Hemp light

HEMP 企业软件分析及设计系列的一部分

（完整的企业机械自动化进程）

David E. Jones 著
最后修改-------2009 年 6 月 20 日

译者：肖奕（540104231@qq.com）
校对：张振宇（zhangzy2402@gmail.com）

目录

HEMP light 简介

HEMP 是一系列用来驱动需求分析和定制开发企业级系统的完整工件集合（actifacts）。它是全面的，因为它的出发点是用来收集，组织并构思所有相关的业务需求（包括偶尔业务关联性不强的需求），并且以这些为基础来设计系统。HEMP light 是 HEMP 的简化版，适用于轻量级的，少量人员参与的项目。

如果有较多的人员参与了明确需求或协同进行设计的工作，则可能会导致这个 HEMP 的精简版留下太多未记录的细节信息，并因此造成一些问题，特别是在设计和实现这些工件的过程中。这就是为什么完整版 HEMP 要花这么多力气写这么多的用例，数据声明，测试脚本了。

在每个业务场景中，系统中客户化部分的工作最好是由最终用户组织的需求原样进行添加和修改。这种方式通常要先描述一个“业务场景”（或者借鉴编辑 OFBIZ 的统一业务过程库中已存在的通用“场景”），然后看系统覆盖了哪些已有的业务，并对通用的业务描述未覆盖的空白需求进行设计工作。当我们要重新搭建一个系统，而不是定制已经有的系统的时候，那就不要用 HEMP light 了，推荐你用 HEMP。

通常构建系统的整个过程主要思路是先做需求，然后做设计，并以此做为实现的依据。在整个过程中产生的工件集合都是为了使下一个工件变得越来越简单。

通常越早做改变，就越容易越少花钱。在每一层讨论到的，建立起来的东西要用来争对需求的某一个层面，当决定了越来越多的东西的时候，这就并且变得越来越有通用性，能应对各种细节。有时在之前的环节的基础上做一些回顾也很值得。比如说建立了一个线性流程以后就可以做一些模拟用户的测试。从这个测试的反馈，就可能可以修改一些用户界面来改进系统。尽量每设计一步，就进行一些测试反馈，这样新的设计依然能站的住脚。这样就给设计带来很大的灵活性，并且能满足自动化的要求和目标。

HEMP light 的过程

下面的这个过程是收集和组织需求的示例。这个过程和文档最后一页的工件流程图相对应。

这个简短的过程融合了很多高度抽象层的业务过程步骤以及每步的细节。如果有比较庞大的业务过程，一般会分开将步骤细节记录在不同的文档中，这些文档可以通过链接语句的方式将低层次的描述进行串联。Apache OFBiz 的 UBPL 就是一个很好的示例。

这里要注意下，我们不推荐在一个业务场景描述中提到系统交互，因为这是设计内容而不是需求文档，“系统”通常被用于特殊的场景中。区别在于这里我们所指的系统是业务描述出来需要构建的系统结构而不是呈现出来的某个业务的功能。采用这个过程，设计实现一个系统，HEMP 进程就能实现自动化。下面就是这个过程…

业务专家收集需求：

业务专家编写业务场景描述。编写业务场景时业务专家可以发现不符合业务过程的需求。争对每一个不属于任何业务过程的需求，业务专家应该都写一个需求声明。

分析师准备初始设计：

分析师分析进行每个环节（业务动作）。如果这个某个业务环节已在原有的系统中实现，分析师就记录下它在系统中是怎么实现的。如果还没有被实现，分析师就记录下缺陷，并描述用户可以怎样修改系统来解决这个步骤。

UI 和 系统设计者规范并细化系统设计：

UI 设计师查找所有分析后的冲突和缺陷，并以此为依据设计一个界面原型或者界面调整来改变用户交互，以弥补这个缺陷。对于比较复杂的并布局不清晰的页面，UI 设计师会创建一个功能性的线框图，以便让界面布局设计看上去更清晰。系统设计师研究所有的缺陷，并创建或者扩展数据模型以兼容缺陷中所涉及到的所有数据。系统设计师依据缺陷来准备初始化数据，并声明数据模型的用途，配置和其它信息。

开发人员按照设计进行开发：

在开发Apache OFBiz应用的时候，开发者编写控制器要求，访问入口，菜单定义，界面定义，表单定义，模版实现，服务定义，以及具体代码实现，全部都按照设计中的内容。在用其它框架开发的时候，开发者也是必须依据设计来创建界面，后台进程等工件的。

收集需求

业务过程描述和业务经验描述

业务过程描述是记录一系列商业行为活动的一种简单的方式。这应该使用动词（verb）来描述谁（角色角色）做了什么（动作）。这个过程描述了商业活动的关联和从属关系，类似订单和分类等。

这种业务描述（就像通常的叙述手法）应该以非正式的方式进行驱动。这些描述会在深入讨论和了解商业活动和策略的过程中经常变化。起步于一个高度抽象层面的商业策略列表将帮助和引导我们组织和文档化定义这些策略的目标和持续发展。。

为了专注于需求，避免过早进入设计实现，我们不能提及 “系统”或系统交互。应用系统只是一种用来交流和协作的工具，所有的业务过程都可以表述成不同角色的用户间的协作。写这些东西的时候，自然而然地那些关于系统设计的思路，以及业务过程的其它内容，就浮现在脑海里了。这是非常有意义，对最终的系统设计实现很有帮助的一件事。最好的方式就是把这些想法以需求说明书的形式记录下来，保证他们在后期的业务工件中体现出来。

这里提一点，你应该避免讨论系统应用实现，因为你会越界开始规划设计系统应用了，，你应该谈论其它仍然并可能持续使用的系统。应当把这些已存在的或者第三方的系统当成整个业务过程中的参与者，就像业务描述中叙述的用户和系统间的交互。我们应该站在一个技术中立的角度（就是说不要轻易推翻别人的系统）去描述用户与系统协作了什么以及流转的数据是什么，而不是别人的系统怎么实现的以及它是怎么处理数据流的。”怎么实现”是基于“做什么”的需求基础上的。

这个看起来价值不大，因为企业机构里的所有人已经知道大致业务步骤是什么样的。当一个小组进行这些业务过程的构建的时候，最有用的结果是会发现不同的人对于步骤和优先级的理解很不一样，这种影响使每个人有机会达成共识，并且建立起来一个结构最终达到完成系统的目标。这些过程将需求以特定的格式很好地表述成了用例和系统设计，要完成这个工作比人们一般最初设想的要难。这些内容不仅复杂而且有用，它以一种有组织的方式记录

下几乎所有的需求。

最初的业务过程是来自于对于组织机构的透视分析，然后再专注于业务过程。通常每一个业务过程只需要几张纸的文字，和一些细节的子业务过程。最上层的业务过程描述了商业的方方面面。最上层的业务过程里，一句话可以扩展成更多的子过程，并且向下扩展出多层完整的细节信息。

这些业务过程基于一系列句法描述，以谁做的什么的形式（noun-verb）。无论如何，这种很有价值的方式能以一种更通用的信息和注解来阐述角色角色的想法和目标，。最底层的业务过程（继承下来的“叶子”业务过程）需要包含大概5-15个步骤（句法描述的操作），每个步骤一般都包含一个用例。如果你对于业务过程的某一点研究的非常细，简单的切换方式是将这个业务过程分离成几个子业务，并记得要修改业务过程的层级。

我们讨论的这些业务场景过程是自上而下组织的，推荐也以这种格式来描述。起步于一个高层次的抽象，去描述贯穿业务场景的通用业务过程，更能接近于商业的战略目标。如果某个公司的策略不是很明确，你应该最先做这件事，并且通过反馈和讨论明确怎么去实现。换句话说，如果有很多种策略可供选择，你可以问问哪个最好，如果没有一个符合要求，就继续寻找，直到找到一个为止。

这些业务场景过程应当专注于理想化的情况，但是当建立到比较细节的过程时，就应该考虑一些特殊情况，并做一些调整。但是我们仍然要专注于主线，并取得进展，不要丢失重要的步骤，特别是在上层的时候（没有涉及太多细节）。

当业务场景过程变得更细化时，业务场景应该被纳入流程中（如果需要，业务场景可以在后期进行设计转化时，变成更为正式的用例文档）。

业务场景过程的第一步是要记录企业目前的运行状况。如果企业不需要改变现状，那就可以保持这种形式。通常公司的一些运行过程是需要进行改变。一旦业务运行过程以场景的形式被记录下来，将极大的促进企业业务过程的改进讨论，形成记录在案的讨论过程，这是个极好的起点。有的时候只通过业务过程场景很难知道改善企业运行过程的方向，这时通过研究商业用例关联的业务场景的反馈信息，将极大改善现有的业务场景。。

因为业务过程场景是源于机构的宏观视角(对立于来自单个业务角色角色的经验场景)，这当中主要的问题包含：机构里面正发生了什么？谁做什么工作？当某项工作发生时，产生了什么关联的业务？为了达成某个业务目标，什么事情是必须提前做的？每一个业务角色角色怎么知道他们什么时候该进行业务工作？业务工作的接力棒是什么时候、怎么从一个业务角色传递到另一个业务角色手上的？一个业务角色如何决定业务过程需要哪些人参与以及需要开展那些其他的业务环节？

一个普遍的疑惑是如何描述已经存在的系统应用。写业务过程场景的一个原则是不要提及“系统”，但是这仅表示我们即将要构建的系统应用，而不是已经存在的系统应用。现存的系统可以简单的视为业务角色与其他的业务角色交互，共同进行业务活动。当发生这种情况的时候，就意味着你需要进行系统集成了。

当我们建立了企业级业务过程场景的宏观视角的时候，就很容易确定一连串业务角色的含义并定义他们。详见角色定义部分的细节描述。也很容易定义业务过程场景需要的功能。

另外一种有价值的业务过程场景形式是从某一个特定业务角色的角度出发。这种方式能更有效的收集每个业务角色关注点的细节，以“我每天的业务工作是。。。”的形式。以业务角色视角出发涉及到的问题包含：业务角色的目标是什么？业务角色想做什么？业务角色如何确保业务完成，以及确定目标完成的很正确？业务角色受影响的资源和约束是哪些？这个业务角色如何于其它业务角色交互，如何与组织机构进行交互？

误区：基于业务角色/角色 的需求组织

从某一个业务角色的角度出发来组织业务过程场景很容易导致组织业务过程场景变的很困难而且冗余。因为同样的业务过程会从多个业务角色从各自的视角被重复描述好多遍。同时，我们也有很大的几率遗漏一些重要的业务场景或业务角色，直到非常后期的时候（项目接近尾声），因为业务场景是由业务角色推导出来的，而不是业务角色从业务场景来的（这其实就是需求分析的步骤：先确认用例角色还是先确认业务场景过程）。必须牢记一点：描述业务场景的时候，描述有哪些业务角色，每一个业务角色做了什么很容易，但是在描述业务角色的时候，很难描述清楚整体的业务过程是什么（这就是说无法通过具体的用例角色将其所有的业务能描述出来，而只能通过具体的业务场景去描述具体的用例角色）。也就是说你可以从一个表述的很好的业务场景中得到用例角色，但是很难从用例角色中来得到完整的业务场景。

误区：基于特征的需求组织

比从业务角色角度出发更成问题的需求组织方式是从功能或特征点视角出发并组织业务场景过程或需求。这里最大的问题是跳过需求收集而直接进行解决方案设计。当选择如何组织文档的时候，记住特征点通常会出现在不同的商业业务场景过程中，所以以这种方式描述业务需求会带来和基于业务角色视角出发收集需求产生类似的问题。而且这里的问题规模更大，会导致大量的冗余和零散需求。必须记住一点：类似基于业务角色视角出发，如果你从功能点去组织需求，就很难得到业务场景过程和用例角色，但如果你从业务场景过程去组织需求，就可以知道业务中要发生那些事，即可更容易的设计功能来实现这些目标。

误区：先标题/题材，后内容（其实也就是先提纲，后细节描述）

你可能很倾向于提前考虑业务场景可能包含哪些部分，并且建立一个大纲，然后再一个个填写细节完善内容。这样做可能用来写书或者组织事情非常好，但是写业务场景过程的时候，最好让它们处于开放的待定状态，让他们自行展开，而不要在开始的时候做太多假设。一步一步地进行并且把业务过程中的每一步表述成业务角色和活动动作（谁做什么事情）的形式。以提纲方式建立需求文档最大的问题是：你定义了一个框框，并且在组织需求环节的时候局限在了这个框框里面。当业务超出你定义的框框的范围或者和别的框框产生交叉的时候，就有可能就被你忽略或者无法恰当表述了，因为你就局限于你的框框之内了。必须要记住一点：一旦你描述清楚了业务场景那么将很容易的去提取出需求的结构，这样就可以很清楚地回过头找到这些业务环节之间的传递，独立点，和频繁的交错点。

误区：描述“为什么”，而不是“什么”

不要记录“为什么”，因为这过于主观并且非常容易变化。不要尝试去描述“为什么”的理由。这种方式就是一个“坑”，会花费你大量的时间并且什么都得不到。

你没必要讨论为什么要这么做，不去记录就行了。基本上，需求说明书记录的业务活动，会给大家讨论和理解业务为什么按照这种方式去执行完成时有一个具体的框架。为什么业务如此执行最终会引导至大家讨论决定去做什么，而“做什么”是很容易被描述记录的。

我们的目标是尽可能快的收集各种关联并有效的信息用以实现商业自动化系统。如果你老是和用户纠缠不休为什么要为什么要，那么不仅用户产生挫折感，而且你也会浪费很多时间。这种挫折感是个很大的问题。通常人们对于如何处理他们自己的业务工作有自己的观点和感受，如果你非要讨论为什么，很容易导致需求说明具有个人色彩或者十分主观反而忽视了真正的要点。如果你调研时，与用户交流按照“什么”（具体的业务动作）的方向，那么就很容易客观地让别人理解怎么做会产生怎样的效果，在决定业务过程中的某个操作是怎么

影响到其他业务环节的。

一些场景中，你无须过多讨论“为什么”，因为某些业务过程已经具备了很好的规划和准备，你只需要在某些细节点上进行下深挖，精炼抽象业务过程，以便促使业务过程的自动化运行。另一些场景中，你很难去描述业务场景。大量的时间被浪费在个别的争议点的讨论上，并往往争执的过程都是过于情绪激动和挫折的。当你去调研一些业务为什么这么执行时，你会发现最终的结果并未契合商业业务需求目标，同时，除非你深入细节点去查看什么在执行并忘记为什么这么做，大家才会意识到这种方式是错误的。总之只需坚持探讨业务活动以及业务活动是怎么有序执行的，又产生什么效果就行了。最后你会找到一种最有效率的方式去抓住必须的关键点，并且规避业务过程中“主要的问题占用大部分的需求时间”的结果。

比如说一个公司在少估了货物的装船量，导致损失了很多钱。如果你问他们为什么要以那样的方式做事，他们肯定很纠结。在打包之前加上核实货物的重量的选项对于他们来说是个很重要的部分，但是他们不会去讨论其他有序的运作方式，因为对于他们来说“为什么”这么做是很重要的。一旦交流的关注点变为什么时候做什么事情，他们会很容易去发现问题的起因，并发现我们无需讨论为什么必须做核实重量的环节，他们只要称重好货物，算好船运的容量，把船只的状态设置为“已打包”，这样就解决错误估算装载量的问题了。

业务角色定义

业务角色定义要包括业务场景过程中涉及到的所有业务角色。业务角色包括用户，组用户，甚至系统。理解业务角色，可以帮助我们组织业务场景过程信息和互相关联的商业活动的重要信息。

一旦定义了业务角色之后，基于各个业务角色视角的业务场景就能够被描述出来。这样可以从不同的角度来理解业务场景，并能定义不同的业务角色对于主线的影响。

需求声明

虽然业务场景对于组织自动化系统和商务活动非常有帮助，但是它并不能包含设计阶段考虑的实现系统所有方方面面的重点。有一些需求没办法写在业务场景描述里面或者普适性的需求。

最常见的一种需求声明是，用来记录适用于所有业务过程的商业需求，而不是针对具体业务场景过程的。对于这种需求，用需求声明的方式记录比写在业务场景过程文档里面要简单和自然的多。

还有一种需求声明的方式是用来描述工具以及系统特征的构思，以及用户与系统交互方式等不适合在业务场景中描述。

需求声明还能用来记录讨论过程中产生的，还没来得及加入业务场景描述的一些想法。这样你就不用暂停讨论来记录东西，只要简单的写下讨论中产生的思路就行了。这种方式十分有价值，特别是一些涉及到关联很多业务场景以及业务点的思路想法，需要待其他的规划及讨论后再去全面的讨论合并这些想法。

需求声明是特意不规范记录的形式，这意味着在头脑风暴中能很方便的抓住思维亮点或者是在描述业务场景文档是无需打断思路去完成想法的合并。然后这些非正式的需求文档描述后期要被合并到一些正式的文档里面。特别是涉及到系统交互用例描述，界面/报表样式，数据声明，和其它的一些设计文档。

就像描述业务场景一样，声明也可以以不同的视角去描述心中成形的系统的各个方面。一种常见的声明是功能声明，通常被用来描述系统本身所需要的功能点。功能声明也可以被用来定义业务角色的功能，包括用户和其他的系统。

慢慢的需求声明会变得越来越多，所以你最好把它们组织起来，把相似、相关联的声明

放一起，或者通过分类来更好的组织它们，否则最后你会被大量无序的声明“弄死”。

当你回顾并组织需求声明的时候，要一个个检查看看它们能不能被纳入到业务场景过程描述中去。有的时候需求没办法被纳入到某个具体的业务场景中，因为它同时涉及好几个业务场景，这是比较常见的。如果出现这种情况，那么最好调整每个相关的业务过程，以免在设计和构建这个业务过程时，遗漏了这个需求。

描述声明的时候不用追求完美。描写业务场景过程的目的是为了确保业务场景中的所有点都是存在且经过深思熟虑的。需求声明的目的是在一些想法纳入系统交互用例描述及之后的设计文档之前，有个地方能记录下这些需求。

误区：声明中的冗余和矛盾

因为声明是在编写需求业务场景中灵光一闪产生的想法，所以这些想法有可能出现好几次，很容易产生写了好几条差不多的声明或者意思一样，甚至更坏的产生互相含义相左的声明。事实上，这些冗余很容易产生矛盾，在我们进行下一步完善需求文档的工作之前，必须理清解决这些矛盾。冗余导致效率低下，矛盾会导致设计问题甚至系统有效性问题。所以我们要很好地组织需求声明，经常回顾、精炼需求声明，尽量把它们纳入到业务场景需求过程里面，这样就能尽量避免和消除那些问题。

初始设计

重叠与缺陷分析

重叠与缺陷分析文档是建立在业务场景需求过程以及需求声明文档的基础上的。重叠缺陷分析是直接以文字的形式组织记录的，它是对于每个特定业务场景需求过程或者需求声明的一种反应。

当现有系统有包含了需求的重叠部分的时候，我们需要把细节记录下来，甚至每个步骤细节描述去说明一个用户或其它系统是怎样与现有系统进行交互，来执行这个业务过程里面的商务动作。

当我们描述缺陷的时候，我们应当记录每个需求重叠的部分，然后再去描述系统需要支持哪些常规的环节部分，来填补缺陷和满足需求。这是重叠、缺陷分析里面最重要的部分，有了这个初始设计分类分析，我们就能更详细地设计 UI 界面和系统设计，最后实现系统。一旦系统实现工作完成（或者细节设计完成），我们就可以回溯这个重叠、缺陷分析文档，把那些缺陷以重叠的方式表述出来。

这个文档的最终目的是描述系统如何来满足业务场景需求过程以及需求声明里面的每一个要求。

这个文档可以是在业务场景过程或需求声明文档上通过加上重叠、缺陷注释的形式，也可以是拆分为多个小文档，与业务场景过程或需求声明子文档关联起来。对于现有不同的系统，可能会有很多不同的重叠、缺陷分析需要和需求去比对，所以这些分析文档最好独立拆分出来。当你回溯这些业务场景目标时，你可以找到和初始的重叠缺陷分析相对独立的文档。

设计

界面和报表大纲

这个大纲是在反复修订完所有的重叠和缺陷分析中的系统交互之后产生的，并且要确保对于每一个交互都要有充足页面元素的页面去实现。每个界面的大纲节点需要包括

展示给用户的界面，用户发出的请求，还有所有跳转出界面的出口。这里应当包含了链接，表单提交，跳转到哪个界面。如果某个跳转会引导至多个页面，那么我们就要描述跳转到具体某个页面的规则。这里介绍一些界面流图的一些概念。

报表的大纲和界面大纲结构上差不多。一些报表是业务过程中的一部分环节，并包含一些决策上的数据信息。包含很多战略决策信息的报表不应该放在通常的商业过程环节里，而应该单独做为大纲的另外部分，罗列出来。

界面/报表大纲是和重叠、缺陷分析相独立的文档，因为它的结构完全不同。重叠缺陷分析由业务过程环节所组成，而界面大纲由界面流以及界面元素组成。一个界面可能在多个业务环节中被调用或是一个业务过程环节中的组成部分，相反的，一个业务过程环节也可能由多个界面组成。

大纲中的每一项都应该来源于缺陷描述（在重叠缺陷分析文档中），商业过程环节或系统交互，并应该被重新提及。虽然这些参照源信息都是可选的，但是如果你遇到缺陷分析与界面大纲不一致时；或是在进行界面大纲设计阶段，发现界面大纲不是基于任何需求或预设计等问题，你也许会考虑要求你的团队去使用这种技术方式了。。一个粗略的大纲大概可以写成这样：

1. Foo Baz 界面
1.1 区域 1（界面最上方）
1.1.1 静态文字 XYZ
1.1.2 动态信息 foo
1.2 区域 2（页面底部）
1.2.1 动态标题文字 baz
1.2.2 baz 表单
1.2.2.1 表单 1-文本框
1.2.2.2 表单 2-下拉
1.2.2.3 提交按钮（跳转到 Bar Foo 界面）
2 Bar Foo 界面
2.1 中间区域等等

仔细检查每个系统缺陷描述和交互并在界面上给其创造一块空间，就形成了界面大纲。这更像是个精细的过程，我们需要不断地进行优化，改变元素的位置，合并或者分割页面等等，作为系统交互的进一步综述。就像大多数的中间工件的目标一样，大纲是为了讨论过程中能方便的创建和修改，以便决策方案确定。

一旦大纲初步完成，并且所有的系统交互都已经包含组织起来之后，我们最好通过一些“角色扮演”来检验设计。这是个快速且明智的校验方式：我们完整的走一遍每个业务过程环节，来确保每个要点都包含了，并且很容易查找和使用，或者说能很容易的达到业务过程中要求的目标。一旦基于大纲内容的线性框架构建出来了，就能进行更多的前瞻的最终用户的“角色扮演”，不过现在，一些不特别正式的东西已经足以帮助我们确认没有遗漏什么东西并且确保每一点都有意义，以便未来进一步完善设计思想，界面组织的方式等。

功能线框

功能线框应当包含用户看到的任何内容：界面（网页或桌面应用的界面），报表，邮件等等。诚然，界面/报表大纲有一些更为细节完善的信息，并可以做为功能线框的参考文档，但是，通过文字的方式有时候很难描述空间立体的关系和布局，用线性框架

就是一种很好的办法。功能线性框架的内容是基于界面/报表大纲的信息创建的。

功能线性框架对于预期的最终用户测试来说也十分有用。一个最简单的方式就是进行角色扮演。就像你高中的时候所做的一样，不过可能稍微枯燥一点，因为你不是扮演小矮人，精灵或者兽人，而是扮演 ERP 系统的一部分或者一个用户。进行角色扮演需要有一个（通常是设计师或分析师）或者几个人来扮演系统，来维持功能线框。然后预期的最终用户就将选择演员来扮演角色。在这个过程中，扮演新的系统的人不要尝试去描述用户应该做什么，怎么做，而仅仅是展示出线框结构，并让扮演使用者的人自己去研究并搞清楚怎么进行。用户扮演者描述他们做了什么，系统扮演者就描述系统会怎么响应。他们都得益于之前阅读的关于系统的各种文档，系统扮演者需要十分小心的依据这个线框文档以及设计文档进行扮演，而用户的扮演者只需要尝试进行日常的工作或达到特定的目标，而无需参考或者按照文档的规定去扮演。

你可以用很专业的画图工具绘制出功能线框，也可以简单地画在纸上或者白板上（然后扫描或者照下来）。图纸很方便，不过改起来比较麻烦。用画图工具可能有点麻烦，不过改起来容易，而且看上去比较清晰和清楚，特别是大量的组合元素的时候。我个人的倾向是一开始在白板上画个草稿布局，然后等线框结构组织差不多完善确定下来了，就用画图工具画个清晰的线框。

根据界面大纲来画功能线框是相当容易的，仅仅只要把大纲中所有的可视元素过一遍，然后一个一个画到线框上就可以了。同时，注意画线框图的时候，要在界面大纲上标注出界面元素的来源。如果画图工具支持浮动层，那么你可以把这些信息单独放在浮动层上，带上一点透明度和阴影，这样主界面还是能看的很清楚，有些界面只要简单通过明确的文字描述它需要做什么就可以了。有的时候画一些比较复杂的，有很多元素的界面可能要花大量的工作。有的界面可能只要几分钟，有的要好几天，甚至需要不同的人去进行复核。

有时候需要画好多个界面，因为界面有的部分可能会扩展，有的时候需要展现弹出框，有的时候做了一些操作，界面就相应发生变化。通常绘制多个线框图的方式会比在一个线框中将所有的内容都放进去要简单的多。还有一种是“绒布板”（这里其实指代类似订做衣服时，看到的选布料的那个本本，这里其实类似内嵌页面或者门户的界面）的形式，线框中的每个部分是空白的，并且里面可以显示各种版本的界面可以在主界面上被添加和删除。

当你进行界面布局的时候，有可能会发现界面大纲的二维界面尺寸无法承纳那么多的元素。实际上，你需要随时进行界面大纲的调整，以确保不会出现不一致的情况。有的时候在需要满足用例描述中系统交互的需求变更，你甚至需要重新画界面。记住这是一种改进的过程，在你进行不同工件的创建过程中，你可以进行多角度的对系统进行透视分析。同时，你也要记住：现在在规划的时候改很容易，等以后实现完成的阶段再去改界面就难了。

数据模型

数据模型是被用以确保系统最终运行的物理持久化的信息。这里的模型文件描述的是关系型数据库的模型。虽然市面上有其他的数据持久化方式，不过关系型数据库仍然是主导，而且将来估计也是，因为其他的数据库持久化技术没办法进行这么灵活的存储和检索。如果现实的业务需要有其他的数据组织和持久化方式（对象型数据库，目录/层次数据存储，本体服务，XML 文档存储等），那么这里就需要创建个更好的工件来支撑这种需求。

数据是极其重要的，因为不管系统的运行过程如何，最后都是形成持久化的数据，来提供给其他的运行环节，不管是本身的系统还是与其他的系统协同交互运行。

创建数据模型是一个不平凡并重要的工作，虽然表现出来的是很简单的一件事情。其中最重要的元素是实体（表）和字段（列）。刚开始第一步工作就是要花大量充足的工作、时间将各种信息组织成实体和字段。然后另一个重要的环节，就是定义唯一标识（主键）以及表间关系（外键），你也许想在第一步初始化实体时或至少初始化完成实体后进行定义。但是在实体细节，如字段数据类型、大小等完成后进行会更妥当。

创建数据模型最简单和有效的方式就是基于数据声明，数据声明描述了数据的本质、数据之间的关系以及系统对于数据的追溯。在定义工件的时候，经常回溯其他相关的工件是很重要的。在 HEMP Light 中，数据声明不仅仅简单的基于重叠和缺陷分析中的数据模型。

数据模型模式和技术的经验对于好的数据结构设计结果是十分重要的，并且有很多不错的模式,和大量的讨论如何构建一个好的模式。在现代系统中，为了保证灵活性和减少未来的修改，通常最好的方法是将信息字段拆分为多个数据表，不管是否字段有明确的一一对应关系。

当创建数据模型时，确保数据模型中已包含了已有系统的数据信息元素并尽量重用已经有的部分，这样就在保持原有关联关系的前提下对模型进行扩展，而避免产生一个独立的毫无关联关系的数据模型，以致于约束了系统的灵活性和功能。如果你清楚现有系统的数据模型和数据关系图，那么就只需要参考数据模型文档，按照那个来就行了。

如果你觉得某些信息可能和原有数据模型的一些部分相似，但是无法确定，那么可以先把这些当成新的实体记录在数据模型文档里，当文档完成以后，再去和原有结构去比较，看看这些部分是不是真的很吻合。也就是说，如果只根据数据声明或追溯的信息说明比较难判断模型是否吻合，那就可以在新数据模型构建完毕后，再去和原模型比较就比较容易了。

初始化数据

初始化数据中包含一些其他术语并且初始化数据也有很多类型,如：种子数据，样例数据，或测试数据。不管是那一种，它们都是数据模型的一种补充，用来描述数据的含义和不同操作的有效性，亦或是更好地描述，数据模型的每个字段如何体现现实世界中的场景实例。

种子数据是维持在代码中的数据，并且代码依据这些数据（包含唯一标识符的直接引用）来运行。种子数据包括很多种，如：状态数据，枚举数据以及数据结构中常被复用的类型（比如销售订单和采购单之间有很多但不是全部字段和关联的实体信息类似，但是通常业务和编码差异很大）。通常系统代码改变的时候，种子数据也要跟着改变，来确保最新的代码能获得正确的选项。

这里有个特例，当系统或者用户修改了代码的时候，种子数据需要重新初始化加载一次。例如，一个后台进程进入数据库操作中，这时系统程序发生变化导致进程响应超时。你不想启用另外一个进程介入或是将原有进程回滚到原有的状态，你需要在更新程序之后进行种子数据的重新加载进行初始化。

样例和测试数据有不同的目的，不过可以共用相同的数据。样例数据是为了方便的呈现系统功能的不同部分和特点。特别是当组合不同的数据来产生某个结果的时候。样例数据可以通过用户界面的录入，来看不同的环境下产出了什么数据结果。并且可以提供给分析师，开发人员，测试人员对于系统如何组织构成以及选项有效性检查的一个简单、直观的指引和自描述。

测试数据更多的是为了测试的目的以及做为样例数据的一种扩展，因为测试数据可能会比较冗余和庞大，通常是用来满足自动化测试过程的需求。测试数据当然可能提供给手动测试用，这种情况下，看上去和模拟数据就差不多。再次重申，目的有两种，但是数据可以用一样的，用于自动化测试的数据可能比较特别。两种数据都是十分有用的，可以更好的理解系统以及不同环境下记录数据预期的状态，或是在不同的业务过程（业务场景中定义的）环节中使系统促进自动化。

这些文件的结构和形式可能根据不同的系统会有区别。通常最初可以在电子表格或者一般的 XML 文件里面创建数据，在没有目标系统或者体系结构定义数据格式之前，然后当确定了系统以后，再把数据转化成需要的形式（永远不要低估通过“邮件合并”电子表格得到你想要的数据格式的能力）。另外，要知道如何规范数据格式，并最好有可以导入数据，并且对照数据模型定义检查（最好在实际的数据库中）的工具去避免简单的错误，在任何情况下，初始化数据都是十分有用的。

实现

不管如何使用工具和利用可复用的实现工件，仍然有很多实现的方面需要采纳。

一些工具是低级别的，需要更多的开发工作并要求很多的灵活性。对于这些工具来说，不能像文字一样逐字的去转换详细设计工件和实现的工件，换句话说，在实现一个特定详细设计工件时，我们需要选择很多的工具去实现它。在这种情况下，尤其是对于大型项目，最好是提前计划使用什么工具来开发，以及它们如何去实现设计的内容。

使用 Apache 的 OFBIZ 框架，或是其他类似的框架，都有很多典型的设计工件的工具可以使用。在完整的 HEMP 的工件集合中，有很多设计以及已经实现的工件一一对应了。对于轻量 HEMP 来说，某些设计工件被遗弃了，因此在实现工作可以引用的记录性的资源会很少甚至是简单的绘制下线框，但是最终可用的信息是一样的。

因为实现涉及最终校验的详细设计工件是常见的，并且设计是基于可行性和层级评估的。

开发人员的基本责任是基于详细设计的工件去创建实现工件，然后确认系统功能结果满足设计的描述。在实施的工作步骤前，需要进行无缺陷的检查，目的是实现后复核并靠拢设计，满足需求，然后进行结构普适性和多角色使用测试。

实现后复查和验收

就像设计工件的复核需要使用需求工件一样，实现工件（系统本身）的复核是需要开发人员通过设计工件来进行的。除了这些工作，UI 和系统设计师也可以审查实现的系统执行是否符合细节的设计描述。

更进一步的，在最终需求和业务应用适用性评审会议中，分析师可以复核实现与重叠、缺陷分析的差异，并能创建各种覆盖了全部重叠的文档。一旦这个工作完成了，业务专家和其他最终用户可以复核和测试系统是否满足之前帮忙收集和记录的需求文档。

一半的思路是，创建某个工件设计到的人员应当去复查相应的实现执行的最终结果。某种程度上，这意味着开发过程需要向后进行复核并测试到底实现了什么。通过每种角色代表的参与和努力，将确保他们能够很自然的对最终结果进行验收。

当某种角色复核时发现问题，他们有责任去判断是否有必要去改变，如果有就去改变它。然后基于此，简单的走一遍，再复核下一个步骤的工件是否需要变化，按照这种方式，如果最终的结果是需要进行变化的话，就需要去修改具体的代码实现了。再次的，

一旦完成了这些工作，需对整个过程的结果向后进行复核，以保证最后业务专家表示没有问题了（或至少没有重大的问题需要返工…）

封底：HEMP 轻量工件流图