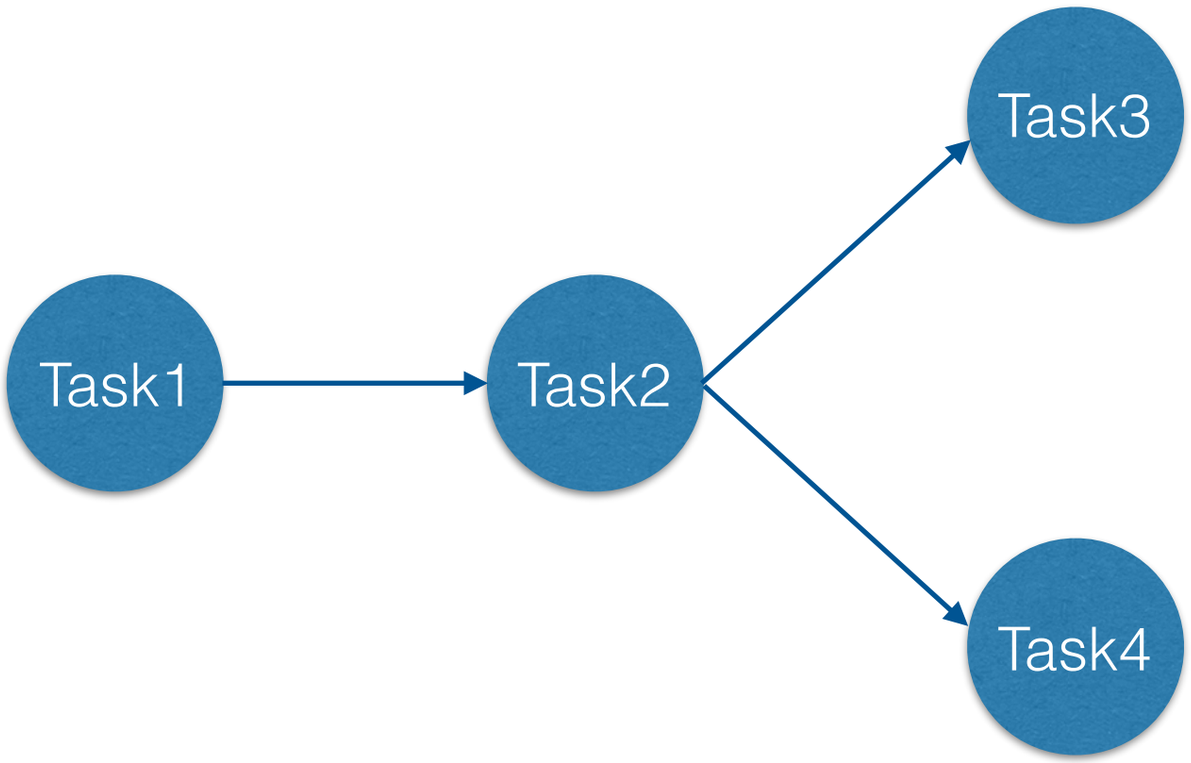
使用领域事件来解耦多任务间的依赖

中兴通讯虚拟化架构师　张晓龙

内容大纲

- 问题域
- 解决方案
- 代码示例

问题域



- 实例化问题域：Ops域性能汇聚
  - Task1: 5minute汇聚任务
  - Task2: 1day汇聚任务
  - Task3: 1week汇聚任务
  - Task4: 1month汇聚任务

内容大纲

- 问题域
- **解决方案**
- 代码示例

原解决方案

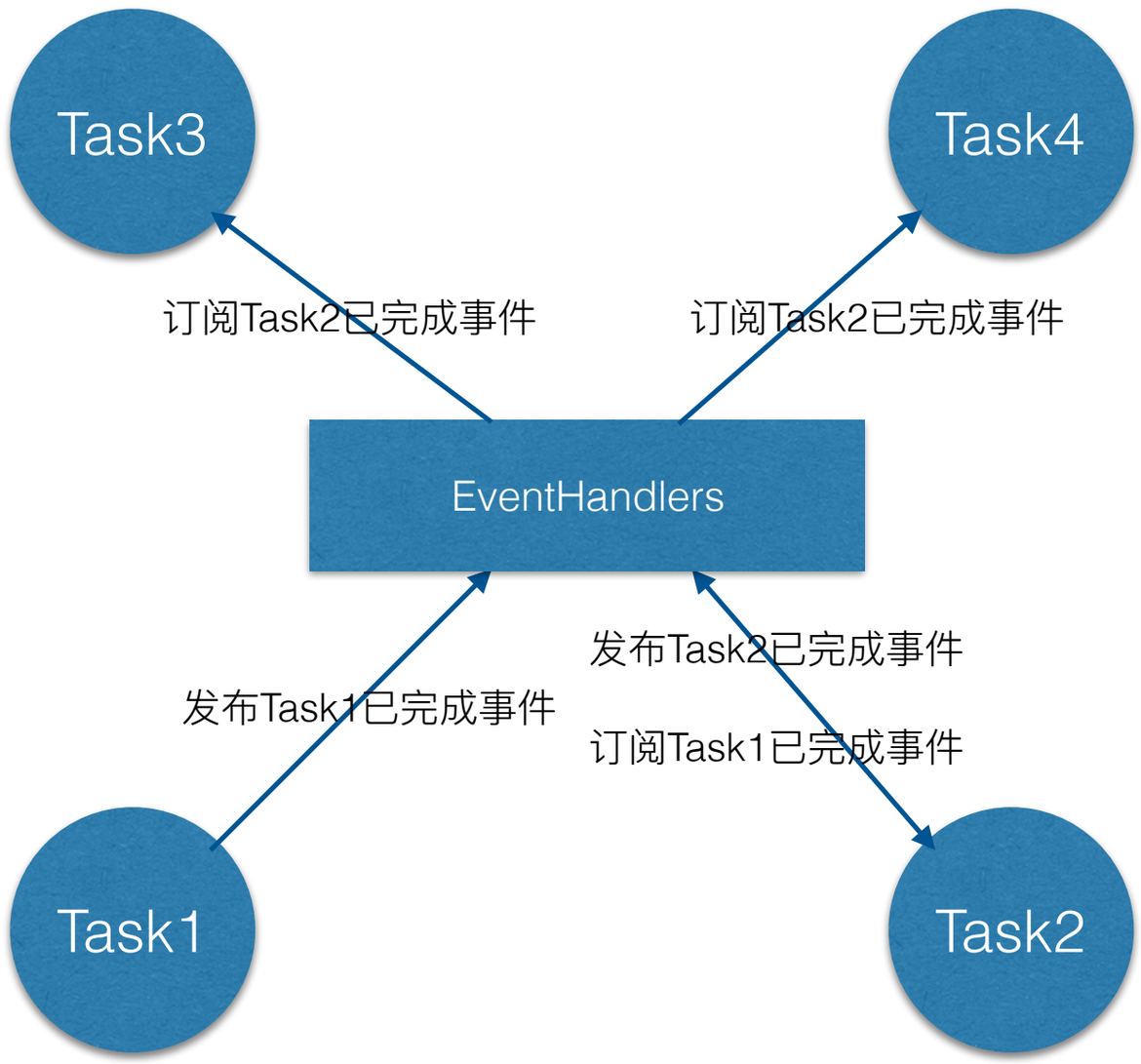
- 方案：下游任务增加偏移时间Δt
  - 优点：简单
  - 缺点：时序关系可能失控，即如果上游任务的某一次执行时间超过了Δt，则下游任务的汇聚就不正确

领域事件介绍

- 定义
  - 领域专家所关心的发生在领域中的一些事件
  - 通用语言
  - 通常用于维护系统的一致性

新解决方案

- 方案：使用领域事件
  - 思路：上游任务发布领域事件，下游任务订阅领域事件
  - 优点：多任务间没有耦合
  - 缺点：引入了第三方，增加了复杂度



内容大纲

- 问题域
- 解决方案
- **代码示例**

领域事件定义示例

```go
package task

import (
    "fmt"
    "domain/event-handlers"
)

const (
    task1HasCompleted eventhandlers.Event = "Task1 has completed"
    task2HasCompleted eventhandlers.Event = "Task2 has completed"
)
```

领域事件订阅示例

```go
type Task3 struct {

}

func (t *Task3) Exec() {
    task3Handler := new(Task3Handler)
    ehs := eventhandlers.GetInstance()
    ehs.Sub(task2HasCompleted, task3Handler)
    fmt.Println("task3 sub task2HasCompleted")
}

type Task3Handler struct {

}

func (t *Task3Handler) Handle() {
    fmt.Println("task3 handler start")
    time.Sleep(200 * time.Millisecond)
    fmt.Println("task3 handler end")
}
```

```go
type Task4 struct {

}

func (t *Task4) Exec() {
    task4Handler := new(Task4Handler)
    ehs := eventhandlers.GetInstance()
    ehs.Sub(task2HasCompleted, task4Handler)
    fmt.Println("task4 sub task2HasCompleted")
}

type Task4Handler struct {

}

func (t *Task4Handler) Handle() {
    fmt.Println("task4 handler start")
    time.Sleep(500 * time.Millisecond)
    fmt.Println("task4 handler end")
}
```

领域事件发布示例

```go
type Task1 struct {

}

func (t *Task1) Exec() {
    task1Handler := new(Task1Handler)
    task1Handler.Handle()
}

type Task1Handler struct {

}

func (t *Task1Handler) Handle() {
    fmt.Println("task1 handler start")
    time.Sleep(50 * time.Millisecond)
    fmt.Println("task1 handler end")
    ehs := eventhandlers.GetInstance()
    ehs.Pub(task1HasCompleted)
    fmt.Println("task1 pub task1HasCompleted")
}
```

领域事件订阅发布示例

```go
func (t *Task2) Exec() {
    task2Handler := new(Task2Handler)
    ehs := eventhandlers.GetInstance()
    ehs.Sub(task1HasCompleted, task2Handler)
    fmt.Println("task2 sub task1HasCompleted")
}

type Task2Handler struct {

}

func (t *Task2Handler) Handle() {
    fmt.Println("task2 handler start")
    time.Sleep(100 * time.Millisecond)
    fmt.Println("task2 handler end")
    ehs := eventhandlers.GetInstance()
    ehs.Pub(task2HasCompleted)
    fmt.Println("task2 pub task2HasCompleted")
}
```

EventHandlers示例

```go
package eventhandlers

import (
    "sync"
)

type Event string

type Handler interface {
    Handle()
}

type EventHandlers struct {
    ehsMap map[Event][]Handler
    lock sync.RWMutex
}
```

```go
var inst *EventHandlers
var once sync.Once
func GetInstance() *EventHandlers {
    once.Do(func() {
        inst = &EventHandlers{ehsMap: make(map[Event][]Handler)}
    })
    return inst
}

func (e *EventHandlers) Pub(event Event) {
    e.lock.RLock()
    defer e.lock.RUnlock()
    if handlers, ok := e.ehsMap[event]; ok {
        for _, handler := range handlers {
            go handler.Handle()
        }
    }

}

func (e *EventHandlers) Sub(event Event, handler Handler) {
    e.lock.Lock()
    defer e.lock.Unlock()
    e.ehsMap[event] = append(e.ehsMap[event], handler)
}
```

运行示例

```go
package main

import "domain/task"

func main() {
    task1 := new(task.Task1)
    go task1.Exec()
    task2 := new(task.Task2)
    go task2.Exec()
    task3 := new(task.Task3)
    go task3.Exec()
    task4 := new(task.Task4)
    go task4.Exec()
    time.Sleep(time.Second)
}
```

```
$go run main.go
task3 sub task2HasCompleted
task2 sub task1HasCompleted
task1 handler start
task4 sub task2HasCompleted
task1 handler end
task1 pub task1HasCompleted
task2 handler start
task2 handler end
task2 pub task2HasCompleted
task4 handler start
task3 handler start
task3 handler end
task4 handler end
```

小结

- 问题域
  - 多任务间时序依赖
  - 服务内
- 解决方案
  - 原解决方案：使用Δt
  - 新解决方案：使用Domain Events
- 代码示例
  - Task代码
  - EventHandlers代码

Q

A

谢谢大家！

- 张晓龙 @ 中兴通讯
- 架构师，技术教练，DDD实践布道者
- zhangxiaolong1980@126.com
- 简书个人主页：http://www.jianshu.com/u/1381dc29fed9