**Anycmd知识目录**

目录

[1 概述 3](#_Toc394649167)

[2 准备 3](#_Toc394649168)

[2.1 接口（Interface） 3](#_Toc394649169)

[2.1.1 标记接口 3](#_Toc394649170)

[2.1.2 接口的显示实现 4](#_Toc394649171)

[2.2 分类法、类（Class） 4](#_Toc394649172)

[2.3 对象（Object） 4](#_Toc394649173)

[2.3.1 值对象（Value Object） 4](#_Toc394649174)

[2.3.2 实体（Entity） 4](#_Toc394649175)

[2.4 时间类型（DateTime） 4](#_Toc394649176)

[2.5 原则、模式、最佳实践 4](#_Toc394649177)

[约定胜于配置原则 4](#_Toc394649178)

[不要重复你自己 4](#_Toc394649179)

[针对接口编程不要针对实现编程 4](#_Toc394649180)

[把文档放在靠近代码的地方 4](#_Toc394649181)

[每WebService方法只应接收一个参数 5](#_Toc394649182)

[一个系统对外只应有一个Url（统一资源定位符） 5](#_Toc394649183)

[3 Anycmd 5](#_Toc394649184)

[3.1 代码度量 5](#_Toc394649185)

[3.2 Anycmd领域概念 6](#_Toc394649186)

[3.2.1. 数据模式（Data Schema） 7](#_Toc394649187)

[3.2.2. 本体（Ontology） 7](#_Toc394649188)

[3.2.3. 实体（Entity） 7](#_Toc394649189)

[3.2.4. 动作（Action） 7](#_Toc394649190)

[3.2.5. 本体元素（Element） 7](#_Toc394649191)

[3.2.6. 实体（Entity） 7](#_Toc394649192)

[3.2.7. 信息字典（InfoDic） 7](#_Toc394649193)

[3.2.8. 组织结构（Organization） 7](#_Toc394649194)

[3.2.9. 编码 7](#_Toc394649195)

[3.2.10. 信息是什么（Info） 7](#_Toc394649196)

[3.2.11. 信息项（InfoItem） 7](#_Toc394649197)

[3.2.12. 信息标识（InfoID） 7](#_Toc394649198)

[3.2.13. 信息值（InfoValue） 7](#_Toc394649199)

[3.2.14. 信息项元组（InfoTuple） 7](#_Toc394649200)

[3.2.15. 操作（Operation） 7](#_Toc394649201)

[3.2.16. 操作的分类 7](#_Toc394649202)

[3.2.17. 命令（Command） 8](#_Toc394649203)

[3.2.18. 查询（Query） 8](#_Toc394649204)

[3.2.19. 用户（Account） 8](#_Toc394649205)

[3.2.20. 资源类型（ResourceType） 8](#_Toc394649206)

[3.2.21. 功能（Function） 8](#_Toc394649207)

[3.2.22. 权限（Permission） 8](#_Toc394649208)

[3.2.23. 委托（Delegation） 8](#_Toc394649209)

[3.2.24. 角色（Role） 8](#_Toc394649210)

[3.2.25. 组（Group） 8](#_Toc394649211)

[3.2.26. 日志（Log） 8](#_Toc394649212)

[3.2.27. 实体类型（EntityType） 8](#_Toc394649213)

[3.2.28. 实体属性（Property） 8](#_Toc394649214)

[3.2.29. 系统字典（Dic） 8](#_Toc394649215)

[3.2.30. 页面（Page） 8](#_Toc394649216)

[3.2.31. 菜单（Menu） 8](#_Toc394649217)

[3.2.32. 按钮（Button） 8](#_Toc394649218)

[3.2.33. 帮助（OperationHelp） 9](#_Toc394649219)

[3.2.34. 开发人员（Developer） 9](#_Toc394649220)

[3.2.35. 节点（Node） 9](#_Toc394649221)

[3.3 其它 9](#_Toc394649222)

[4 架构是什么 9](#_Toc394649223)

[5 整体架构 9](#_Toc394649224)

[5.1. 结构视图 9](#_Toc394649225)

[5.2. 功能视图 9](#_Toc394649226)

[5.3. 协作视图 9](#_Toc394649227)

[6 详细设计 9](#_Toc394649228)

[5.1 原则（Principle） 9](#_Toc394649229)

[5.2 模式（Pattern） 9](#_Toc394649230)

[5.3 实践（Practice） 10](#_Toc394649231)

[5.4 基础架构（Infrastructure） 10](#_Toc394649232)

[5.4.1 分层（Layer & Tier） 10](#_Toc394649233)

[5.4.2 模型（Model） 12](#_Toc394649234)

[5.4.3 模型数据验证（Validation） 14](#_Toc394649235)

[5.4.4 事务 15](#_Toc394649236)

[5.4.5 运行时（Runtime） 15](#_Toc394649237)

[5.4.6 总线（Bus） 15](#_Toc394649238)

[5.4.7 查询（Query） 15](#_Toc394649239)

[5.4.8 命令（Command） 15](#_Toc394649240)

[5.4.9 序列化 15](#_Toc394649241)

[5.4.10 数据库（Database） 15](#_Toc394649242)

[5.4.11 数据访问（DataAccess） 15](#_Toc394649243)

[5.4.12 业务逻辑（BusinessLogic） 15](#_Toc394649244)

[5.4.13 时间戳（TimeStamp） 15](#_Toc394649245)

[5.4.14 传输（Transfer） 16](#_Toc394649246)

[5.4.15 展示（Presentation） 16](#_Toc394649247)

[5.4.16 异常（Exception） 16](#_Toc394649248)

[5.4.17 框架、工具（Framework、Tool） 16](#_Toc394649249)

[5.4.18 配置（Config） 16](#_Toc394649250)

[5.4.19 文档（Doc） 16](#_Toc394649251)

[5.4.20 启动（Boot） 16](#_Toc394649252)

[5.4.21 生命周期与上下文（LifeCycle & Context） 16](#_Toc394649253)

[5.4.22 缓存（Cache） 16](#_Toc394649254)

[5.4.23 类（Category） 16](#_Toc394649255)

[5.5 业务架构 16](#_Toc394649256)

[5.6 数据架构 16](#_Toc394649257)

[5.7 应用架构 16](#_Toc394649258)

[5.8 技术架构 16](#_Toc394649259)

# 概述

**读者对象：**

Anycmd开发人员（维护、二次开发）。

**阅读建议：**

如果对“准备”章节中提到的知识不太熟悉最好先补充这些知识再回过头来阅读。

# 准备

这里不会对每一个知识点做详细的解说，只会对Anycmd中用到的和作者认为重要的知识做必要的表述。如果这些东西您都已经掌握了，不妨把这节作为知识的回顾。

## 接口（Interface）

### 标记接口

### 接口的显示实现

## 分类法、类（Class）

## 对象（Object）

### 值对象（Value Object）

### 实体（Entity）

## 时间类型（DateTime）

## 原则、模式、最佳实践

### 约定胜于配置原则

理解此原则需分清技术与业务边界。把握技术与业务的边界的关键是清楚哪些是所要解决的领域问题而哪些只是解决领域问题的手段。在Anycmd中约定胜于配置原则主要用于指导技术而非业务。如：Anycmd中的插件从何处导入就是技术问题，根据约定胜于配置原则约定插件从应用程序域基目录下的名为Plugins的文件夹中导入，除非有充分的理由否则不要将约定提取为配置。

### 不要重复你自己

Anycmd团队痛恨重复的功能、重复的代码，所以Anycmd团队在架构层次和代码编写层次均严格遵守不要重复自己的原则。这表现在Anycmd的展示层没有重复的页面，业务逻辑层没有重复的流程和代码等。

### 针对接口编程不要针对实现编程

Anycmd项目中全是针对接口编程，代码中找不到直接new服务对象（服务对象一般是无状态的对象，如数据访问层的仓储对象等）的代码，它们全是通过AppHost容器注入的。

### 把文档放在靠近代码的地方

要放在程序员容易看到的地方，放在别处没人去看。另外，文档要写在代码上写在别处等于日久天长必定与代码不同步，错误的文档不如没有文档。

### 每WebService方法只应接收一个参数

这里的WebService包括WebApi、WCF、WebService，这个参数属于“DTO（数据传输对象）”。换句话说凡是需要通过网络远程调用的方法，其接收的参数列表中应该只有一个参数。

1. 将服务的参数列表封装为一个数据传输对象提高了接口的易用性；
2. 将服务的参数列表封装为一个数据传输对象打破了参数列表的顺序关系，使接口参数与顺序无关。
3. 提高了接口的稳定性，众所周知参数是方法契约的一部分，这个契约包括参数的顺序。将服务的参数列表封装为一个数据传输对象的话就将参数列表的变化封装在了数据传输对象中。

### 一个系统对外只应有一个Url（统一资源定位符）

具体的说就是一个系统对外只应有一个服务接口，这个服务接口只对应一个定位符。不应通过不同类型的DTO对象来区分不同类型的消息，而应通过消息属性区分。一个系统只应有一个DTO对象类型。

# Anycmd

Anycmd虽是一个开源项目，但作者是将它视为产品来做的。

根据性质的不同可以将开发工作分为这样几种：做项目、做产品、做标准。一流的团队做文化、做标准；二流的团队做品牌、做资本；三流的团队做产品、做项目。由于Anycmd试图定义一套权限数据标准，并且试图提供一个拿来即可使用的最佳实践，所以它更接近“做标准”、“做产品”。做产品和做标准是需要一流的团队的，然而作者并不是，所以在Anycmd的开发中我付出了巨大的努力，如果您有学习或参与Anycmd的构建的想法的话希望您不要害怕前面的苦难。

Anycmd不能有“破窗户”，否则前人极力维护的干净的环境很快将不复存在。

破窗理论：一个房子如果窗户破了，没有人去修补，隔不久，其它的窗户也会莫名其妙地被人打破；一面墙，如果出现一些涂鸦没有被清洗掉，很快的，墙上就布满了乱七八糟、不堪入目的东西；一个很干净的地方，人们不好意思丢垃圾，但是一旦地上有垃圾出现之后，人就会毫不犹疑地抛，丝毫不觉羞愧。——摘自百度

## 代码度量

目前Anycmd有10万行纯C#代码（非C#代码不计），属于大型项目。

如何划分大中小型项目？下面是依据代码行数的一种参考方案，真正的要划分项目大小的话绝不能只基于一个视图观察，这里把Anycmd基于代码行数度量一下只是要说明Anycmd项目不算小项目，希望给参与进来的新成员一个基本的度量上的把握。

* 微型：只是一个人，甚至是半日工作在几天内完成的软件；
* 小型：一个人半年内完成的 2000 行以内的程序；
* 中型： 5 个人在 1 年多的时间内完成的 5000-50000 行的程序；
* 大型： 5-10 人在两年内完成的 50000-100000 行的程序；
* 甚大型： 100-1000 人参加用 4-5 年完成的具有 100 ， 0000 行的软件项目；
* 极大行： 2000-5000 人参加， 10 年内完成的 1000 万行以内的程序；

以上摘自：《软件工程概论》 郑人杰、殷人民编

这样的观点是以代码行作为计量标准的，认为代码行多的自然项目也就大了。

## Anycmd领域概念

这一章定义的概念均在Anycmd的代码实现上找到对应。它们对应的不是隐藏在边边角角的一两行丑陋的代码，而是多数对应的类或接口。这些概念和实现是Anycmd的核心。

### 数据模式（Data Schema）

### 本体（Ontology）

### 实体（Entity）

### 动作（Action）

### 本体元素（Element）

### 实体（Entity）

### 信息字典（InfoDic）

### 组织结构（Organization）

### 编码

### 信息是什么（Info）

### 信息项（InfoItem）

### 信息标识（InfoID）

### 信息值（InfoValue）

### 信息项元组（InfoTuple）

### 操作（Operation）

### 操作的分类

### 命令（Command）

### 查询（Query）

### 用户（Account）

### 资源类型（ResourceType）

### 功能（Function）

### 权限（Permission）

### 委托（Delegation）

### 角色（Role）

### 组（Group）

### 日志（Log）

### 实体类型（EntityType）

### 实体属性（Property）

### 系统字典（Dic）

### 页面（Page）

### 菜单（Menu）

### 按钮（Button）

### 帮助（OperationHelp）

### 开发人员（Developer）

### 节点（Node）

## 其它

# 架构是什么

# 整体架构

## 结构视图

## 功能视图

## 协作视图

# 详细设计

## 原则（Principle）

## 模式（Pattern）

Http请求响应架构模式

Restfull架构风格

DDD架构模式

CQRS架构模式

事件架构模式

## 实践（Practice）

最佳实践

## 基础架构（Infrastructure）

### 分层（Layer & Tier）

初级者可以通过拖拽控件和堆砌代码构建出复杂界面的应用但往往在分层面架构面前变得力不从心。这是为什么？在单层开发模式下几乎所有的逻辑被放在表现层，这种开发模式表现出了可观的生产力，生产力来自于强大的表现层框架而不是来自于我们的控件拖拽和代码堆砌。这里没有贬低某种开发模式的意思，每一个成熟的模式背后都有一套合理的理论，而这里要说的是：基于表现层框架的单层应用不适于开发复杂的程序。

分层对应了两个单词：Layer和Tier有何不同？简单来说Layer是逻辑分层，通常大家糊里糊涂的说的层就是逻辑层。而Tier指物理分层，如数据库层部署在一台服务器、WebService部署在另一台服务器这里分层就属于Tier的范畴。

本节着重说的是Layer。

#### Anycmd的分层

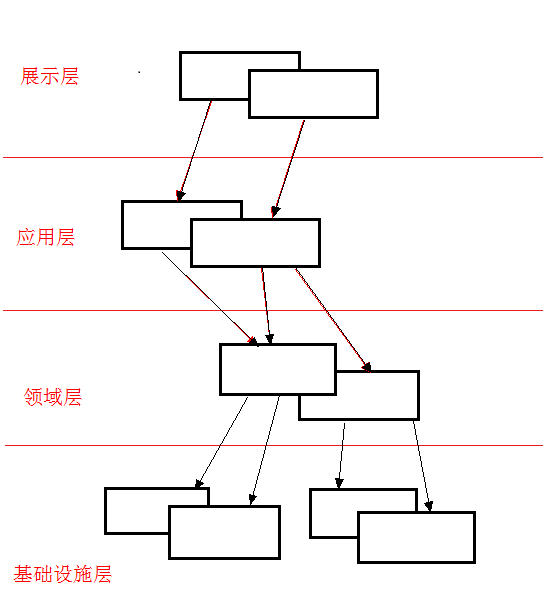
Anycmd按照DDD的推荐分为四层：展示层、应用层、领域层和基础结构层

|  |  |
| --- | --- |
| 展示层 | 负责向用户显示（View的职责）信息，并且解析用户命令（Controller的职责）。外部的执行者有时可能会是其他的计算机系统，不一定非是人（WebService） |
| 应用层 | 定义软件可以完成的工作（Task、Command、CommandHandler、ApplicationService），并且指挥具有丰富含义的领域对象来解决问题。这个层所负责的任务对业务影响深远，对跟其他系统的应用层进行交互非常必要。这个层要保持简练。它不包括业务处理规则或知识，只是给下一层中相互协作的领域对象协调任务、委托工作。在这个层次中不反映业务情况的状态，但反映用户或程序的任务进度的状态。 |
| 领域层 | 负责标识业务概念、业务状况的信息以及业务规则。尽管保存这些内容的技术细节由基础结构层来完成，反映业务状况的状态在该层中被控制和使用。这一层是Anycmd的核心。 |
| 基础结构层 | 为上层提供通用的技术能力：应用的消息发送、领域持久化，为用户界面绘制窗口等。通过架构框架，基础机构层还可以支持这四层之间的交互模式。 |

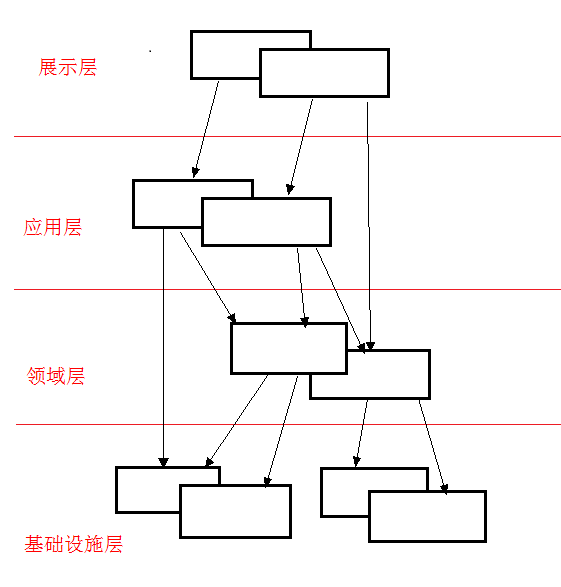
层是抽象的，当我们说层时它指的不是具体的实现，能够区分开层和层的实现两个概念很重要。在分层结构中，上层依赖于下层而不是依赖于下层的具体实现。有利于更好的理解概念，可以这样认为：所谓层表现出来就是一簇接口，而所谓实现就是一套实现了这簇接口的类。

**Anycmd是灵活分层**

如果在分层结构中加入一个约束，这个约束是上层依赖于且只依赖于离它最近的那个下层的话这种分层叫做严格分层，如下图所示：



Anycmd的分层不是这种激进的严格分层而是灵活分层，即Anycmd的上层可以依赖于任何下层，如下图所示：



#### 数据库层

#### 数据访问层

#### 业务逻辑层

#### 展示层

### 模型（Model）

这一节会提到很多种模型，将我们通常笼统的所说的模型做个分类。可能有人会说分的那么细累不累啊？恰好相反，人类发明分类法是为了帮助解决复杂的问题，怎么会累呢？

这一节会提到这么几种模型：领域模型、关系模型、数据访问模型、输入模型、数据传输模型、展示模型等。

#### 什么是领域模型（Domain Model）

**1 结构**

**2 运动**

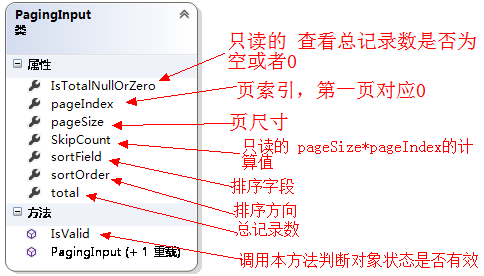
**什么是领域模型（Domain Model）**

我们可以在概念层次认为Domain Model就是在大的领域边界中的，可以用基于离散的思想来限定出的，承载“数据”和“运动”的小边界（个人给的定义，仅供参考）。定义中蕴含着这样一层意思：所谓模型乃是我们限定出的用于解决问题的承载着“数据”与“运动”的“问题边界”，也就是Model不一定跟现实中的真实物理对象一一对应，虽然大都一一对应。领域大边界由Model小边界来明确，Model小边界需要由领域大边界来给限定出问题的范围（因为宇宙是无穷的开放的，除了那个终极规律外的任何规律都是有适用范围的，不限定出范围就会寸步难行，无法认识任何问题了）。

领域模型承载的数据可以分为两个类别：Entitys和Value Objects。问题是客体，我们是主体。我们使用自己的能动性加入自己的意愿将数据组织起来，形成一个个单元，模拟出一个个有机的问题模型。问题模型不是静止不动的，宇宙是运动的。模型在收到外来的信号后将发生运动。

#### 什么是输入模型（InputModel）

通常方法的形参都是基本类型的依次排开的，像这样GetPlistNodes(int pageIndex, int pageSize,string sortField,string sortOrder, out long total)，这里的形参都是基本类型的依次排开的，当然也可以不是基本类型的但通常都是依次排开的。所谓输入模型就是封装这里的依次排开的形参的模型。比如上面那个例子中的pageIndex、pageSize、sortField、sortOrder和total就可以封装为下面这样一个类：



输入模型的书写习惯通常和其它模型不同，输入模型的对应形参的属性是采用camel格式编写的，非形参的属性依然采用Pascal格式。

将这些参数封装为输入模型后上面那个GetPlistNodes的方法就变成了GetPlistNodes(PagingInput pagingData);

有了输入模型这一层抽象就可以带来很多好处，比如它是放置验证输入数据合法性的逻辑的好场所。

#### 什么是数据访问模型（DataAccessModel）

通常情况下领域模型需要持久化，但领域模型不是专为持久化而设计的而是为业务逻辑的实现而设计的，所以持久化的时候就需要转换为另一种模型——数据访问模型。数据访问模型与领域模型不同，这是由于通常数据的持久层是关系数据库的缘故，关系模型和对象模型之间的阻抗使得两种模型之间需要一个转化工作。而数据访问模型就是抽象用来帮助实现这个转换工作的模型。

关系模型

数据访问模型

领域模型

面向关系数据库的数据访问模型通常都是扁平的，比如ADO.NET的IDataReader模型。

#### 什么是数据传输模型（DataTransferModel）

#### 什么是展示模型（ViewModel）

合法性

传输模型

展示模型

输入模型

合法性

模型转换

### 模型数据验证（Validation）

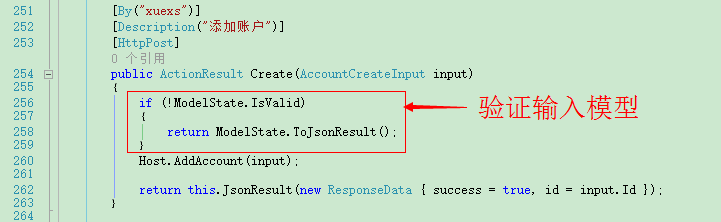
Anycmd对数据合法性的验证主要分布在表现层模型（ViewModel）中和领域模型（DomainModel）中，表现层所做的是基本的数据输入合法性的验证，而领域模型中所做的验证是涉及到业务逻辑的验证。表现层模型又分好几类，具体来说Anycmd对数据合法性的验证主要是在表现层输入模型（InputModel）中完成的。具体的验证方案可以依赖.NET的DataAnnotations验证框架实现，也可以通过anycmd的元数据系统配置实现。

拿添加账户举例，添加账户的功能对应有一个AccountAddInput输入模型



输入模型上封装的是添加账户所需要是输入项，在这些输入属性上添加DataAnnotations标记实现对输入数据的验证。

相应的Action方法代码如下：



### 事务

### 运行时（Runtime）

### 总线（Bus）

### 查询（Query）

### 命令（Command）

### 序列化

数据模型序列化而非领域模型序列化

### 数据库（Database）

### 数据访问（DataAccess）

### 业务逻辑（BusinessLogic）

### 时间戳（TimeStamp）

涉及到分布式时间戳就不那么不起眼了，时间戳必须被拿到台面上说。关键的只有一句话：在Anycmd系统中所有本地的时间戳全部使用本地时间，而当时间戳需要传送到其它系统时全部转化为Utc时间戳，同样Anycmd接收来自其它系统是时间戳的时候全部视为Utc时间戳，接收时转化为本地时间戳。

### 传输（Transfer）

### 展示（Presentation）

### 异常（Exception）

### 框架、工具（Framework、Tool）

### 配置（Config）

### 文档（Doc）

### 启动（Boot）

### 生命周期与上下文（LifeCycle & Context）

### 缓存（Cache）

### 类（Category）

## 业务架构

## 数据架构

## 应用架构

## 技术架构