**KS BLOCK设计与使用文档**v1.2 by 海平

**设计思路：**

KS BLOCK依赖于系统基础消息机制（见IOS版本k歌二期设计文档）和IOS的GCD机制，结合gcc的BLOCK支持，通过一系列宏，将一段代码封装成一个BLOCK，通过一个特定消息id，交由消息管理器进行同步、异步、延时分发，最终在主线程执行，或者交由特定的线程（GCD队列或者NSThread）执行。

**同步、异步、延时BLOCK：**

一个KS BLOCK宏被预处理之后的结构类似于（不同类型BLOCK略有差异）：

dispatch\_block\_t \_\_func=^(void)

{

@autoreleasepool {

。。。。。。需要执行的代码主体

}

};

SYN\_NOTIFY(OBSERVER\_ID\_RESERVE, MAGIC\_NUM,Block\_copy(\_\_func));//或者异步延时NOTIFY

在MessageObserver主线程分发的时候：

if (msg==OBSERVER\_ID\_RESERVE && iMethod==KS\_BLOCK\_RUN\_MAGIC\_NUM)

{

dispatch\_block\_t \_\_func=vecParams[0];

\_\_func();

Block\_release(\_\_func);

}

这样借助于系统消息机制一个BLOCK就以同步、异步、延时执行。

**线程BLOCK：**

一个KS BLOCK宏被预处理之后的结构类似于：

dispatch\_block\_t \_\_func=^(void)

{

@autoreleasepool {

。。。。。。需要执行的代码主体

}

};

dispatch\_async(dispatch\_get\_global\_queue(priority,0),\_\_func);

**详细构成：**

主要由几个宏构成，位于Base/Core/Block.h文件中。该文件默认被MessageManager.h包含，而MessageManager.h在大多数文件里都需要，所以此时不必单独包含Block.h。

1. #define KS\_BLOCK\_DECLARE

声明一个用于执行的代码块。

2. #define KS\_BLOCK\_SYNRUN()

在主线程中执行代码块，同步执行，当前线程会等待代码执行完毕之后再继续执行。

3. #define KS\_BLOCK\_ASYNRUN(delay)

在主线程中执行代码块，异步执行，可以通过delay设置延时，无论delay为多少，当前线程均不会被阻塞。此操作可以代替部分timer。

4. #define KS\_BLOCK\_RUN\_THREAD(priority)

在非主线程里执行代码，priority为优先级，可以不写，默认DEFAULT级别，该方法其实是在系统默认的线程队列里执行的，根据优先级不同，放到不同的优先级队列里。此操作可以用在需要起线程的大多数地方，但不提供对线程的控制，比如挂起和强杀等。

优先级类型：

KS\_BLOCK\_PRIORITY\_HIGH

KS\_BLOCK\_PRIORITY\_DEFAULT

KS\_BLOCK\_PRIORITY\_LOW

5.#define KS\_BLOCK\_RUN\_TARGET\_THREAD(Pnsthread)

在特定线程里执行代码，参数为一个NSThread对象指针，通过performSelector onThread执行，要求目标线程有消息循环。

**Block支持无限嵌套。**

**例子：**

void test()

{

int a=100;//在调用test函数的线程里执行

KS\_BLOCK\_DECLARE //声明一个BLOCK，用来在线程里执行

{//现在是在一个低优先级线程里了

KS\_BLOCK\_DECLARE //声明一个BLOCK，用来在主线程里延时执行

{//现在进入主线程

m\_pView->SetNum(a);//在主线程执行

}

KS\_BLOCK\_ASYNRUN(100);//执行BLOCK，异步，延时100毫秒

KS\_BLOCK\_DECLARE //声明一个BLOCK，异步到主线程里执行

{

m\_pView->NeedUpdate();//操作界面需要在主线程执行

}

KS\_BLOCK\_RUN\_TARGET\_THREAD([NSThread mainThread]);//指定线程

//与KS\_BLOCK\_ASYNRUN(0)等价

}

KS\_BLOCK\_RUN\_THREAD(KS\_BLOCK\_PRIORITY\_LOW)// 执行BLOCK，，低优先级线程

……

}

**另一个常用的场景例子：**

void testFuncInMainThread(SONGLIST& lst) //在主线程里

{

string url=list.getUrl();

KS\_BLOCK\_DECLARE //声明BLOCK用来线程中执行

{

string file =HTTP.Download(url);//下载不能阻塞主线程，要放到线程里。

KS\_BLOCK\_DECLARE

{

lst.filepath=file; //结果放回lst，因为lst非线程安全，回主线程

}

KS\_BLOCK\_SYNRUN() //阻塞直到BLOCK在主线程里执行完毕

... do other thing in thread

}

KS\_BLOCK\_RUN\_THREAD() //在一个默认优先级线程里执行

}

下载和之后的清理操作需要在非主线程里异步执行，下载结束修改界面或者数据结构需要在主线程执行，如果用传统的多线程编程实现相当麻烦，代码分散在代码文件的不同位置，也增加了阅读维护成本，KS BLOCK很好的解决了这个问题。

**注：**

1. 默认在BLOCK外的所有c/c++类型，如果在BLOCK内访问，编译器会自动复制一份const版本，此时原始变量修改结果在BLOCK内不会得到体现，同时因为是复制得到的是const版本，在BLOCK内不允许修改。在变量声明前面加上\_\_block 可以改变此行为，加上\_\_block之后，BLOCK内直接访问变量，不会复制，要注意线程中执行的BLOCK访问BLOCK外部栈上的变量可能导致崩溃，因为可能此时栈上变量已经离开作用域而析构。
2. 默认在BLOCK外的栈上的object-c对象，如果在BLOCK内访问，进入BLOCK时引用计数会自动加1，BLOCK执行完毕的时候会自动减1，如果想改变此行为，请在变量声明前面加上\_\_block。
3. 默认情况下，如果BLOCK在某个object-c类的成员函数里，如果在BLOCK内访问该类的对象的成员变量，在进入BLOCK时对象的引用计数会加1，BLOCK执行完毕会自动减1。