FEB 2022

I find it unusual that an elegant mechanism for a cross-platform shared library code between Android and iOS is possible in the very powerful language of Go, and yet, such mechanism is little documented to the point of being almost obscure and unrecognized. Having previously done a dynamic library in C/C++ that is usable on different platforms Android ARM, Linux, Windows and later after much wrestling on a MacOS, the smooth tooling provided by Go to do the same thing is a breath of fresh air.

The `fyne.io` mobile platform produces a whole `*.apk` application, with its own UI widget system. Hence, it cannot interop well with existing Android mobile apps.

In order for Go to interop with existing Android mobile projects, it must produce either of the following:

- A shared `*.so` library and a header file
- An Android `*.aar` library

The Go technologies that is used here are:

- *cgo* - Works with Android NDK toolchain to convert `file.go` to `libfile.so` and associated `libfile.h` header file.
- *gomobile*, *gobind* - A super tool on top of `cgo` to produce `app.aar` from `app.go`. It can also target *iOS*.

Go is now using a package management file `go.mod` which is like `package.json` in *nodejs*. Here are the overall steps on how a Go language function can be used in an Android application.

Set needed environment variables:

```
export ANDROID_NDK_HOME=/home/dx/Android/Sdk/ndk/23.1.7779620
export ANDROID_HOME=/home/dx/Android/Sdk

cd $HOME
mkdir downloads && cd downloads
wget https://go.dev/dl/go1.17.7.linux-amd64.tar.gz
tar xvzpf go1.17.7.linux-amd64.tar.gz
export GOPATH=$HOME/downloads/go
export PATH=$GOPATH/bin:$PATH

cd $HOME
go version

mkdir testproj && testproj
```

Inside `testproj/` folder, only two files are needed to produce `testlib.aar` Android library.

testlib.go (*this is the Go object that we need to use in Android Java)*

```
package testlib

type Counter struct {
    Value int
}

func (c *Counter) Inc() {
    c.Value++
}

func NewCounter() *Counter {
    return &Counter{ 5 }
}
```

go.mod

```
module testlib
go 1.17
```

Now that Go and the necessary environment variables are setup, it is time to produce the `testlib.aar`. First, install globally the Go mobile tools:

```
go install golang.org/x/mobile/cmd/gomobile@latest golang.org/x/mobile
/cmd/gobind@latest
```

The *gomobile* tools is a higher-level tool that uses *cgo* under the hood.

Then inside `testproj` folder:

```
go get -d golang.org/x/mobile/cmd/gobind
gomobile bind -v -o testlib.aar -target=android .
```

A `testlib.aar` Android library is produced. Copy `testlib.aar` into your Android project's `app/libs/` folder.

In your Android project's `app/build.gradle` add under *dependencies*:

```
dependencies {
    ....
    implementation files('libs/testlib.aar')
}
```

Here, a little Go language familiarity is needed. So far, all we know from `testlib.aar` is the `testlib.go` file. In your `MainActivity.java` , nevertheless try to instantiate the object:

```
Counter c = new Counter();
counter.setValue(99);
counter.inc();
long v= counter.getValue();
System.out.println(v);
```

If you unzip the `*.aar`, then you'll see it bundled `libgojni.so` and JNI-stitched it together with `classes.jar`. However, if you had used the lower-level `cgo` (not using the gomobile tool) method, the `testlib.go` gets fully compiled into `libtestlib.so` completely.