Atitit golan 异常处理 以及与java php js的对比

目录

[1. Java php js tsql等的异常处理 1](#_Toc9062)

[1.1. Try catch 模式 代码块作用域 1](#_Toc7864)

[2. golan 异常处理 和mysql对异常处理很类似 1](#_Toc16403)

[2.1. defer 函数要定义在 panic前 1](#_Toc27777)

[2.2.  Defer 函数收尾方法 1](#_Toc5733)

[2.3. 调用时机 向 defer 关键字传入的函数会在函数返回之前运行 2](#_Toc10524)

[2.4. 函数块作用域 非代码块 2](#_Toc9724)

[2.5. 多次调用 defer 时执行顺序 倒序执行多次 3](#_Toc7615)

[2.6. 得到异常信息recover() 3](#_Toc769)

[2.7. 循环中defer，封装到匿名函数中即可 4](#_Toc26253)

# Java php js tsql等的异常处理

## Try catch 模式 代码块作用域

# golan 异常处理 和mysql对异常处理很类似

## **defer 函数要定义在 panic前**

**后面，该函数在 panic 后就无法被执行到**

## Defer 函数收尾方法

很多现代的编程语言中都有 defer 关键字，Go 语言的 defer 会在当前函数或者方法返回之前执行传入的函数。它会经常被用于关闭文件描述符、关闭数据库连接以及解锁资源

使用 defer 的最常见场景就是在函数调用结束后完成一些收尾工作，例如在 defer 中回滚数据库的事务：

**func** insertOne(db \*sql.DB, i int) {

// 处理异常的函数 same befFunExit()

**defer** **func**() {

fmt.Println("开始处理异常")

// 获取异常信息

**if** err := recover(); err != nil {

// 输出异常信息

fmt.Println("error:", err)

}

fmt.Println("结束异常处理")

}()

res, \_ := db.Exec("create database db" + strconv.Itoa(i))

// CheckErr(err)

//查询删除多少条信息

// if(res)

// {

num, \_ := res.RowsAffected()

// CheckErr(err)

fmt.Println(num) //1 if creted database ok ..

// }

}

## 调用时机 向 defer 关键字传入的函数会在函数返回之前运行

## 函数块作用域 非代码块

，defer 传入的函数不是在退出代码块的作用域时执行的，它只会在当前函数和方法返回之前被调用。

## 多次调用 defer 时执行顺序 倒序执行多次

会倒序执行所有向 defer 关键字中传入的表达式，最后一次 defer 调用传入了 fmt.Println(4)，所以会这段代码会优先打印 4

defer语句：****当Golang的代码执行时，如果遇到defer语句，则压入堆栈，当函数返回时，会按照后进先出的顺序调用defer语句****。

## 得到异常信息recover()

为什么可以这么写？

这是因为 panic 的函数签名显示它可以接收 interface{} 类型，我们可以将它理解为 Go 中的 "任意类型"

这是 panic 的签名

func panic(v interface{})

recover 的签名

func recover() interface{}

因此，基本上它会这样运行

panic(value) -> recover() -> value

recover 会把传入 panic 的值返回出来

recover都是在当前的goroutine里进行捕获的，这就是说，对于创建goroutine的外层函数，如果goroutine内部发生panic并且内部没有用recover，外层函数是无法用recover来捕获的，这样会造成程序崩溃

## 循环中defer，封装到匿名函数中即可

//准备插入操作

//

**for** i := 0; i <= 10; i++ {

**func**() {

// 处理异常的函数 same befFunExit()

**defer** **func**() {

fmt.Println("开始处理异常")

// 获取异常信息

**if** err := recover(); err != nil {

fmt.Println("error:", err)

}

fmt.Println("结束异常处理")

}()

// sql := "drop database db" + strconv.Itoa(i)

sql := "create database db" + strconv.Itoa(i)

res, \_ := db.Exec(sql)

num, \_ := res.RowsAffected()

fmt.Println(num) //1 if creted database ok ..

}()

}