ITSC IOS客户端开发实验报告

学号	姓名
191220162	张乐简

ITSC IOS客户端开发实验报告

- 概述
- 实验内容
  - 最上层
  - 基本功能
    - 下载数据
    - 解析html
    - 支持浏览条目
    - 应对断网情况
  - 搜索功能
    - 搜索UI搭建
    - 搜索模块算法设计
- 反思

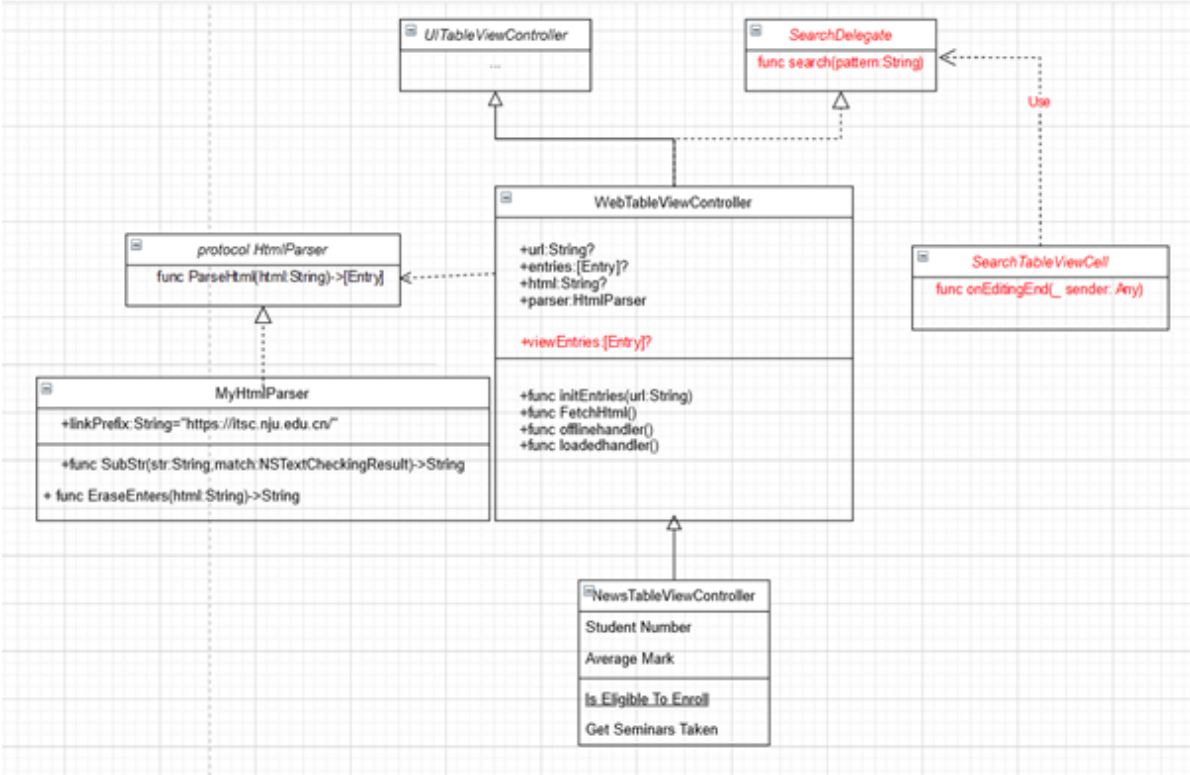
概述

该实验为https://itsc.nju.edu.cn开发一个IOS客户端，用户界面基于给定的模板设计，要求可以点开每一个具体的条目进行浏览。同时，也采用了GCD进行多线程编程，异步地从网站上下载数据，并根据屏幕的方向设置了不同的界面布局。另外，也实现了搜索条目的功能。

实验内容

最上层

这一节，报告将根据如下的类图，介绍App的大体结构。



其中，一些不重要的或重复类没有画出。另外，红色的字体是与搜索功能有关的类和协议。WebTableViewController类负责完成数据的下载、并调用HtmlParser进行解析，利用offlinehandler和loadedhandler控制activity indicator的动画播放以及断网情况的处理；NewsTableViewController和负责其他三个页面的类继承WebTableViewController，它们通过设定不同的url、调用基类中的API来完成具体的条目浏览功能。

MyHtmlParser实现HtmlParser协议，利用正则表达式NSRegularExpression来解析Html，并生成便于使用的，包含条目标题、url和日期的Entry结构体数组返回给调用的WebTableViewController。

另外，为支持搜索功能，WebTableViewController实现SearchDelegate协议，它将以委托的方式处理从SearchTableViewCell中发出的搜索请求。

“关于”界面的实现没有太多特别之处，其中的html解析、断网处理在前面均有设计，不单独拿出来介绍。

基本功能

下载数据

这个功能在WebTableViewController中的FetchHtml方法中实现。该方法为initEntries方法异步地调用，在获取到数据，或发现网络有问题时调用对应的Handler。

```
    func initEntries(url:String){
        self.viewEntries=[Entry]()
        self.url=url
        let queue=DispatchQueue(label: "ITSC.web")
        queue.async {
            self.FetchHtml()
        }
    }
```

如上，initEntries异步地调用FetchHtml。而这个初始化函数将在viewDidLoad中被继承WebTableViewController的子类以不同的url调用。FetchHtml具体代码如下：

```
        let task=URLSession.shared.dataTask(with: URL(string: url!)!,
completionHandler: {
            data,response,error in
            if let error=error{
                print("\(error.localizedDescription)")
                self.offlinehandler()
                return
            }
            guard let httpResponse=response as? HTTPURLResponse,
                  (200...299).contains(httpResponse.statusCode) else{
                      print("server error")
                      return
                  }
            if let mimeType=httpResponse.mimeType,mimeType=="text/html",
               let data=data,
               let string=String(data:data,encoding: .utf8){
                DispatchQueue.main.async {
                    self.html=string
                    self.entries=self.parser.ParseHtml(html: self.html!)
                    self.viewEntries=self.entries
                    self.tableView.reloadData()
                    self.loadedhandler()
                }
            }
        })
        task.resume()
```

完全参照PPT。在成功得到Html文件后，FetchHtml将控制转给解析器，在主线程中异步地解析Html，并初始化需要的条目。

解析html

这部分在MyHtmlParser中实现。首先观察得到的Html文件中所需信息出现的形式，再制作对应的正则表达式进行解析，最后将解析得到的产物装入[Entry]返回。代码如下。

```
    func ParseHtml(html: String) -> [Entry] {
        var ret:[Entry]=[Entry]()
        var test:String=html
        test=EraseEnters(html: test)
        let range=NSRange(location: 0, length: test.count)
        let regex=try! NSRegularExpression(pattern: "<spanclass=\"news_title\">
<ahref='.*?'target='_blank'title='.*?'>(.*?)</a></span>")

        let matches=regex.matches(in: test, options: [], range: range)
        for match in matches{
            let substr=SubStr(str: test, match: match)
            let linkregex=try! NSRegularExpression(pattern: "<ahref=
(.*?)'target='")
            let subrange=NSRange(location: 0, length: substr.count)
            let linkmatch=linkregex.firstMatch(in: substr, options: [], range:
subrange)
            var linkstr:String=SubStr(str: substr, match: linkmatch!)

 linkstr=linkPrefix+linkstr.substring(with:linkstr.index(linkstr.startIndex,offs
etBy: 9)..<linkstr.index(linkstr.startIndex,offsetBy: 37))
```

```
            let titleregex=try! NSRegularExpression(pattern: "'title='(.*?)'")
            let titlematch=titleregex.firstMatch(in: substr, options: [], range:
subrange)
            var titlestr:String=SubStr(str: substr, match: titlematch!)
            titlestr=titlestr.substring(from:
titlestr.index(titlestr.startIndex,offsetBy: 8))
            titlestr=titlestr.substring(to:
titlestr.index(titlestr.startIndex,offsetBy: titlestr.count-1))

            var datestr:String=test.substring(with:
test.index(test.startIndex,offsetBy:
match.range.location+match.range.length+35)..
<test.index(test.startIndex,offsetBy:match.range.location+match.range.length+45)
)

            let entry=Entry(title: titlestr, url: linkstr,date: datestr)
            ret.append(entry)
        }

        return ret
    }
```

其中，EraseEnters用于消除Html文件中的空格和回车，方便匹配；SubStr则是为了方便取子字符串而封装的方法。

支持浏览条目

这个功能相对比较简单，在某个cell被点击时，即在prepare函数中传入对应的url，让下一个ViewController加载该页面，准备url代码如下：

```
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        let detail = segue.destination as! WebDetailViewController
        let cell=sender.self as! MyTableViewCell
        detail.currentUrl=cell.entry?.url
    }
```

加载url部分完全与PPT中相同，不再赘述。

应对断网情况

若在获取条目时时发现网络断开，如前所述，控制权将被移交给offlinehandler。由于不太清楚如何让四个子类共用一个outlet变量来承接它们的indicator和retrybtn，我将offlinehandler的具体任务移交给子类自己完成。具体来说，即是停止indicator播放，同时显示重试的按钮。代码如下：

```
    override func offlinehandler() {
        DispatchQueue.main.async {
            self.indicator.stopAnimating()
            self.retrybtn.isHidden=false
            self.retrybtn.isEnabled=true
        }
    }
```

注意此处回到主线程执行这段代码，是因为之前有过报错提示不可在主线程外调整indicator和retrybtn的状态。最后，为retrybtn加上响应函数。代码如下：

```
    @IBAction func onretry(_ sender: Any) {
        DispatchQueue.main.async {
            self.retrybtn.isEnabled=false
            self.retrybtn.isHidden=true
            self.indicator.startAnimating()
        }
        self.initEntries(url: "https://itsc.nju.edu.cn/xwdt/list.htm")
    }
```

此处用于演示的代码在控制新闻动态的ViewController中。其目的即是恢复初始状态，再次加载页面。

搜索功能

搜索UI搭建

由于TableView顶边栏的两个位置已经被indicator和重试按钮占据了，单独开辟一个Section，并在里面放入一个SearchBar，一个搜索按钮来完成与用户的交互。当接受到点击搜索事件时，Cell将会以委托的形式将控制转给它所在的ViewController。Cell的代码如下：

```
class SearchTableViewCell: UITableViewCell {
    @IBOutlet weak var searchBar: UISearchBar!
    var delegate:SearchDelegate?=nil
    ...
    @IBAction func onEditingEnd(_ sender: Any) {
        delegate!.search(pattern: searchBar.text!)
    }
}
```

其中，delegate的初始化在对应的ViewController在loadData时完成。如下：

```
        if indexPath.row==0{
            let sCell=tableView.dequeueReusableCell(withIdentifier:
"searchCell")
            let scs=sCell as! SearchTableViewCell
            scs.delegate=self as! SearchDelegate
            return sCell!
        }
```

搜索模块算法设计

在tableView和entries，即网站上的所有条目，之间加一层中间人。tableView不直接获取entries的数据，而是获取viewEntries的数据，viewEntries则在search方法被调用时更新。具体的搜索方法仍然是通过正则表达式匹配。search代码如下：

```
    func search(pattern:String){
        self.viewEntries!.removeAll()
        let regex=try! NSRegularExpression(pattern: "(.*?)"+pattern+"(.*?)")
        for var entry in self.entries!{
            let text=entry.title
            let range=NSRange(location: 0, length: text.count)
            let matches=regex.matches(in: text, options: [], range: range)
            if matches.isEmpty==false{
                self.viewEntries!.append(entry)
            }
        }
        self.tableView.reloadData()
    }
```

反思

四个界面的内容实在大同小异，应该使用同一个类。下次可试着寻找共用一个outlet变量的方法。