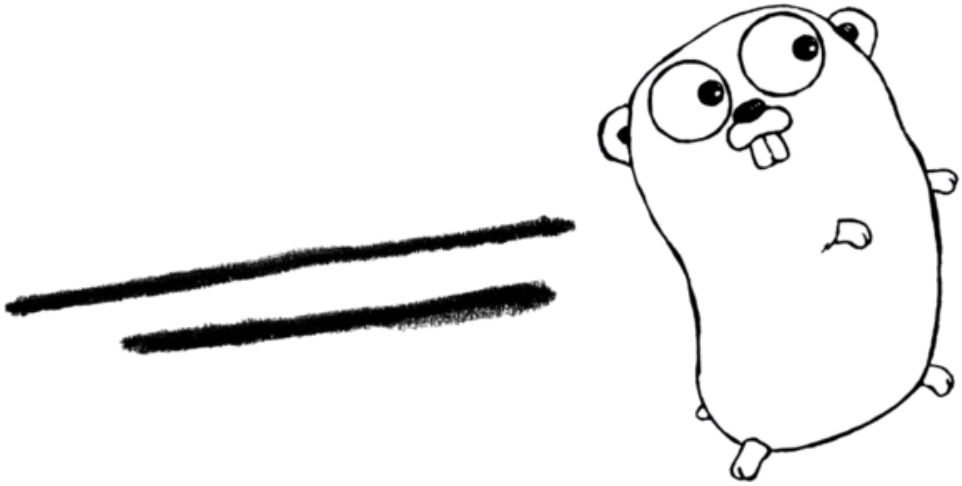
Investigación sobre Lenguajes para desarrollo de aplicaciones web: **Lenguaje Go.**



Materia: Programación Web
Docente: Ing. Sergio Octavio Rosales Aguayo
Unidad 1
Instituto Tecnológico de Tepic
Ingeniería en Sistemas Computacionales.

Integrantes:
Arellano del Aguila Jorge Arturo
Becerra Casillas Efraín Alejandro
Garnica López Luis Alberto
Pardo Pérez Jessica Lizbeth
Pérez González Cesar Antonio
Rivera Torres Alejandro

Sumario

Historia....3

Precursores....3

El proyecto Go....4

Descripción....6

Características Generales....7

Alfabeto....7

Letras y Dígitos....7

Comentarios....7

Tokens....7

Identificadores....7

Keywords....8

Operadores y delimitadores....8

Literales Enteras....8

Estructura de un Programa....9

Características del lenguaje....11

Usos del Lenguaje....11

Aplicaciones Web con Go....12

Un ejemplo de la codificación de la aplicación en Go....16

Arquitectura de una aplicación web en Go....17

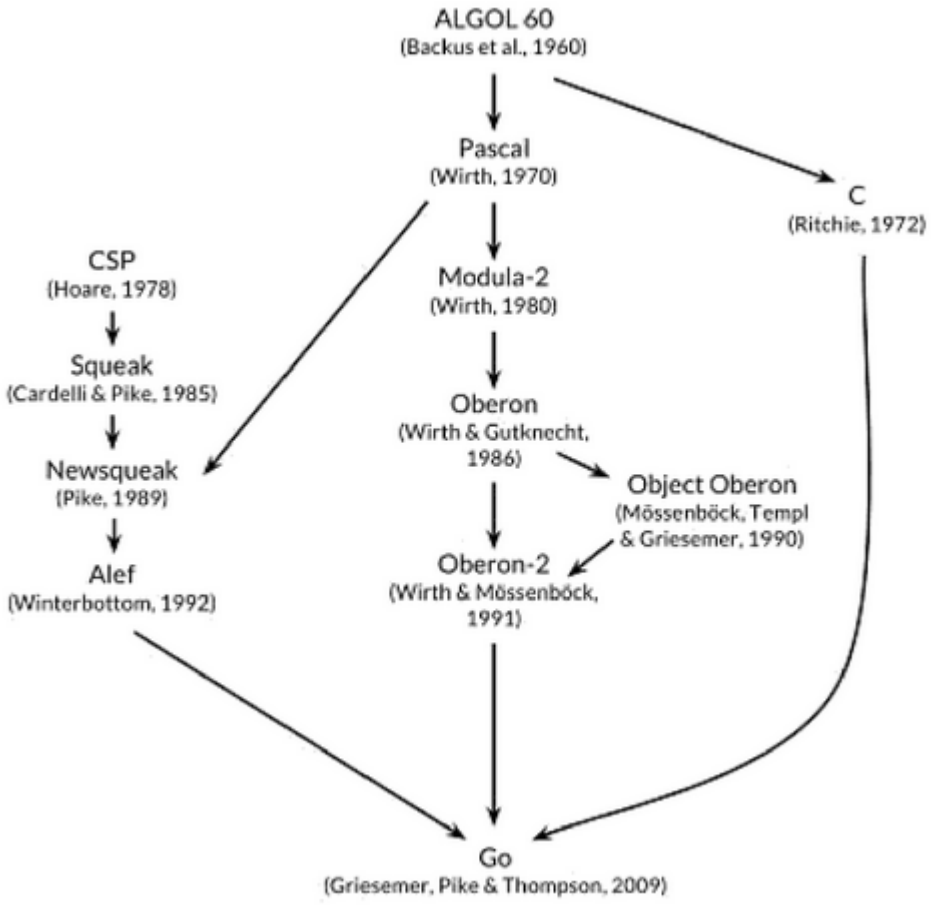
Fuentes de información....18

Historia

Precursores

Podemos aprender mucho sobre porqué un lenguaje es de la forma en que es y para qué ambiente ha sido adaptado mirando a los lenguajes que le influyeron.

La figura inferior muestra las influencias más importantes de lenguajes de programación tempranos en el diseño de Go.



Go es descrito algunas veces como un lenguaje similar a C, o como “C para el siglo 21.” De C, Go heredó su sintaxis de expresiones, sentencias de control de flujo, tipos de datos básicos, paso de parámetros, punteros, y sobre todo, el énfasis de C en programas que compilan en código máquina eficiente y cooperan naturalmente con las abstracciones de los sistemas operativos actuales.

Pero existen otros ancestros de Go en el árbol familiar. Una gran influencia viene de lenguajes hechos por Niklaus Wirth, comenzando con Pascal. Modula-2 inspiró el concepto de paquetes. Oberon eliminó la distinción entre archivos de interfaz de módulos y archivos de implementación de módulos. Oberon-2 influenció la sintaxis para paquetes,

importaciones y declaraciones, y Object Oberon proveyó la sintaxis para declaraciones de métodos.

Otro linaje sobre ancestros de Go, y que lo hace distinguirse entre los lenguajes de programación recientes, es una secuencia de poco conocidos lenguajes de investigación desarrollados en Bell Labs, todos inspirados por el concepto de Comunicación de procesos secuenciales (CSP communicating sequential processes) desde el documento de 1978 de Tony Hoare sobre las bases de la concurrencia. En CSP, un programa es una composición paralela de procesos que no tienen un estado compartido; los procesos se comunican y sincronizan usando canales. Pero el CSP de Hoare era un lenguaje formal para describir los conceptos fundamentales de concurrencia, no un lenguaje de programación para escribir programas ejecutables.

Rob Pike y otros comenzaron a experimentar con implementaciones de CSP como lenguajes de programación reales. El primero fue llamado Squak (“A language for communicating with mice”), el cual proveyó un lenguaje para manejar eventos del ratón y teclado, con canales creados estáticamente. Esto fue seguido por Newsqueak, el cual ofrecía sentencias y sintaxis de expresiones similares a C y notación de tipos similar a pascal. Se trataba de un lenguaje puramente funcional con recolección de basura, de nuevo dirigido a la administración de los eventos de teclado, ratón y ventanas. Los canales se volvieron guardables en variables y creados dinámicamente.

El sistema operativo Plan 9, llevó adelante esas ideas en un lenguaje llamado Alef. Alef intentó hacer de Newsqueak un lenguaje de programación de sistemas viable, pero su omisión de la recolección de basura hizo la concurrencia demasiado difícil.

Otras construcciones en Go muestran la influencia de “genes no-ancestrales” por aquí y por allá; por ejemplo iota es un poco de APL, y el análisis léxico con funciones anidadas es de Scheme. Las partes innovadoras de Go proveen arreglos dinámicos con acceso aleatorio eficiente pero también permiten sofisticados arreglos compartidos remanentes de listas enlazadas. Y la sentencia defer es nueva con Go.

El proyecto Go.

Todos los lenguajes de programación reflejan la filosofía de programación de sus creadores, quienes usualmente incluyen un componente significativo de reacciones a las deficiencias percibidas en lenguajes anteriores. El proyecto Go nació de la frustración con varios sistemas de software en Google que estaban sufriendo de una explosión de complejidad (Este problema no es, por cierto, exclusivo de Google).

Cómo lo planteó Rob Pike, “la complejidad es multiplicativa”: arreglar un problema haciendo una parte del sistema más compleja, lenta pero inevitablemente añade más

complejidad a las otras partes. Con presión constante para añadir características, opciones y configuraciones, y para entregar código con rapidez, es fácil perder simplicidad, incluso aunque la simplicidad es la clave para el buen software a la larga.

La simplicidad requiere más trabajo al inicio de un proyecto para reducir una idea a su esencia y más disciplina a lo largo del ciclo de vida de un proyecto para distinguir los cambios buenos, de los malos o perniciosos. Con suficiente esfuerzo, un buen cambio puede ser adaptado sin comprometer lo que Fred Brooks llamó la "integridad conceptual" del diseño pero un mal cambio no puede serlo, y un cambio pernicioso intercambia la simplicidad por su primo superficial, la conveniencia. Sólo a través de la simplicidad del diseño, un sistema puede permanecer estable, seguro y coherente a medida que crece.

El proyecto Go incluye el lenguaje en sí mismo, sus herramientas y librerías estándar, y por último pero no menos importante, una agenda cultural de simplicidad radical. Como un reciente lenguaje de alto nivel, Go tiene el beneficio de comprensión retrospectiva, y los aspectos básicos están bien hechos: tiene recolección de basura, un sistema de paquetes, funciones, análisis léxico, una interfaz de llamadas al sistema, y cadenas inmutables en cuyo texto generalmente está codificado en UTF-8. Pero tiene, comparativamente pocas características y es poco probable añadirle más. Por ejemplo, no tiene conversión numérica implícita, no hay constructores o destructores, no tiene sobrecarga de operadores, no hay valores de parámetros por defecto, no hay herencia, no hay genéricos, no hay excepciones, macros ni funciones, no hay anotaciones y no hay almacenamiento del hilo local. El lenguaje es maduro y estable, y garantiza compatibilidad ascendente: programas viejos de Go pueden ser compilados y corridos con las nuevas versiones de los compiladores y las librerías estándar.

Go tiene suficiente de un sistema de tipos como para evitar la mayoría de los errores descuidados que plagan a los programadores en los lenguajes dinámicos, pero tiene un sistema de tipos más simple que los lenguajes tipados con los que puede ser comparado. Este acercamiento algunas veces puede llevar a las isoladas definiciones de "no-tipado" al programar con un framework de tipos más amplio, y los programadores de Go no van tan lejos como los de C++ o los de Haskell al expresar las propiedades de seguridad como pruebas de que se encuentra basado en tipos. Pero en la práctica Go brinda a los programadores muchos de los beneficios de la seguridad y rendimiento en tiempo de ejecución de un sistema de tipos relativamente fuerte sin la carga de tener uno muy complejo.

El Go anima y se preocupa del diseño contemporáneo de sistemas de cómputo, y particularmente de la importancia de la localidad. Sus tipos de datos primitivos y gran parte de sus librerías de estructuras de datos están diseñados para trabajar naturalmente sin inicialización explícita o constructores implícitos, así que relativamente pocas alojaciones y escritura de memoria están escondidos en el código. Los tipos agregados de

Go (Estructuras y arreglos) almacenan sus elementos directamente, requiriendo menos almacenamiento y menores alojamientos y direcciones de punteros que lenguajes que usan campos indirectos. Y desde que la computadora moderna es una máquina paralela, Go tiene características de concurrencia basadas en CSP, como se mencionó anteriormente. Las pilas de tamaño variable de los hilos ligeros de Go o “goroutines” son inicialmente tan lo suficientemente pequeñas que crear una goroutine es barato y crear un millón es práctico.

La librería estándar de Go, usualmente descrita como que viene con “baterías incluidas”, provee bloques de construcción limpios y APIs para entrada y salida, procesamiento de texto, gráficos, criptografía, redes y aplicaciones distribuidas, con soporte para muchos formatos de archivo estándar y protocolos. Las librerías y herramientas hacen extensivo el uso de la convención para reducir la necesidad de configuración y explicación, simplificando así la lógica de programación y haciendo programas de Go diversos más similares unos con otros y siendo así más fácil de aprender. Los proyectos construidos usando la herramienta Go usan solo archivos y nombres de identificadores y un comentario especial ocasional para determinar todas las librerías, ejecutables, pruebas, ejemplos, variantes para plataformas específicas, y documentación para un proyecto; la fuente Go en si misma contiene la especificación del proyecto.

Descripción

La página oficial de Go describe al lenguaje de programación de la siguiente forma:

> Go es un lenguaje de programación de propósito general diseñado con el proposito de la programación de sistemas. Está fuertemente tipado, cuenta con un recolector de basura y tiene un apoyo explícito a la programación concurrente. Los y programas se construyen a partir de paquetes, con propiedades que permitan una gestión eficiente de las dependencias. Las implementaciones existentes utilizan un modelo de compilación/enlace tradicional para generar los binarios ejecutables.
>
> La gramática es compacta y regular, lo que facilita el análisis mediante herramientas automáticas tales como entornos de desarrollo integrado.

Características Generales

Alfabeto

El código fuente está codificado en texto Unicode UTF-8. Cada punto de código es distinto; por ejemplo, las letras mayúsculas y minúsculas son diferentes caracteres.

Letras y Dígitos

```
letter              = unicode_letter | "_" .

decimal_digit  = "0" … "9" .

octal_digit         = "0"… "7" .

hex_digit           = "0"… "9" | "A" … "F" | "a" … "f" .
```

Comentarios

Comentarios sirven como documentación del programa. Hay dos formas:

- Las líneas de comentario comienzan con la secuencia de caracteres // y se detienen al final de la línea.
- Comentarios generales comienzan con la secuencia de caracteres / * y termina con la primera secuencia de caracteres posterior * /.

Un comentario no puede comenzar dentro de una runa o cadena literal, o dentro de un comentario. Una observación general que no contiene saltos de línea actúa como un espacio. Cualquier otro comentario actúa como una nueva línea.

Tokens

Los tokens forman parte del vocabulario de Go, existen cuatro clases: Identificadores, keywords, operadores y delimitadores y literales.

Identificadores

Los identificadores nombran a las entidades de programas tales como variables y tipos. Un identificador es una secuencia de una o más letras y dígitos. El primer carácter de un identificador debe ser una letra.

```
identifier = letter { letter | unicode_digit } .
```

a
_x9
ThisVariableIsExported

Keywords

Las siguientes palabras clave están reservados y no se pueden utilizar como identificadores.

```
break          default              func      interface
     select
```

```
case            defer                go              map
          struct

chan            else                 goto            package
          switch

const           fallthrough          if              range
          type

continue         for                 import          return
          var
```

Operadores y delimitadores

Las siguientes secuencias de caracteres representan los operadores, los delimitadores y otros símbolos especiales:

```
+     &      +=     &=      &&     ==     !=     (     )

-     |      -=     |=      ||     <      <=     [     ]

*     ^      *=     ^=      <-     >      >=     {     }

/     <<     /=     <<=     ++     =      :=     ,     ;

%     >>     %=     >>=     --     !      ...    .     :

      &^           &^=
```

Literales Enteras

Un literal entero es una secuencia de dígitos que representan una constante entera. Un prefijo opcional establece una base no decimal: 0 para octal, 0x o 0X para hexadecimal. En literales hexadecimales, las letras A-F y A-F representan los valores de 10 a 15.

```
int_lit         = decimal_lit | octal_lit | hex_lit .

decimal_lit     = ( "1" … "9" ) { decimal_digit } .
```

```
octal_lit      = "0" { octal_digit } .


hex_lit        =   "0"   (   "x"   |   "X"   )   hex_digit   {
hex_digit } .

42
0600
0xBadFace
170141183460469231731687303715884105727
```

Estructura de un Programa

Miremos al siguiente programa con más detalle.

```
package main

import “fmt”

// Esto es un comentario

func main(){
        fmt.Println(“Hola, mundo!”)
}
```

Los programas de Go se leen de arriba a abajo, de izquierda a derecha (como un libro). La primera línea dice esto:

package main

Esto es conocido como una declaración de paquete, y todo programa en Go debe iniciar con ella. Los paquetes son la manera de Go para organizar y reutilizar código. Existen dos tipos de programas en Go: ejecutables y bibliotecas. Las aplicaciones ejecutables son el tipo de programas que podemos correr directamente desde la terminal (en Windows, terminan con .exe). Las bibliotecas son una colección de código que nosotros empacamos junto de manera que podamos usarlo en otros programas.

En la siguiente línea vemos lo siguiente:

import “fmt”

La palabra reservada import es como nosotros incluimos código desde otros paquetes para usarlo con nuestro programa. El paquete fmt (abreviación de format) implementa formatos para entrada y salida.

Nótese que fmt está rodeado por comillas dobles. El uso de comillas dobles como esta es conocido como cadena de caracteres, la cual es un tipo de expresión. En Go, las cadenas representan una secuencia de caracteres (letras, números, símbolos, etc.) de una determinada longitud.

La línea que inicia con // es conocida como un comentario. Los comentarios son ignorados por el compilador de Go y existen para nuestro propio uso (o el de cualquier persona que tome el código fuente de tu programa). Go soporta dos tipos diferentes de estilos de comentarios: // comentarios en los que todo el texto entre el simbolo // y el fin de línea es parte del comentario, y /* */ comentarios en los cuales todo lo que está entre los asteriscos es parte del comentario (y puede incluir líneas múltiples).

Después de ello, vemos la declaración de una función.

```
func main() {
        fmt.Println(“Hola, Mundo!”)
}
```

Las funciones son los bloques de construcción de un programa en Go. Tienen entradas, salidas, y una serie de pasos llamados sentencias que son ejecutados en orden. Todas las funciones inician con la palabra reservada func seguida del nombre de la función (main, en este caso), una lista de cero o más parámetros rodeados de paréntesis, un tipo de retorno opcional, y un cuerpo el cual está rodeado por llaves. Esta función no tiene parámetros, no retorna nada y solo tiene una sentencia. El nombre main es especial porque es la función que se manda llamar cuando se ejecuta el programa.

La pieza final de nuestro programa es esta línea:

```
fmt.Println(“Hola, Mundo!”)
```

Esta sentencia está hecha de 3 componentes. Primero, accedemos a otra función dentro del paquete fmt llamada Println; Println significa “print line”. Entonces creamos una nueva cadena que contiene “Hola, Mundo!” e invocamos (también llamado llamar o ejecutar) esa función con la cadena como su primer y único argumento.

Características del lenguaje.

Es esencialmente un lenguaje de tipo imperativo (procedural, estructural), construido pensando en la concurrencia.

No es orientado a objetos en el sentido tradicional como Java y C++ porque no tiene el concepto de clases y herencia. Sin embargo si tiene el concepto de interfaces, con las cuales mucho del polimorfismo puede ser hecho. Go tiene un claro y expresivo sistema de

tipos, pero es ligero y sin jerarquía. Así que sobre esto se podría decir que es un lenguaje híbrido.

La orientación a objetos como en los lenguajes que siguen este paradigma, fue considerada como demasiado “pesada”, generando desarrollo usualmente incomodo construyendo grandes jerarquías de tipos, y por lo tanto no ayudando a la meta de velocidad del lenguaje.

Las funciones son los bloques básicos de construcción en Go, y su uso es muy versátil. Es estáticamente tipado, así que es lenguaje seguro, y compila a código nativo, así que tiene una ejecución muy eficiente.

Es fuertemente tipado: conversiones implícitas de tipo (también llamadas castings o coerciones) no son permitidas; el principio es: ¡mantén las cosas explicitas!

Tiene ciertas características de los lenguajes dinámicamente tipados (mediante la palabra reservada var).

Así que también apela a los programadores que dejaron el mundo de java y .Net por Python, Ruby, PHP y Javascript.

Go tiene soporte para compilación cruzada: Ejemplo, Desarrollar en una máquina Linux para una aplicación que será ejecutada en Windows. Es el primer lenguaje de programación en el que se puede usar UTF-8, no solo en cadenas, sino también en el código del programa: ¡Go es verdaderamente internacional!

Usos del Lenguaje

Go fue originalmente concebido como un lenguaje de programación de sistemas, para ser utilizado en el pesado mundo centrado en servidores de los servidores web (Google), almacenamiento y parecidos. Para algunos dominios como sistemas distribuidos de alto rendimiento Go ya ha probado ser un lenguaje más productivo que muchos otros. Go brilla y hace la concurrencia masiva más sencilla, así que debería ser muy bueno para el desarrollo en servidores de videojuegos.

Procesamiento de eventos complejo (CEP, complex event processing), donde uno necesita tanto concurrencia masiva como alto nivel de abstracciones y rendimiento, es también un excelente objetivo en el cual usar Go. Mientras nos movemos al Internet de las Cosas, CEP vendrá al frente.

Pero resulta que también es un lenguaje de programación de propósito general, útil para resolver problemas de procesamiento de texto, hacer frontends, o incluso aplicaciones tipo script.

Sin embargo Go no es adecuado para software en tiempo real, debido a la recolección de basura y la asignación de memoria automática.

Aplicaciones Web con Go.

Go es un lenguaje de programación relativamente nuevo, con una comunidad próspera y en crecimiento. Es muy adecuado para la escritura de programas de servidor que deben de ser rápidos. Es muy sencillo y familiar a la mayoría de los programadores que se utilizan para la programación de procedimientos, además de que también proporciona características de programación funcional. Es compatible con la concurrencia de forma predeterminada, tiene un sistema de paquetes ideal, hace posee con un recolector de basura, y tiene una amplio y potente conjunto de bibliotecas integradas.

Un gran número de bibliotecas de código abierto de buena calidad están disponibles que pueden complementar lo las bibliotecas estándar no tienen, pero las bibliotecas estándar que vienen con Go son bastante amplias y de gran alcance.

Go está ganando rápidamente popularidad como un lenguaje de programación web. Muchas compañías, incluyendo empresas de infraestructuras como Dropbox y SendGrid, orientadas a la tecnología.

Go ofrece una alternativa viable a los lenguajes existentes y plataformas para el desarrollo de aplicaciones web a gran escala. Las aplicaciones web a gran escala por lo general tienen que brindar las características siguientes:

- Escalabilidad
- Modularidad
- Mantenibilidad
- Alto rendimiento

Aplicaciones Web Escalables y Go

Las aplicaciones web a gran escala deben ser escalables. Esto significa que debe ser capaz de rápida y fácilmente aumentar la capacidad de la aplicación para asumir un mayor volumen de las solicitudes por parte de los clientes. La aplicación debe escalar también linealmente, lo que significa que debe ser capaz de añadir más hardware y procesar un número correspondiente de peticiones. Podemos mirar a la ampliación de dos maneras:

- La escala vertical, o el aumento de la cantidad de CPU o de la capacidad en una sola máquina
- La escala horizontal, o el aumento del número de máquinas para expandir la capacidad

Go escala bien verticalmente con su excelente soporte para la programación concurrente.

Aplicaciones Web Modulares y Go

Las aplicaciones web a gran escala deben ser construidas con componentes que funcionan de manera intercambiable. Este enfoque le permite añadir, eliminar o modificar características fácilmente y le da la flexibilidad para satisfacer las necesidades cambiantes de la aplicación. También le permite menores costes de desarrollo de software mediante la reutilización de componentes modulares. Aunque Go es de tipo estático, tiene un mecanismo de interfaz que describe el comportamiento y permite tipado dinámico. Las funciones pueden tener las interfaces, lo que significa que puede introducir un nuevo código en el sistema y aun así ser capaz de utilizar las funciones existentes mediante la implementación métodos requeridos por esa interfaz.

Aplicaciones Web Mantenible y Go

Al igual que todas las aplicaciones grandes y complejas, teniendo de base un código que es fácil de mantener, es de gran importancia para aplicaciones web a gran escala tener un código sencillo y mantenible. Es importante porque las aplicaciones a gran escala a menudo lo necesitan para crecer y evolucionar, por lo se suele revisar y cambiar la código regularmente. El código complejo, es difícil de manejar, tarda mucho tiempo para cambiar y tiene el riesgo de dañar componentes, así que tiene sentido para mantener el código fuente bien organizado y mantenible.

Go fue diseñado para fomentar las buenas prácticas de ingeniería de software. Tiene una limpia y la sintaxis simple que es fácil de leer. Los sistemas de paquetes de Go son flexibles y sin ambigüedades, y hay un buen conjunto de herramientas para mejorar la experiencia de desarrollo y ayudar los programadores escribir código más legible.

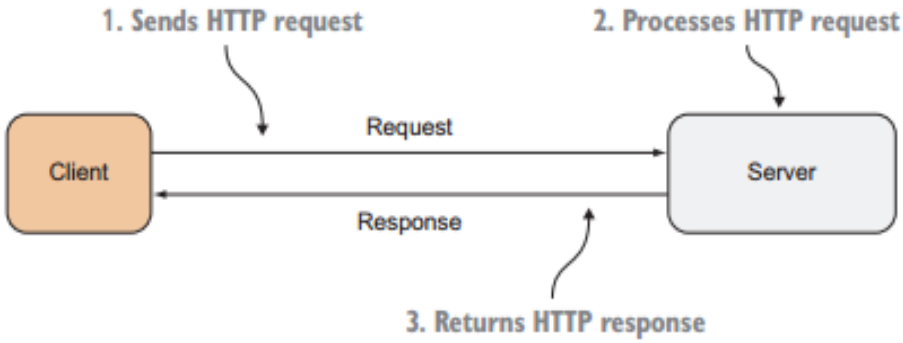
Aplicaciones Web de Alto Rendimiento y Go

Alto rendimiento significa ser capaz de procesar un gran volumen de solicitudes dentro de un corto periodo de tiempo. También significa ser capaz de responder al cliente de forma rápida y haciendo que las operaciones más rápido para los usuarios finales.

Uno de los objetivos de diseño de Go es acercarse al rendimiento de C, y aunque no ha alcanzado este objetivo, los resultados actuales son bastante competitivos. Go compila código nativo, lo que generalmente significa que es más rápido que otros lenguajes interpretados y macros.

Como se describió anteriormente, Go también tiene un gran soporte de concurrencia con goroutines, que permite que múltiples solicitudes sean procesadas al mismo tiempo.

Un ejemplo adecuado de una aplicación Web hecha con Go es un foro o una aplicación para hablar por red. El diseño de una aplicación de comunicación es típico para cualquier otra aplicación web. Las aplicaciones web tienen el flujo en general de que el cliente envía una solicitud a un servidor y el servidor responde a esa solicitud.



La lógica de la aplicación está codificada en el servidor. Mientras el cliente dispara la solicitud y provee información al servidor, el formato y los datos solicitados son sugeridos por el servidor, provistos en hipervínculos en las páginas HTML que el servidor sirve a los clientes.

El formato de la solicitud normalmente es una prerrogativa de la aplicación en sí. Para una aplicación de comunicación, se puede usar el formato http://<servername>/<handler-name>?/<parameters>

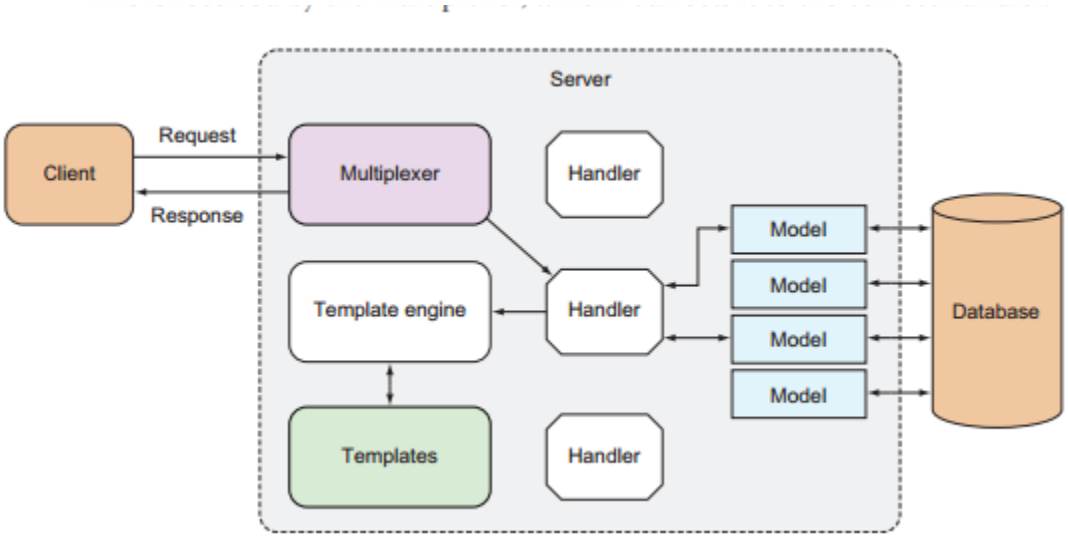
El servername es el nombre del servidor de la aplicación de comunicación; el handler-name es el nombre del manejador que está siendo llamado. El nombre del manejador es jerárquico: la raíz del nombre del manejador es el módulo que está siendo llamado, la segunda parte el submódulo, y así sucesivamente, hasta que se lleva a la hoja, la cual es el

manejador de la solicitud con ese submódulo. Si tenemos un módulo llamado hijo y necesitamos tener un manejador para leer el hilo, el nombre del manejador es /thread/read.

Los parámetros de la aplicación, los cuales son consultas URL, son lo que sea que se necesite pasar al manejador para procesar la solicitud. En el caso del foro, se necesita proveer un identificador único del hijo al manejador, así que los parámetros serán Id = 123, donde 123 es el identificador único.

Recapitulando la solicitud, se vería de la siguiente manera http://foro/thread/read?id=123.

Cuando la solicitud llega al servidor, un multiplexor inspecciona la URL que se solicita y redirige la solicitud al manejador correcto. Una vez que la solicitud llega al manejador, el manejador recibirá información de la solicitud y la procesará. Cuando el procesamiento está completo, el manejador pasa los datos al motor de la plantilla, el cual usará plantillas para generar HTML que será retornado al cliente.



Un ejemplo de la codificación de la aplicación en Go

Se inicia todas las aplicaciones Go con un archivo de código fuente main, el cual es el archivo que contiene la función main y es el punto inicial donde se ejecuta el binario compilado. En el foro podemos llamar este archivo como main.go

```
package main
import (
"net/http"
)
func main() {
mux := http.NewServeMux()
files := http.FileServer(http.Dir("/public"))
mux.Handle("/static/", http.StripPrefix("/static/", files))
```

```
mux.HandleFunc("/", index)
server := &http.Server{
Addr: "0.0.0.0:8080",
Handler: mux,
}
server.ListenAndServe()
}
```

En main.go, primero creamos un multiplexor, la pieza de código que redirige una solicitud a un manejador. La librería estándar net/http provee de un multiplexor por defecto que puede ser creado llamando la función NewServeMux.

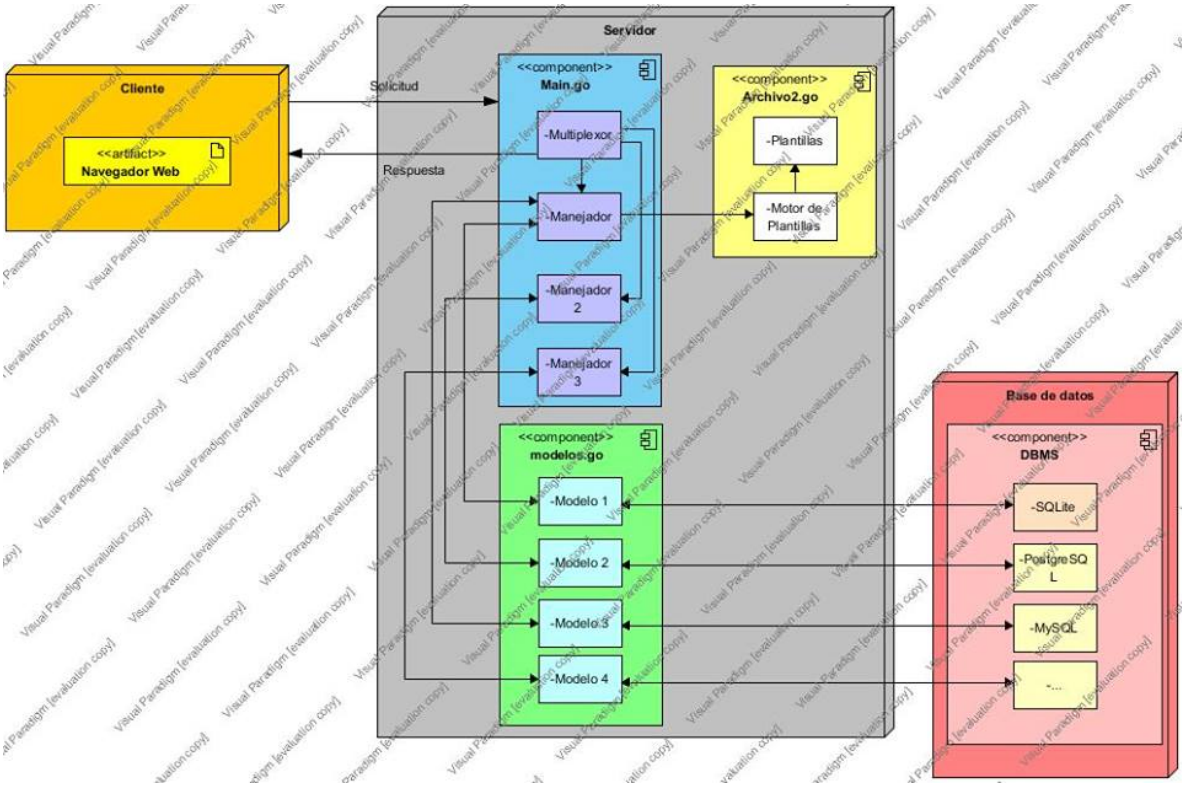
Para redirigir de la URL raíz al manejador de la función, use utiliza la función HandleFunc, HandleFunc toma el URL como su primer parámetro, y el nombre del manejador de la función como su segundo parámetro, así cuando una solicitud llega desde la URL raíz (/), es redireccionada a un manejador de función llamado index. No se necesita proveer los parámetros del manejador de la función porque todos los manejadores de función tomar ResponseWriter como su primer parámetro y un puntero a la solicitud como su segundo parámetro.

Lo siguiente podría ser resumido de la siguiente forma:

1. El manejador procesa la solicitud.
2. Cuando se necesitan datos, se usa una o más estructuras de datos que modelan la información de la base de datos.
3. Los modelos se conectan con la base de datos, disparados por funciones o métodos en la estructura de datos.
4. Cuando el procesamiento está completo, el manejador dispara el motor de la plantilla, algunas veces enviando información desde el modelo.
5. El motor de plantillas traduce archivos para crear plantillas, las cuales posteriormente son combinadas con datos para producir HTML.
6. El archivo HTML generado es enviado de vuelta al cliente como parte de la respuesta.

Arquitectura de una aplicación web en Go.

A continuación se muestra un diagrama de despliegue (deployment diagram) de una aplicación en Go.



Fuentes de información.

Donovan, A. & Kernighan, B. (2016). *The Go Programming Language.* Estados Unidos. Addison-Wesley

Doxsey, C. (2016). *Introducing Go.* Estados Unidos. O'Reilly Media

Balbaert, I. (2012). *The Way to Go.* Estados Unidos. Iuniverse.

Sheong, S. (2016). Go Web Programming. Estados Unidos. Manning Publications.

Berenguel Gómez, J. L. (2016). *Desarrollo de aplicaciones web en el entorno servidor.* España: Parainfo.

Galli, R. (2015). *Principios y algoritmos de concurrencia.* Createspace Independent Publishing Platform.

The Go Team. (15 de Agosto de 2016). *The Go Programming Language*. Obtenido de Golang website: https://golang.org/