目录

[一、 框架 1](#_Toc496094867)

[1. 简介 1](#_Toc496094868)

[2. 土木工程 1](#_Toc496094869)

[3. 施工工艺流程 2](#_Toc496094870)

[4. 应用框架 2](#_Toc496094871)

[5. 网页设计 5](#_Toc496094872)

[6. 程序设计 6](#_Toc496094873)

[二、 开发框架 6](#_Toc496094874)

[1. 前言 6](#_Toc496094875)

[2. 什么是框架 6](#_Toc496094876)

[3. 为什么要使用框架 7](#_Toc496094877)

[4. 搭建框架时，我们要如何定位 8](#_Toc496094878)

[5. 框架和设计模式 9](#_Toc496094879)

[6. 框架的主要特点和要求 9](#_Toc496094880)

1. 框架

# 简介

框架（framework）是一个框子——指其约束性，也是一个架子——指其支撑性。是一个基本概念上的结构，用于去解决或者处理复杂的问题。

框架这个广泛的定义使用的十分流行，尤其在软件概念。框架也能用于机械结构。

# 土木工程

（框，读kuàng，不读kuāng。）

由梁和柱组成的能承受垂直和水平荷载的结构，梁和柱是刚性连结的。主要要用于工业与民用建筑物的承重骨架，桥梁构架或工程构筑物。

一般指建筑工程中，由粱或尾架和柱联结而成的结构。由于我国古代家具自宋以后曾吸收古代大木构制的作法，故传统家具采用框档、立柱结构形式的即称框架，并将此类结构形式的家具称之谓“框架式家具”。

# 施工工艺流程

## 3.1基础工程

测量放线→土方开挖→钎探、验槽→静压桩→砼垫层→承台、基础钢筋、模板、砼→墙、柱钢筋→水电预留、预埋→墙柱模板→墙柱砼→拆模养护→梁板模板→梁板钢筋→水电预留、预埋→梁板砼→养护→外墙防水→防水保护层→基础土方回填。

## 3.2主体工程

测量放线→墙柱钢筋→水电预留、预埋→墙柱模板→墙柱砼→拆模养护→梁板模板→梁板钢筋→水电预留、预埋→梁板砼→养护

## 3.3装饰装修工程

测量放线→二次结构砌筑、隔墙安装→屋面防水→立门窗框→内墙抹灰→楼地面→墙面踢脚→设备安装→门窗扇安装→吊顶→精装修→油漆、涂料。

# 应用框架

随着管理信息应用范围的拓展，交易类应用为管理对象的电子信息获取提供了丰富的手段，这些信息的存储、加工、增值、展现等处理事务，均属于管理决策类应用系统的范畴。传统的信息系统通常将这些事务，与交易类应用合并在一个应用系统中实现，随着一个组织中应用系统不断地涌现，关联数据的组织和共享、历史数据的积累和重用、全面数据的挖掘和增值等需求，促生了基于数据仓库技术的，面向整个组织，独立性的管理决策类应用环境的实现。这类应用有一个最大的特点，就是用户需求是持续发展和不断完善的，特别是应用的初期，用户甚至提不出充分和全面的需求。同时，这类应用又存在非常强的应用共性。为此，可以通过一组与业务无关的，产品化的技术支撑环境，去实现对海量数据获取和储存的支持能力；去实现描述加工规则发展和完善的能力；去实现提供数据组织访问和管理的能力；去实现反映结果信息简洁和人性化的能力，这便是“管理决策框架”的意义。该框架由以下部分构成：

## 4.1业务架构

所有的应用系统只有在覆盖了相应的业务以后，才具有应用的实际意义。与业务无关的管理决策框架在没有加载业务以前，只能称之为框架，加载业务以后则成为了针对特定业务的管理决策系统了。所谓业务架构一方面是为用户加载和组织业务提供的一个手段和环境（开始也为用户加载了一些通用的业务如查询、报表等）；另一方面也是在实际应用时的业务门户。类似于智能终端的桌面，用户拿到手的时候桌面是空的，只是有一些通用业务，如时钟、画图、记事簿等，随着应用的发展，每个人的桌面会呈现出各不相同的个性化业务。管理决策框架中向用户提供了业务操作和管理的操作2类业务，其中业务操作是面向大众用户的，涉及业务管理活动的流程、查询、分析、决策等日常作业，这些操作绝大多数都需要用户后续自行加载；管理操作是面向小众用户的，涉及对业务管理活动的流程定义、数据组织、分析规则、决策算法、展现效果等定义和描述操作，并对它们进行加载和维护的作业。与传统的应用系统不同的是，这些作业的形成不是由程序员编码实现的，而是用户骨干或者第三方团队（小众用户），利用管理决策框架提供的管理操作（由应用架构的相关产品提供）加载实现的。加载的结果通常以“方案”的形式打包，每个“方案”对应一个管理活动的过程、规则、算法、结果展现等等。训练成熟的方案经过该架构的解析和执行，面向用户提供直观高效的，可持续发展的，智能化的最佳用户体验。该架构涉及的技术包括：统一门户、统一权限、工作流、商务智能（BI）等等。

## 4.2应用架构

该架构主要是面向业务架构提供软件功能的支持，既提供运行时的业务功能支持，又提供加载时的管理功能支持。与传统的应用架构的最大不同在于，该架构能为整个组织实现业务需求的变化和满足覆盖地域的不同，提供可持续发展的支撑能力和具有更长的生命周期。同时，也是确保业务无关性，实现管理决策类应用产品化的关键。为此，组成该架构的一系列产品，均以人机交互的模式，将管理决策业务需要实现的数据源获取、数据口径描述、数据的组织、加工规则、管理过程、结果展现等进行定义、描述和发布管理；其所涉及的每个定义和描述的结果，均分为2个部分，一是以代码段或脚本形式保存的，可以由业务架构解析并执行的部分，称为“方案元”；二是对相应的方案元按照标准的元模型，以人工语境描述的数据集合，称为“元数据”。方案元也作为元数据的一部分一并打包，并加以保存和管理，每项业务所涉及方案元的集合称为“方案”，每个“方案”经过测试和训练，面向特定的用户发布。这个过程称为“主动式元数据管理”。应用架构发布的结果就是业务架构面向大众用户提供的“业务功能”；使用应用架构所提供产品进行业务实现（加载）的过程，就是小众用户在业务架构中使用“管理功能”的过程。该架构涉及的技术包括：元数据标准、元数据管理、方案的形成和管理、知识的形成和管理等等。

## 4.3数据架构

它面向全局提供统一的数据综合利用及管理环境。与传统的数据架构不同，该架构提出了对“数据空间 ”进行“数据管理”的概念。“数据空间”是整个组织所有管理对象所涉及的数据全集，以及它们所有的数据属性。传统的数据架构关注的重点，局限在所有管理对象涉及的实体数据（内容），而“数据空间”关注和管理的对象，还要扩展到：一是以人工语境对“内容”进行解释性描述的元数据（变化）；二是记录“内容”和“变化”的归档数据（历史）；三是反映管理决策框架运行环境的日志数据（状态）等。这里记录内容、变化、历史、状态的数据集合，称之为“数据全集”；“变化”与传统的只供机器识别的技术元数据（传统数据属性），一并称之为新的“数据属性”。“数据管理”指的是要对进入管理决策框架的数据源进行完整性、原始性、不可抵赖性的实现管理；要对基础数据口径和后续加工规则、算法等进行标准化、规范化、可追溯的描述管理；要对数据空间涉及的所有数据，进行合理组织、物理分区、数据结构的定义，实现全面科学、统分结合、访问高效的控制管理；要对内容、变化、历史、状态等涉及的所有数据增值过程，进行全面质量管理和全过程的生命周期管理。该架构涉及的技术包括：非结构化数据处理、档案管理、“大数据”技术、数据仓库（特别是DW2.0）涉及的相关技术等等。

## 4.4技术架构

该架构是构成信息系统物理环境的产品集合，包括服务器、操作系统、中间件、网络环境等基础技术环境。在进行配置管理时，管理决策框架除了要考虑灾备的异地支持环境，还要在物理上分为生产环境和训练环境。生产环境是训练环境的子集，其主要是将正式发布的“方案”经过加载、解析、执行，针对特定业务提供日常作业的支持服务，从而确保了生产环境的简化、高效、可靠、稳定；训练环境除了能模拟实际生产环境，还要提供业务（方案）加载、测试、训练、维护、发布等管理作业的支持服务，从而确保该框架对业务应用无关性、对业务需求的可持续发展、保证支持环境的长生命周期。该架构涉及的技术包括：虚拟技术、云计算、容灾管理、数据中心监控等等。

## 4.5安全架构

在最大限度满足管理决策框架运行的基础上，构建网络、硬件和软件相结合的安全体系，通过监控管理手段来确保系统稳定，削弱高度信息化的应用系统受单点故障的影响程度，使系统能够将风险分散和具备一定的自救能力。要考虑一体化的，整体安全架构的设计要求；要符合信息安全标准(物理安全、运行安全、数据安全、内容安全等)规定而采取的技术和管理要求；要实现信息安全和数据质量管理的技术环境，能够提供安全策略的具体管理机制。信息安全不仅体现在物理环境的实现上，更要强调信息安全管理机制的建立和持续完善，并且管理机制要能够体现在物理环境上，要能够通过物理环境管理、记录、分析各类信息安全事件，避免其再次发生。

## 4.6标准体系

作为与业务无关的应用软件产品，管理决策框架需要一系列标准，以规范整个框架对外部的衔接、规范框架内各架构间的衔接，以及每个架构内部的，对处理对象的获取、加工、处理等描述的规范等等。它们涉及到相互衔接、处理对象描述、加工规则描述等等方面的标准化，以及如何对它们进行描述的标准化（元数据标准，即元模型）；涉及到包括组织、制定、维护、发布、遵循等内容的标准管理机制；特别是要有一个支撑标准管理机制的技术支持环境，这个环境不仅要提供对每个标准生命周期的管理，还要提供整个框架对标准遵循和使用的一致性和易用性保障和服务。

## 4.7运维体系

管理决策框架的运维体系分为2类：一类是物理环境运维。即传统的数据中心环境和设备的运行保障和安全保障；另一类是应用环境运维。它包括涉及业务架构、应用架构、数据架构、标准体系的运维管理。通过这些运维管理活动，实现业务需求的有效支撑和可持续发展；实现管理功能（即整个框架本身）的可靠运行和可持续发展。为了这2类管理活动的有效进行，需要建立一套运维管理机制，包括运维组织、制度、职责等等；同样，也要有一个支撑运维管理活动的运维技术支持环境，它不仅要提供对运维管理活动的过程和监控提供服务，还要提供对运维事项的发起、发现、定位、预警、处置、恢复等手段提供功能性支持，并且能够通过对运维事件多角度信息的捕获、积累、分析、挖掘，实现智能化的运维辅助和事件预测。

# 网页设计

框架是网页中经常使用的页面设计方式，框架的作用就是把网页在一个浏览器窗口下分割成几个不同的区域，实现在一个浏览器窗口中显示多个HTML页面。使用框架可以非常方便的完成导航工作，让网站的结构更加清晰，而且各个框架之间决不存在干扰问题。利用框架最大的特点就是使网站的风格一致。通常把一个网站中页面相同的部分单独制作成一个页面，作为框架结构的一个子框架的内容给整个网站公用。

一个框架结构有两部分网页文件构成：

框架Frame：框架是浏览器窗口中的一个区域，它可以显示与浏览器窗口的其余部分中所显示内容无关的网页文件。

框架集Frameset：框架集也是一个网页文件，它将一个窗口通过行和列的方式分割成多个框架，框架的多少根据具体有多少网页来决定，每个框架中要显示的就是不同的网页文件。

所谓框架就是把网页分成几个框窗，同时取得多个 URL。<frameset>用来划分框窗，每一框窗有一个<frame>，<frame>必须在<frameset>范围中使用。如：

　　<frameset cols="50%,\*">

　　<frame name="hello" src="1.html"> <frame name="hi" src="2.html">

　　</frameset>

此例中 <frameset> 把画面分成左右两相等部分，左面显示 1.html，右面显示 2.html

# 程序设计

程序设计中的框架包含DoitPHP(原Tommyframework)是一个基于BSD开源协议发布的轻量级PHP框架，还包含如thinkphp、codeigniter（简称CI）、yii framework、doophp、qeephp、等主流的程序设计框架，根据个人习惯和性能要求，其各也有不同的优缺点！

从软件设计角度，框架是一个可复用的软件架构解决方案，规定了应用的体系结构，阐明软件体系结构中各层次间及其层次内部各组件间的毅力关系，责任分配和控制流程，表现为一组接口，抽象类以及实例间协作的方法。

1. 开发框架

# 前言

作为一个程序员，在开发的过程中会发现，有框架与无框架，做起事来是完全不同的概念，关系到开发的效率、程序的健壮、性能、团队协作、后续功能维护、扩展......等方方面面的事情。很多朋友在学习搭建自己的框架，很多公司也在创建或使用自己的框架，网上开源的框架多如牛毛，每年新上线的各种框架也不知多少。而不停的重复造轮子，也让更多的朋友鸟枪换炮，越跑越快，工作效率越来越高。那么什么是框架呢？

# 什么是框架

度娘上说：框架（Framework）是整个或部分系统的可重用设计，表现为一组抽象构件及构件实例间交互的方法; 另一种定义认为，框架是可被应用开发者定制的应用骨架。前者是从应用方面而后者是从目的方面给出的定义。

　　可以说，一个框架是一个可复用的设计构件，它规定了应用的体系结构，阐明了整个设计、协作构件之间的依赖关系、责任分配和控制流程，表现为一组抽象类以及其实例之间协作的方法，它为构件复用提供了上下文(Context)关系。因此构件库的大规模重用也需要框架。

　　应用框架指的是实现了某个应用领域通用完备功能（除去特殊应用的部分）的底层服务。使用这种框架的编程人员可以在一个通用功能已经实现的基础上开始具体的系统开发。框架提供了所有应用期望的默认行为的类集合。具体的应用通过重写子类(该子类从属于框架的默认行为)或组装对象来支持应用专用的行为。

　　应用框架强调的是软件的设计重用性和系统的可扩充性，以缩短大型应用软件系统的开发周期，提高开发质量。与传统的基于类库的面向对象重用技术比较，应用框架更注重于面向专业领域的软件重用。应用框架具有领域相关性，构件根据框架进行复合而生成可运行的系统。框架的粒度越大，其中包含的领域知识就更加完整。

　　框架，即framework。其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。

　　上面是度娘百科——框架里讲的内容，讲得挺理论挺全面的，直接复制了一些过来。

　　简单来讲，软件开发框架可以理解为我们起楼房时，用梁+柱子+承重墙做起来的钢筋混凝土结构框架。而实现的软件功能，也就像在这个框架结构中所要实现的不同类型、功能的房子，比如停车场、商场、酒店、饭店、商住房......

　　功能强大，结构扎实的框架，可承载的类型就更多，适用性就更强。

　　不论是什么开发框架，都是为实现不同的业务功能而创建的。好的开发框架能提高企业的竞争能力，包括降低开发成本、提高产品质量、改善客户满意程度、控制开发进度等。同时，开发框架也是某一领域为实现某种业务的知识积累，它是一种持续性的活动，随着特定业务功能实现而创建，且跳出软件开发周期而存活。一个成熟的框架，不单单只是代码而已，它除了将某一领域业务的基础功能以代码的方式实现外，还必须编写对应的文档、模型、流程图、数据字典、开发使用说明.....等一系列的文档来支撑，这样的框架才更加成熟和健壮。

# 为什么要使用框架

软件系统随着业务的发展，变得越来越复杂，不同领域的业务所涉及到的知识、内容、问题非常非常多。如果每次都从头开发，那都是一个很漫长的事情，且并不一定能将它做好。团队协作开发时，没有了统一标准，大家各写各的，同样的重复的功能到处都是。由于没有统一调用规范，很难看懂别人写的代码，出现Bug或二次开发维护时，根本无从下手。

而一个成熟的框架，它是模板化的代码，它会帮我们实现很多基础性的功能，我们只需要专心的实现所需要的业务逻辑就可以了。而很多底层功能操作，就可以完完全全不用太多的考虑，框架已帮我们实现了。这样的话，整个团队的开发效率可想而知。另外对于团队成员的变动，也不用太过担心，框架的代码规范让我们能轻松的看懂其他开发人员所写的代码。

# 搭建框架时，我们要如何定位

　　是不是框架的扩展性、可移值性、功能越强大就越好呢？

　　好的框架是相对的，它都有自己特定的应用领域，合适才是最好。

　　个人觉得在实际开发中要根据具体情况来看的，因为功能越全面它的复杂度就越大，所需要的开发人员能力和技能就会要求更高，付出的成本也就最大。比如做一个还未发展起来的电商网就想 将系统做成像京东那样，直接用京东分模块分布式的框架来开发，那得怎么来组建这个团队？更不用说开发成本了。就算团队有能力做到，也没有那个必要这么去做，因为从成本预算和开发周期等方面来看，得不偿失，更多的可能项目还未完成公司就给拖垮了。

　 　一般来说，一个中小型项目，1到5人左右的开发团队，使用一般的三层结构就可以了，不用去细想框架要分三层还是五层，每个层之间要怎么实现解耦，要用什么设计模式.....因为当今飞速发展的互联网时代，快才是王道，做一个中小型项目能用一周完成的，绝不能拖了一个月还未做完。人工与时间成本才是重点中 的重点，唯有快才能更好的生存下来并壮大。至于扩展功能、接口、分布式、并发、大数据......等等问题，实际上过早考虑太多并不是好事情，有经验的程序员在写这个框架时早已留下扩展方案或思路，而没到这一层次的开发人员你想再多也可能想不明白，还不如先做出来积累一定经验后再慢慢学习，慢慢升级框架。

　　当然也不是说设计框架时不用考虑高内聚低耦合，而是要根据自己的能力与经验来设计出自己能把控的框架出来。因为框架不是开发出来后就不再变动，它也需要不停的进行升级，将你所学到的新知识新技术融合到框架中，使它的功能更加强大，更加健壮。而对于自己不能把控的框架，在团队协作开发和上生产环境后，你就发现有一大堆的坑等着你去填埋，这种框架只能拿来先练练手，有空再慢慢完善。

　　框架通过小步快跑，不断的迭代升级来慢慢扩展的，当项目上生产环境后，根据新的需求和所碰到的问题，去不停的调整，最终越来越强大。所有框架都是从1.0版本到2.0、3.0......发展而来，而不是直接跳过最初版本到最终成熟版本。

　 　所以说我们在创建一个框架时，必须根据我们当前个人的技术能力、团队成功技术水平、时间、投入成本、项目现状（规模与需求复杂程度）、以后的发展前景来决定所要开发的框架的最终设计方案。当然也不是说不能一步到位，心有多大世界就有多大，只要个人能力和团队能力配得上，老板资金成本雄厚，时间充足，直接上大项目使用超级框架也完全没有问题。

# 框架和设计模式

　　度娘上说：框架、设计模式这两个概念总容易被混淆，其实它们之间还是有区别的。构件通常是代码重用，而设计模式是设计重用，框架则介于两者之间，部分代码重用，部分设计重用，有时分析也可重用。在软件生产中有三种级别的重用：内部重用，即在同一应用中能公共使用的抽象块;代码重用，即将通用模块组合成库或工具集，以便在多个应用和领域都能使用；应用框架的重用，即为专用领域提供通用的或现成的基础结构，以获得最高级别的重用性。

　　框架与设计模式虽然相似，但却有着根本的不同。设计模式是对在某种环境中反复出现的问题以及解决该问题的方案的描述，它比框架更抽象；框架可以用代码表示，也能直接执行或复用，而对设计模式而言只有实例才能用代码表示;设计模式是比框架更小的元素，一个框架中往往含有一个或多个设计模式，框架总是针对某一特定应用领域，但同一设计模式却可适用于各种应用。可以说，框架是软件，而设计模式是软件的知识。

# 框架的主要特点和要求

## 6.1代码模板化

　　框架一般都有统一的代码风格，同一分层的不同类代码，都是大同小异的模板化结构，方便使用模板工具统一生成，减少大量重复代码的编写。在学习时通常只要理解某一层有代表性的一个类，就等于了解了同一层的其他大部分类结构和功能，容易上手。团队中不同的人员采用类同的调用风格进行编码，很大程度提高了代码的可读性，方便维护与管理。

## 6.2重用

　　开发框架一般层次清晰，不同开发人员开发时都会根据具体功能放到相同的位置，加上配合相应的开发文档，代码重用会非常高，想要调用什么功能直接进对应的位置去查找相关函数，而不是每个开发人员各自编写一套相同的方法。

## 6.3高内聚（封装）

　　框架中的功能会实现高内聚，开发人员将各种需要的功能封装在不同的层中，给大家调用，而大家在调用时不需要清楚这些方法里面是如果实现的，只需要关注输出的结果是否是自己想要的就可以了。

## 6.4规范

　　框架开发时，必须根据严格执行代码开发规范要求，做好命名、注释、架构分层、编码、文档编写等规范要求。因为你开发出来的框架并不一定只有你自己在用，要让别人更加容易理解与掌握，这些内容是非常重要的。

## 6.5可扩展

　　开发框架时必须要考虑可扩展性，当业务逻辑更加复杂、数量记录量爆增、并发量增大时，能否通过一些小的调整就能适应？还是需要将整个框架推倒重新开发？当然对于中小型项目框架，也不必考虑太多这些内容，当个人能力和经验足够时水到渠成，自然就会注意到很多开发细节。

## 6.6可维护

　　成熟的框架，对于二次开发或现有功能的维护来说，操作上应该都是非常方便的。比如项目要添加、修改或删除一个字段或相关功能，只需要简单的操作，十来分钟或不用花太多的工夫就可以搞定。新增一个数据表和对应的功能，也可以快速的完成。功能的变动修改，不会对系统产生不利的影响。代码不存在硬编码等等，保证软件开发的生产效率和质量。

## 6.7协作开发

　　有了开发框架，我们才能组织大大小小的团队更好的进行协作开发，成熟的框架将大大减轻项目开发的难度，加快开发速度，降低开发费用，减轻维护难度。

## 6.8通用性

同一行业或领域的框架，功能都是大同小异的，不需做太大的改动就可以应用到类似的项目中。在框架中，我们一般都会实现一些同质化的基础功能，比如权限管理、角色管理、菜单管理、日志管理、异常处理......或该行业中所要使用到的通用功能，使框架能应用到某一行业或领域中，而不是只针对某公司某业务而设定（当然也肯定存在那些特定功能的应用框架，这只是非常少的特殊情况，不在我们的考虑范围）。