# 中山大学数据科学与计算机学院本科生实验报告

| 课程名称 | 现代操作系统应用开发 | 任课老师 | 郑贵锋 |
| --- | --- | --- | --- |
| 年级 | 2017级 | 专业（方向） | 软件工程专业 |
| 学号 | 17343141 | 姓名 | 姚东烨 |
| 电话 | 13246859092 | Email | 894816193@qq.com |
| 开始日期 | 2019.11.18 | 完成日期 | 2019.11.18 |

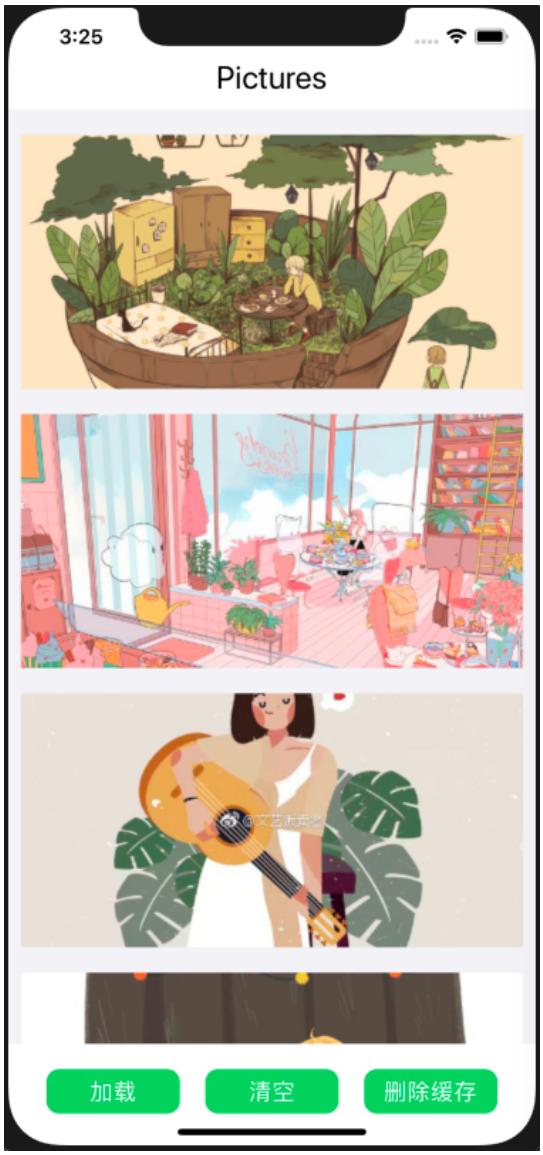
## 一、实验题目

**多线程与本地存储**

1. 学习使用NSOperation或GCD进行多线程操作
2. 学习iOS沙盒机制，进行文件读写操作

## 二、实现内容

实现一个简单的图片浏览应用，页面如下：

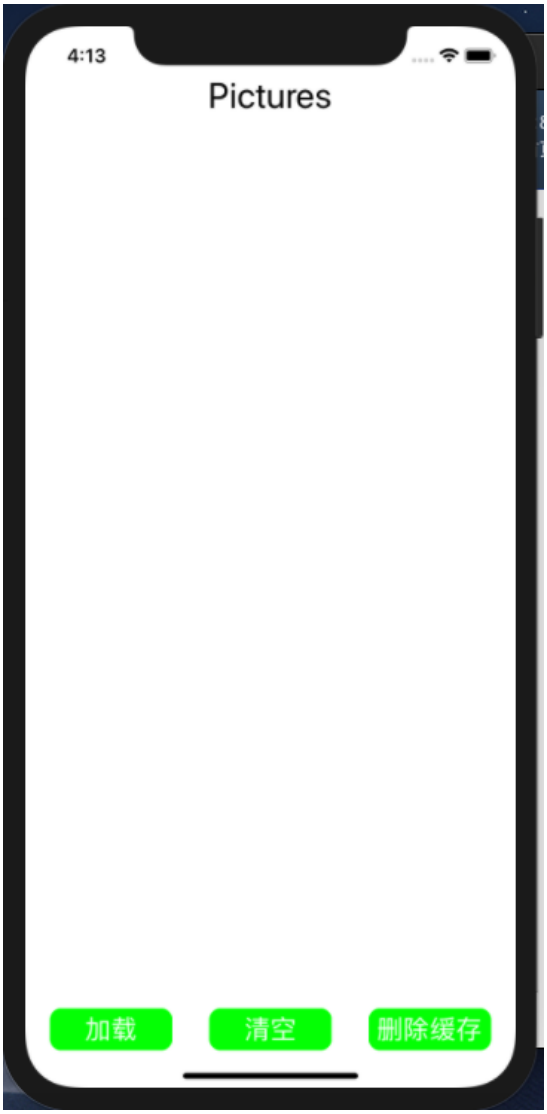
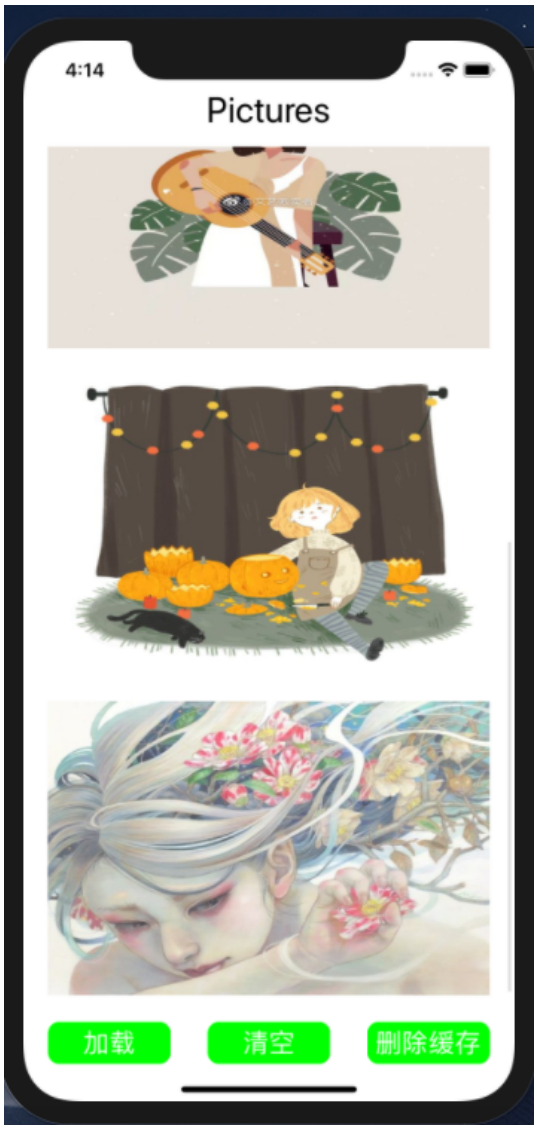
| 初始状态 | 加载图片 |
| --- | --- |
|  |  |





# 三、实验结果

## (1)实验截图

| 初始状态 | 加载图片 |
| --- | --- |
|  |  |





## (2)实验步骤以及关键代码

本次实验所做的应用主要有3个部分组成：顶部的**UILabel** 、中间的**UICollectionView** 、底部的3个**UIButton** 。下面将依次对他们进行介绍。

**UILabel**

> 这个组件十分简单，主要是要调整其位置和字体，除此之外就没有什么难点了。
>
> 具体代码如下：

```
- (UILabel *)titleLabel{
    if(_titleLabel == nil) {
        _titleLabel = [[UILabel alloc] initWithFrame:CGRectMake(0,
self.view.frame.size.height*0.03, self.view.frame.size.width,
self.view.frame.size.height*0.07)];
        [_titleLabel setText:@"Pictures"];
        [_titleLabel setFont: [UIFont systemFontOfSize: 30]];
        [_titleLabel setTextColor: [UIColor blackColor]];
        [_titleLabel setTextAlignment:NSTextAlignmentCenter];
    }
    return _titleLabel;
}
```

**UIButton**

- LeftButton

  这个按钮做的是资源的下载，这里要使用的多线程进行下载。做一个5次循环，建立5个 **NSInvationOperation** ，并给这些Operation对应分配一个 **NSDictionary** ，传递所需要下载的图片（图片的链接被保存在之前设好的数组中），最后将这些Operation加入到子线程任务队列 **NSOperationQueue** 中。

  ```
  - (void) leftEvent:(UIButton *) btn
  {
      NSOperationQueue *queue =[[NSOperationQueue alloc]init];
      for(int i=0;i<5;i++){
          NSDictionary*dict= [NSDictionary dictionaryWithObject:[NSString
  stringWithFormat:@"%d",i] forKey:@"key"];
          NSInvocationOperation *op=[[NSInvocationOperation
  alloc]initWithTarget:self selector:@selector(downloadImg:) object:
  dict];
          [queue addOperation:op];
      }
  }
  ```

  在NSInvationOperation中，加入了一个@selector **downloading** ，在 **downloading** 中，我们先要根据传过来的dict获取参数，接着根据这个参数获取对应的图片url，接着就可以通过 **NSURL** 获取图片。等到图片获取成功后，**downloading** 会执行回调函数，在这个回调函数中，我们要将更新UI的 @selector **rd** 加入到主线程的执行队列中。

  ```
  - (void) downloadImg:(NSDictionary*)dict
  {
      self.Clicked=true;
      NSLog(@"%@",[dict valueForKey:@"key"]);
      NSString* temp=[dict valueForKey:@"key"];
      int value = [temp intValue];
      NSURLSessionConfiguration *defaultConfigObject =
  [NSURLSessionConfiguration defaultSessionConfiguration];
      NSURLSession *delegateFreeSession = [NSURLSession
  sessionWithConfiguration: defaultConfigObject

   delegate: self

  delegateQueue: [NSOperationQueue mainQueue]];
      NSLog(self.dataSourceArray[value]);
      NSURL *url = [NSURL URLWithString:self.dataSourceArray[value]];
  ```

```
    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    NSURLSessionDataTask * dataTask = [delegateFreeSession
dataTaskWithRequest:request completionHandler:^(NSData *data,
NSURLResponse *response, NSError *error) {
        if(error == nil)
        {
            NSArray *paths =
NSSearchPathForDirectoriesInDomains(NSCachesDirectory,
NSUserDomainMask, YES);
            NSString *cachePath = [paths objectAtIndex:0];
            NSString *filePath = [cachePath
stringByAppendingPathComponent:[NSString stringWithFormat: @"%ld.png",
value]];
            [UIImagePNGRepresentation([UIImage imageWithData:[NSData
dataWithContentsOfURL:url]]) writeToFile:filePath atomically:YES];
            NSOperationQueue *queue =[NSOperationQueue mainQueue];
            NSInvocationOperation *op=[[NSInvocationOperation
alloc]initWithTarget:self selector:@selector(rd) object: NULL];
            [queue addOperation:op];
        }
    }];
    [dataTask resume];
}
```

至于**rd** 函数的功能就十分简单了，只需要调用一下 **UICollectionView** 的 **reloadData** 函数就可以了。

```
- (void)rd{
    [self.collectionView reloadData];
}
```

- MiddleButton

  中间这个按钮就只是将图片都设为不显示而已。在这我选择维护一个 **Clicked** 变量，当值为 **false** 的时候，重载cell时会将其中的 **UIImage** 设为**NULL** 。值为 **true** 的时候才会根据图片是否在缓存中进行图片显示。

```
- (void) middleEvent:(UIButton *)btn{
    self.Clicked=false;
    [self.collectionView reloadData];
}
```

- RightButton

  右边这个按钮就更加简单了，只是清除缓存的功能。

```
- (void) rightEvent:(UIButton *) btn
{
    for(int i=0;i<5;i++){
        [self.fileManager removeItemAtPath:self.fileSourceArray[i]
error:nil];
    }
}
```

**UICollectionView**

> 位于应用中部的这个collectionview其实也没什么特别的，重点就在于如何选择cell的图片。其逻辑是这样的：当 **Clicked** 为 **false** 的时候，就什么都不显示，将cell的image设为NULL，当 **Clicked** 为**true** 的时候，就判断此时cell对应的图片在不在缓存中，若不在则显示转圈等待图片，否则则显示对应图片。
>
> 具体代码如下：

```
- (UICollectionViewCell *)collectionView:(UICollectionView *)collectionView
cellForItemAtIndexPath:(NSIndexPath *)indexPath{

    NSLog(@"%d",indexPath.item);
    MyCell *cell =[collectionView
dequeueReusableCellWithReuseIdentifier:@"MyCell" forIndexPath:indexPath];
    if(self.Clicked==false){
        cell.imageView.image=NULL;
        return cell;
    }
    else{
        if([self.fileManager
fileExistsAtPath:self.fileSourceArray[indexPath.section]]==false){
            NSLog(@"Download not complete");
            cell.imageView.image=[UIImage imageNamed:@"loading.png"];
        }else{
            cell.imageView.image=[UIImage
imageWithContentsOfFile:self.fileSourceArray[indexPath.item]];
         //   cell.imageView.image=[UIImage imageNamed:@"loading.png"];
        }
    }
    return cell;
}
```

### (3)实验遇到的困难以及解决思路

- @Selector不会传参。因为在之前的实验中，我们很少接触过这个Selector，而这次的实验中，我的思路是5次调用Selector就要对应下载5个不同的图片，那么就要传过去0~4这5个数。但是到了真正要传参的时候就找不到要把参数填到哪里了。最后通过网络查询，发现可以设置一个UIDictionary传过去，这样才解决了问题。
- 不知道如何更新UI。实验中要求我们要在主线程中更新UI，但是当我设置改变了一个cell的image后，我不知道应该如何和主线程交互来更新UI。后来经过和同学的讨论，发现可以使用UICollectionView里的reloadData，这样就可以更新UI了。

## 四、实验思考及感想

本次实验我认为是我们这几次实验中比较简单的一次。只需要实现5个组件就做完了。我觉得学到最多的还是这个重新加载cell的方法，一开始我一直想获取对应的cell并更新其图片。但是就遇到一个问题：由于重用机制，我很可能拿不到在当前页面以外的cell。而且在我们划到页面以外的cell时，这个cell会重新加载，那么我们所做的更改就无效了。所以最终我使用了简单粗暴的方法：reloadData，更新整个UICollectionView。但其实这种做法的花费是比较大的。在我网上冲浪的过程中，我发现了有个api可以获取当前能看到的cell，这样配合上cell的加载函数，应该可以更高效的实现本次作业。但是由于当时做的时候脑子非常混乱，想不清楚各种逻辑，所以就直接使用“暴力刷新”了。

至于多线程，我们在之前的操作系统课中已经被折磨了很久，对于其运作的原理算是比较清楚。在操作系统课中我们用pthread实现过很多程序，所以现在用上封装好的多线程api，难度并不是很大。

以上就是本次实验中我的收获与感想，希望在之后为数不多的作业中能做的更好。