Go语言 数据操作 Memcached

Memcached 介绍

Memcached是一个自由开源的，高性能，分布式内存对象缓存系统。
基于内存的key-value存储，用来存储小块的任意数据（字符串、对象）。这些数据可以是数据库调用、API调用或者是页面渲染。
一般的使用目的是，通过缓存数据库查询结果，减少数据库访问次数，以提高动态Web应用的速度、提高可扩展性
Memcached 官网：https://memcached.org/
github地址：https://github.com/memcached/memcached/wiki/Install

Go 操作 Memcached

Go使用memcached需要第三方的驱动库，这里有一个库是memcached作者亲自实现的，代码质量效率肯定会有保障。

安装：

```
go get github.com/bradfitz/gomemcache/memcache
```

使用：

```
import "github.com/bradfitz/gomemcache/memcache"
```

示例：

```
package main

import (
    "fmt"

    "github.com/bradfitz/gomemcache/memcache"
)

// memcached server地址
var (
    server = "127.0.0.1:11211"
)

func main() {
    // 创建memcached实例
    m := memcache.New(server)
    if m == nil {
        fmt.Printf("memcache 实例化失败!")
    }

    //set 操作
    err := m.Set(&memcache.Item{Key: "name", Value: []byte("包子")})
    if err != nil {
```

```
    fmt.Printf("写入memcached失败, %s", err)
}

// get 操作
it, _ := m.Get("name")
if string(it.Key) == "name" {
    fmt.Println("值：", string(it.Value))
} else {
    fmt.Println("获取失败")
}

// add 操作
m.Add(&memcache.Item{Key: "food", Value: []byte("米饭")})
i, err2 := m.Get("food")
if err2 != nil {
    fmt.Println("添加失败")
} else {
    if string(i.Key) == "food" {
        fmt.Println("添加的值：", string(i.Value))
    } else {
        fmt.Println("获取失败")
    }
}

// replace 操作
m.Replace(&memcache.Item{Key: "food", Value: []byte("花卷")})
i2, err3 := m.Get("food")
if err3 != nil {
    fmt.Println("替换失败")
}
if string(i2.Key) == "food" {
    fmt.Println("替换的值是 : ", string(i2.Value))
} else {
    fmt.Println("获取失败")
}

// delete 操作
err4 := m.Delete("food")
if err4 != nil {
    fmt.Println("删除失败")
} else {
    fmt.Println("删除成功")
}

// increment 操作
err5 := m.Set(&memcache.Item{Key: "number", Value: []byte("1")})
if err5 != nil {
    fmt.Printf("设置失败, %s", err)
}
newValue, err6 := m.Increment("number", 5)
if err6 != nil {
    fmt.Printf("加值失败, %s", err)
} else {
    fmt.Println("新值是：", newValue)
}
```

```
    // decrement 操作
    err7 := m.Set(&memcache.Item{Key: "number2", Value: []byte("5")})
    if err7 != nil {
        fmt.Printf("设置失败, %s", err)
    }
    newValue2, err8 := m.Decrement("number2", 3)
    if err8 != nil {
        fmt.Printf("减值失败, %s", err)
    } else {
        fmt.Println("新的值是：", newValue2)
    }
}
```