Web Architecture With Golang
Google's Go Language

Web Architecture, Git, Curl, Interfaces, Context, Error Handling, & Escape Analysis

Todd McLeod & Daniel Hoffmann

Table of Contents

Getting started **3**
Welcome 3
Course outline 3
Git and versions **4**
Introduction 4
Github repo 4
.gitignore 4
Versioning codebase 4
Access versions 4
Code and curl **5**
Introduction 5
Separation of concerns 5
Reviewing example code 5
Post with curl 5
Get with curl 6
Hands-on Exercises - ninja level 1 **6**
Hands-on exercise #1 6
Hands-on exercise #2 6
Hands-on exercise #3 7
Software architecture **7**
Introduction 7
Exploring architecture 7
Overview of buzzwords 8

Monolith vs microservices 8
Serverless 9
Virtualization 10
Containers 10
Container orchestrators 11
Containers and orchestration illustrated 11
Continuous integration, delivery, deployment 12
REST 12
SPA 13
MVC 13

Interfaces 14
Introduction 14
One value, many types 14
Concrete & abstract types 15
Using an interface 15
Using an interface II 15
Decoupling data storage 15
Coding the decoupling 16
Improving the decoupling 16
Standard library interfaces 16

Housekeeping 17
Code organization conventions 17
Adding documentation 17
Testing 17
Example tests 17
No return 17
The testify mock package 17
Introduction to gomock 17
Generating gomock code 18
Testing with gomock 18
Method sets explained 19
Method sets example 19
Empty interface 20

Hands-on Exercises - ninja level 2 20
Hands-on exercise #1 20
Hands-on exercise #2 20
Hands-on exercise #3 20
Hands-on exercise #4 21

Hands-on exercise #5 21

Context **21**

Where we are in the course 21

Understanding context 21

Reading documentation 22

With timeout 22

With value 22

Abstracting with value 22

Done 23

Hands-on exercises - ninja level 3 **23**

Hands-on exercise #1 23

Hands-on exercise #2 - attempt 1 23

Hands-on exercise #2 - attempt 2 23

Hands-on exercise #3 24

Hands-on exercise #4 - attempt 1 24

Hands-on exercise #4 - attempt 2 24

Hands-on exercise #5 24

Dealing with errors **26**

Introduction 26

Go and errors 26

Dealing with errors 26

Creating an error using the standard library 27

Creating a custom error 27

Wrapping errors 27

Accessing wrapped errors 28

Errors in-depth **28**

Introduction 28

Provide meaning 28

Why compare errors 29

Why assert errors 29

Switch case 29

Why unwrap 29

Using unwrap 30

Errors in practice **30**

Checking errors after processing 30

Best practices 30

Understanding best practices 30

Abstracting into a package 31

Hands-on exercises - ninja level 04 **31**

Hands-on exercise #1 31

Hands-on exercise #2 31

Hands-on exercise #3 32

Hands-on exercise #4 32

Hands-on exercise #5 32

Hands-on exercise #6 32

Nice to know **33**

Overview 33

String method 33

Equality on structs 33

Stack, heap, escape analysis intro 33

Exploring stack, heap, escape analysis 33

Coding stack, heap, escape analysis 33

Review **34**

Todd 34

Daniel 34

Farewell **34**

Congratulations! 34

Bonus lecture 35

Getting started

Welcome

Welcome to the course. You are taking a great step by enrolling in this course. Better skills create a better life. ***You are on your way to a better life.*** As you learn new skills, you are building a better life. ***I commend you for your efforts to improve your life.*** As you improve your life, you are improving the world --- one person at a time. You are making the world better, and you are making your life better. This is a win-win for everybody. Great work! Also, ***this is your course. Use it in the way which is best for you.*** If you want to skip ahead, skip ahead. As your teacher, my job is to help you succeed. The content here is designed to help you succeed both with visual studio code, and also as a student and in life.
Video file: 01 Welcome

Course outline

You can find everything I use in the course and all of the courses resources here:

- github - https://github.com/GoesToEleven/golang-architecture

Video file: 03 Course Resources

Git and versions

Introduction

Introducing Daniel Hoffmann
video: 11 Intro

Github repo

- setting up
- cloning
- ssh
- adding a collaborator
  - allows Daniel to push

video: 12 Git Repo

.gitignore

- adding a gitignore file

video: 12-b gitignore

Versioning codebase

- https://semver.org/
- **git tag v0.0.1**
- **git push --tags**

tag v0.0.1
video 13 Versioning

Access versions

- **git tag v0.0.2**
- **git push --tags**
- **git log**

Click on RELEASES in github. Then you can

- click on the commit number and click BROWSE FILES

You can pull down different versions of the code with

- **git checkout v0.0.1**

remember to checkout master when you're done

- ***git checkout master***

tag v0.0.2
video: 14 Access Versions

Code and curl

Introduction

- This is the code we're starting with
    - monolith → microservices
- focusing on architecture first
    - then applying it to some application
- **goal is to organize code**
    - **efficient execution**
    - **ease of program**

Example Repo - https://github.com/oralordos/separation
video: 15 Intro

Separation of concerns

- https://en.wikipedia.org/wiki/Multitier_architecture

video 16 Separation of Concerns

Reviewing example code

- **to explore in future videos ...**
    - interfaces
    - json encode / decode
    - context

video 17 Review Code Example

Post with curl

Here are some curl commands

- curl google.com
- curl www.google.com/teapot
- curl -v www.google.com/teapot
    - > what you're sending to website
    - < what the website is sending back to you
- curl localhost:8080/
    - 404
- curl localhost:8080/register
    - post request required
- curl -XPOST localhost:8080/register

- curl -v -XPOST localhost:8080/register
- curl-v -XPOST -d “{}” localhost:8080/register
    - email cannot be empty
- curl-v -XPOST -d ‘{“email”:””}’ localhost:8080/register
    - email cannot be empty
- curl-v -XPOST -d ‘{“email”:”test”}’ localhost:8080/register
    - must include “@” symbol
- curl-v -XPOST -d ‘{“email”:”test@test.com”}’ localhost:8080/register
    - name cannot be empty
- curl-v -XPOST -d ‘{“email”:”test@test.com”, “name”:”Bob”}’ localhost:8080/register
    - success

video 18 Post With Curl

Get with curl

- curl -v -XGET localhost:8080/user
    - needs email
- curl -v -XGET localhost:8080/user?email=blah
    - email needs “@” symbol
- curl -v -XGET localhost:8080/user?email=test@test.com
    - 200 OK

Sites to help you learn curl

- curl builder

video 19 Get With Curl

Hands-on Exercises - ninja level 1

Hands-on exercise #1

Git versioning with tags
tag 1.5.0
video: 20 Hands On 1

Hands-on exercise #2

Accessing a version of a git repo

- do it on github
- do it on the terminal

video: 21 Hands On 2

Hands-on exercise #3

Use curl to view the headers from https://www.google.com/teapot
video: 22 Hands On 3

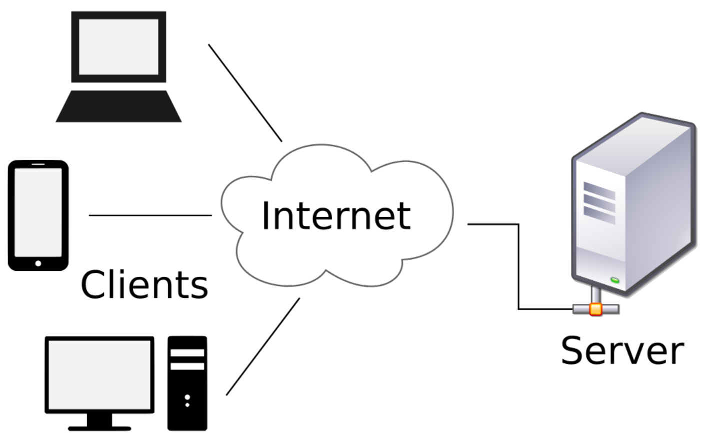
Software architecture

Introduction

video: 23 intro

Exploring architecture

- many possibilities
    - client / server



server	**database**
1	1
many	1
many	many

- layers that can be included
    - load balancer
    - in-memory database / redis
    - interface layers
    - & much more
    - images

video: 24 Exploring Architecture

Overview of buzzwords

- monolith
- microservices
- REST
- SPA
  - angular
  - react
  - vue
- Serverless
  - IAAS - hardware
  - PAAS - hardware + software
  - SAAS - hardware + software + software
- Other
  - Layers
  - Multi-tier Architecture / three-tier architecture / hierarchical
  - Presentation Abstraction Control (PAC)
  - Model View Controller (MVC)
  - Model View Presenter (MVP)
  - Model View ViewModel (MVVM)
  - Pipeline
  - Implicit Invocation
  - Blackboard System
  - Peer-to-Peer
    - sometimes client & sometimes server
  - Service Oriented Architecture (SOA)
  - Naked Objects
- source: stackoverflow & Wikipedia & Anthony Ferrara & hackr.io

- **goal is to organize code**
  - **efficient execution**
  - **ease of program**

video: 25 Overview of Buzzwords

Monolith vs microservices

Monolith advantages

- Simpler to **develop**
- Simpler to **test**
- Simpler to **deploy**
- Simpler to **launch**

Monolith disadvantages

- can lead to **code complexity** if poorly designed

- **language restrictions**

Microservice advantages

- **modularized** code
- **separation of concerns**
- scales to **large dev teams**
- ease of **refactoring**
- asymmetric **scaling**
- **language flexibility**

Microservice disadvantages

- **secure communication** between services

source1 & source 2 & images

video 26 Mono vs Micro

Serverless

“Serverless computing is a cloud-computing execution model in which **the cloud provider runs the server, and dynamically manages the allocation of machine resources.** Pricing is based on the actual amount of resources consumed by an application, rather than on pre-purchased units of capacity. It can be a form of utility computing. **Serverless computing can simplify the process of deploying code into production. Scaling [elasticity], capacity planning and maintenance operations may be hidden from the developer or operator.** Serverless code can be used in conjunction with code deployed in traditional styles, such as microservices. Alternatively, applications can be written to be purely serverless and use no provisioned servers at all.” wikipedia

- IAAS
  - infrastructure as a service
  - “Infrastructure as a service (IaaS) are online services that provide high-level APIs used **to dereference various low-level details** of underlying network infrastructure like **physical computing resources, location, data partitioning, scaling, security, backup etc.** … large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements.” wikipedia
- **PAAS**
  - **serverless is PAAS**
  - “Platform as a Service (PaaS) or Application Platform as a Service (aPaaS) or platform-based service is a category of cloud computing services that provides a **platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an app.**” wikipedia
- SAAS
  - examples: gsuite (gmail, calendar); salesforce; activecampaign; …
  - “Software as a service (SaaS) is a software licensing and delivery model in which **software is licensed on a subscription basis and is centrally hosted.** …

> SaaS is typically accessed by users using a thin client, e.g. via a web browser. SaaS has become a common delivery model for many business applications, including office software, messaging software, payroll processing software, DBMS software, management software, CAD software, development software, gamification, virtualization, accounting, collaboration, customer relationship management (CRM), Management Information Systems (MIS), enterprise resource planning (ERP), invoicing, human resource management (HRM), talent acquisition, learning management systems, content management (CM), Geographic Information Systems (GIS), and service desk management. **SaaS has been incorporated into the strategy of nearly all leading enterprise software companies.** According to a Gartner estimate, SaaS sales in 2018 were expected to grow 23% to $72 billion. SaaS applications are also known as Web-based software, on-demand software and hosted software." wikipedia

video 27 Serverless

Virtualization

- **simulated hardware**

"In computing, virtualization refers to the act of **creating a virtual (rather than actual) version of something, including virtual computer hardware platforms,** storage devices, and computer network resources. Virtualization began in the 1960s, as a method of **logically dividing the system resources** provided by mainframe computers between different applications. Since then, the meaning of the term has broadened." wikipedia
video 28 Virtualization

Containers

- **simulated operating systems**

"OS-level virtualization refers to an **operating system paradigm** in which the kernel allows the existence of **multiple isolated user-space instances.** Such instances, called containers (Solaris, **Docker)**, Zones (Solaris), virtual private servers (OpenVZ), partitions, virtual environments (VEs), virtual kernel (DragonFly BSD) or jails (FreeBSD jail or chroot jail), may **look like real computers from the point of view of programs running in them.**

- ***A computer program running on an ordinary operating system can see all resources (connected devices, files and folders, network shares, CPU power, quantifiable hardware capabilities) of that computer.***
- ***However, programs running inside of a container can only see the container's contents and devices assigned to the container.***

… The term "container," while most popularly referring to OS-level virtualization systems, is sometimes ambiguously used to refer to fuller virtual machine environments operating in varying degrees of concert with the host OS, e.g. Microsoft's "Hyper-V Containers."" wikipedia video 29 Containers

Container orchestrators

- **orchestration**

“Orchestration is the **automated configuration, coordination, and management of computer systems and software.** A number of tools exist for automation of server configuration and management, including Ansible, Puppet, Salt, Terraform, and AWS CloudFormation.” wikipedia

- **container orchestration**

“For **Container Orchestration** there are different solutions such as **Kubernetes** software or managed services such as AWS EKS, AWS ECS or Amazon Fargate.” wikipedia

video 30 Container Orchestrators

Containers and orchestration illustrated

video 31 Containers and Orchestration Illustrated

Continuous integration, delivery, deployment

- **continuous integration**
  - “In software engineering, continuous integration (CI) is the practice of **merging all developers' working copies to a shared mainline several times a day.** Grady Booch first proposed the term CI in his 1991 method, although he did not advocate integrating several times a day. Extreme programming (XP) adopted the concept of CI and did advocate integrating more than once per day – perhaps as many as tens of times per day.” wikipedia
- **continuous delivery**
  - “Continuous delivery (CD or CDE) is a software engineering approach in which **teams produce software in short cycles, ensuring that the software can be reliably released at any time and, when releasing the software, doing so MANUALLY**. It aims at **building, testing, and releasing software with greater speed and frequency.** The approach helps reduce the cost, time, and risk of delivering changes by **allowing for more incremental updates to applications in production.** A straightforward and repeatable deployment process is important for continuous delivery. wikipedia
- **continuous deployment**
  - **continuous deployment** ... a similar approach in which software is also produced in short cycles but through **AUTOMATED** deployments rather than manual ones. wikipedia

video 32 Continuous Integration, Delivery, Deployment

REST

- **state**

“In information technology and computer science, a program is described as **stateful** if it is **designed to remember preceding events or user interactions; *the remembered information is called the state of the system***.” wikipedia

- **Representational state transfer (REST)**

“Representational state transfer (REST) is a software architectural style that defines a set of **constraints to be used for creating Web services.**

- ***Web services that conform to the REST architectural style, called RESTful Web services (RWS), provide interoperability between computer systems on the Internet.***
- RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a ***uniform and predefined set of stateless operations.***

Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations. "Web resources" were first defined on the World Wide Web as documents or files identified by their URLs. However, today they have a much more generic and abstract definition

that encompasses every thing or entity that can be identified, named, addressed, or handled, in any way whatsoever, on the Web. In a RESTful Web service, requests made to a resource's URI will elicit a response with a payload formatted in HTML, XML, JSON, or some other format. The response can confirm that some alteration has been made to the stored resource, and the response can provide hypertext links to other related resources or collections of resources. When HTTP is used, as is most common, the operations (HTTP methods) available are GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE. ***By using a stateless protocol and standard operations,*** RESTful systems aim for fast performance, reliability, and the ability to grow by reusing components that can be managed and updated without affecting the system as a whole, even while it is running. The term representational state transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Fielding's dissertation explained the REST principles that were known as the "HTTP object model" beginning in 1994, and were used in designing the HTTP 1.1 and Uniform Resource Identifiers (URI) standards. ***The term is intended to evoke an image of how a well-designed Web application behaves: it is a network of Web resources (a virtual state-machine) where the user progresses through the application by selecting resource IDENTIFIERS such as http://www.example.com/articles/21 and resource OPERATIONS such as GET or POST (application state transitions), resulting in the next resource's representation (the next application state) being transferred to the end user for their use.***“ wikipedia
video 33 REST

SPA

“**A single-page application (SPA)** is a web application or web site that interacts with the user by ***dynamically rewriting the current page rather than loading entire new pages from a server.*** This approach ***avoids interruption of the user experience between successive pages, making the application behave more like a desktop application.*** In a SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the ***appropriate resources are dynamically loaded and added to the page as necessary, usually in response to user actions.*** The page does not reload at any point in the process, nor does control transfer to another page, although the location hash or the HTML5 History API can be used to provide the perception and navigability of separate logical pages in the application. Interaction with the single page application often involves dynamic communication with the web server behind the scenes.” wikipedia
video 34

MVC

“Model–View–Controller (usually known as MVC) is a **software architectural pattern** commonly used **for developing user interfaces** which divides the related program logic into three interconnected elements. This is done to **separate internal representations of information** from the ways information is presented to and accepted from the user. Following the MVC architectural pattern **decouples these major components allowing for code reuse** and parallel development. **Traditionally used for desktop graphical user interfaces (GUIs),** this pattern has become popular for designing web applications. Popular programming languages like JavaScript, Python, Ruby, PHP, Java, and C# have MVC frameworks that are used in web application development straight out of the box." wikipedia

- **Model**
  - The central component of the pattern. It is the application's dynamic **data** structure, independent of the user interface. It directly manages the data, logic and rules of the application. ~wikipedia
- **View**
  - Any **VISUAL representation** of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. ~wikipedia
- **Controller**
  - **The controller responds to the user input and performs interactions on the data model objects**. The controller receives the input, optionally validates it and then passes the input to the model. As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system. Particular MVC designs can vary significantly from the traditional description here. ~wikipedia

video 35 MVC

Interfaces

Introduction

Interfaces allow for ***substitutability*** and flexibility. "Interface types express generalizations or abstractions about the **behaviors** of other types. By generalizing, **interfaces let us write functions that are more flexible** and adaptable because they are not tied to one particular implementation." (Donovan & Kernighan)
video 36 Intro

One value, many types

In Go, values can be of more than one type. An interface allows a value to be of more than one type. We create an interface using this syntax: "keyword identifier type" so for an interface it would be: "type human interface" We then define which method(s) a type must have to implement that interface. If a TYPE has the required methods, which could be none (the empty interface denoted by interface{}), then that TYPE *implicitly implements* the interface and is ***also*** of that interface type. **In Go, values can be of more than one type.**
tag v0.0.3
video: 37 One Value, Many Types

Concrete & abstract types

“A **concrete type** specifies the exact representation of its values and exposes the intrinsic operations of that representation, such as arithmetic for numbers ... or indexing, append, and range for slices. A **concrete type** may also provide additional behaviors through its methods. When you have a value of a **concrete type**, you know exactly ***what it is*** and ***what you can do with it.***

There is another kind of type in Go called an **interface type**. An interface is an **abstract type**. It doesn’t expose the representation or internal structure of its values, or the set of basic operations they support; **it reveals only some of their methods.** When you have a value of an interface type, **you know nothing about *what it is,* you know only *what it can do,*** or more precisely, what behaviors are provided by its methods.” (Donovan & Kernighan)
https://play.golang.org/p/rZH2Efbpot
video 38 Concrete & Abstract

Using an interface

tag v0.1.0
video 39 Using Interface

Using an interface II

tag v0.2.0
video 40 Using Interface 2

Decoupling data storage

The plan, stepped out:

- type person
- two “fake” data storage systems
  - map: mongo
    - map[int]person
  - map: postgres
    - map[int]person
- give methods to data storage types
  - save
  - retrieve
- “accessor” interface for the data storage system types with these methods
  - save
  - retrieve
- use the above - demonstrate
  - func put(a accessor)

        - func get(a accessor)

video 41 Decoupling Data Storage

Coding the decoupling

Writing an interface for saving data
tag v1.0.0
video 42 Coding The Decoupling

Improving the decoupling

Here is Daniel's improvement on the previous code:

- type person
- two "fake" data storage systems
    - map: mongo
        - map[int]person
    - map: postgres
        - map[int]person
- give methods to data storage types
    - save
    - retrieve
- "accessor" interface for the data storage system types with these methods
    - save
    - retrieve
- create a type "personService" struct
    - include a field of type "accessor"
- add methods to type "personService"
    - put
    - get
- use type personService
    - create a value of one of the database types
    - create a value of type personService
    - for the field of type "accessor" associate the database
    - call the methods attached to type "personService"

tag v1.1.0
video 43 Improving The Decoupling

Standard library interfaces

s
video 44 Standard Library Interfaces

Housekeeping

Code organization conventions

tag v1.2.0
video 45 Code Organization

Adding documentation

- above every
    - package
        - doc.go
    - func

running the **go doc <module path>** command shows you documentation, or you can go to **godoc.org/<module path>**
tag v1.2.1
video 46 Adding Documentation

Testing

s
tag v1.2.2
video 47 Testing

Example tests

tag v1.2.3
video 48 Example Tests

No return

Some funcs, you don't have to deal with their returns. What gives?
video 49 No Return

The testify mock package

https://github.com/stretchr/testify
video: 49-add-on

Introduction to gomock

1. make sure GOBIN is set to somewhere on your path
2. go get gomock

    a. go get github.com/golang/mock/mockgen
3. create a "tools.go" file to have an import statement so that go.mod keeps the dependency for
    a. github.com/golang/mock
    b. https://github.com/golang/go/wiki/Modules#how-can-i-track-tool-dependencies-for-a-module
    c. make sure you have a **// +build tools** comment at the top of the tools.go file
4. look at your go.mod file
    a. see the import?
    b. if it still says "indirect" run **go mod tidy**

tag v1.2.4
video 50 Intro Gomock

Generating gomock code

5. enter a command to generate your code
    a. mockgen has two modes of operation: source and reflect.
        - ***use the SOURCE mode***
        - ***reflect doesn't seem to work with go modules***
    b. command
        - **mockgen -self_package github.com/GoesToEleven/golang-architecture -destination mock.go -package architecture -source service.go**
        - meaning of flags
            - **self_package**
                - the full package name of where the destination is located
                - prevents cyclical imports / self-imports
            - **destination**
                - where to put the generated code
                - ***should be in the same folder to prevent cyclical import statements***
            - **package**
                - the package the generated code should have as its package statement
            - **source**
                - the file(s) that has the interface(s) you want to test

video 51 Gen Gomock Code

Testing with gomock

6. use your mock
    a. Create a gomock controller
    b. Create the mock of your interface

c. EXPECT the usage of you functions
d. Use the functions
e. Call the controller.Finish() function

tag v1.2.5
video 52 Testing With Gomock

Method sets explained

Interface types

- "An interface type specifies a **method set** called its interface. A variable of interface type can store a value of any type with a **method set** that is any superset of the interface. Such a type is said to implement the interface. The value of an uninitialized variable of interface type is nil." golang spec

Method sets

- "A type may have a method set associated with it. The **method set** of an interface type is its interface.
  - **type T**
    - The method set of any other **type T** consists of all methods declared with **receiver type T**.
  - **type *T**
    - The method set of the corresponding pointer **type *T** is the set of all methods declared with **receiver *T or T** (that is, it also contains the method set of T).
  - Further rules apply to structs containing embedded fields, as described in the section on struct types. Any other type has an empty method set. In a method set, each method must have a unique non-blank method name. **The method set of a type determines the interfaces that the type implements and the methods that can be called using a receiver of that type.**

IMPORTANT: "The method set of a type determines the INTERFACES that the type implements....." ~ golang spec

TYPE	RECEIVERS
T	(t T)

TYPE	RECEIVERS
*T	(t T)
	(t *T)

video 53 Method Sets Explained

Method sets example

tag v1.3.0
video 54 Method Sets Example

Empty interface
video 55 Empty Interface

Hands-on Exercises - ninja level 2

Hands-on exercise #1
Create an interface and use it.
tag v1.6.0
video: 56 Hands On 1

Hands-on exercise #2
Recreate the database example using two maps to simulate the databases.

- Create an “accessor” interface that has two methods
    - save
    - retrieve
- have your databases implement the “accessor” interface
- create functions that take in values of type accessor
    - put
    - get
- use “put” and “get” to store data in either database

tag v1.7.0
video: 57 Hands On 2

Hands-on exercise #3
Follow these steps

- fork
- clone
- checkout v1.7.0
- branch
- make changes:
    - Organize your code to follow the proper go conventions
        - **cmd** folder
            - subfolder for what you want to call your executable
            - put in here
                - package main
                - func main
        - create package architecture
        - create database implementations in separate folders

        - mongo
            - package mongo
            - db.go
        - harddrive
            - package harddrive
            - db.go

tag v1.8.0
video: 58 Hands On 3

Hands-on exercise #4

Add some documentation, both package level and on your interface

- Tip: place the package documentation in a **doc.go**

tag v1.8.1
video: 59 Hands On 4

Hands-on exercise #5

In this exercise, copy the contents from one file to a new file. Create a file "file-01.txt" from which to copy. To complete this exercise, use

- **io.Copy**
- **os.Create**
- **os.Open**

tag v1.9.0
video: 60 Hands On 5

Context

Where we are in the course

video 61 Where Are We

Understanding context

- https://godoc.org/context
- https://blog.golang.org/context
    - Talk Video: https://vimeo.com/115309491
    - Slide Deck: https://talks.golang.org/2014/gotham-context.slide#1
- Ardan labs Article

video: 62 Understanding Context

Reading documentation

In Go servers, each incoming request is handled in its own goroutine. Request handlers often start additional goroutines to access backends such as databases and RPC services. The set of goroutines working on a request typically needs **access to request-specific values** such as the ***identity of the end user, authorization tokens, and the request's deadline.*** **When a request is canceled or times out,** ***all the goroutines working on that request should exit quickly so the system can reclaim any resources*** they are using. At Google, we developed a context package that makes it easy to pass ***request-scoped values, cancelation signals, and deadlines*** across API boundaries to all the goroutines involved in handling a request. The package is publicly available as context. This article describes how to use the package and provides a complete working example. source
video: 63 Reading Documentation

With timeout

WithTimeout sets a timeout for a context. If the timeout is exceeded, the context is cancelled, which leads to the request being cancelled. Canceling this context releases resources associated with it, so code should call cancel as soon as the operations running in this Context complete:
tag v1.10.0
video: 64 With Timeout

With value

WithValue returns a copy of parent in which the value associated with key is val. Use context Values only for request-scoped data that transits processes and APIs, not for passing optional parameters to functions. The provided key must be comparable and should not be of type string or any other built-in type to avoid collisions between packages using context. Users of WithValue should define their own types for keys. To avoid allocating when assigning to an interface{}, context keys often have concrete type struct{}. Alternatively, exported context key variables' static type should be a pointer or interface. The four common values stored in context that Daniel mentioned:

- **userID**
- **session ID**
- **isAdmin**
- **IP address**

tag v1.11.0
video: 65 With Value

Abstracting with value

Creating a package for context to use the withValue() func

tag v1.11.1
video: 66 Abstracting With Value

Done

Done returns a channel that's closed when work done on behalf of this context should be canceled. Done may return nil if this context can never be canceled.
tag v1.12.0 && 1.12.1
video: 67 Done

Hands-on exercises - ninja level 3

Hands-on exercise #1

- Create a background context.
- Create a child context with that background as a parent
- store two values in the child context
  - key of type string
  - key of type int
- pull the values out

tag v1.13.0
video: 67 Hands On 1

Hands-on exercise #2 - attempt 1

There are two versions of this video: a first try and a second try. You will see the SECOND try first. After that, if you want to see more, you can watch the longer and slower, and with more discussion, FIRST try.

- ***SECOND try is first***
- ***FIRST try is second***

Using the code from the previous exercise

- have each key be its own type
- abstract all of the context code into its own
  - functions as applicable
  - package, separate from main
- pull the values out

tag v1.14.0 & v1.16.0
video 69 Hands On 2 - Attempt 1

Hands-on exercise #2 - attempt 2

video 68 Hands On 2 - Attempt 2

Hands-on exercise #3

Answer these two questions:

- you should store **LESS** or **MORE** values in context?
- what are the four common values stored in context that Daniel mentioned?

video: 70 Hands On 3

Hands-on exercise #4 - attempt 1

There are two versions of this video: a first try and a second try. You will see the SECOND try first. After that, if you want to see more, you can watch the longer and slower, and with more discussion, FIRST try.

- ***SECOND try is first***
- ***FIRST try is second***

Context is not magic. It doesn't end stuff automatically. If something takes in a context, it knows about the context, and then can end stuff <u>if it is coded to do so</u>.

- create a background context
- create a child context **with a timeout**
- show the code working two ways
  - not timing out
  - timing out

Things that will be useful to use in this example:

- time.**Time** is a specific ***point*** in time
- time.**Duration** is a ***period*** of time
- **time.Millisecond**
  - this is a duration
- **time.Sleep**
  - takes in a duration

tag v1.15.0 & v1.17.0

video: 71 Hands On 4 - Attempt 2

Hands-on exercise #4 - attempt 2

video: 72 Hands On 4 - Attempt 1

Hands-on exercise #5

- create a background context
- create a child context **with cancel**
  - assign this to a variable "ctx"
- launch 100 goroutines
  - give a unique number to each goroutine
    - print "running" and the number

    - have an eternal loop with
        - select
            - case <- ctx.Done():
                - return
            - default:
                - print “still working” and the number
                - sleep for a 1/10 of a second
        - show the number of goroutines running
- after your eternal for loop
    - sleep for a bit so that all goroutines can be launched
    - call your cancel func
        - this will cancel your context, and all launched goroutines
    - sleep for a bit so that your program can close goroutines
- show the goroutines running
- exit main

Things that will be useful to use in this example:

- runtime.NumGoroutine()
- time.Sleep

func WithCancel(parent Context) (ctx Context, cancel CancelFunc)

WithCancel returns a copy of parent with a new Done channel. The returned context's **Done channel is closed when the returned cancel function is called** or when the parent context's Done channel is closed, whichever happens first. **Canceling this context releases resources associated with it,** so code should call cancel as soon as the operations running in this Context complete. source

A receive expression used in an assignment or initialization of the special form

x, ok = <-ch
x, ok := <-ch
var x, ok = <-ch
var x, ok T = <-ch

yields an additional untyped boolean result reporting whether the communication succeeded. The value of ok is true if the value received was delivered by a successful send operation to the channel, or false if it is a zero value generated because the channel is closed and empty.
source

tag v1.18.0
video: 71 Hands On 5

Dealing with errors

Introduction

- where we've been
- where we are
- where we're going

video: 72 Intro

Go and errors

- https://golang.org/doc/faq#exceptions
    - Go does not favor try / catch / finally
- https://en.wikipedia.org/wiki/Exception_handling#Criticism
    - Notice Hoare's work also influenced goroutines and channels
- https://github.com/golang/go/wiki/ErrorValueFAQ
- **go blog**
    - https://blog.golang.org/errors-are-values
    - https://blog.golang.org/error-handling-and-go

other things to know

- builtin.error

```
type error interface {
    Error() string
}
```

The error built-in interface type is the conventional interface for representing an error condition, with the nil value representing no error.

Reading documentation

- https://godoc.org/errors

video 73 Go And Errors

Dealing with errors

- **panic(error)**
    - this shuts your program down
    - so only "panic" if you need your program to shut down
- **returning**
    - don't just return your error
    - **wrap your error in more detail, then return that**

All **errors should be handled just once.** You can handle your error in a variety of ways:

- you can print it to the command line with the **LOG** or **FMT** package
  - your program can then end, or move along
  - either way, your error is handled
  - **log**
    - Print
    - Fatal
    - Panic
  - **fmt**
    - Print
- Special code, based upon the error
  - Return custom http status code
  - Show instructions to user to resolve the error

Also, you can wrap an error to add more details

- fmt
  - Errorf

video 74 Dealing With Errors

Creating an error using the standard library

- **fmt.Errorf**
- **errors.New()**

tag v1.19.0
video 75 Create Error Using Standard Library

Creating a custom error

Create a type that fulfills the error interface

```
type error interface {
    Error() string
}
```

You can use the “is(error) bool” function to customize the comparability of custom errors

- Is(error) bool
- An error is considered to match a target if it is equal to that target or if it implements a method Is(error) bool such that Is(target) returns true.

tag v1.20.0
video 76 Creating Custom Error

Wrapping errors

- Add more details to your errors - two ways to do this
  - fmt.Errorf with %w
  - create a custom error with your own time
    - add an **unwrap() error** function

tag v1.21.0
video 77 Wrapping Errors

Accessing wrapped errors

```
if errors.Is(err, os.ErrExist)
```

is preferable to

```
if err == os.ErrExist
```

```
var perr *os.PathError
if errors.As(err, &perr) {
      fmt.Println(perr.Path)
}
```

is preferable to

```
if perr, ok := err.(*os.PathError); ok {
      fmt.Println(perr.Path)
}
```

tag v1.22.0
video 78 Accessing Wrapped Errors

Errors in-depth

Introduction

Personal anecdote about sincerity and wanting you to master this material on errors
video 79 Intro

Provide meaning

main take-aways:

- deal with your errors **OR**
- provide meaning so that your error tells **what, where, how, whom, and when** (as applicable) and return it

If we just **fmt.Println** or **log.Println** or **panic** our "**error**" then we don't get this extra meaning. We want to **add meaning to our errors - what, where, how, whom, and when**

- go 1.13 gives us the ability to add meaning without losing
    - **original error**
    - **comparability of errors**
        - **IS** the error this other type of error

        - **assertion of errors**
            - the error **AS** this other type of error allows us to access functionality of that other type
- **ADD MEANING TO YOUR ERRORS WITH**
    - **fmt.Errorf** and use **%w** and pass in the error to %w
        - this allows us to **unwrap** our error to one level of func calls
        - this allows us to use **IS** or **AS** to many levels of func calls
    - **errors.New()** when you don't need to pass in a previous error
        - good for creating a "base" error that everything is going off
        - you can also use fmt.Errorf for the "base" error and just not pass into %w

Pass errors up through the call stack as necessary, adding something to each pass
video 80 Provide Meaning

Why compare errors

os file to demonstrate
reviewing using **IS**
tag v1.23.0
video 81 Why Compare Errors

Why assert errors

os file to demonstrate
reviewing using **AS**
We **ASSERT** so that we can access fields or methods associated with another error type. So if we get an error, we can **ASSERT** if this error is also of this other type, which we **ASSERT** to see if that is so, then we can have access to methods and fields of that other type.
tag v1.24.0
video 82 Why Assert Errors

Switch case

switch case is for multiple conditions

- **switch** is **switching** between conditions

select case is for multiple conditions RELATED TO CHANNELS

- **select** is **selecting** a channel

tag v1.25.0
video: 83 Switch Case

Why unwrap

When you create a custom type that is an error, eg,

```
type ErrFile struct {
	Filename string
	Base     error
}

// type ErrFile implementing the error interface
func (e ErrFile) Error() string {
	return fmt.Sprintf("File %s: %v", e.Filename, e.Base)
}

func (e ErrFile) Unwrap() error {
	return e.Base
}
```

you can add the **Unwrap() error** method so that you can get back to the base (original) error
video: 84 Why Unwrap

Using unwrap

this is why **%w** is important with **fmt.Errorf**
tag v1.26.0
video 85 Using Unwrap

Errors in practice

Checking errors after processing

bufio.Scanner
tag v1.27.0 & v.1.27.1
video: 86 Checking Errors After Processing

Best practices

- abstract away errors if it is going to be used over and over in your code base
    - pull it into a package
- if just used once, check errors at each step
- adding in fmt.Errorf and %w in the wrapping functions

tag v1.28.0
video: 87 Best Practices

Understanding best practices

- going through the code

tag v1.28.1
video: 88 Understanding Best Practices

Abstracting into a package

-

tag v1.29.0
video: 89 Abstract To Package

Hands-on exercises - ninja level 04

Hands-on exercise #1

- create a type
- have it implement the error interface
- create a value of that type
- pass that into a function, or return it from a function, that takes type error

tag v1.30.0
video: 90 Hands On 1

Hands-on exercise #2

Answer these questions

- Do you ever want to use
    - **panic(error)**?
        - programmer error happened
        - prints out stack trace
    - **log.Panic(error)**?
        - programmer error happened
        - prints out stack trace
    - **log.Fatal(error)**?
        - turns off the program
        - does not print out stack trace
        - appropriate for, “Something has gone so catastrophically wrong, the program shouldn’t try to recover.”
- Do you ever want to **return** an error without adding any information to the error?
    - always add information about the error
- Do you ever want to **log.Print** or **fmt.Print** an error without adding any information to the error?
    - always good to add info
- Do you always want to use **fmt.Errorf()**?
    - whenever you’re returning an error, wrap it with more info

        - always doesn't necessarily as you might have created your own error struct, or something

video: 91 Hands On 2

Hands-on exercise #3

Using **fmt.Errorf()**, handle all errors in a program that does the following:

- open file #1
- create file #2
- copy the contents of file #1 to file #2

tag v1.31.0
video: 92 Hands On 3

Hands-on exercise #4

Using the code in the previous example, use **errors.Is()** to give a special message if the file you are trying to open does not exist. Notes:

- https://godoc.org/os#pkg-variables

tag 1.31.1
video: 93 Hands On 4

Hands-on exercise #5

Using the code in the previous example, use **errors.As()** to access *PathError and print out some fields or run some methods attached to *PathError. Notes:

- https://godoc.org/os#Open
- https://godoc.org/os#Create
- https://godoc.org/os#PathError

tag v1.31.2
video: 94 Hands On 5

Hands-on exercise #6

Recreate the foo bar moo cat example. To do this, follow these general steps:

- create these functions, each of which returns a value of type error
        - main calls foo
    - foo
        - foo calls bar
    - bar
        - bar calls moo
    - moo
        - moo calls cat
    - cat
        - cat calls nothing

- include the error information from any previous calls
- use **errors.Unwrap()** to access all of the individual errors
  - name the variables that hold the unwrapped error
    - fooErr
    - barErr
    - mooErr
    - catErr
    - baseErr
      - this err will be nil

tag v1.32.0
video: 95 Hands On 6

Nice to know

Overview
video 96 Overview

String method
Stringer interface
tag v1.33.0
video: 97 String Method

Equality on structs
You can use a struct as a key to a map
tag v1.34.0
video: 98 Equality On Structs

Stack, heap, escape analysis intro
- As a general rule, don't use pointers if you don't need them

video: 99 Stack, Heap, Escape Analysis Intro

Exploring stack, heap, escape analysis
video: 100 Exploring Stack, Heap, Escape Analysis

Coding stack, heap, escape analysis
go run -gcflags "-m" main.go
tags v1.35.0
video: 101

Review

Todd

- Git versioning
- Errors
    - How they should be dealt with
    - Is
    - As
    - Unwrap
    - fmt.Errorf
- Basic curl
- Architecture concepts
- Interfaces
- Method Sets
- Context

video 102 Todd

Daniel

- git versions
- interfaces
    - a lot of detail
    - a little painfully slow at times!
- documenting
    - goc.go
- testing
- **best practices**
    - **document**
    - **write tests**
    - **handle errors**
- context
- KEEPING IT REAL

video 103 Daniel

Farewell

Congratulations!

- Great job completing the course. I am so proud of you.

Video file: 104 Congrats

Bonus lecture