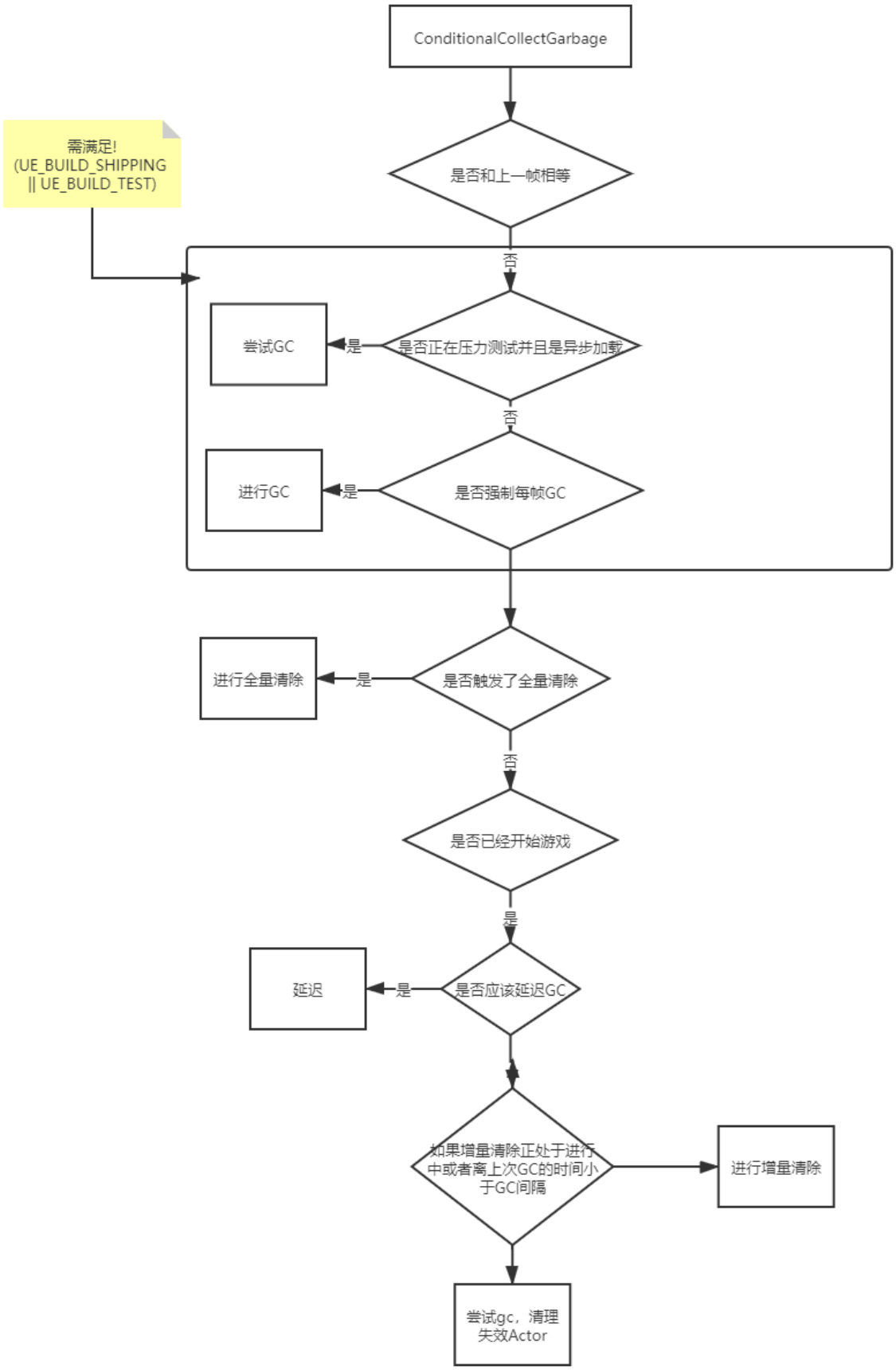
入口

通过world的tick，每帧调用：

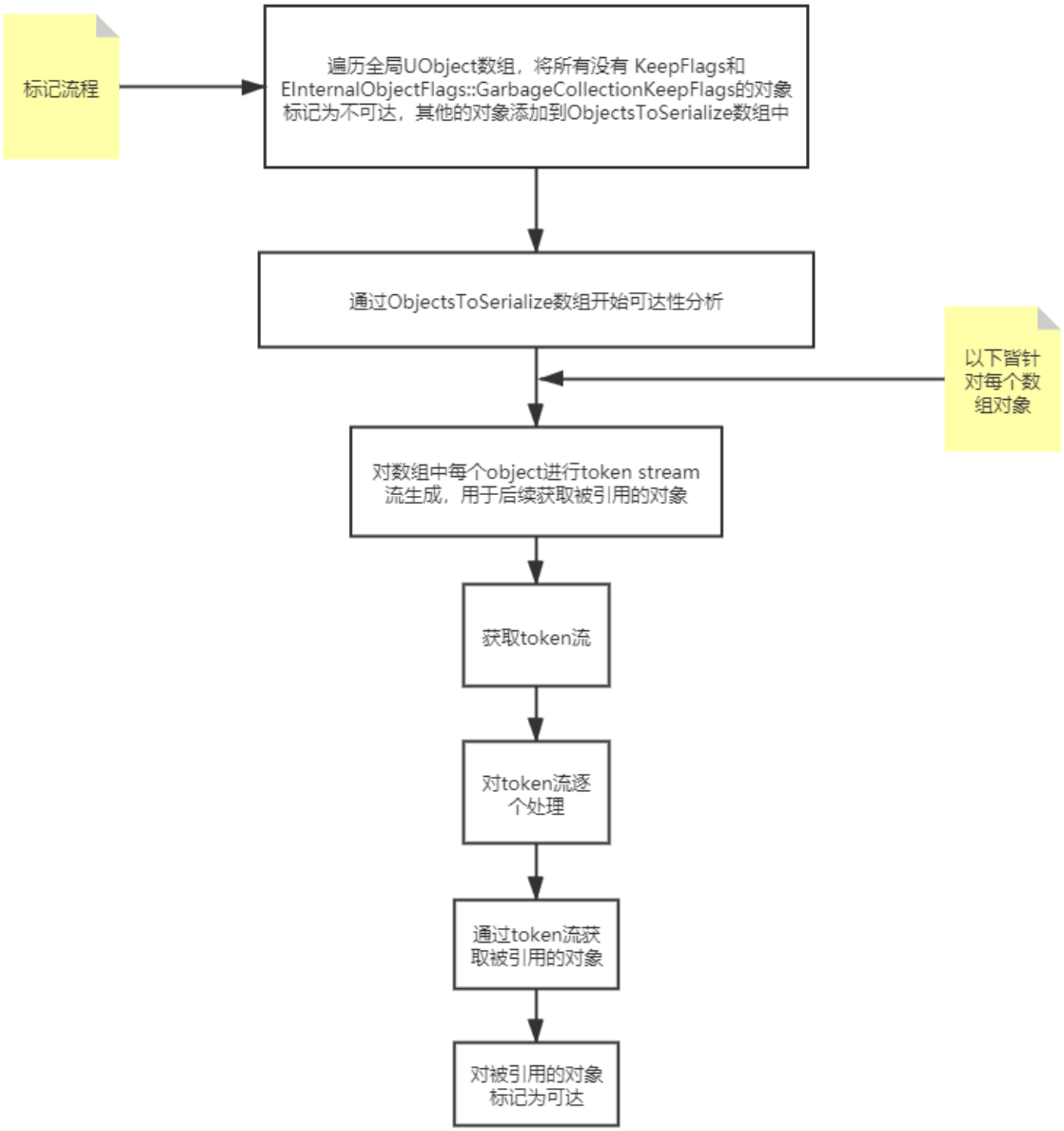


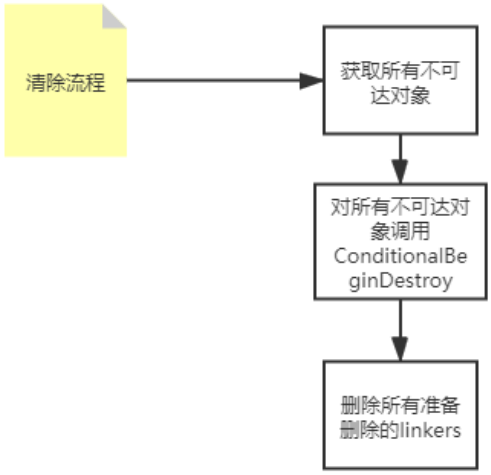
CollectGarbage

```
void CollectGarbage(EObjectFlags KeepFlags, bool bPerformFullPurge)
{
    // No other thread may be performing UObject operations while we're running
    AcquireGCLock();

    // Perform actual garbage collection
    CollectGarbageInternal(KeepFlags, bPerformFullPurge);

    // Other threads are free to use UObjects
    ReleaseGCLock();
}
```

标记和清除流程

标记是一次性的，清除可以是增量性的。





关于token流的获取和使用：

gc中对每个object产生标记流，用以保存必要的引用信息，用于后续的被引用的uobject的获取，

```
1 /**
```

```
 * Convenience struct containing all necessary information for a reference.
 */
//包含了所有必要的引用信息
struct FGCReferenceInfo
{
    FORCEINLINE FGCReferenceInfo( EGCReferenceType InType, uint32 InOffset )
    :   ReturnCount( 0 )
    ,   Type( InType )
    ,   Offset( InOffset )
    {
        check( InType != GCRT_None );
        check( (InOffset & ~0x7FFFF) == 0 );
    }
    FORCEINLINE FGCReferenceInfo( uint32 InValue )
    :   Value( InValue )
    {}
    FORCEINLINE operator uint32() const
    {
        return Value;
    }
    union
    {
        struct
        {
            uint32 ReturnCount  : 8;//嵌套深度
            uint32 Type         : 5;//类型
            uint32 Offset       : 19;//偏移量
        };
        uint32 Value;
    };
};
```

```
/**
 * Reference token stream class. Used for creating and parsing stream of
object references.
 */
//引用的token流 用来创建和分析object的引用流
struct FGCReferenceTokenStream
{
        /** Token array */
    TArray<uint32> Tokens;//存为线性数组
}
```

token的添加

对每个objectToSerialize中的对象，通过其ClassPrivate生成其token流

可以看到此处需要反射的支持，遍历类中的所有属性，将每个属性的信息添加到token数组中，用的时候取出来。

```
UClass* ObjectClass = CurrentObject->GetClass();
ObjectClass->AssembleReferenceTokenStream();
```

```
for( TFieldIterator<FProperty> It(this,EFieldIteratorFlags::ExcludeSuper);
It; ++It)//遍历objectclass中的所有属性信息
```

```
{
    FProperty* Property = *It;
    Property->EmitReferenceInfo(*this, 0, EncounteredStructProps);
}

void FObjectProperty::EmitReferenceInfo(UClass& OwnerClass, int32
BaseOffset, TArray<const FStructProperty*>& EncounteredStructProps)
{
    FGCReferenceFixedArrayTokenHelper FixedArrayHelper(OwnerClass,
BaseOffset + GetOffset_ForGC(), ArrayDim, sizeof(UObject*), *this);
    //将类型和偏移量存入token中
    OwnerClass.EmitObjectReference(BaseOffset + GetOffset_ForGC(),
GetFName(), GCRT_Object);
}
void UClass::EmitObjectReference(int32 Offset, const FName& DebugName,
EGCReferenceType Kind)
{
    FGCReferenceInfo ObjectReference(Kind, Offset);
    //将ObjectReference作为int型传入token流，objectreference由union类型，int32的
前8位，中5位，后19位对应不同的信息。
    ReferenceTokenStream.EmitReferenceInfo(ObjectReference, DebugName);
}
int32 FGCReferenceTokenStream::EmitReferenceInfo(FGCReferenceInfo
ReferenceInfo, const FName& DebugName)
{
    int32 TokenIndex = Tokens.Add(ReferenceInfo);
    return TokenIndex;
}
```

token的获取

```
FGCReferenceTokenStream* RESTRICT TokenStream = &CurrentObject->GetClass()-
>ReferenceTokenStream;
FGCReferenceInfo ReferenceInfo = TokenStream-
>AccessReferenceInfo(ReferenceTokenStreamIndex);
FORCEINLINE FGCReferenceInfo AccessReferenceInfo( uint32 CurrentIndex ) const
{
    return Tokens[CurrentIndex];
}
```

可以看到，获取到的int值直接转换为FGCReferenceInfo类型，此类型支持int的隐式转换。

token的使用

```
switch(ReferenceInfo.Type){
case GCRT_Object:
UObject**    ObjectPtr = (UObject**)(StackEntryData + ReferenceInfo.Offset);
UObject*&    Object = *ObjectPtr;
}
```

通过所获取的info中的偏移量，获取对应uobject中的对象引用，可以对此引用进行可达性标记了！

优化

Cluster

通过Cluster减少标记时间，将一些声明周期相同的对象声明为一个cluster，这样对其进行标记的时候可以作为一个整体进行。

作为整体的话，只要检测到根不可达或可达，直接就将整个簇中的成员标记，即检查集群本身，而不是每个对象，加速垃圾回收。可以将整个集群视为一个对象，但是这也意味着这些对象将在一帧被删除，如果集群较大，可能会有麻烦。一般来讲，可以提高垃圾收集性能。

增量清除

每次删除只用固定的时间，到了时间就会退出。

多线程

标记和引用分析都使用了多线程。

内存池

`FGCArrayPool`,减少GC分配。