**触发系统简介**

本文将简述触发系统的组成和运行机制，适合有一定触发基础的熟练用户阅读。本作品采用[知识共享署名-非商业性使用-相同方式共享 4.0 国际许可协议](http://creativecommons.org/licenses/by-nc-sa/4.0/)进行许可，转载请注明作者。

作者：Handama

最后修订：2025.8.19

1. **触发系统的术语**

在进行讨论前，需要首先统一术语。许多新人犯的错误都来自于术语理解不清晰，同时一些正在使用的术语存在模糊性，因此本文首先对他们进行定义：

**触发（Trigger）**：广义上是对整个触发系统的统称。狭义指触发编辑器中看到的那些信息，包括触发本体、事件、行为，分别对应INI中[Triggers]、[Events]、[Actions]中有相同ID（INI键，如01000002=）的内容。

**标签（Tag）**：控制触发可执行次数、将触发与其他游戏对象（小队、单位、建筑、单元标记）相关联的结构。对应INI中[Tags]的内容。触发编辑器中的“重复类型”是标签属性而不是触发属性。一个标签只能关联一个触发，但一个触发可以有多个标签。

**启动（Spring）：**触发被特定游戏内事件（函数）调用，所有条件满足，成功执行的过程。

**满足（meet）**：使一个事件为真的条件。原有术语为“……时，触发此事件”。由于触发易于前者的触发相混淆，并且“事件为真”是一个“状态”而不是一个“动作”，因此改为“……时，此事件被满足”。

**事件（Event）**：触发启动的条件。当一个触发的所有事件同时满足时，才能被启动。

**行为（Action）**：触发启动时执行的具体内容。

**下级触发（Attached Trigger）**：触发可以拥有一个下级触发，并且可以形成一个下级触发链条。该链条不能形成循环。原有术语为关联触发，但是这个词没有体现关联的方向性（丢失了英语中的被动语义），且会与将标签关联到单位、小队的操作混淆（许多新人误把关联触发当作关联标签），因此弃用。

1. **触发的基本结构**

触发系统是一个**类（class）**，也是一个**对象（object）**。类与对象是编程中的概念，在此不再赘述。总之，在INI中填写的内容都只是对触发类的定义，触发的**实例（instance）**需要在游戏内创建对象时建立。触发的基本结构可以这样表示：

[标签A]---→[触发A]---→[事件A]---→[行为A]

↓下级触发

[标签B]---→[触发B]---→[事件B]---→[行为B]

↓下级触发（无限叠加）

[标签C]---→[触发C]---→[事件C]---→[行为C]

标签是一切触发的起点。标签实例中存在一个属性[[TriggerClass\* FirstTrigger]](https://github.com/Phobos-developers/YRpp/blob/phobos-dev/TagClass.h)，指向标签关联的第一个触发。触发实例中有属性[[TriggerClass\* NextTrigger]](https://github.com/Phobos-developers/YRpp/blob/phobos-dev/TriggerClass.h)，指向与触发关联的下一个触发。若存在多个NextTrigger，他们可以组成一个下级触发链条，共用同一个标签（pTag->FirstTrigger->NextTrigger……->NextTrigger，pTag为标签实例）。在上图中，标签A同时拥有触发A、触发B、触发C三个实例，标签B拥有触发B与触发C两个实例，标签C仅拥有触发C的实例（注意，三个触发C和两个触发B的实例是独立的）。

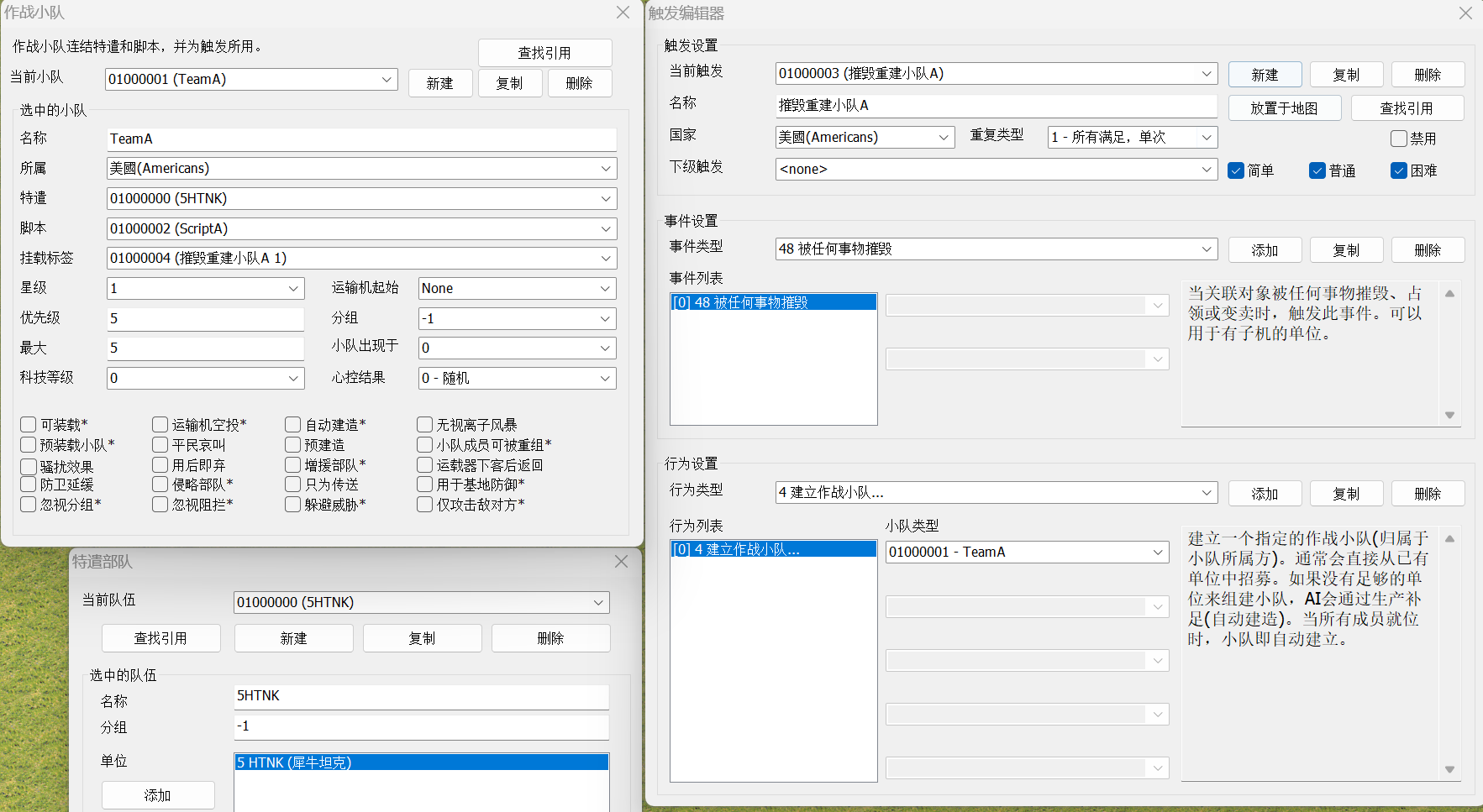
标签的重复类型有三种，分别是重复类型0（单次或，Volatile）、1（单次与，Semi-Persistent）、2（重复或，Persistent）。重复类型0、1的标签在启动一次后就会被摧毁，重复类型2的不会。如果标签关联到单位，则会有一个属性[[InstanceCount]](https://github.com/Phobos-developers/YRpp/blob/phobos-dev/TagClass.h)，记录关联单位的数量，重复类型0、2的标签会在任一单位满足条件后触发，重复类型1的标签每有一个单位满足条件，InstanceCount减1，直至InstanceCount=0时启动触发（可以参考[TriggerClass::Spring](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/TRIGGER.CPP#L274)）。若有单位在没有满足条件的情况下死亡，则InstanceCount永远无法归零。

多个触发事件会进行与运算，当触发被调用的那一瞬间，当且仅当所有触发事件同时满足时，触发才能被执行。新手常见的一个误解就是，认为标签重复类型会影响多个事件之间的运算。事件可分为瞬时事件与状态事件。

当触发启动时，触发行为会依次顺序执行。所有行为都会在启动的这一帧内执行。有部分触发行为会对触发进行操作，如[12 摧毁触发...]、[22 强制启动触发...]、[53 允许触发]、[54 禁止触发]、[70 摧毁标签...]。这其中，12、22、70都是对触发或标签实例进行操作，因此[12 摧毁触发...]的说明就很好理解：*“摧毁所有特定触发类型的当前实例。但不会阻止未来可能建立的实例。”*53、54是对触发类型的操作，效果等同于勾选触发编辑器中的“禁用”选项，对触发类型的禁用、启用会影响所有实例，既包括已存在的，也包括未来建立的。

1. **触发的实例化**

触发的实例化可分为**游戏载入时创建**与**小队创建时创建**。前者是绝大多数触发的创建方法，而后者则是小队标签所对应的触发创建的方法。只要有标签的触发，都会在游戏载入时进行创建。需要注意的是，即使这个触发的标签被关联到地图单位上，也只会创建一份实例。由于小队关联的也是标签，因此这种触发会同时在载入时与小队创建时创建。由于小队可以被多次创建，这种触发也可以拥有多个实例。以最简单的小队被摧毁重建触发为例：



在这个触发组中，由于需要确保小队成员全部被毁才重建，重复类型需要设置为1，但是，这个小队却能无限重建。这就是因为，虽然单个触发实例只能执行一次，但每次执行都会创建一个新的触发实例，不同实例间的执行次数是互相独立的。行为[12 摧毁触发...]描述中的“*未来可能建立的实例”*，指的就是通过小队建立的实例。

1. **触发的启动**

触发的启动方式可分为**一般启动、所属方启动**，**对象启动**，（可以参考[Attaches\_To](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/TEVENT.CPP" \l "L677)、[LogicTriggers](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/EXTERNS.H" \l "L251)）。一个触发具体通过哪种方式启动，取决于其事件类型。当一个触发的事件全部与所属方、单位或单元标记无关，如[13 流逝时间...]、[60 科技类型存在]等，则为一般启动；当一个触发的事件全部与所属方相关，如[9 单位全部被摧毁]、[12 金钱超过...]，则为所属方启动；对象启动可以细分为两类：**单位启动**、**单元标记启动**。虽然单元标记启动的触发需要关联到单元标记而不是单位上，但由于这两种类型实际上均由与单位有关的函数进行调用，可以将它们视为一类。如果一个触发的事件全部与单位或单元标记相关，如[6 被任一所属方攻击]、[33 被玩家选中]、[1 进入事件]、[25 越过水平线]，则对象启动。[0 -无事件-]不属于任何启动方式，**[8 任何事件]属于所有启动方式**。

如果一个对象启动的触发的标签没有关联到任何单位或小队上，这个触发永远无法启动。如果一个触发同时包含多种方式的事件，则可以分别通过对应方式启动。如果触发实例是通过小队创建的，则触发只能通过小队成员（也就是单位）启动。

一般启动是经由游戏逻辑的某个函数执行的（可以参考[LogicClass::AI](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/LOGIC.CPP" \l "L209)），此函数调用为**每帧一次**。对于一个事件只有[13 流逝时间...]的重复类型为2的触发，会每帧都执行一次。通过所属方的启动是经由所属方的某个函数执行的（可以参考[HouseClass::AI](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/HOUSE.CPP" \l "L1235)），此函数调用为**每8帧一次**，根据所属方列表从上而下依次执行。所以对于一个事件为[12 金钱超过...]的重复类型为2的触发，会每8帧执行一次。如果有两个所属方有事件完全相同的触发，则位于列表靠前位置的所属方的触发会优先执行。对于一个事件只有[8 任何事件]的重复类型为2的触发，由于该事件属于所有启动方式，会在每帧执行一次的基础上，每8帧多执行一次。通过单位和单元标记的启动是经由单位的某个函数执行的（可以参考[ObjectClass::Take\_Damage](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/OBJECT.CPP" \l "L1796)、[FootClass::Per\_Cell\_Process](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/FOOT.CPP" \l "L1472)）。这种函数并不会无条件的循环调用，而是在单位执行特定动作或经历特定事件时被调用（如单位被攻击则触发Take\_Damage()）。因此，可以把通过对象启动与通过单元标记启动粗略地归于同一类型中。

如果一个触发包含一般启动事件，它在实例化时会被加入[LogicTriggers](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/EXTERNS.H" \l "L251)列表；如果一个触发包含所属方启动事件，它在实例化时会被加入对应所属方的[HouseTriggers](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/EXTERNS.H" \l "L253)列表；包含对象启动事件的触发实例并不会被加入到特定列表中，他们的实例化依赖于标签关联的单位、小队或单元标记。也就是说，即使触发事件是[0 -无事件-]，甚至完全不存在事件，只要标签被关联到某一对象上，该触发就可以被实例化。触发的启动方式，本质上就是触发在哪个函数中被调用，在调用时读取的是哪一个列表。可以使用工具中附带的**ObjectInfo.dll的Dump Trigger Info功能**查看当前触发的不同列表与启动方式。

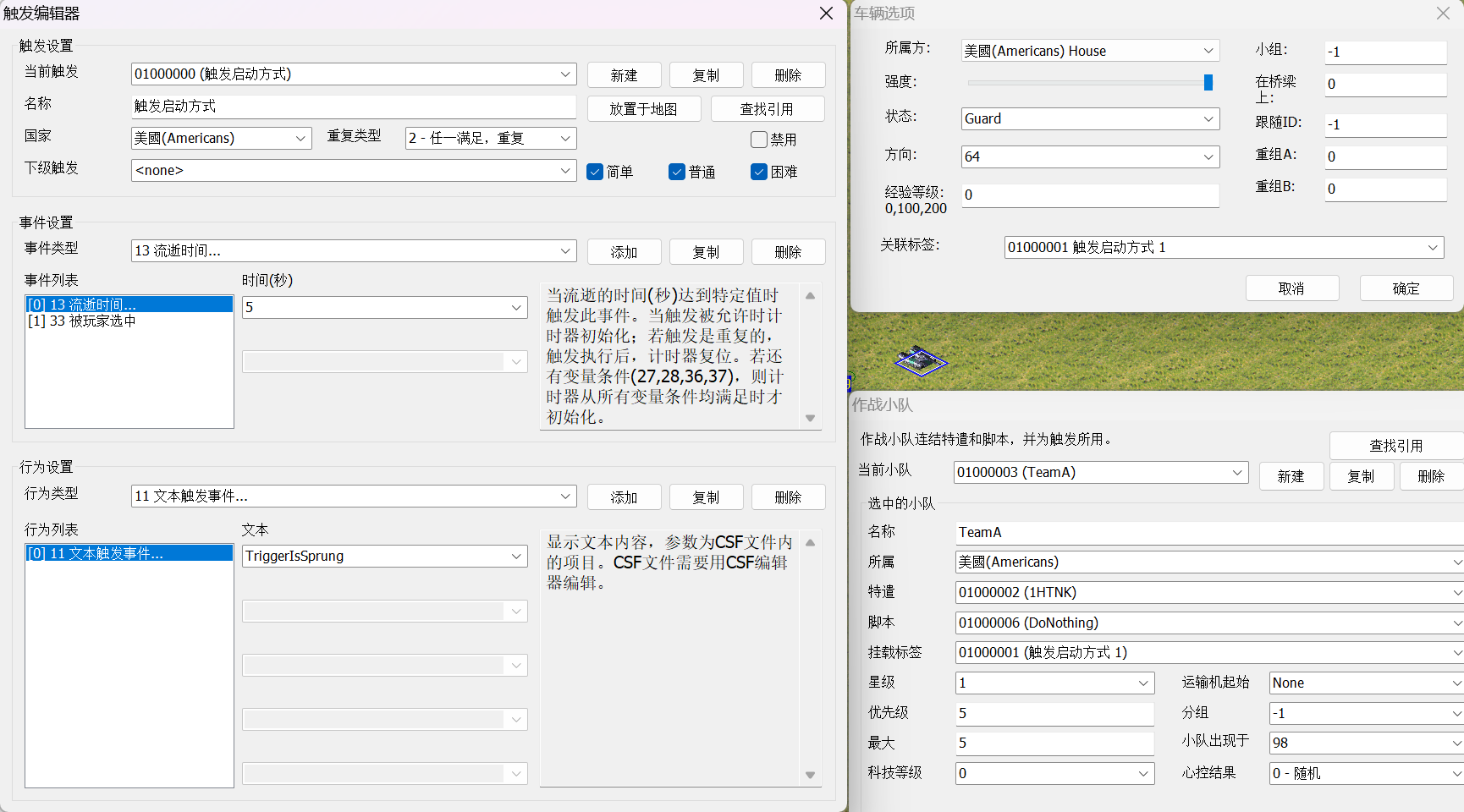
以下GIF为一般启动与所属方启动的对比（使用了Phobos的Frame by

Frame功能。同时，Yuricountry的触发ID比Americans更靠前）：

文本

描述已自动生成

以下面的触发组对多种启动方式进行详细说明：



对于上述触发，它同时具有两种创建方式，（1）游戏载入时被创建一次，这个实例的标签被关联在了灰熊坦克上。由于这个触发同时包含了一般启动事件与对象启动事件，它可以通过这两种方式进行启动。也就是说，这个单位可以被与所属方有关的函数和与单位有关的函数分别调用。（2）犀牛坦克小队被创建时也创建了一次，这个实例的标签被关联在犀牛坦克上，只能通过对象启动。最后，还需要记住事件[33 被玩家选中]的描述：*“首次触发后，若对象受到任何「扰动」(移动、受到伤害、被选中等...)，也会被满足。”*在了解以上知识的情况下，进入游戏实验：

等待5秒钟以上的时间，然后选中灰熊坦克。可以发现，触发会首先立即执行一次，然后无论灰熊坦克处于什么状态，如静止、移动、被攻击、乃至已经被摧毁，这个触发均会每隔5秒执行一次。虽然小队创建的犀牛坦克也关联到了同一个触发类型（注意是类而不是对象），并且创建5秒后的首次选中也会立即执行一次，但后续执行则需要犀牛坦克受到「扰动」才行被执行。

从这个例子可以看出，灰熊坦克对应的触发实例会通过一般与单位两种方式启动。第一次选中立即执行，说明是单位被选中的函数调用了此触发。由于[33 被玩家选中]在首次满足后就恒为真，而后续的启动是一般启动，因此无论灰熊坦克处于什么状态，触发都能正常启动。犀牛坦克对应的触发实例则只能通过对象启动，因此后续也必须在单位受到「扰动」，也就是各种对应的单位函数执行时才会启动。

1. **持续事件与非持续事件**

本段翻译自[Ares说明文档](https://ares-developers.github.io/Ares-docs/new/triggerevents.html)：持续（persistent）事件只需满足一次，游戏就会记住它们的发生。因此，非持续事件只能在事件启动的瞬间发生。如果整个触发实际上是通过该事件来满足的，那么该事件就是伴随（incidental）事件。伴随事件发生在特定的时间点，并与正在发生的某些特定操作相关联。例如，[1 进入事件]在单位进入建筑物时触发。如果一个事件取决于游戏的情境，并且在条件当前为真的任何时候都能满足，那么它就是情境（situational）事件。也就是说，条件通常会在一段时间内满足，并有可能随时间发生变化。例如 [17 不再有工厂]和[30 电力不足...]。这意味着包含两个**非持续伴随事件**的触发永远不会启动，因为这两个事件是相继发生的，而不是同时发生的，因此永远不可能同时满足条件。这也意味着，如果一个触发器包含一连串事件，其中有一个非持续伴随事件，那么它必须是最后一个被满足的事件，才能启动触发并执行行为。

Ares文档中提到的“伴随事件”，指的就是前文对象启动的事件。例如，从程序上讲，一个单位不可能在被攻击的同时进入特定单元格。即使他们是在同一帧发生的，但在程序执行中，仍然分别对应着被攻击函数与移动函数，这两个函数分别激活对应的事件，而两个函数是无法同时执行的。

所有一般启动与所属方启动的事件都是持续事件。然而，并不是所有对象启动事件都是非持续事件。例如，[4 关联对象被玩家发现]、[33 被玩家选中]、[34 特定对象到达路径点附近]、[38 首次受损(仅战斗伤害)]等，在第一次满足条件后恒成立，此时单位若受到「扰动」，仍会启动触发，因此此类触发的重复类型应尽量避免设置为2。

1. **强制启动触发**

执行[22 强制启动触发...]时，该行为本身会直接寻找并启动对应触发的实例，并且**不消耗重复类型次数**。但如果触发本身被禁止，或触发类型当前并不存在任何实例，就无法被强制执行。因此，一个触发无法被强制启动的原因也仅有这两种情况。对于第二种情况，结合上述知识，可以总结出**一个触发类型不存在实例的原因**：（1）重复类型为0或1的触发实例已经启动，该实例被摧毁；（2）触发没有一般启动或所属方启动的事件，并且标签没有被关联到任何对象上，因此未能成功实例化。

因为强制启动触发是在行为中执行的，因此被启动的触发会进行插队，并且早于[22 强制启动触发...]的后续行为。又因为不消耗重复类型次数，因此强制启动触发不能启动自身，否则这个触发会在这一帧内无限循环。

1. **触发执行时序总结**

经过上述讨论，可以解决触发的执行时序问题（多个触发在同一帧满足条件时，执行先后的问题）：对于不同启动方式的触发，一般来说**一般启动优先于对象启动优先于所属方启动**。对于同一启动方式，触发会按照遍历顺序依次执行。

对于一般启动，程序会遍历[LogicTriggers](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/LOGIC.CPP" \l "L209)列表，该列表是加载地图时依据[Tags]小节的顺序自上而下读取的。在绝大多数情况下，这意味着ID越大的触发顺序越靠后。但是，程序在处理触发实例时存在一个漏洞。若触发的重复类型为0或1，在触发启动后会删除触发实例（可以参考[TriggerClass::Spring](https://github.com/electronicarts/CnC_Remastered_Collection/blob/f1f0d42bc2dcd06d5d1df943c6150ab34bf307ae/REDALERT/TRIGGER.CPP" \l "L339)），同时删除列表中对应的引用。这就导致当列表读取第n+1项时，实际上读取的是原先第n+2项的内容。这导致，如果有ABCDE五个重复类型为0或1的一般启动触发在同一帧满足，这一帧实际上仅有ACE能够执行，BD被跳过；而到了第二帧，D又被B跳过，因此它们的实际执行顺序为(ACE)(B)(D)。重复类型为2的一般启动触发执行后不会删除实例，因此也不会导致这个bug。**如果想保证某一一般启动的触发能在每一帧都被检查，需要在LogicTriggers中此触发实例的前一位插入一个占位触发**，这样即使发生了bug，也只会跳过占位触发。这个占位触发需要添加任一一般启动事件，如[13 流逝时间...]，并且勾选禁止，保证它能够被正确实例化，同时永远不会执行。在编写触发时，需要同时建立这两个触发，保证占位触发的ID紧邻并小于第二个触发。

对于所属方启动，程序会遍历所有所属方的[HouseTriggers](https://github.com/electronicarts/CnC_Remastered_Collection/blob/7d496e8a633a8bbf8a14b65f490b4d21fa32ca03/REDALERT/EXTERNS.H#L253)列表。因此，当第一个所属方的所有触发执行完后，才会执行第二个所属方的触发。单人任务中，所属方的顺序由[Houses]小节的顺序决定。多人游戏中，所属方按照游戏者位置排序，Neutral和Special排在最后。所属方启动的触发不存在一般启动的bug，因此也不会跳过触发。

对于对象启动，程序会遍历所有单位，依据单位的UID顺序执行（可以使用**ObjectInfo.dll**的**Display Object Info**查看）。

下级触发链会按照[pTag->FirstTrigger->NextTrigger……->NextTrigger]的顺序执行。FirstTrigger反而是下级触发链中的最后一个触发，因此执行顺序与关联顺序是相反的。需要注意的是，若链条中多个触发能够在同一帧运行，且标签重复类型是0或1，删除标签实例的操作会在所有触发执行后才进行。

由于下级触发链在实例化时，会通过上级触发的标签和本触发的标签创造多个实例，在部分情况下，可能会干扰预期的效果。例如，某触发A将关联单位改所属至House A，下级触发B将关联单位改所属至House B。触发的事件不包含对象启动事件，如[13 流逝时间...]或[32 建筑存在...]。在触发B本身的标签仍然存在时，游戏会创建两个触发B实例，其中一个由触发A的标签创建，关联到单位，第二个由触发B的标签创建，没有关联到单位。为了使这个下级触发链能够重复启动，需要将触发A的标签设为重复类型2，且两个触发执行时需要禁用自身。此时，触发B的任意一个实例启动后，均会禁用触发B类型，导致另一个实例也被禁止。由于上述一般启动的bug，实际游戏中两个触发B实例的先后顺序是不确定的。此时的表现为，被关联的单位有概率不能改变所属方。因此，在大多数情况下，**建议删除下级触发链中最上级标签以外的所有标签**。