Main Packages in Go

Go has a very simple package system. It is based on the idea of a workspace. A workspace is a directory that contains Go source files, and a src subdirectory that contains source code for Go packages. The src subdirectory contains subdirectories, each of which represents a package. The path to a package's source code is its import path. The import path determines the import path of any package that imports it. The import path of a package is the import path of its parent directory, followed by the package's subdirectory name. For example, the import path of the time package is "time", and the import path of the io/ioutil package is "io/ioutil".

The packages in the standard library are:

- fmt: for formatting input and output
- math: for mathematical operations
- net: for networking operations
- os: for working with the operating system
- time: for working with time and dates
- bytes: for working with byte slices
- strings: for working with strings
- sync: for synchronization primitives
- encoding: for encoding and decoding data
- reflect: for reflection
- testing: for testing
- context: for managing context across API boundaries

fmt package

The fmt package implements formatted I/O with functions analogous to C's printf and scanf. The format 'verbs' are derived from C's but are simpler.

The main functions in the fmt package are:

- Print: formats using the default formats for its operands and writes to standard output.
- Printf: formats according to a format specifier and writes to standard output.
- Println: formats using the default formats for its operands and writes to standard output. Spaces are always added between operands and a newline is appended.
- Scan: scans text read from standard input, storing successive space-separated values into successive arguments.

Scanf: scans text read from standard input, storing successive space-separated values into successive arguments as determined by the format.

- Scanln: scans a line of text read from standard input, storing successive space-separated values into successive arguments.

```
package main

import "fmt"

func main() {
    // print the message "Hello, World!" to the console
    fmt.Print("Hello, World!", 4)
    // print the message "Hello, World!" to the console in a new line
    fmt.Println("Hello, World!")
    // print the message "Hello, World!" to the console
    fmt.Printf("Hello, World!: %d", 4)
    // scan the input from the console
    fmt.Println("")
    var name string
    fmt.Println("Enter your name:")
    fmt.Scan(&name)
    fmt.Println(name)
    fmt.Println("Enter your name:")
    fmt.Scanf("%s", &name)
    fmt.Println(name)
    fmt.Println("Enter your name:")
    fmt.Scanln(&name)
    fmt.Println(name)
}
```

math package

The math package provides basic constants and mathematical functions.

The main functions in the math package are:

- Abs: returns the absolute value of x.
- Ceil: returns the smallest integer value greater than or equal to x.
- Cos: returns the cosine of the radian argument x.
- Exp: returns e**x, the base-e exponential of x.
- Floor: returns the largest integer value less than or equal to x.
- Max: returns the larger of x or y.
- Min: returns the smaller of x or y.
- Mod: returns the floating-point remainder of x/y.
- Pow: returns x**y, the base-x exponential of y.
-

Sin: returns the sine of the radian argument x.

- Sqrt: returns the square root of x.
- Tan: returns the tangent of the radian argument x.

```
package main

import (
    "fmt"
    "math"
)

func main() {
    fmt.Println(math.Abs(-1))
    fmt.Println(math.Ceil(1.2))
    fmt.Println(math.Cos(1))
    fmt.Println(math.Exp(1))
    fmt.Println(math.Floor(1.2))
    fmt.Println(math.Max(1, 2))
    fmt.Println(math.Min(1, 2))
    fmt.Println(math.Mod(1, 2))
    fmt.Println(math.Pow(2, 3))
    fmt.Println(math.Sin(1))
    fmt.Println(math.Sqrt(4))
    fmt.Println(math.Tan(1))
}
```

net package

The net package provides a portable interface for network I/O, including TCP/IP, UDP, domain name resolution, and Unix domain sockets.

The main functions in the net package are:

- Dial: connects to the address on the named network.
- Listen: listens for incoming connections.
- ResolveTCPAddr: resolves TCP addresses.
- ResolveUDPAddr: resolves UDP addresses.
- ResolveUnixAddr: resolves Unix domain socket addresses.
- ResolveIPAddr: resolves IP addresses.

```
package main

import (
    "fmt"
    "net"
)

func main() {
    fmt.Println(net.Dial("tcp", "google.com:80"))
    fmt.Println(net.Listen("tcp", ":8080"))
    fmt.Println(net.ResolveTCPAddr("tcp", "google.com:80"))
    fmt.Println(net.ResolveUDPAddr("udp", "google.com:80"))
    fmt.Println(net.ResolveUnixAddr("unix", "google.com:80"))
    fmt.Println(net.ResolveIPAddr("ip", "google.com:80"))
}
```

os package

The os package provides a platform-independent interface to operating system functionality.

The main functions in the os package are:

- Exit: exits the current program with the given status code.
- Getenv: retrieves the value of the environment variable named by the key.
- Getwd: returns a rooted path name corresponding to the current directory.
- Hostname: returns the host name reported by the kernel.
- IsExist: reports whether the error is known to report that a file or directory already exists.
- IsNotExist: reports whether the error is known to report that a file or directory does not exist.
- IsPathSeparator: reports whether c is a directory separator character.
- Mkdir: creates a new directory with the specified name and permission bits.
- MkdirAll: creates a directory named path, along with any necessary parents, and returns nil, or else returns an error.
- Remove: removes the named file or (empty) directory.
- RemoveAll: removes path and any children it contains.
- Rename: renames (moves) oldpath to newpath.
- Setenv: sets the value of the environment variable named by the key.
- Unsetenv: deletes the named environment variable.

```
package main

import (
    "fmt"
    "os"
)

func main() {
    fmt.Println(os.Exit(1))
    fmt.Println(os.Getenv("PATH"))
    fmt.Println(os.Getwd())
    fmt.Println(os.Hostname())
    fmt.Println(os.IsExist(os.ErrExist))
    fmt.Println(os.IsNotExist(os.ErrNotExist))
    fmt.Println(os.IsPathSeparator(os.PathSeparator))
    fmt.Println(os.Mkdir("test", 0777))
    fmt.Println(os.MkdirAll("test/test", 0777))
    fmt.Println(os.Remove("test"))
    fmt.Println(os.RemoveAll("test"))
    fmt.Println(os.Rename("test", "test2"))
    fmt.Println(os.Setenv("PATH", "test"))
    fmt.Println(os.Unsetenv("PATH"))
}
```

strings package

The strings package contains many useful string-related functions.

The main functions in the strings package are:

- Contains: reports whether substr is within s.
- Count: counts the number of non-overlapping instances of substr in s.
- Fields: splits the string s around each instance of one or more consecutive white space characters, as defined by unicode.IsSpace, returning a slice of substrings of s or an empty slice if s contains only white space.
- Index: returns the index of the first instance of substr in s, or -1 if substr is not present in s.
- Join: concatenates the elements of a to create a single string. The separator string sep is placed between elements in the resulting string.
- Map: returns a copy of the string s with all its characters modified according to the mapping function.
- Repeat: returns a new string consisting of count copies of the string s.
- Replace: returns a copy of the string s with the first n non-overlapping instances of old replaced by new. If old is empty, it matches at the beginning of the string and after each UTF-8 sequence, yielding up to k+1 replacements for a k-rune string. If n < 0, there is no limit on the number of replacements.
- Split: slices s into all substrings separated by sep and returns a slice of the substrings between those separators.
-

ToLower: returns a copy of the string s with all Unicode letters mapped to their lower case.

- ToTitle: returns a copy of the string s with all Unicode letters mapped to their title case.
- ToUpper: returns a copy of the string s with all Unicode letters mapped to their upper case.
- Trim: returns a slice of the string s with all leading and trailing Unicode code points contained in cutset removed.
- TrimSpace: returns a slice of the string s, with all leading and trailing white space removed, as defined by Unicode.

```
package main

import (
    "fmt"
    "strings"
)

func main() {
    fmt.Println(strings.Contains("test", "es"))
    fmt.Println(strings.Count("test", "t"))
    fmt.Println(strings.Fields("test test"))
    fmt.Println(strings.Index("test", "e"))
    fmt.Println(strings.Join([]string{"test", "test"}, "-"))
    fmt.Println(strings.Map(func(r rune) rune { return r + 1 }, "test"))
    fmt.Println(strings.Repeat("test", 2))
    fmt.Println(strings.Replace("test", "t", "T", 1))
    fmt.Println(strings.Split("test test", " "))
    fmt.Println(strings.ToLower("TEST"))
    fmt.Println(strings.ToTitle("test"))
    fmt.Println(strings.ToUpper("test"))
    fmt.Println(strings.Trim("!!!test!!!", "!"))
    fmt.Println(strings.TrimSpace(" test "))
}
```

time package

The time package provides functionality for measuring and displaying time.

The main functions in the time package are:

- After: returns a channel that will send the time at the specified duration after the current time.
- AfterFunc: waits for the duration to elapse and then calls f in its own goroutine.
- Date: returns the year, month, and day in which t occurs.
- Format: returns a textual representation of the time value formatted according to layout, which defines the format by showing how the reference time, defined to be Mon Jan 2 15:04:05 MST 2006, would be displayed if it were the value; it serves as an example of the desired output. The same display rules will then be applied to the time value.
- Now: returns the current local time.
-

Parse: parses a formatted string and returns the time value it represents.

- Sleep: pauses the current goroutine for at least the duration d.
- Tick: returns a channel that will send the time with a period specified by the duration argument.
- Unix: returns the local Time corresponding to the given Unix time, sec seconds and nsec nanoseconds since January 1, 1970 UTC. It is valid to pass nsec outside the range [0, 999999999].

```
package main

import (
    "fmt"
    "time"
)

func main() {
    fmt.Println(time.After(1 * time.Second))
    fmt.Println(time.AfterFunc(1*time.Second, func() { fmt.Println("test") }))
    fmt.Println(time.Date(2012, 12, 21, 12, 0, 0, 0, time.UTC))
    fmt.Println(time.Now())
    fmt.Println(time.Parse("2006-01-02", "2012-12-21"))
    fmt.Println(time.Sleep(1 * time.Second))
    fmt.Println(time.Tick(1 * time.Second))
    fmt.Println(time.Unix(1355971200, 0))
}
```

sync package

The sync package provides basic synchronization primitives such as mutual exclusion locks.

The main functions in the sync package are:

- Mutex: a mutual exclusion lock.
- Once: an object that will perform exactly one action.
- WaitGroup: a counter for the number of goroutines that must execute before a continuation can be executed.

```
package main

import (
    "fmt"
    "sync"
)

func main() {
    var mutex sync.Mutex
    mutex.Lock()
    mutex.Unlock()

    var once sync.Once
    once.Do(func() { fmt.Println("test") })

    var wg sync.WaitGroup
    wg.Add(1)
    wg.Done()
    wg.Wait()
}
```