 在iOS开发过程中，不管是做什么应用，都会碰到数据保存的问题。将数据保存到本地，能够让程序的运行更加流畅，不会出现让人厌恶的菊花形状，使得用户体验更好。下面介绍一下数据保存的方式：

1.NSKeyedArchiver：采用归档的形式来保存数据，该数据对象需要遵守NSCoding协议，并且该对象对应的类必须提供encodeWithCoder:和initWithCoder:方法。前一个方法告诉系统怎么对对象进行编码，而后一个方法则是告诉系统怎么对对象进行解码。例如对Possession对象归档保存。

定义Possession：

@interface Possession：NSObject<NSCoding>{//遵守NSCoding协议

       NSString \*name;//待归档类型

}

@implementation Possession

-(void)encodeWithCoder:(NSCoder \*)aCoder{

            [aCoder encodeObject:name forKey:@"name"];

}

-(void)initWithCoder:(NSCoder \*)aDecoder{

            name=[[aDeCoder decodeObjectforKey:@"name"] retain];

}

归档操作：

如果对Possession对象allPossession归档保存，只需要NSCoder子类NSKeyedArchiver的方法archiveRootObject:toFile: 即可。

NSString \*path = [self possessionArchivePath];

[NSKeyedArchiver archiveRootObject:allPossessions toFile: path ]

解压操作：

同样调用NSCoder子类NSKeyedArchiver的方法unarchiveRootObject:toFile: 即可

allPossessions = [[NSKeyedUnarchiver unarchiveObjectWithFile:path] retain];

缺点：归档的形式来保存数据，只能一次性归档保存以及一次性解压。所以只能针对小量数据，而且对数据操作比较笨拙，即如果想改动数据的某一小部分，还是需要解压整个数据或者归档整个数据。

2.NSUserDefaults：用来保存应用程序设置和属性、用户保存的数据。用户再次打开程序或开机后这些数据仍然存在。NSUserDefaults可以存储的数据类型包括：NSData、NSString、NSNumber、NSDate、NSArray、NSDictionary。如果要存储其他类型，则需要转换为前面的类型，才能用NSUserDefaults存储。具体实现为：

保存数据：

NSUserDefaults \*defaults =[NSUserDefaults standardUserDefaults];  
NSString \*name =@”default string“;  
[defaults setObject:firstName forKey:@"name"];  
  //获得UIImage实例  
  
 UIImage \*image=[[UIImage alloc]initWithContentsOfFile:@"photo.jpg"];  
  
 NSData \*imageData = UIImageJPEGRepresentation(image, 100);//UIImage对象转换成NSData  
  
 [defaults synchronize];//用synchronize方法把数据持久化到standardUserDefaults数据库

读取数据：

NSUserDefaults \*defaults =[NSUserDefaults standardUserDefaults];  
NSString \*name = [defaults objectForKey:@"name"];//根据键值取出name  
NSData \*imageData = [defaults dataForKey:@"image"];  
UIImage \*Image = [UIImage imageWithData:imageData];//NSData转换为UIImage

3. Write写入方式：永久保存在磁盘中。具体方法为：

第一步：获得文件即将保存的路径：

NSArray \*documentPaths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  NSUserDomainMask,YES);//使用C函数NSSearchPathForDirectoriesInDomains来获得沙盒中目录的全路径。该函数有三个参数，目录类型、he domain mask、布尔值。其中布尔值表示是否需要通过～扩展路径。而且第一个参数是不变的，即为NSSearchPathDirectory 。在IOS中后两个参数也是不变的，即为：NSUserDomainMask 和 YES。  
 NSString \*ourDocumentPath =[documentPaths objectAtIndex:0];  
  
还有一种方法是使用NSHomeDirectory函数获得sandbox的路径。具体的用法为：  
  
NSString \*sandboxPath = NSHomeDirectory();  
// Once you have the full sandbox path, you can create a path from it，但是不能在sandbox的本文件层上写文件也不能创建目录，而应该是此基础上创建一个新的可写的目录，例如Documents,Library或者temp。  
NSString \*documentPath = [sandboxPath  
            stringByAppendingPathComponent:@"Documents"];//将Documents添加到sandbox路径上，具体原因前面分析了！

这两者的**区别**就是：**使用NSSearchPathForDirectoriesInDomains比在NSHomeDirectory后面添加Document更加安全。因为该文件目录可能在未来发送的系统上发生改变。**

第二步：生成在该路径下的文件：

NSString \*FileName=[documentDirectory stringByAppendingPathComponent:fileName];//fileName就是保存文件的文件名

第三步：往文件中写入数据：

[data writeToFile:FileName atomically:YES];//将NSData类型对象data写入文件，文件名为FileName

最后：从文件中读出数据：

NSData data=[NSData dataWithContentsOfFile:FileName options:0 error:NULL];//从FileName中读取出数据  
4. SQLite：采用SQLite数据库来存储数据。SQLite作为一中小型数据库，应用ios中，跟前三种保存方式相比，相对比较复杂一些。还是一步步来吧！

第一步：需要添加SQLite相关的库以及头文件：在项目文件的Build Phases下，找到Link Binary Library(ies)，添加libsqlite3.0.dylib（libsqlite3.dylib与前者的区别暂时不知，两者应该差不多）；在项目文件中头文件或者源文件中添加头文件#import "/usr/include/sqlite3.h"

第二步：开始使用SQLite：

使用前注意：**如果不往数据库里面添加任何的表，这个数据库等于没有建立，不会在硬盘上产生任何文件，如果数据库已经存在，则会打开这个数据库**。

NSArray \*documentsPaths=NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask , YES);  
NSString \*databaseFilePath=[[documentsPaths objectAtIndex:0] stringByAppendingPathComponent:@"mydb"];  
//上面两句已经比较熟悉了吧！   
//打开数据库  
if (sqlite3\_open([databaseFilePath UTF8String], &database)==SQLITE\_OK) {   
        NSLog(@"sqlite dadabase is opened.");   
}  
else{ return;}//打开不成功就返回

在打开了数据库的前提下，如果数据库没有表，那就开始建表了哦！  
char \*error;   
const char \*createSql="create table(id integer primary key autoincrement, name text)";  
if (sqlite3\_exec(database, createSql, NULL, NULL, &error)==SQLITE\_OK) {   
        NSLog(@"create table is ok.");   
}  
else  
{  
       NSLog(@"error: %s",error);  
       sqlite3\_free(error);//每次使用完毕清空error字符串，提供给下一次使用  
}

建表完成之后，就开始插入记录：

const char \*insertSql="insert into a person (name) values(‘gg’)";   
if (sqlite3\_exec(database, insertSql, NULL, NULL, &error)==SQLITE\_OK) {   
        NSLog(@"insert operation is ok.");   
}  
  
else  
{  
       NSLog(@"error: %s",error);  
       sqlite3\_free(error);//每次使用完毕清空error字符串，提供给下一次使用  
}

下一步，查询记录：

const char \*selectSql="select id,name from a person";   
sqlite3\_stmt \*statement;   
if (sqlite3\_prepare\_v2(database,selectSql, -1, &statement, nil)==SQLITE\_OK) {   
        NSLog(@"select operation is ok.");   
}  
else  
{  
       NSLog(@"error: %s",error);  
       sqlite3\_free(error);  
}   
while(sqlite3\_step(statement)==SQLITE\_ROW) {   
int \_id=sqlite3\_column\_int(statement, 0);   
NSString \*name=(char\*)sqlite3\_column\_text(statement, 1);   
NSLog(@"row>>id %i, name %s",\_id,name);   
}  
sqlite3\_finalize(statement);

最后，关闭数据库：

***sqlite3\_close(database);***

注意：写入数据库，字符串可以采用char方式，而从数据库中取出char类型，当char类型有表示中文字符时，会出现乱码。这是因为数据库默认使用ascII编码方式。所以要想正确从数据库中取出中文，需要用NSString来接收从数据库取出的字符串。