По чему изучать: Udemy, Rebrain – 2 levels, Slurm – 1 kurs?!

За основу взять Rebrain – следовать оглавлению

Сюда вставить осн треб для джунов и потом добавлять в этот файл и в оглавление

Сделать оглавление здесь в начале файла

Пункты добавлять с сылками на плейграунд скрины из VS code + текст+ ссылки на сторонние ресурсы

**#Подготовка окружения. Основы языка**

Переменные, типы данных

Указатели в Go

Слайсы

Map’ы в Go

Конструкции языка и функции

defer - обработка выхода из функции

Panic и их обработка

Обработка ошибок

Продвинутая работа с ошибками

**#Модули и пакеты**

Области видимости, инициализация через init()

Работа с зависимостями, go mod

Создание модулей и их версионирование

layout проекта (структура проекта)

**#Структуры и интерфейсы**

Структуры в Go

Методы структур

Интерфейсы и утиная типизация

Пустой интерфейс

Композитное наследование

**#Асинхронность**

Goroutines

Go sheduler

Race condition

Пакеты sync и atomic

Пакет errgroup

sync.Pool

Каналы ч.1. Deadlocks

Каналы ч.2. Context

**#Тестирование, бенчмарки и профилирование**

Unit-тестирование в Go

Моки, стабы и генерация через GoMock

Table driven test vs closure driven tests

Test coverage

Benchmarks

Профилирование с pprof

**#Кодогерация**

Рефлексия

AST

Templates

Решение проблем рефлексии

Враппинг

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ADVANCED\*\*\*\*\*\*\*\*\*\*\*\***

**#Работа с БД**

Конфигурирование подключения к БД (postgres)

Работа с БД (postgres)

Миграции (goose)

Работа с базой на примере использования GORM

Работа с NoSql (Mongo)

**9 #Сервер на Go (обработка запросов, context, middleware)**

+++Поднимаем сервер, роутинг, первый handler

+++Работа с параметрами

+++Роутер Gorilla

+++Роутер Chi

+++Middleware

+++Контекст запроса

fasthttp

WebSockets

**10 #Низкоуровневость + продвинутая сборка**

Продвинутая работа с модулями

Сборка с использованием Idflags

Сборка для разных ОС

Пакет unsafe

Cgo

**11 #Микросервисная архитектура**

+++Заворачиваем наш сервис в образ

+++Продвинутая сборку образа

Подтягиваем конфигурации из key-value store (consul)

**12#Межсервисное взаимодействие**

Синхронное взаимодействие через REST

gRPC взаимодействие

Interceptors

Асинхронное взаимодействие (kafka) и pub-sub

Работа с GraphQL в Go

**#Оптимизация**

in-memory хранение

Redis

Самописный LRU cache

**#Работа сервиса в кластере**

Формат логов и уровни логирования

Пишем логи (Greylog)

Сквозное логирование

Метрики (Prometheus + Grafana)

graceful обработка сигналов

healthcheck

**1 Basics**

**1.1 Обьявление переменных**

Var myVar int (обьявление имяПеременной тип )

**1.2 Функции в GO**

func funcName (arg1 string, arg2 int, [argN type])(res1 int, res 2 error, [recN type]){

//operators

}

**1.3 Обьявление фцнкций:**

<https://play.golang.com/p/aq8MUZv8uys>

Анонимные функции можно обьявлять внутри других функций и сохрфнить результат в переменной

s:=strings Map(func(r rune)rune{return r+1}, “SDWS”)

fmt.Println(s)

**1.4 Функции как тип данных** – функции можно обьявлять как собственный тип

var f, f2 func(s string) int

f=func(s string)int{return len(s)}

f2=func()int{return 1} // оибка компиляции

**1.5 Замыкание** – функции которые в своем теле используют переменные обьявленные вне этой функции

Пример:

https://play.golang.com/p/FYH-U6h-A9y

**1.6 Отложенный вызов функций (defer)**

- Вызываются в обратом порядке

- аргументы определяются на момент обьявления

- выполнеине defer гарантируется

Пример:

https://play.golang.com/p/Yyn94frU-oZ

**1.7 Циклы и Условные операторы**

Виды цыклов:

-бесконечный

- цикл с параметро

- цикл с условием

- range(slice, map, chan) итерация по строке с range происходи не побайтово а посимвольно

Пример:

<https://play.golang.com/p/ulV_zNo93zg>

**1.8 Модули и пакеты**

Пакеты в гоу бывают двух видов -executabl and reusable

Директива package для обьявления пакета import для подключения

Пакеты могут быть импортированы из разных источников – стандартные пкеты библиотеки Гоу, пакеты из удаленных репозиториев, и локальные пакеты.

Бывают экспортируемые(Exported) элементы пакета. Которые начинаются с заглавной буквы и неэкспортируемые(unexported) – начинаются с малеькой

Функция init()выполняется при инициализации пакета в том числе при импорте.

Директива импорт имеет четыре вида – обычный, через псевдоним, через нижнее подчеркивание и через точку

Пример:

Import(

“fmt”

“githyb.com/faith/color”

. “gopackages/wordz”

newcolor ”gopackages/color” - псевдоним

)

1. **Adv Basic**
   1. **Struct**

Пример: https://play.golang.com/p/HyefNfRuIon

**2.2 Metods**

Помимо полей для структуры мы можем определить и методы с помошщью конструкции:

func (p Point) myMethod(param int) string{…}

(p Point) – привязка типа с указанием роли получателя

Тоесть обьявляем функцию для структуры Point

Пример: <https://play.golang.com/p/KSguuM3vxeT>

**2.3 Область видимости**

Если поля или структуры и переменные обьявлены с маленькой буквы то они не видны запределами пакета (с большой = как публичные)

* 1. **Встравивание структур**

-Структура может содержать в себе другую структуру как поле

-Структура может быть встроенна в другую структуру тоесть создаваьб композтицию структур. Они не являются полем а дополняют структуру(расширяют)

Перекрытие возникет если а расширяющей(дополняющей) стрктуре есть поле с таким же именем. В приоритете всегда имя то что выше.

* 1. **Интерфейсы - абстракция поведения другх типов**

type Writer interface{

Write(p[]byte)(n int, error) //обьявляем сигнатуру методов интерфейсов

}

Пример: <https://play.golang.com/p/23HxaybWUXI>

* 1. **Утверждение Типов (type assertion)-**

email, ok:=s.(Email) здест происходит инициализация двух переменных

значения переменных происходят отхначения стоящего справа

(Email) – утверждение типа.

Пример: <https://play.golang.com/p/btWYZhN0Uw0>

* 1. **Встраивание Интерфейсов(embading interfaces)**

Интерфейсы можно встраивать друг в друга, чтобы удовлетворять интерфейсу со встроенными в него другими интерфейсами надо реалзовывать методы всех встроенных интерфейсов и реализовать собственные методы интерфеса.

пример: https://play.golang.com/p/tApi1P0BAXi

* 1. **Пустой Интерфейс**

Пустым называют – тот у которого нет никакихтребований к реализации. Тоесть нет методов. Так как он пустой то почти ве переменные могут подходить под его требования

I:=interface{}

Пример: <https://play.golang.com/p/XxcRrHmochI>

* 1. **ООП**

Основные принцыпы: Абстракция инкапсюляция наследование полиморфизм.

Абстракция – выделение значимой инфрормации , исключение из рассмотрения незначимой, набор характеристик создаваемой нами сущности (структуры)

Инкапсуляция – объединяем данные и методы в одну сущность, скрываем и разграничиваем доступы к разным частям программы

Наследование – с помощью наследование наш абстрактный тип данных может принимать(наследовать)данные и функциональность другого типа.(интерфейсы)

Полиморфизм – возможноть обьектов с олинаковой спецификацией иметь различную реализацию. Один интерфейс множество реализаций.

**3 Concurrency**

**3.1 GOROUTINES** – операция которая может выполняться независимо то функции в которой была запущена. Выполнение горутин может выполняться паралельно

go Myfunc()

пример: <https://play.golang.com/p/YXWsG5FAkwD>

**3.2 ERRORS** – ошибки в го это интерфейс error

Пример 1 errors : <https://play.golang.com/p/R3Mn_djVUBt>

Пример 2 errors :

**3.3 PANICS -**

Пример:

* 1. **Errors Loging and Printing**

1. **Потоки и файлы**
2. **Сетевое програмирование**

**5.1 Docker**

Простое приложение Go на Docker пример:

<https://play.golang.com/p/v_f_cedvTke>

**5.2 сервер на Go + кастомные хендлеры**

Пример: <https://play.golang.com/p/LJy1scUWswc>

**5.3 Работа с параметрами запросов(через url после?, FORMDATA, Body, basic author)**

[**https://play.golang.com/p/o3-0BV1uq55**](https://play.golang.com/p/o3-0BV1uq55)

* 1. **Gorilla/mux router**
* Посмтореть и сделаттью и там же можно подкл.чить бд вроже как к айпиай:  
  [REST-API with Golang and Mux.. In this article we are building a… | by Hugo Johnsson | Medium](https://hugo-johnsson.medium.com/rest-api-with-golang-and-mux-e934f581b8b5)
  1. **Gorilla/mux router**
* Особенность что он очень легковесный и он без внешних зависимомтей

Он совместим http пакетом скоростной. Модульный – фуекционал наращивается модулями.

* 1. **Middleware**   
     \* Создавать такие обработчики в go достаточно просто, главное придерживаться определенного интерфейса.  
      type MiddlewareHandler = func(next http.Handler) http.Handler

Это функция в которую мы передаем что то что реализует интерфейс http.Handler и отдаем на возврат что то что реализует интерфейс http.Handler

(bynthatqc http.Handler – это интерфейс с одной функцией ServeHTTP)

// <https://golangify.com/http-handler-interface> : про handle interace, http.handerFunc, Handlefunc(сахар)

|  |  |
| --- | --- |
|  | type Handler interface {      ServeHTTP(ResponseWriter, \*Request)  } |

Если объяснить по-простому, это значит, что для того чтобы объект считался обработчиком, у него должен быть метод ServeHTTP() со следующей сигнатурой:

ServeHTTP(http.ResponseWriter, \*http.Request)

* 1. **Context**

Контекст - это информация, которая описывает запрос или выполнение какой-нибудь команды.

- Контекст это обьект который имеет возможность отмеить из вне потенциально долгой операции

- с помощью контекста можно хранить и передаватьь информацию между фуекциями и методами в программе

Отменять долгие операции с помощью контекста можно

1. По явному сигналу отмены context.WithCancel()
2. По истечению промежутка времени context.WithTimeout()
3. По наступлении временной отметки context.WithDeadLIne()
4. type Context interface {
5. Deadline() (deadline time.Time, ok bool)
6. //возвращает веремя когда контекст должен быть отменен
7. // ok bool – индикатора задан ли дед лайн
8. Done() <-chan struct{} // канал done
9. //канал закрывается когда работы выполненная для контекста должна быть отменена. Тоесть если работа отменена то канал закрывается
10. Err() error
11. // если канал done не закрыт то err==nil
12. //если закрыт то err вернет ошибку обьясняющуб почему контекст был закрыт
13. Value(key any) any
14. // принимает интерфейси возвращает интерфейс нужно для того чтобы какие то данные класть в контекст

Методы:

Context.Background()

Context.ToDo()

Context.WithCancel(ctx) - требует родительского контекста. Возвращает копию роительского контекста в переданный контекст будет передан новый канал done,

Этот канал будет закрыт когда вызовется функция отмены cancel() или когда у родителльского контекста канал done будет закрыт. Если у родителького контекста закроется канад done то и этот закроется тоже. Тоесть ои связанны

Пример: context.WithCancel <https://play.golang.com/p/HnT7imCBM_q>

Пример: context.Withdeadline and with value <https://play.golang.com/p/HHwzkBOSZ-p>

**6. Продвинутая работа с модулями**

#собираем два контейнера - название техники - multistage

#первый контейнер -builder

FROM golang:latest as builder

# builder - даем ему имя

# на какой опер системе собрать.

WORKDIR /app

# если нет такой папки то она создаст папаку и перейдет в нее

COPY . .

# копируем все файлы на хост машине на докер машину в папаку опис выше

RUN CGO\_ENABLED=0 GOOS=linux GOARCH=amd64 go build -o ./rebrainapp ./main.go

# !!!what cases to goos t be changed to windows???

# компилируем приложение

# отключаем с

# первый контейнер служит сборщиком

#контейнер scratch - маленький контенер (файловая система с возм запуска бинарников)

# стартуем контейнер - он и будет результатом выполнения

FROM scratch

COPY --from=builder /app/rebrainapp /usr/bin/rebrainapp

#копируем из билдера

ENTRYPOINT ["/usr/bin/rebrainapp"]

# то что будет запущенно когда напишем докер ран

Makefile  
REPOSITORY := rebrain

APP\_NAME:=module11-task02

VERSION:=0.1

build:

    @docker build -t $(REPOSITORY)/$(APP\_NAME):$(VERSION) .

run:

    @docker run --name $(APP\_NAME) --rm -d -p 8080:8080 $(REPOSITORY)/$(APP\_NAME):$(VERSION)

images:

    @docker images --format "{{.Repository}} {{.Size}}"

check:

    @curl localhost:8080/

stop:

    @docker container stop $(APP\_NAME)

Теперь представим, что наше приложение завершило свою работу критическим образом, то есть просто упало. Контейнер сам по себе не поднимется, нам придется перезапустить его руками. Но можно указать специальный флаг для работы данного контейнера - always. docker run -p 7777:7777 --restart=always --name rebraincontainer rebrainapp

Теперь при возникновении паники контейнер docker перезапустит наш контейнер. Также у флага restart есть и другие значения: ● no - не перезапускать контейнер (устанавливается по умолчанию); ● unless-stopped - перезапускает контейнер всегда, за исключением явной остановки самим пользователем; ● on-failure[:2] - перезапускать в случае сбоя, в скобках указывается значение максимального количества перезапусков.

**Docker-compose:** Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения. Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями. Docker-compose позволяет поднять ваш сервис и рядом все сервисы, от которых он зависит, например, базу данных или брокер сообщений. Если бы нам требовалось поднять несколько сервисов с помощью обычных команд докер, мы бы повторяли несколько действия подряд: создать образ, поднять контейнер. Docker-compose позволят оптимизировать эти шаги