1. 进程：是指在系统中正在运行的一个应用程序，一个应用程序可以有多个进程，比如打开2个qq
2. 线程：1个进程要执行任务，必须要有线程（每1个进程至少要有1条线程），是CPU调度的最小单位
3. 对于单核系统：同一时间，CPU只能处理1条线程，只有1条线程在工作，多线程并发执行，其实是CPU快速地在多线程之间调度（切换）

CPU在N多线程之间调度，CPU会累死，降低程序的性能，消耗大量的CPU资源（一般开3-5条）

1. 多线程优点：能适当提高程序执行效率；缺点：线程越多，cups调度开销越大，而且存在多线程数据抢夺安全问题
2. 
3. 安全隐患解决：互斥锁

格式：@synchronize(锁对象) {

//需要锁定的代码

//优点：能有效防止多线程抢夺资源造成数据安全为题

//缺点：需要消耗大量的CPU资源

}

注意：1.锁对象必须全局唯一，一般用self

2.锁定1份代码只能用1把锁，用多把锁是无效的

线程同步：多条线程按顺序地执行任务），

怎样实现线程同步：给线程加一把互斥锁，默认是异步的

7．原子和非原子属性‘

atomic [ə'tɒmɪk]：原子属性，为setter方法加锁（默认），线程安全，但需要消耗大量的资源

nonatomic:非原子属性，不会为setter方法加锁，非线程安全

8．线程间通讯(刷新UI必须在主线程)

常用方法：

* (void)performSelectorOnMainThread(SEL)aSelector withObject(id)arg waitUntilDone(BOOL)wati;
* (void)performSelector:(SEL)aSelector onThread:(NSThread \*)thr withObject:(id)arg wa….

9.图片下载过程

1）确定url NSURL \*url = [NSURL URLWithString。。。

2）根据url下载图片 二进制数据

NSData \*data = [NSData dataWithContentsOfURL:url];

3)把二进制数据转换图片格式

UIImage \*image = [UIImage imageWithData:imageData];

4）显示图片

10．GCD（Grand Central Dispatch）共产党

执行任务两种方式：

同步：只能在当前线程中执行任务，不具备开启新线程能力

异步：可以在新的线程中执行任务，具备开启新线程能力

队列两种类型：

串行队列：让任务一个接着一个执行

并发队列：可以让多个任务并发（同时）执行

GCD快速迭代：---应用：文件移动

dispatch\_apply(遍历次数, 队列（传全局并发队列，如果是主队列，会出现死锁，传串行队列，没有作用）, index索引)

---》内部会开启子线程并发执行遍历任务

GCD队列组：

dispatch\_group\_t = dispatch\_group\_create();

dispatch\_group\_notify(group, queue, ^{

//内部本身是异步，不会阻塞主线程

//组中所有任务执行完毕后发这个通知

})

dispatch\_group\_wait(group, DISPATCH\_TIME\_FOREVER);//该方法等组所有任务执行完才会走后面内容，但会阻塞只线程

注意：

1.并不是有多少个任务就开多少条线程，这个是有系统决定

2.主队列的任务都在主线程运行，同步+主队列===死锁，

队列作用：存放任务，和安排任务到哪个线程执行

3.dispatch\_once函数保证代码在程序运行过程只被执行一次，而且不能用在懒加载中

4.栅栏函数不能使用全局并发队列，dispathc\_barrier\_async（queue， ^）{……});

5.使用for遍历时，不要在for 里面写 i < array.count,因为array.count会调用[array count]方法，这样，每次循环都调用这个方法，应该用一个变量接收

使用create函数创建但并发队列和全局并发队列区别：

1. 全局并发队列在整个程序中默认存在的，并且有优先级，而create函数创建的队列要从头设置
2. 栅栏函数只对create函数有效，对全局并发队列中无效

dispatch\_async()和dispatch\_saync\_f()区别：

封装任务对方法，第一使用block，第二使用函数

栈只有一个入口，先进后出，而队列是先进先出

延迟操作几种方式：

1. [self performSelector:@selector(delayMethod) withObject:nil/\*可传任意类型参数\*/ afterDelay:2.0];

取消：

[NSObject cancelPreviousPerformRequestsWithTarget:self];

1. NSTimer定时器

取消：[timer invalidate];

1. [NSThread sleepForTimeInterval:2.0];

只有该方法阻塞主线程，直接让线程休眠

1. GCD : dispatch\_after，改方法可以指定在哪个线程里面执行

11.单例模式：

保证程序运行过程中，一个类只有一个实例对象

步骤0:提供全局变量 static MMTool \*\_instance;

1:提供类方法 +(instancetype)shareTool {

retrun [[self alloc] init] }

2.重写allocWithZone方法

static dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

\_instance = [super allocWithZone:zone];

});

return \_instance;

3:重写copyWithZone和mutableCopyWithZone

return \_instance(直接返回该对象，因为已经存在)

注意：单例模式不能继承

系统单例有：[UIApplication shareApplication]

[NSFileManager defalutManager]

[NSNotificationCenter defaultCenter]

[NSUserDefault standerUserDefaults]

MRC模式下的单例：

4.重写relesase 5.retain方法中，return \_instance

6.retainCount方法中，return MAXFLOAT;

项目改为MRC：target-build Setting-搜automatic-改NO

宏使用 \ 连接下一行代码

12.NSOperation是个抽象类，并不具备封装操作的能力，必须使用它的子类

子类有3种：

* NSInvocationOperation
* NSBlockOperation
* 自定义子类继承NSOperation，实现内部相应的方法(main方法) ，外面只需要 【alloc init】创建任务

NSOperationQueue队列：

* 主队列 [NSOperationQueue mainQueue] 和GCD的主队列一样，在主线程执行
* 非主队列 [[NSOperationQueue alloc] init] 非常特殊（同时具备并发和串行的功能）
* 默认情况下，非主队列是并发队列，可以设置maxConcurrentOperationCount > 1 并发队列

== 1 串行队列

== 0 不会执行任务

（表示最大值，不受限制）== -1 默认值

串行执行任务不等于只开一条线程（线程同步）

1. 创建任务

* [NSInvocationOperation alloc] initWithTarget….
* [NSBlockOperation blockOperationWithBlock….

1. 创建队列

NSOperationQueue \*queue = [[NSOperationQueue alloc] init]

1. 添加操作到队列

[queue addOperatio:op1];

[queue addOperatio:op2];

NSBlockOperation 简便创建方法：

* [queue addOperationWithBlock:^{}];

队列的状态

暂停： [self.queue setSuspended:YES];

继续： [self.queue setSuspended:NO];

取消: [self.queue cancelAllOperations];

注意：1。暂停和取消都不能暂停当前正在执行都任务，等当前任务执行完毕再暂停 2。取消不可以恢复

自定义任务的好处：

1. 有利于代码隐藏 2.复用性

操作依赖：不能循环依赖，如果循环依赖，哪个都不执行，但可以跨队列依赖

[op1 addDependency:op2];任务2执行完毕才执行任务1

操作监听：

[op3.completionBlock = ^{ … }];任务3执行完毕才执行

主线程更新UI：

[[NSOperationQueue mainQueue]

addOperationWithBlock:^{ //更新UI操作 }];

图片合并步骤：

1. 开启上下文(设置大小)

UIGraphicsBeginImageContext(CGSizeMake(200,200));

1. 画图1

[image1 drawInRect:CGRectMake(0,0,100,200)];

1. 画图2

[image2 drawInRect:CGRectMake(100,0,100,200)];

1. 根据上下文得到图片

UIImage \*image =

UIGraphicsGetImageFromCurrentImageContext();

1. 关闭上下文

UIGraphicsEndImageContext();

沙盒（磁盘缓存）目录

Documents:会备份，苹果不允许把缓冲数据放在里面，否则上架被拒

Tmp:临时路径（随时会被删除），所以不可以保存缓存数据

Library:

Preferences：偏好设置，保存账号

Caches:缓存文件

获取caches路径：

NSString \*cachesPath =

[NSSearchPathForDirectoriesInDoMains(NSCachesDirectory,NSUserDomainMask,YES).lastObject];

UITableView列表展示图片：

//1.UI很不流畅 ---》 应该开启子线程下载图片

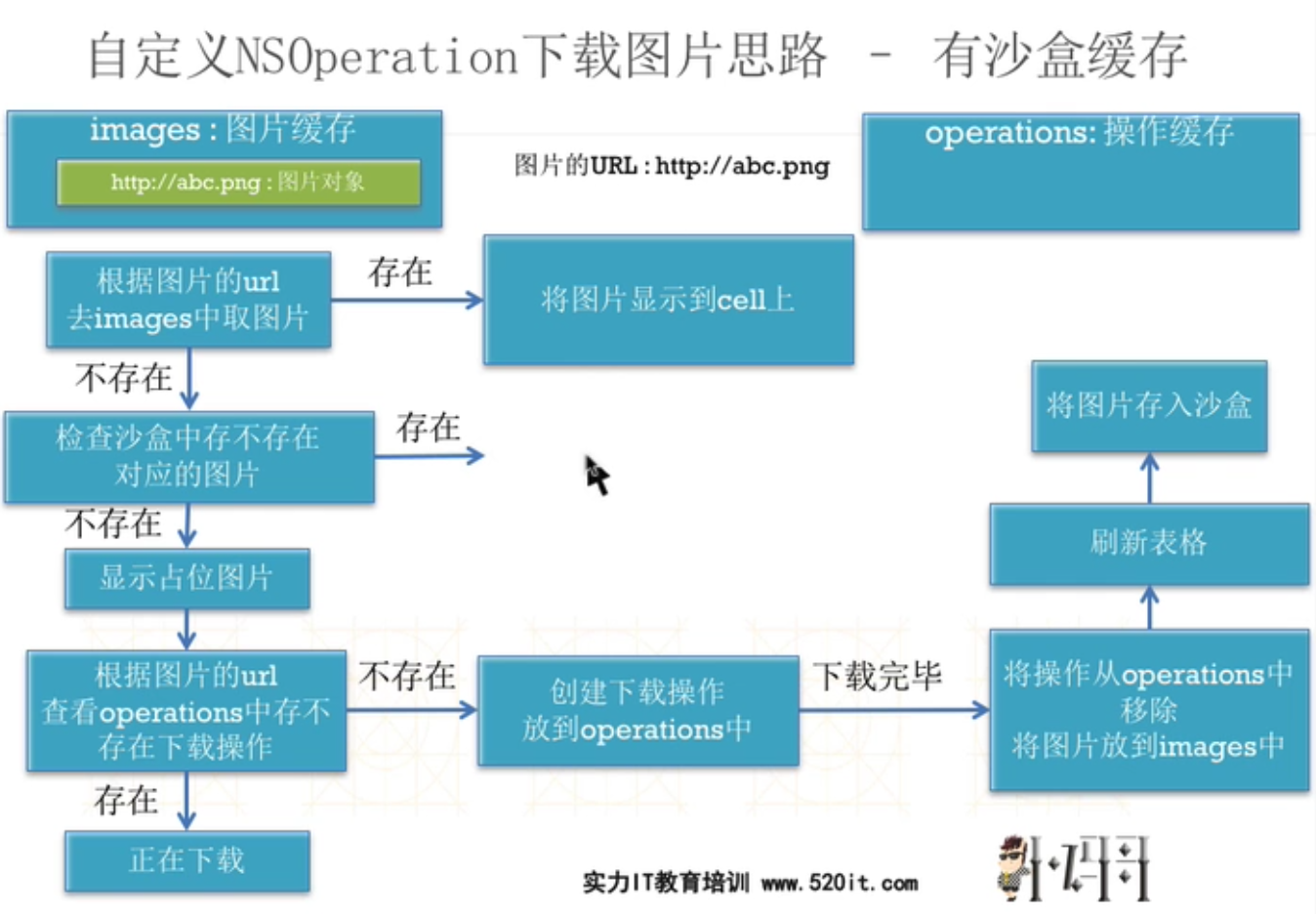
//2.图片重复下载 ---》 应先把之前已经下载好的图片用字典保存起来---（先内存缓存----再沙盒缓存）

//3.开启子线程下载不会刷新---收到刷新某行

//4.图片还会出现重复下载（因为图片下载需要事件，当网速较慢，第一张还么下载完，然后画出屏幕，又滑回来，就会出现又下载一次）--- 用字典把任务NSBlockOperation保存，先判断是否在下载，否则再添加任务，当图片下载完成后，再把该任务移除

//4.当图片下载完，刷新某一行会出现数据错乱：因为下一个显示当cell图片复用顶部滚出cell当图片--- 下载前设置显示一个默认图片

//5.当tableview内存缓存过多时，在didReceiveMemoryWarning方法把缓存图片当字典清空，同时把队列当任务都取消。



1.SDWebImage清空缓存：

ClearDisk:直接删除，然后重新创建

CleanDisk:清除过期缓存（默认7天），计算当前缓存都大小，和设置都最大缓存数比较，如果超出会继续删除（按文件创建都先后顺序）

[[SDWebImageManager sharedManager].imageCache

clearDisk];

2,取消当前所有都操作

[[SDWebImageManger sharedManager] cancelAll];

在appdelegate的applicationDidReceiveMemoryWarning

方法写上面2个方法

3.最大并发数量 == 6

4. 缓存文件的保存名称如何处理：拿到图片的URL路径，对该路径进行MD5加密

5.该框架内部对内存警告对处理方式：内部通过监听通知的方式处理，接收到警告后清空缓存

6.进行缓存处理的方式：内部使用NSCache类进行缓存处理（AFNetworking也是）

7.如何判断图片的类型：判断图片二进制数据的第一个字节，[nsdata datawithcontentoffile]

8.队列中任务的处理方式：FIFO

9.如何下载图片：使用NSURLConnection类发送网络请求下载图片

10.请求超时事件 == 15s

NSCache

1. NSCache是专门用来进行缓存处理的，苹果官方提供的缓存类，在AFN和SDWebImage框架中被使用，在系统内存很低是，会自动释放对象（模拟器不会释放 ）
2. NSCache是线程安全的，在多线程中，不需要对NSCache加锁
3. NSCache对key只是对对象进行strong引用，不是copy