本文从设计思路、原则和技巧讲述如何使你的模组具有更好的兼容性，更好的兼容性一般也便于维护适配新版本。
假设读者已有一定的模组基础知识。
~~本文包含：一定浓度的计算机/编程术语；滥用未定义行为~~

作者：Steam@Saltfish

基本思路

让我们从几个问题开始。

1. 兼容性是什么？是的，这个词在模组制作中也能引申出不同的含义。

- 最经典的定义：和其他模组一起使用时能无冲突正常运行的能力
- 完美兼容？不崩就是成功！
- 方便制作兼容补丁在某种意义上也是兼容性好的体现
- 兼容多个版本的原版是不是兼容性？
- 与其他模组的联动是不是一种兼容？

当然，这一切在实践中是高度统一的，良好的编码习惯和文件管理策略对以上各种意义的“兼容性”都有帮助。

2. 什么样的模组不需要考虑兼容问题？

- 真空球形鸡。 兼容性是相对的，不存在100%兼容。
  即使一个模组做到：
- 完全不触及原版文件（更改冲突）
- 不依赖原版对象（避免访问被其他模组移除的原版内容）
- 也不涉及常驻UI（界面遮挡） 也仍然可能在逻辑上导致冲突。
  写错事件命名空间或ID格式导致的串ID有时也被误报为兼容性问题，不过这就是另一个故事了。

3. 完美的兼容是什么样的？

- 能单独或在只有硬依赖模组的情况下正常运行
- 在和其他模组一起运行时也能正常运行
- 在此基础上，双方对相同内容的修改按自然逻辑同时或择一生效且无冲突，对相似概念的处理能达成一致
- 并且不需要补丁，甚至不需要排序

前置知识

参考基础教程 P语言：从入门到弃坑，此处仅简单列举需要明确的特性。

加性与设置性

设置性这词我自己编的。
为角色增加威望显然是一种加性效果，其数值可加，效应可叠加。
设置角色的年龄则为设置性，多次执行只有最新的执行保留效果。

当不愿覆盖某些原始代码又希望阻止其效果时，可以通过action外挂代码回溯其效应再实现你的修改，这种情况下就必须区分这两者（可用于Vic3的历史，但真的有人坐这牢吗？） ~~点名批评Vic3的`create_building`对当前等级和指令等级取高，人不人鬼不鬼~~

PDX虚拟文件系统逻辑

游戏启动后，引擎首先读入全部原版文件，然后按DLC和模组载入顺序（Jomini游戏中，DLC的装载和模组在技术上是一致的）依次读入虚拟文件系统。
在这个过程中，相对路径一致（同名、同位置）的文件将覆盖已载入的对应文件，因此**排序靠后的模组在这种文件名覆盖策略下具有优先权**。
这是第一轮覆盖，也称为完整文件覆盖等

当虚拟文件系统确认需要访问的文件集合之后即进入第二个阶段，引擎访问每一个文件，枚举其中的条目以构建列表。
在这一阶段，引擎将对所有脚本文件**按照完整（含路径的）文件名，以字符串顺序**进行一次完整遍历，然后提取关键字。
（Jomini）游戏的两个引擎在此具有不同的行为特征：

- Clausewitz引擎（管理.txt脚本文件）用后载入数据覆盖已读入的数据
- Jomini引擎（管理.gui图形化界面文件、.yml本地化文件、.bank音频库文件等）由先载入的数据抢占阻断后载入同名数据对象的读入 较早的游戏则基本上遵循覆盖逻辑。
  各游戏均有少量特例无法采用覆盖策略运行，特别地，名为`replace`的本地化子文件夹具有高优先权。
- 事件不支持覆盖策略，游戏会对ID冲突的每个事件分别**随机**抽取版本
- 通常地图相关数据对覆盖策略的支持较差
- Vic3的历史本质是执行的效果，多为加性，不能使用覆盖策略
- CK3的角色历史具有加性特征，不能使用覆盖策略
- action具有特别的逻辑 注意，覆盖逻辑在绝大多数情况下均只支持按最高层级为单位进行覆盖，如覆盖决议必须覆盖整个决议，不能只覆盖其中的效果块。
  显然，如果只修改一个文件中的少量内容，这种策略更容易保证兼容性。

这是第二轮覆盖，称为补丁式覆盖、单条目/实体覆盖等

启用热重载时对已读入虚拟文件系统的文件进行保存也会触发读入，**这种载入忽略文件名顺序，相当于在已有数据库的基础上再次强制读入保存的文件**

空访问行为

当游戏试图访问不存在/非法的内容时，会根据具体代码归属作出不同行为。

- Clausewitz引擎中，所有无效代码会被直接忽略，因此**未定义的条件恒为真**，并在每次执行/生成提示时报错
- Jomini引擎中，所有无效代码同样会被忽略，但由于Jomini代码写在一行的特性，**整个方括号内的代码将一起被忽略**，并在每次执行/生成提示时报错 这就是UI类错误会快速填满报错日志和C盘的原因。

结合覆盖机制，我们可以很容易想到通过低优先权的文件定义占位符来避免空访问引发的问题。

兼容性技巧

排序与功能自选

当两个模组对同一个功能有不同实现，且模组之间本身没有必须的排序时，便可允许玩家通过模组排序自选使用此功能的何种版本。

文件管理

再次推销Git，保证你修改的原版文件及时合并原版更新内容对兼容性有巨大帮助。
补丁策略更难排查冲突，推荐将补丁覆盖原版的内容全部按原版文件结构分开，即一个补丁文件覆盖一个原版文件，并在文件名注明此文件是补丁以及对应原版文件名，如
`<prefix>_override_<vanilla_file_name>.txt`。
当然，避免维护原版文件最好的方法就是少碰、不碰原版文件。

- 慎重考虑是否修改更多原版文件，增加的游戏内容/实现的特性带来的吸引力是否足以抵过兼容性损失？~~（Vic3省略此条，已经没救了）~~
- 标注对原版文件的修改来自哪个模组，方便检查和适配
- 通过`gui/scripted_widgets`引入自定义UI可以避开大多数增加UI需要的修改，虽然仍然有一些Jomini对象只能在对应窗口访问且无法被Access。
- 使用外挂式action处理历史（有抢执行顺序的风险，避免过于宽泛的条件）

无感知兼容

在不涉及原版UI文件和不能补丁覆盖的文件的情况下，模组可以做到排序不敏感的兼容，也就是说以任意顺序均可正常运行。
这通常需要各模组文件名之间的协调。

载入标志

用于声明模组的载入与否，便于其他模组根据情况实现联动功能或冲突处理逻辑。

- 全局变量 （无需占位符，但在存档外不存在）
- 封装条件 （占位符恒为假，载入恒为真，早期游戏只能用这个）
- 脚本化GUI （同上）
- 宏 （同上）

封装化/接口化

对原版文件的改动尽可能封装为单一一行的接口，这样既方便他人整合改动，又方便在原版更新时快速合并更改。
在新文件中使用的代码也宜尽量封装方便复用和保证一致性，同时便于联动/子模组等情况。
常用的接口类型包括： Clausewitz:

- on_action （Jomini）
- script_value （Jomini）
- scripted_<>系列
- 达成一致的变量/旗标等 Jomini:
- data_binding （特别注意，虽然宏只在Jomini中使用，但它是.txt文件，遵循Clausewitz的后覆盖前原则）
- type
- template

特性

GUI （Jomini）

覆盖原则：

- 直接声明的属性优先级**高于**模板中的属性
- 多个模板中的重复属性，靠后的模板优先
- 调用时声明的属性优先级高于类中定义时的属性 利用这些特性的组合，可以实现载入A模组时呈现A模组的界面，载入B模组则呈现B模组的界面，两者同时载入时同时体现两者的更改，或固定显示其中一方的更改。

action （Jomini）

由于action不需要完整定义的特性，只需在需要时直接使用即可，不需要双方协调文件内容。
常用于保证逻辑的一致性，但需要特别注意，由于action的异步特性，不要将依赖执行顺序的代码用这种方式兼容。

社区生态

很重要，专门开一节。 CK3模组作者们可能会注意到，许多模组的UI文件里有很多带着方括号的注释，比如许多UniUI字样的代码——虽然已经停更了，但此模组留下了宝贵的社区遗产。
建立在互相埋入接口上的广泛跨模组兼容合作使得许多模组可以无补丁兼容，许多早期的QoL模组都得到了这样的支持。

功能设计：整合还是依赖？

一些模组会开放整合授权，甚至直接在Git托管平台上开源。
就目前而言，除了群星建立了可靠的社区依赖生态之外，其他P社游戏的模组大多倾向于整合。这与作者们对前置项目保持持续更新缺乏信心和习惯缺失不无关系。
依赖策略的优点：

- 减少维护压力（外包/不必重复处理代码，避免缺漏）
- 减少冲突风险（不必担心不同分支的不同实现导致冲突）
- 依赖项可作为兼容平台 缺点：
- 依赖项未必及时更新
- 对部分用户来说略为繁琐

另，如果整合后有较大分支改动，建议改文件名和键值，但整合后接口无改动仅改变实现则以不改为宜，以便其他模组沿用兼容信息。

社区公德

非必要不使用高权重值、大量z或0抢文件名顺序等策略。
除非你做的是万能兼容补丁之流生来就是要覆盖别人的模组，否则不要使用超过4位（原版只使用2位）字符来调整文件名载入顺序，超过2位的载入顺序调整应有明确适用对象，而非“滥用抗生素”。