一、特性

1. 资源自动管理。在一定时间没有引用的资源，会被销毁。
2. 资源引用计数：加载引用和实例化引用。
3. 资源最近访问时间。
4. 资源销毁策略：手动，自动（没有引用，且一段时间内不被使用）
5. 协程并发限制

二、协程数据结构

1. 概述

并发的协程数量可以控制。

1. 数据结构
2. eCorroutineType:协程类型

{

Low, //并发协程数量较少，任务有可能放置在队列中

Normal, //并发协程数量较有限制，任务有可能放置在队列中

High, //并发协程数量较有限制，任务有可能放置在队列中

Emergency, //并发协程数量没有限制

}

1. CorroutineTask:执行或者等待的任务。

int Id:任务id

virtual void OnLoaded();加载成功后的回调

{

}

1. virtual void OnLoaded():加载成功后的回调
2. 实现细节

（1）CorroutineTask:: OnLoaded

1. S

三、资源管理数据结构

1. ResLoadTask //继承自CorroutineTask, 资源加载的任务，加载完毕后回调Resource

{

Resource res; //对资源的引用。

overide void OnLoaded();//加

}

1. ResUseCallback：资源使用的回调函数类型 delegate void ResUseCallback(List<Obj>);
2. eResUseCallbackStrategy// 资源加载后的回调策略

{

Immediate，//立刻回调

Delay, // 延迟回调，由管理的回调算法决定回调实际，一般是FILO

}

1. eResClearStrategy// 资源加载后的回调策略

{

Immediate，//立刻回调

Delay, // 延迟回调，由管理的回调算法决定回调实际，一般是FILO

}

1. eResStatus //资源状态

{

Loading，//加载中

Success, //加载完毕

}

1. LoadRef //对资源的加载引用，在加载后，会根据实例化策略转化为实例化引用

{

ResUseCallback callback; //加载充公后的回调，这里

int refId;//同意资源下的引用id。

}

1. InsRef //对资源的实例化引用，是由延迟实例策略产生的。

{

ResUseCallback callback；

int refId;//同意资源下的引用id。

Resource res;

}

1. Resource: 资源数据结构，对应一个已经加载或者正在加载中的资源。

{

float lastestVivitTime;//最近访问时间

eResStatus staus;//资源状态，加载中or加载完毕。

int loadTaskId;//加载任务

Dictionary<int, LoadRef> loadRefDict;//加载请求词典

Dictionary <int, InsRef> insRefDict;//实例化请求词典

int nextRefId;//对该资源访问的内部id最大值

int CreateLoadRef(ResUseCallback callback, ResUseCallbackStrategy callbackStrategy);

int CreateInsRef(ResUseCallback callback);

List<UnityEngine.Object> objlist;

AddObj(UnityEngine.Object obj);

}

1. Map<string,int> pathIDDict;资源路径和资源id的词典。
2. Map<int,Resouce> idResourceDict; 资源id和资源的数据的词典。
3. List< InsRef > delayInsList //实例化队列
4. ResourceManager//资源管理器类

四、资源管理逻辑

1. 请求资源加载

输入：资源路径，使用回调，协程类型，资源清理类型，实例化策略。

输出：资源id

Int LoadResource:

string path,

ResUseCallback callBack

eCorroutineType corroutineType,

eResClearStrategy clearStrategy

eResUseCallbackStrategy insStrategy

算法逻辑

2. 创建新的资源并加载

输入

2. 取消资源加载

输入：资源id

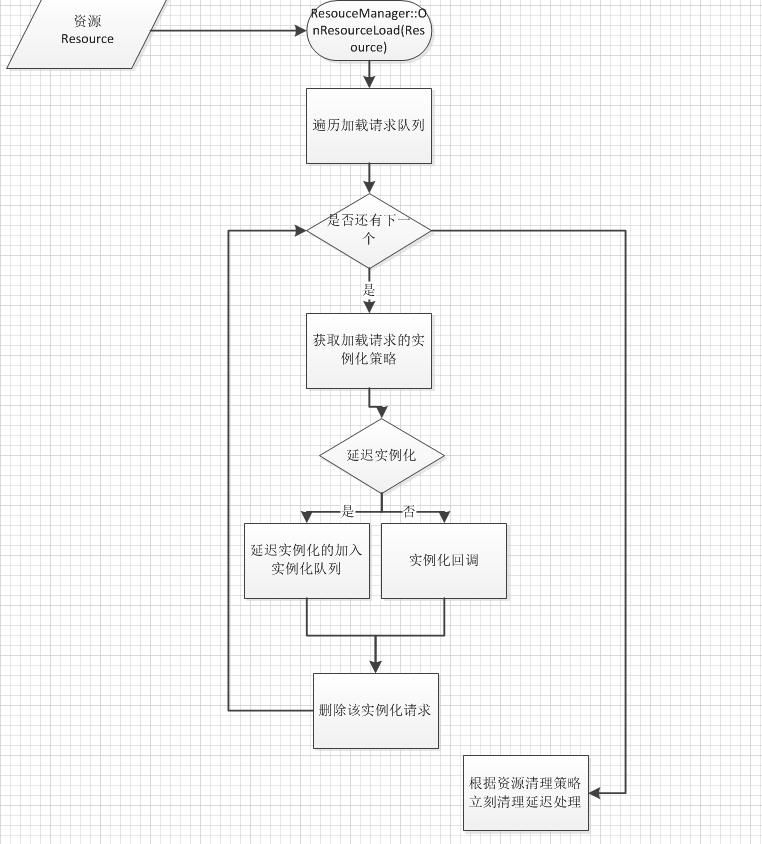
输入：是否成功(0成功)

Int ReleaseResource: int resID

算法流程图

3. 资源加载回调

输入：Resource（ResLoadTask的回掉）



4. 资源管理（内存）

核心逻辑

Int Resource：：CreateLoadRef(ResUseCallback callback, ResUseCallbackStrategy callbackStrategy)

nextRefId++;

LoadRef ref = new LoadRef(nextRefId ,callback, callbackStrategy);

loadRefDict[nextRefId] = ref;

return ref.Id;

Int LoadResource:

string path,

eCorroutineType corroutineType,

eClearStrategy clearStrategy

eResUseCallbackStrategy insStrategy

ResUseCallback callBack

加载中或者已经加载完毕

If pathIDDict.contians(path)

nResId = pathIDDict[path]

Resource res= idResourceDict [nResId];// 获取资源

res.UpdateVisitTime();

nRefId =0;

If(res.IsLoading)

{

//加载中

nRefId = res. CreateLoadRef(callback,insStrategy);

}

else

{

//已经加载完毕

If(insStrategy == eResUseCallbackStrategy.Immediate)

{

nRefId = 0;

nResId = 0;

Callback(res.objlist)

}

Else

{

nRefId = res .CreateInsRef(CallBackFunc);

}

}

return nResId + refid ；

else

return NewSource(nResId ,path, clearStrategy, insStrategy );

endif

创建新的资源并加载

NewSourceToLoaD

String path

enumCortinueType corroutineType,

enumClearStrategy clearStrategy

enumInstantaiteStrategy insStrategy

CallBackFunc callBack

nResId = NextRefId();

Resouce res = new Resouce(nResId, path, corroutineType, clearStrategy, insStrategy, callback)

pathIDDict[path] = RescourdId;

idResourceDict [RescourdId] = res;

res. loadTask = LoadingCorroutine.singleton.GetLoadSession(corroutineType).Load(LoadTask);

refid = res. CreateLoadRef(callback,insStrategy);

return nResId + refid;

实例化相关

Instantiate

Resource res, CallBackFunc callback, enumInstantaiteStrategy insStrategy

If( insStartegy == Immediate)

Callback(res)

If( insStartegy == Delayed)

AddInstanceQueue();