っている**」
  - 要はどのようにインフラを構築するかを理解していないと使えない
- 最初はUbuntuなどをインストールしてそこでなれる方がよいかもしれない

クラウド

クラウドとは

- ネットワーク上にあるコンピューターにアクセスし利用する形態
- ネットワークを介してコンピューターに接続し、操作する
- 多くの企業がサービスを展開している
  - Amazon : Amazon Web Serice
  - Google : Google Cloud Platform
  - Microsoft: Azure

なんでクラウドを使うのか

- 自社サーバーを持ちたくない！！
  - 場所は取るし金もかかる
  - メンテナンスする人も必要になる
  - 壊れたら死ぬ
- インフラにリソースを割けない
  - 少人数で開発をすすめる場合、コードを書く人しかいない事が多い
  - 管理の比較的ラクなクラウドに頼りたい

クラウドの悪いところ

- 技術がロックインされがち
  - あるクラウドで動くプログラムが他の場所で動かないことは多い
    - あるクラウドサービスを利用する場合に、そのサービス独自のシステムを利用することになる(ことが多い)
  - トレードオフ

まとめ

まとめ

- 様々なはやりすたりがある
  - どうして流行ってるのかには理由がある
  - どうしてその技術を使うのか？ということはしっかりと考える必要がある
- 新しい技術・新しいワードに騙されないこと
  - 実は既存技術の言い換えであったりする
  - ビジネスがどうしても関わってくるので

まとめ

- 一方で新しいものに対してある程度アンテナを張っておくことは重要
  - 自分が毎度30分かけているものが1分で終わるようになるかもしれない
  - 30行のコードがライブラリを使って1行で書けるようになったりするのと同じ
  - もちろん一長一短ではあるが

まとめ

- **自分が作りたいものに集中するべき**
  - そのためにはいかに「楽するか」が大事
  - いかに「同じ作業をなくすか」も大事
  - 楽をできる部分で楽をしよう