iOS 并发编程

GCD & Operation

串行 vs. 并行　同步 vs. 异步

- 如何用GCD创立串行和并行队列?
- 如何用OperationQueue创立串行和并行队列?
- Playground是否运行在主线程上？

```
serialQueue.sync {
   print(1)
}
print(2)
serialQueue.sync {
   print(3)
}
print(4)
```

```
serialQueue.async {
   print(1)
}
print(2)
serialQueue.async {
   print(3)
}
print(4)
```

```
serialQueue.async {
   print(1)
   serialQueue.sync {
      print(2)
   }
   print(3)
}
print(4)
```

```
serialQueue.sync {
   print(1)
   serialQueue.async {
      print(2)
   }
   print(3)
}
print(4)
```

```
concurrentQueue.sync {
    print(1)
}
print(2)
concurrentQueue.sync {
    print(3)
}
print(4)
```

```
concurrentQueue.async {
    print(1)
}
print(2)
concurrentQueue.async {
    print(3)
}
print(4)
```

```
concurrentQueue.async {
    print(1)
    concurrentQueue.sync {
        print(2)
    }
    print(3)
}
print(4)
```

```
concurrentQueue.sync {
    print(1)
    concurrentQueue.async {
        print(2)
    }
    print(3)
}
print(4)
```

GCD vs. Operation

- DispatchQueue
- main, global(), qos
- sync, async, asyncAfter
- DispatchGroup

- Operation
- BlockOperation
- OperationQueue
- completionBlock

竞态条件（Race Condition）

```
var num = 0

DispatchQueue.global().async {
    for _ in 1...10000 {
        num += 1
    }
}

for _ in 1...10000 {
    num += 1
}
```

- 用串行队列去访问共享资源
- 用Disptach Barrier去解决读写问题

死锁问题（Dead Lock）

```
serialQueue.async {
   serialQueue.sync {
   }
}

let operationA = Operation()
let operationB = Operation()

operationA.addDependency(operationB)
operationB.addDependency(operationA)
```

- 少用依赖
- 慎用同步

优先倒置（Priority Inversion）

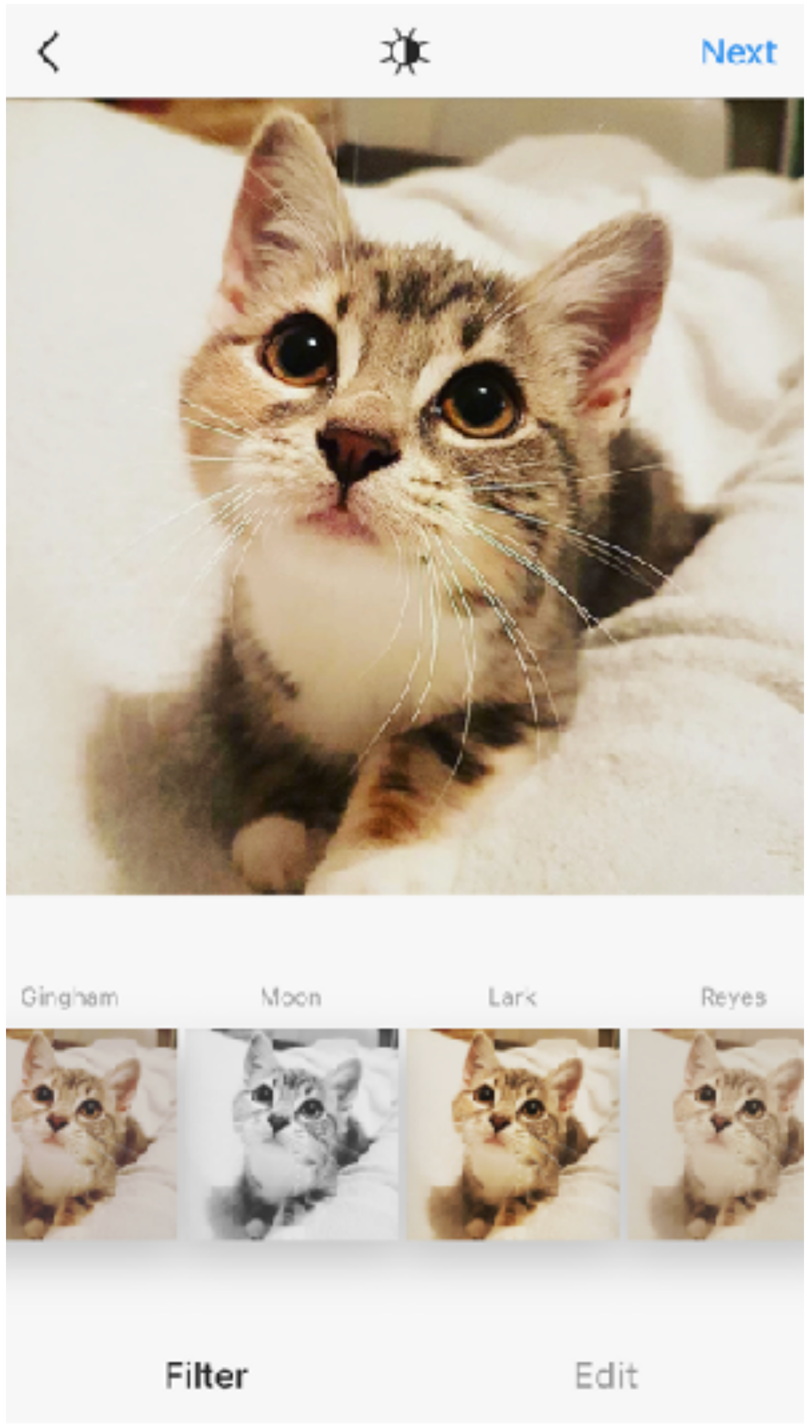
```
var highPriorityQueue = DispatchQueue.global(qos: .userInitiated)
var lowPriorityQueue = DispatchQueue.global(qos: .utility)

let semaphore = DispatchSemaphore(value: 1)

lowPriorityQueue.async {
    semaphore.wait()
    for i in 0...10 {
        print(i)
    }
    semaphore.signal()
}

highPriorityQueue.async {
    semaphore.wait()
    for i in 11...20 {
        print(i)
    }
    semaphore.signal()
}
```

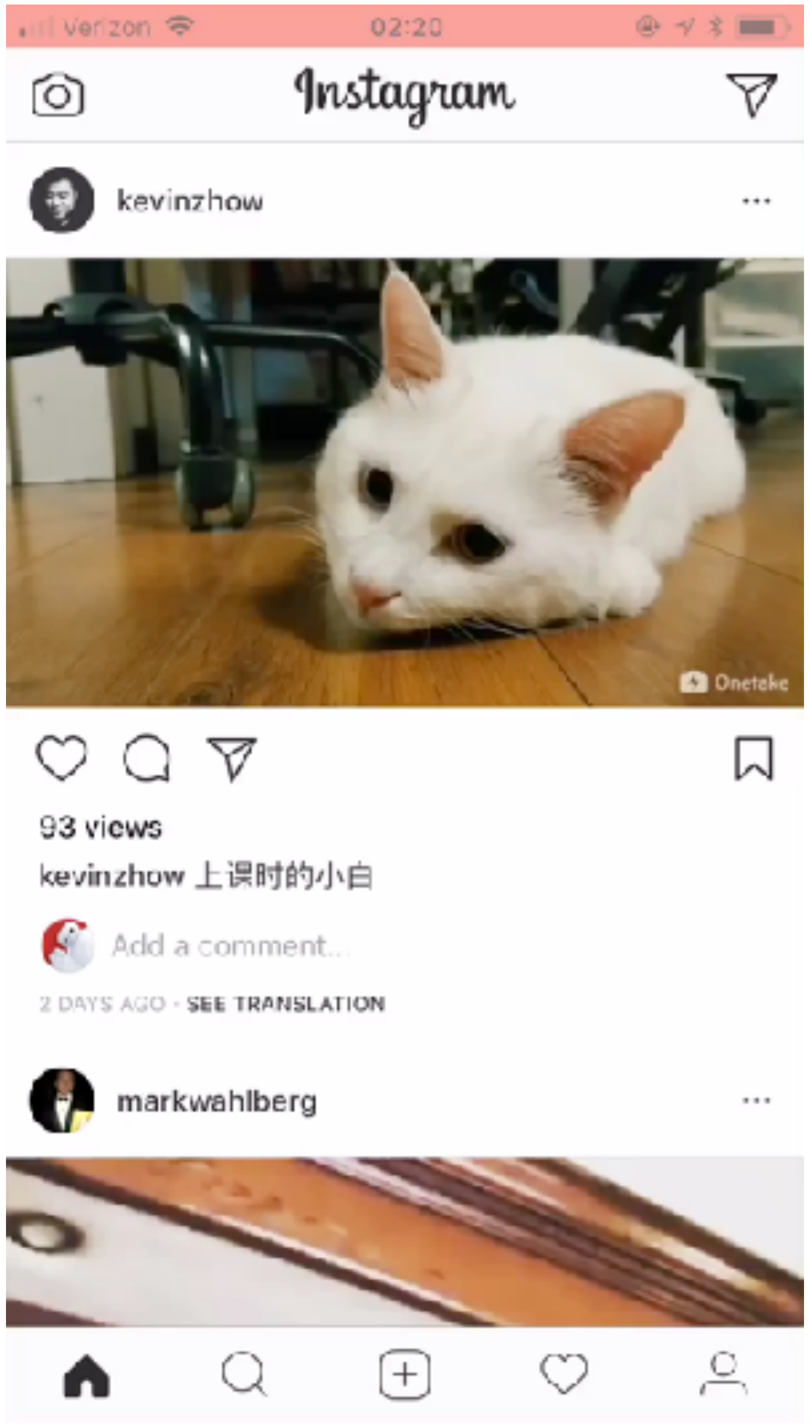
- 同一个资源
- 同一个QoS

“记得用TSan”

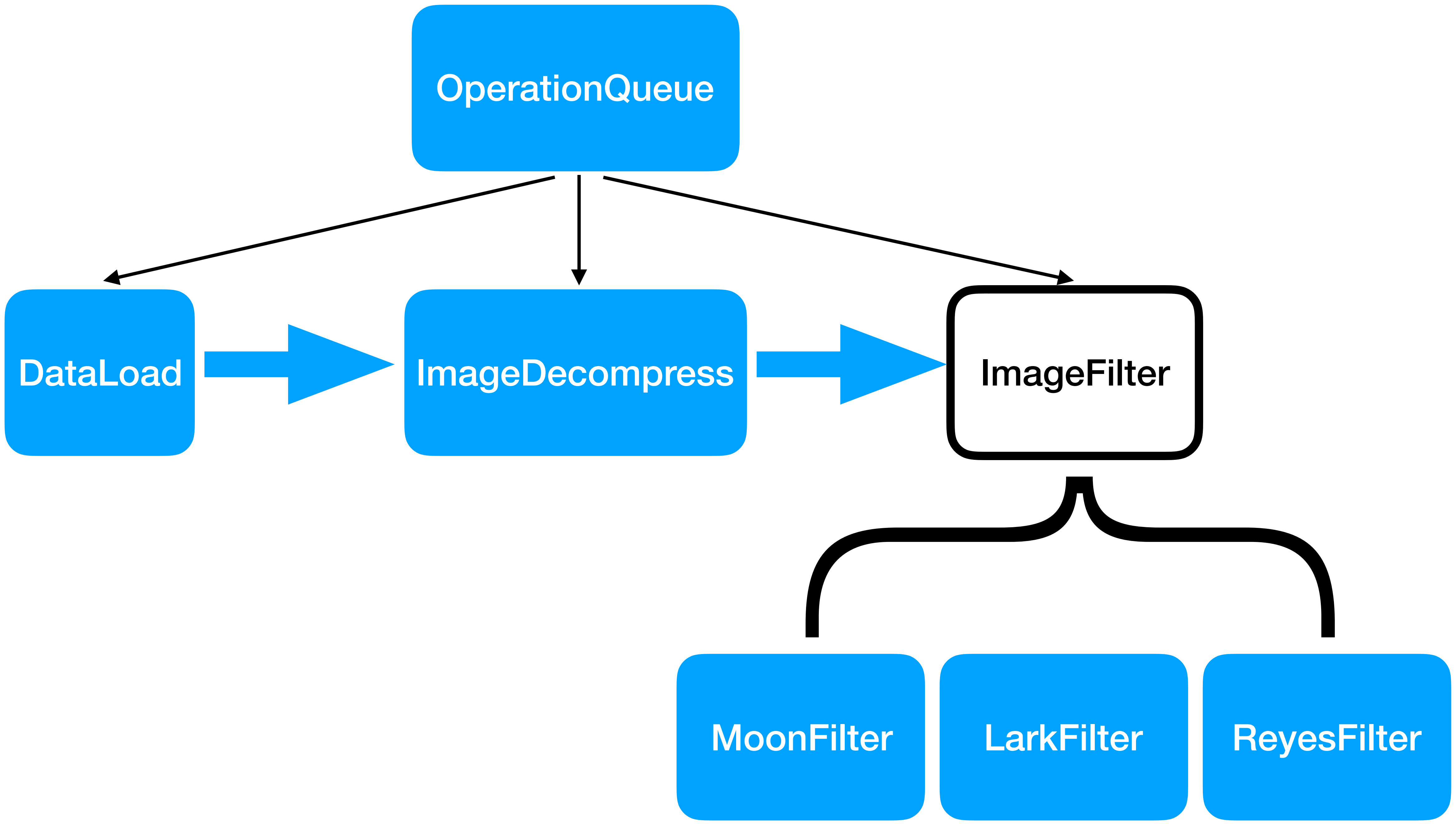
–Thread Sanitizer and Static Analysis, WWDC 2016

因斯腾格雷姆





93 views
kevinzhow 上课时的小白
Add a comment...
2 DAYS AGO · SEE TRANSLATION


OperationQueue
DataLoad
ImageDecompress
ImageFilter
MoonFilter
LarkFilter
ReyesFilter

Data Load Operation

```
let url: URL
var dataLoaded: Data?
let completion: ((Data?) -> ())?

init(url: URL, completion: ((Data?) ->
())? = nil) { ... }
```

```
override func main() {
  if isCancelled {
    return
  }

  ImageService.loadData(at: url) { data in
    if self.isCancelled {
      return
    }
    self.dataLoaded = data
    self.completion?(data)
  }
}
```

Image Decompress Operation

```
let imageData: Data?
var imageDecompressed: UIImage?
let completion: ((UIImage?) -> ())?

init(imageData: Data?, completion:
((UIImage?) -> ())? = nil) { ... }
```

```
override func main() {
  let dataCompressed: Data?
  if isCancelled { return }

  if let imageData = imageData {
    dataCompressed = imageData
  } else {
    let dataProvider = dependencies
      .filter { $0 is
ImageDecompressOperationDataProvider }
      .first as?
ImageDecompressOperationDataProvider
    dataCompressed = dataProvider?.dataCompressed
  }

  if self.isCancelled { return }
  if let data = Utility.convertData(dataCompressed)
{
    imageDecompressed = UIImage(data: data)
  }
  completion?(imageDecompressed)
}
```

```swift
protocol ImageDecompressOperationDataProvider {
  var dataCompressed: Data? { get }
}

extension DataLoadOperation: ImageDecompressOperationDataProvider {
  var dataCompressed: Data? { return dataLoaded }
}
```

```swift
protocol ImageFilterDataProvider {
  var imageRaw: UIImage? { get }
}

extension ImageDecompressOperation: ImageFilterDataProvider {
  var imageRaw: UIImage? { return imageDecompressed }
}
```

Image Filter Operation

```swift
let imageRaw: UIImage? {
  var image: UIImage?

  if let imageRaw = imageRaw {
    image = imageRaw
  } else if let imageProvider = dependencies
    .filter({ $0 is ImageFilterDataProvider })
    .first as? ImageFilterDataProvider {
    image = imageProvider.imageRaw
  }
  return image
}
var imageFiltered: UIImage?
let completion: (UIImage?) -> ()

init(imageRaw: UIImage?, completion:
(UIImage?) -> ()) { ... }
```

```swift
class MoonFilterOperation : ImageFilterOperation {
  override func main() {
    if isCancelled { return }
    guard let imageRaw = imageRaw else { return }


    if isCancelled { return }
    imageFiltered = imageRaw.applyMoonEffect()

    if isCancelled { return }
    completion(imageFiltered)
  }
}
```

Operation Queue

```swift
let operationQueue = OperationQueue()

let dataLoadOperation = DataLoadOperation(url: url)
let imageDecompressOperation = ImageDecompressOperation(data: nil)
let moonFilterOperation = MoonFilterOperation(image: nil, completion: completion)


let operations = [dataLoadOperation, imageDecompressOperation, moonFilterOperation]

// Add dependencies
imageDecompressOperation.addDependency(dataLoadOperation)
moonFilterOperation.addDependency(imageDecompressOperation)


operationQueue.addOperations(operations, waitUntilFinished: false)
```

总结

- iOS并发编程的基本概念
- 并发编程中的三大问题
- 用Operation流程化编程