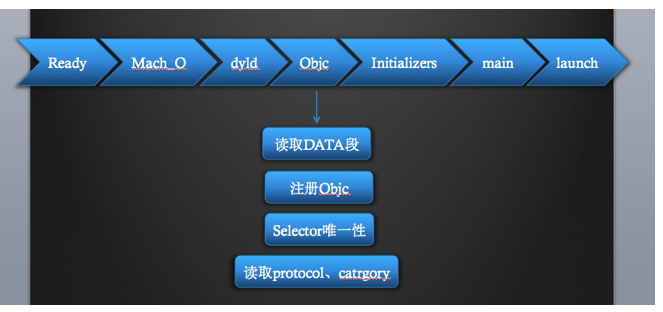
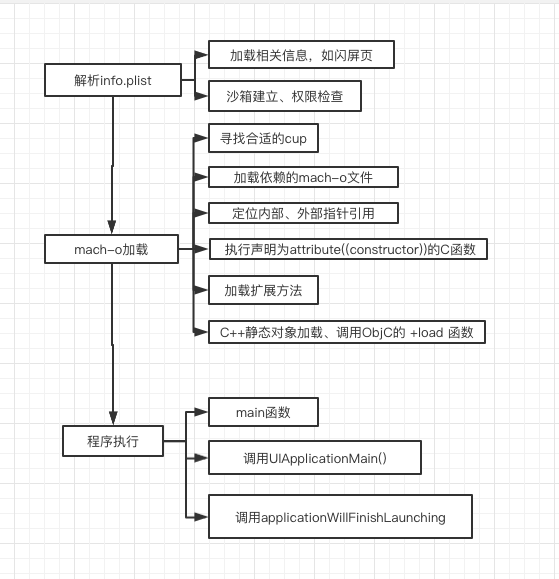
关于iOS 启动流程以及加载过程面试题

1. 启动过程：  
     
   

2、影响启动性能因素。  
1.main()函数之前耗时的影响因素

* 动态库加载越多，启动越慢。
* ObjC类越多，启动越慢
* C的constructor函数越多，启动越慢
* C++静态对象越多，启动越慢
* ObjC的+load越多，启动越慢

1. main()函数之后耗时的影响因素

* 执行main()函数的耗时
* 执行applicationWillFinishLaunching的耗时
* rootViewController及其childViewController的加载、view及其subviews的加载

3、启动优化：

3.1 启动时间 = pre-main耗时+main耗时

3.2 pre-main阶段优化：通过xcode查看启动时间

3.3删除无用代码

3.4抽象重复代码

3.5 +load方法做的事情延迟到initialize中，或者+load的事情不宜花费太多时间。因为load方法是启动应用就会调用。Initialize只有初始化了实例对象才会调用。

3.6 减少不必要的framework，或者优化已有framework

3.7 Main阶段优化

3.8 didFinishLauchingwithOptions里代码延后执行

3.9 首次启动渲染的页面优化

4、视图等的加载顺序

4.1、AppDelegate加载顺序

1.application:didFinishLaunchingWithOptions:

2.applicationDidBecomeActive:

4.2、ViewController中的加载顺序

1.loadView

2.viewDidLoad

3.viewWillAppear

4.viewWillLayoutSubviews

5.viewDidLayoutSubviews

6.viewDidAppear

4.3、View中的加载顺序

1.initWithCoder（如果没有storyboard就会调用initWithFrame，这里两种方法视为一种）

2.awakeFromNib

3.layoutSubviews

4.drawRect  
5、load和initalize的理解  
5.1、+load方法调用顺序是：类或分类添加到object-c runtime时被调用 SuperClass -->SubClass --> CategaryClass。 不同类调用先后顺序不固定。

5.2、+initialize是在类或者它的子类接受第一条消息前被调用，但是在它的超类接收到initialize之后。也就是说+initialize是以懒加载的方式被调用的，如果程序一直没有给某个类或它的子类发送消息，那么这个类的+initialize方法是不会被调用的。

5.3、+initialize方法和+load方法还有个区别，就是运行期系统完整度上来讲，此时可以安全使用并调用任意类中的任意方法。而且，运行期系统也能确保+initialize方法一定会在“线程安全的环境”中执行，这就是说，只有执行+initialize的那个线程可以操作类或类实例，其他线程都要阻塞等着+initialize执行完。

5.4、+initialize方法和其他类一样，如果某个类未实现它，而其超类实现了，那么就会运行超类的实现代码。如果本身和超类都没有实现，超类的分类实现了，就会去调用分类的initialize方法。如果本身没有实现，超类和父类的分类实现了就会去调分类的initialize