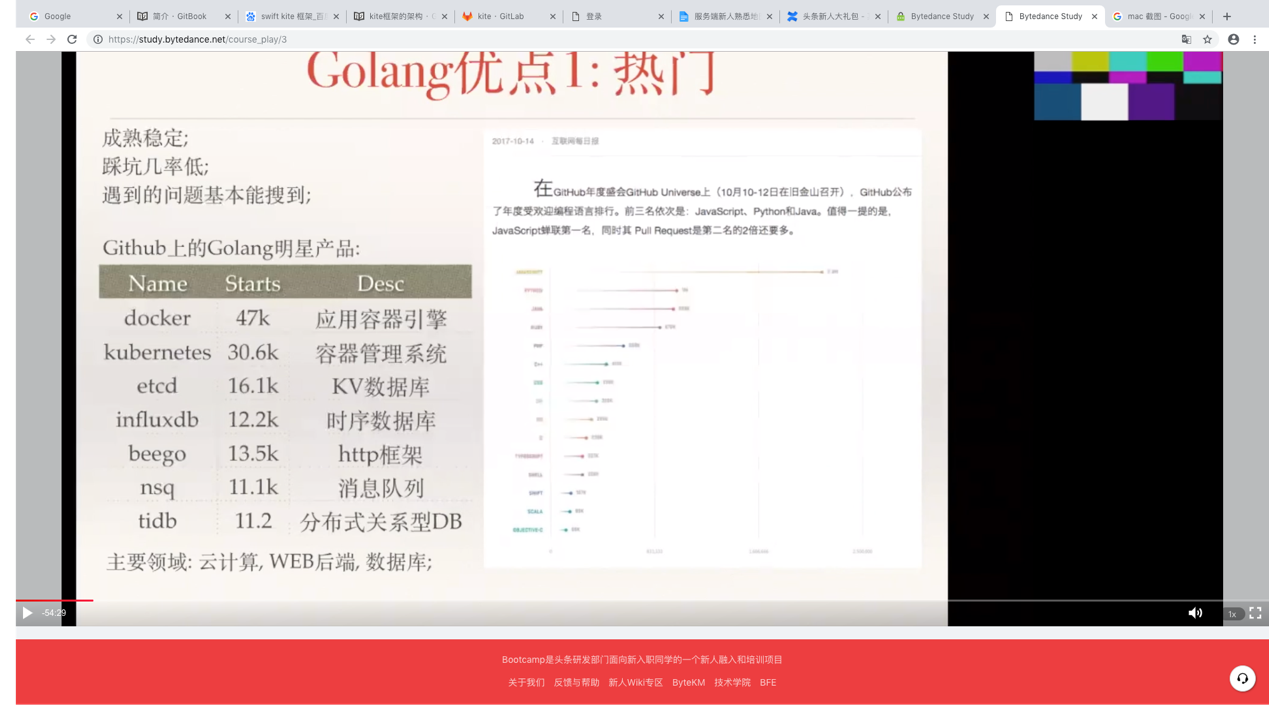
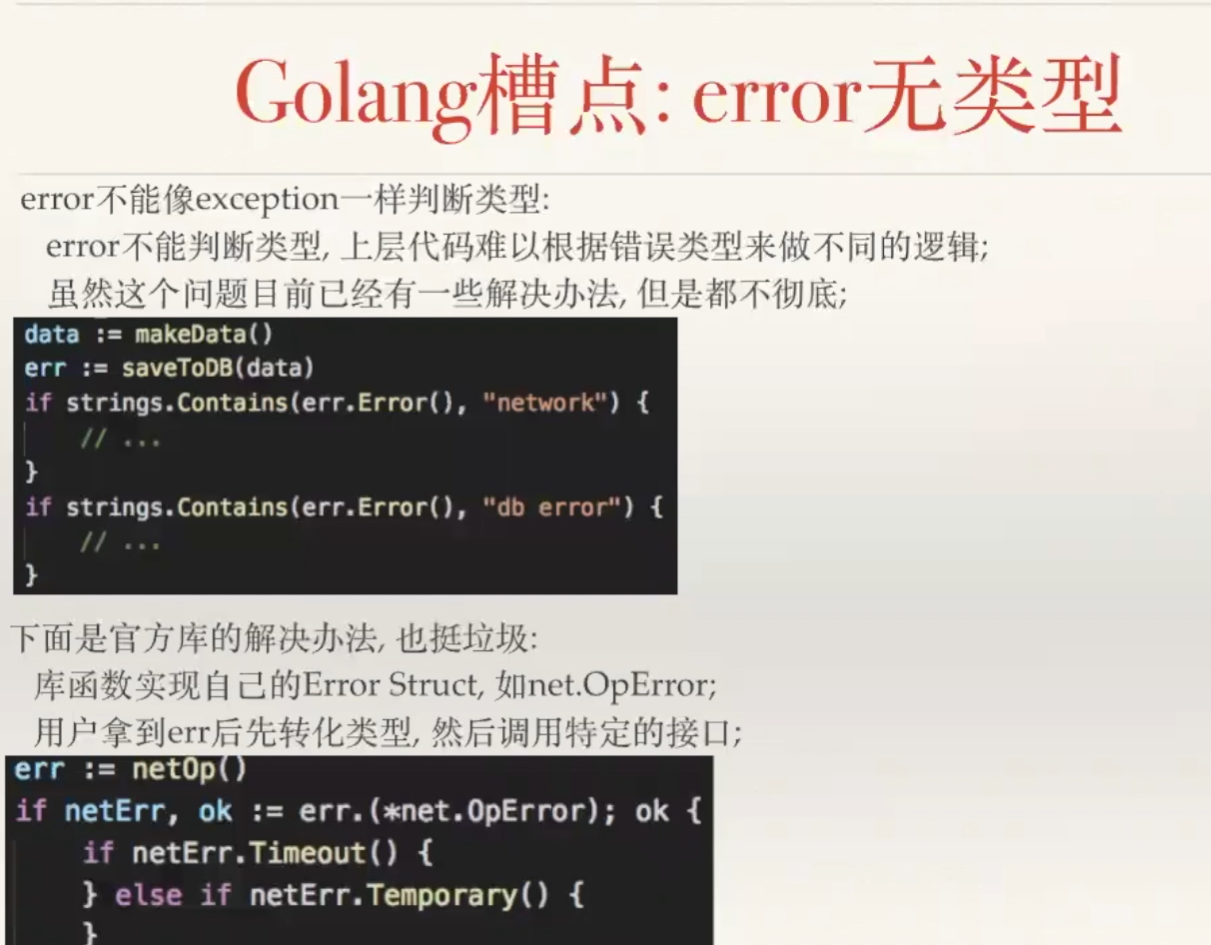
Go（Golang）是Google开发的一种编译型，可并行化，并具有垃圾回收功能的编程语言，所以只要有引用就不会被回收，这对一些写法会比较方便。

Golang 最能打动的是服务器底层开发者。这一部分人群真心不多，自然看起来 Golang 就没那么“火”了。为什么最能打动服务器底层开发者？原因很多，只有尝试才有深刻体会。使用 Golang 开发一个网络底层库会轻松舒服太多，相比之下，用 C/C++ 就太可怕了。goroutine 、channel也是 Golang 的一个特色。我记得协程消耗的资源更加小了，当时好像是一种到用户空间线程的映射，对于CPU密集型应用肯定没有直接线程好。



error：

go没有异常，统一使用包含字符串的error来表示函数出错（函数可以返回两个变量，通常第二个变量是error）。在写法上需要注意，重新抛出的时候，要带上原有err中的内容。

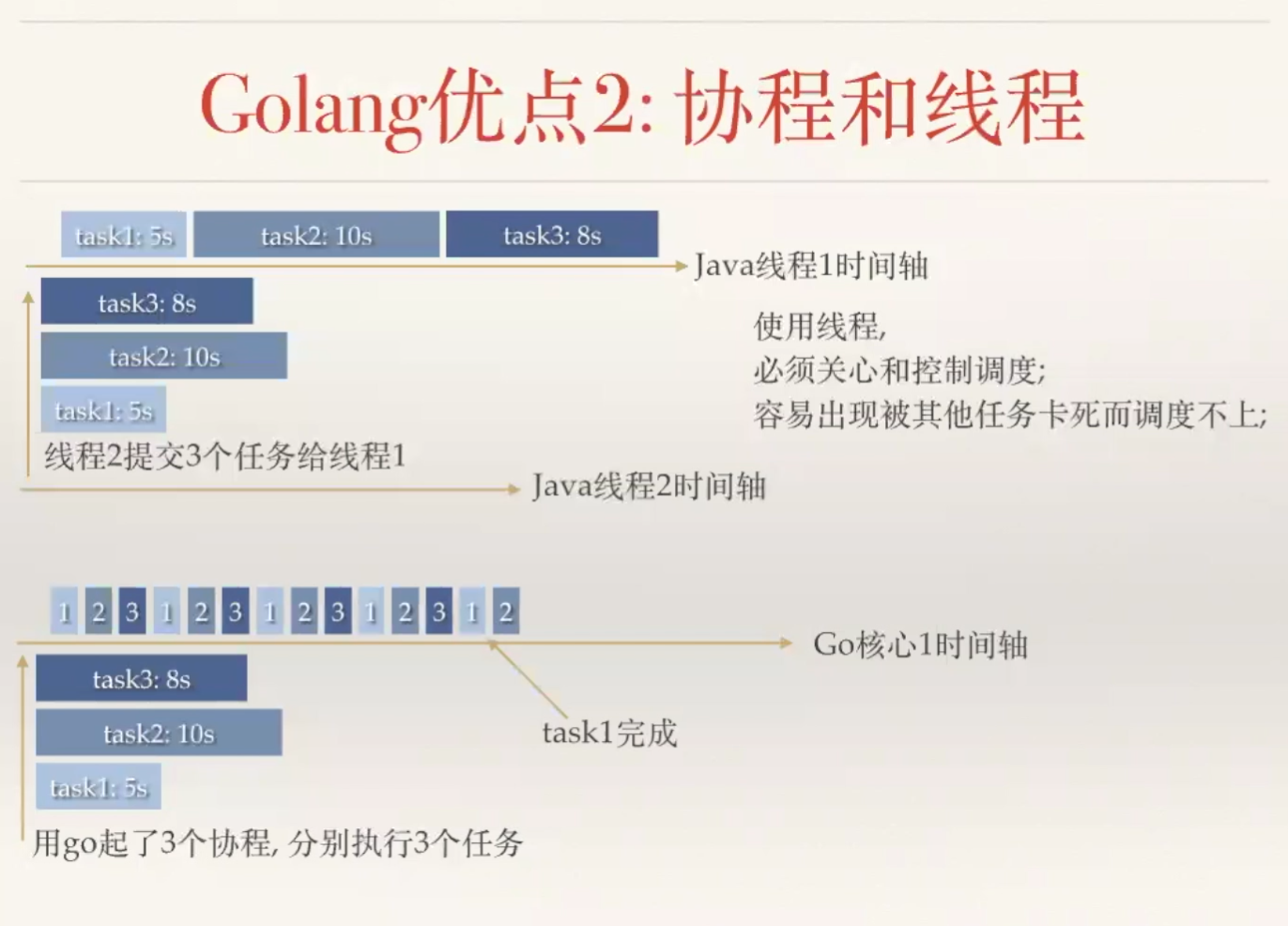


出线对error中内容判断来区分error类型，说明·代码结构上稍微有点问题了。

goroutine：

在用户空间存在一个调度器，一旦进行go库函数的调用，runtime就有机会进行调度。

如果编写了一个单纯的死循环会持续占用掉一个核心。设置占用的核心数：runtime.GOMAXPROCS(number)



上面关于Java线程的图形是错误的，没有考虑到CPU的调度。

context：

go 1.8 引入context，避免过深的函数调用

vendor：govendor godep

将所有的外部依赖打包到项目的vendor目录，编译时路径优先级还高于$GOPATH/src，但是vendor中没有版本概念，如果出现不同的库依赖于同一个库但是不向前兼容的两个版本，会GG。

govendor init

govendor add +e(相当于+external，所有外部依赖)

govendor update +v(相当于+vendor)

+all 会包含所以依赖，包含了当前GO SDK中的库

beego是一个快速开发Go应用的http MVC框架，RESTful风格编程，这意味着前后端都包括了。

package main

import "fmt"

func appendChange(buf []int) {

    buf = append(buf, 0)

    buf = append(buf, 0)

    buf[0] = 1

    fmt.Printf("%p: %v\n", buf, buf)

}

func main() {

    buffer := make([]int, 0, 2)

    buffer = append(buffer, 0)

    buffer = append(buffer, 0)

    appendChange(buffer)

    fmt.Printf("%p: %v\n", buffer, buffer)

}

lipeinandeMacBook-Pro:test lipeinan$ go run main.go

0xc0000120e0: [1 0 0 0]

0xc000014070: [0 0]

因为原先限制大小为2，所以超过之后就会被append重新创建。