今日交流粗略梳理

1. context.WithValue 的取值时间复杂度是 O(n) 的, 效率低. 骚操作是自定义实现 context.Context 并重写 context.Value().
2. 主流意见: context.Context 主要用途是取消 goroutines. Context.WithValue 会让人误用.

```
// 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.
func WithValue(parent Context, key, val any) Context {
```

误用 case1 : context.Value 实现是倒排多叉树链表结构, 所以 context 中的上下文数据并不是全局的, 他只查询本结点及父结点们的数据, 不能查询兄弟结点的数据.

误用 case2 : context.WithValue Key 不推荐使用基础导出类型. 主要原因是防止不同包之间误用.

```
package main

import "fmt"

// go run package.go
func main() {
        type Key1 int
        type Key2 int

        var key1 interface{} = Key1(1)
        var key2 interface{} = Key2(1)

        // main.Key1:1 != main.Key2:1
        if key1 == key2 {
                fmt.Printf("%T:%v == %T:%v\n", key1, key1, key2, key2)
        } else {
                fmt.Printf("%T:%v != %T:%v\n", key1, key1, key2, key2)
        }
}
```

更多详细阅读 proposal: Go 2: update context package for Go 2 #28342 提案. 其中有非常多讨论, goroutine-local storage 和 cancel 相关功能分开, 不一定强相关绑定在一起 proposal: Go2: rename "context" and (maybe) trim functionality #27987.

精彩讨论提案非常的多 proposal: Replace Context with goroutine-local storage #21355, 其中有个提议小插曲非常有意思, 讲 context 变为内置类型.

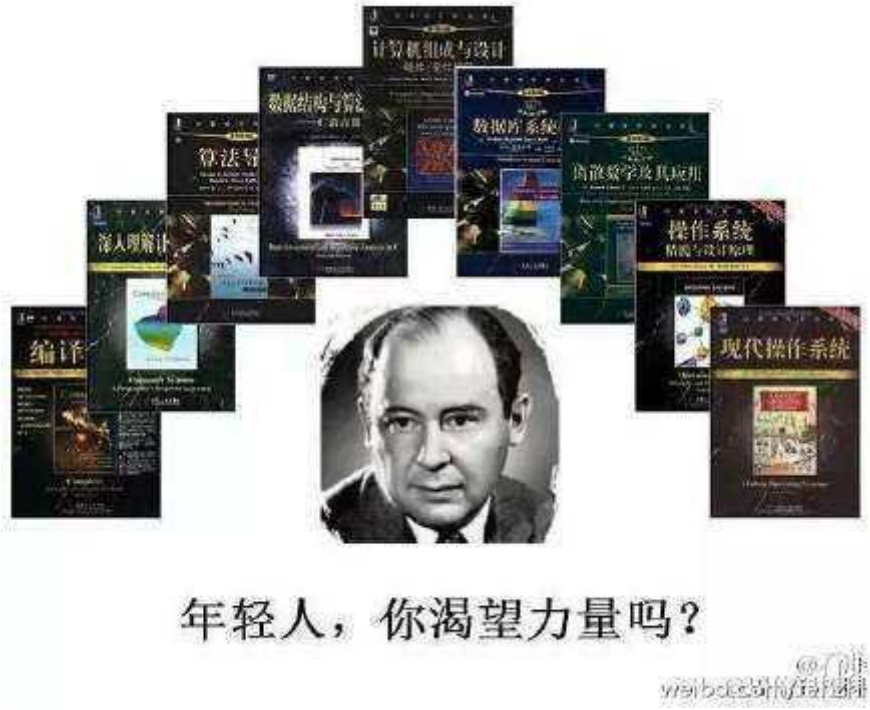
```
-func Foo(ctx context.Context) error {
+func Foo(ctx context) error {
```

非常精彩, 推荐同行收藏递归看完所有的交流. 目前粗读, 事后再细读.

3.Go context.Context 工程使用请阅读, 官方 blog Go Concurrency Patterns: Context

4.神父有话说 Context should go away for Go 2

5.如何提高基础我知道，但「快速」这个有难度



6. Go 有哪些劣势?
7. go doc -src context

```
package context // import "context"

Package context defines the Context type, which carries deadlines,
cancellation signals, and other request-scoped values across API boundaries
and between processes.

Incoming requests to a server should create a Context, and outgoing calls to
servers should accept a Context. The chain of function calls between them
must propagate the Context, optionally replacing it with a derived Context
created using WithCancel, WithDeadline, WithTimeout, or WithValue. When a
Context is canceled, all Contexts derived from it are also canceled.

The WithCancel, WithDeadline, and WithTimeout functions take a Context (the
parent) and return a derived Context (the child) and a CancelFunc. Calling
the CancelFunc cancels the child and its children, removes the parent's
reference to the child, and stops any associated timers. Failing to call the
CancelFunc leaks the child and its children until the parent is canceled or
the timer fires. The go vet tool checks that CancelFuncs are used on all
control-flow paths.

Programs that use Contexts should follow these rules to keep interfaces
consistent across packages and enable static analysis tools to check context
propagation:

Do not store Contexts inside a struct type; instead, pass a Context
explicitly to each function that needs it. The Context should be the first
parameter, typically named ctx:

    func DoSomething(ctx context.Context, arg Arg) error {
        // ... use ctx ...
    }

Do not pass a nil Context, even if a function permits it. Pass context.TODO
if you are unsure about which Context to use.

Use context Values only for request-scoped data that transits processes and
APIs, not for passing optional parameters to functions.

The same Context may be passed to functions running in different goroutines;
Contexts are safe for simultaneous use by multiple goroutines.

See https://blog.golang.org/context for example code for a server that uses
Contexts.

var Canceled = errors.New("context canceled")
var DeadlineExceeded error = deadlineExceededError{}
func WithCancel(parent Context) (ctx Context, cancel CancelFunc)
func WithDeadline(parent Context, d time.Time) (Context, CancelFunc)
func WithTimeout(parent Context, timeout time.Duration) (Context, CancelFunc)
type CancelFunc func()
```

```
type Context interface{ ... }
    func Background() Context
    func TODO() Context
    func WithValue(parent Context, key, val any) Context
```

8. 目前 context 有个非常重要点, 在 Go 只有阻塞式调用没有非阻塞调用，所以必须要有 context cancel (or Done() or Err()) 相关操作，否则操作一旦陷进去没响应 (本质是 goroutines 调度控制), goroutine 就无法继续了.