DaaS 概要与建模

成都双链科技有限公司

修订历史

日期	变更类型	变更人	说明
2019.3.13	创建	张喜来	创建初期版本
2019.3.14	更改	张喜来	增加后端定制功能
2019.3.15	更改	张喜来	增加前端定制功能
2019.10.14	更改	李亚青	重构模型结构原理
2020.01.02	新增内容	李亚青	新增 Change Request 高级应用描述
2020.02.04	新增内容	王煜东	新增 images()

目录

前言

本手册分为三个部分，第一个部分是讲述如何建模，并且在 5 分钟内生成一个基本系统并部署，产品经理和开发均可阅读，第二部分针对后端开发人员定制系统后端，第三部分针对前端开发人员定制前端系统。每个部分都讲述原理，工具，方法，过程，验证和最佳实践。

1. 概述

第一部分为建模：建模的过程主要是通过客户或者产品经理提供领域模型和界面原型，使用 XML 语法，建立不同类型的字段和对象之间的关联。

第二部分为后端定制：后端开发通过在特定的目录中使用或者加入新的 Java 类，这些 Java 类继承生成的代码来扩展系统的后端功能。

第三部分为前端定制：前端端开发通过在特定的目录中使用或者加入新的 React 类，这些 React 类继承生成的代码来扩展系统的界面功能。

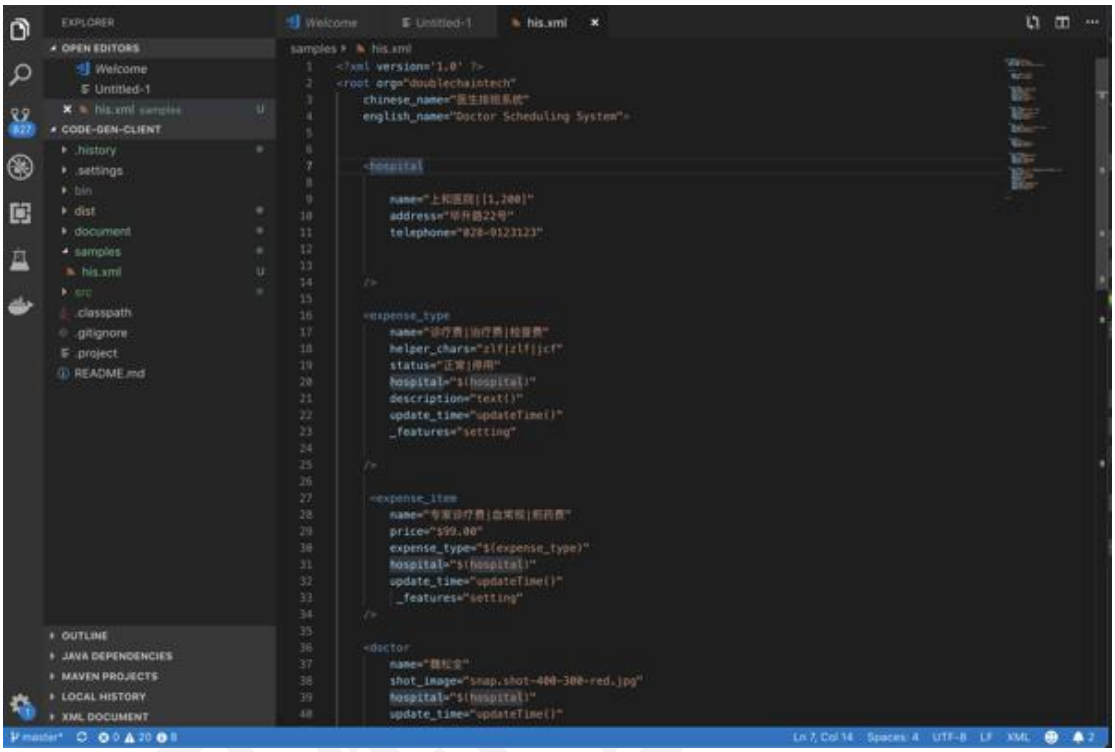
2. 建模

2.1 建模原理

建模的过程主要是通过特定格式的 XML 文件描述将要开发的软件功能业务对象之间的关系和自身属性的过程，系统通过这些关系和属性进行关系推演，发

现基本的逻辑和业务。这些信息会在后续的处理过程中，生成前端界面，后台服务，数据库规格定义和用于测试和演示的数据。

2.2 建模工具



理论上，编写 XML 文件任何文本编辑器都可以，我们推荐使用微软公司出品的 VS Code，该软件在 Windows， Mac，Linux 下均可以使用。

该软件的主界面如下:

2.3 建模准备

在开始建模之前，需要对即将要开发的业务系统的业务有一个比较深入的认识，展开业务系统的需求分析工作，充分了解高层次的分析领域模型：

首先，有哪些角色会使用这个系统，也就是我们在系统设计过程中所提到的主动对象，然后进一步完成角色的归类与抽象；

其次，将要建模的系统将要执行或处理哪些事物——有哪些事物可以被管理，也就是我们在设计过程中将会提到的被动对象；

再次，进一步分析主动对象与被动对象之间的业务逻辑关联关系，进行需求分析的用例整理，也就是进行对象关系的整理；

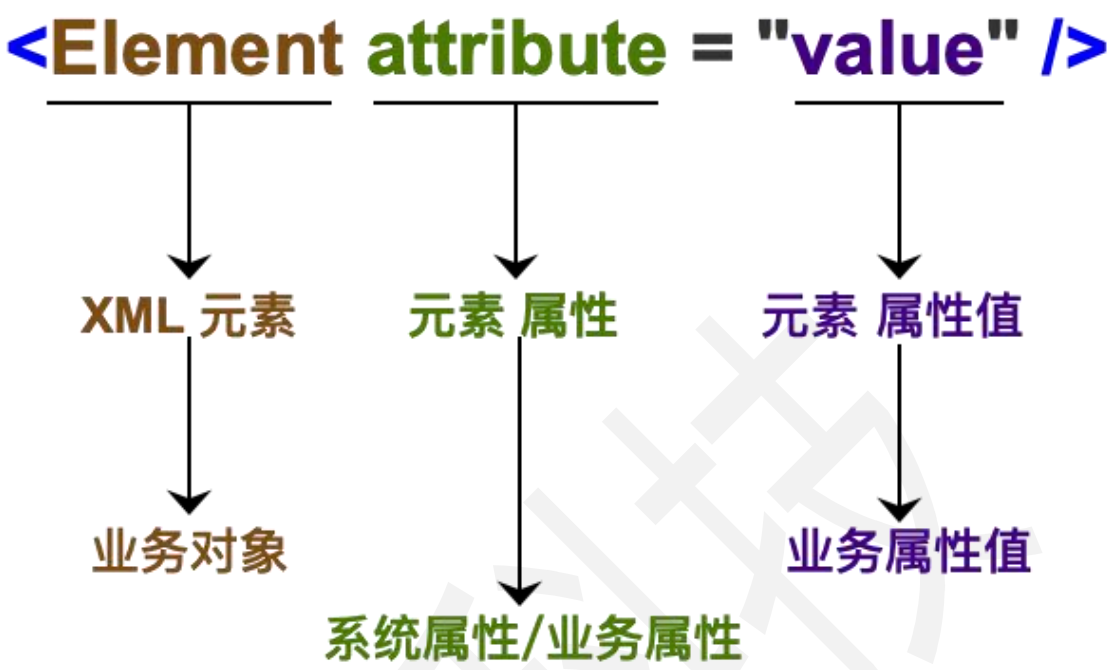
在需求分析阶段的，UI 界面原型设计、业务系统中的业务名词及术语定义、系统交互状态定义等输入信息，都可以帮助模型定义人员定义业务对象，确定业务属性字段，快速完成建模准备工作，进行模型 XML 定义工作；

2.4 模型结构原理

2.4.1 初识 XML 文件

XML 文件是一种 W3C 规范的国际通用标记语言，利于计算机处理和人工编写。XML 有很多高级特性，本手册使用 XML 标记语言一个最简单的子集来构建业务模型。

2.4.2 业务建模 XML 结构



如上图所示，使用 Code Generator 系统进行建模，只使用了 XML 标记语言中定义 XML 标签的最基本的元素，很简单吧：

✓ XML 标签：

每一对尖括号闭合的 XML 标签用来定义一个业务对象模型。

```
<user
name = "张三|[2,10]"
gender = "男|女"
user_status = "$(user_status)"
/>
<user_status
name = "已登录|已退出|已注销"
code = "LoggedIn|LoggedOut|Destroyed"
platform = "$(platform)"
/>


<user
name = "张三|[2,10]"
user_status = "$(user_status)"
>
<user_status
name = "已登录|已退出|已注销"
code = "LoggedIn|LoggedOut|Destroyed"
platform = "$(platform)"
```

```
>
</user_status>
</user>
```

注，以上两种尖括号对闭合方式，所标识的功能都是一样，通过尖括号对 XML 标签的嵌套，只是在编辑视图上对定义对模型数据进行包裹；

- ✓ XML 元素名称：

  每对闭合标签中的 XML 元素的名称，用来作为业务对象模型的名字，所以我们在定义业务对象模型的时候，要注意对象的取名；

  当需要添加对象模型的时候，模型定义用户可以通过增加多个不同的闭合的 XML 标签，来完成业务对象的添加；

- ✓ 元素属性：

  通过添加属性，可以为业务对象添加用于系统配置的系统属性以及用于业务描述和业务定义的业务属性；

  在业务对象的属性添加过程中，属性以及属性值都是成对出现的；

- ✓ 元素属性值：

  通过属性值，可以完成对业务对象属性字段在可理解语义层面上的属性定义；

  模型定义人员在进行模型定义的时候，可以使用用例中的例子或数据实例化的对业务属性字段进行描述，在业务属性值描述完成的时候，也就完成了业务属性字段的功能定义；

  并且，在完成模型定义以后，系统将可以利用这些输入的例子数据，生成后续开发调试过程中所需要的用例数据。

2.4.2.1 XML 元素 vs 业务对象

通过创建语义化的、能够具有业务描述特征的 XML 元素来创建和管理业务对象；

通过 XML 元素的属性以及属性值来描述业务对象的详细属性、系统约束，以及建立起不同业务对象之间的关联关系。

例如

```
<user
name = "张三|李四|王二|[2,10]"
avatar = "avatar.jpg"
mobile = "13812341234"
user_status = "$(user_status)"
last_login_time = "updateTime()"
platform = "$(platform)"
/>

<platform
chinese_name = "代码生成平台"
englisth_name = "Code Generation Platform"
organization = "doublechaintech"
/>

<user_status
name = "已登录|已退出|已销户"
code = "LoggedIn|SignOut|Destroyed"
 platform="$(platform)"
   _features="status"
   _identified_by="code"
/>

<order_status
   name="处理中|待支付|支付中|已完成|已取消"
   code="NotReady|PendingPay|Paying|Completed|Cancelled"
   platform="$(platform)"
   _features="status"
   _identified_by="code"
/>
<!--
业务对象：  order_status
业务属性：name, code, platform
业务属性值：处理中|待支付|支付中|已完成|已取消,
NotReady|PendingPay|Paying|Completed|Cancelled,
$(platform)
系统属性：_features,  _identified_by
系统属性值：  status, code
-->
```

业务对象的对象名称命名规则和创建注意事项：

使用语义清晰、且规范的英文单词--便于后续生成中台用户界面时的翻译；

使用小写英文单词进行对象描述；

使用下划线来连接多个单词；

不建议在业务对象中使用英文缩写；

业务对象使用英语单词的名词进行描述；

2.4.2.2 元素属性 vs 系统属性/业务属性

通过元素的属性，对创建的各业务对象进行业务描述。

在 Code Generator 系统中，我们通过下面两类属性来增强对对象的业务描述以及建立对象与对象之间的关系：

系统属性：由 Code Generator 平台提供的、带来智能化自动化开发以及功能设计的辅助属性，系统属性随着 Code Generator 平台的不断演进，其功能不断的增强，给开发者的开发工作带来更大的便利。

业务属性：业务属性为描述业务对象自身数据字段以及描述业务对象之间管理关系的字段。

业务属性名称命名规则和创建注意事项：

- ✓ 使用语义清晰、且规范的英文单词--便于后续生成中台用户界面时的翻译；
- ✓ 使用小写英文单词进行对象描述；
- ✓ 使用下划线来连接多个单词；
- ✓ 不建议在业务对象中使用英文缩写；
- ✓ 业务对象使用英语单词的名词进行描述；

Code Generator 平台当前提供的系统属性有：

- ✓ **_identified_by**：用于标识当前业务对象在系统中如何进行索引
  默认未添加该属性的业务对象都是通过 id 进行关联查找
  _identified_by 的取值范围为当前业务对象的业务属性字段名称
- ✓ **_feature**：根据属性值来标识当前业务对象的业务特性

当前阶段，平台提供以下属性值来描绘业务对象的业务特性。

系统属性标识	含义	影响
setting	将当前业务对象标识为 设置类型	在中台的菜单和仪表盘，与该对象相关的对象视图上，可以在设置设置该对象
log	将当前业务对象标识为 日志类型	log 为只读列表，不可以增删改
status	将当前业务对象标识为 状态类型	在界面上不提供增删改的功能，并且数据只是生成在模型 XML 定义的
custom	标识当前业务对象生成的业务中台代码在中台需要进行定制，并生成默认的定制仪表盘和表格	会生成前端默认的定制仪表板和表格
blockchain	标识当前业务对象可以自动上链	自动上链，目前还不支持
iot	标识当前业务对象为物联网节点	物联网节点，目前暂不支持
uiform	标识当前业务对象生成后台开发的 UIForm 基础	

当某个业务对象具有多个特性时，可以使用逗号分隔符分隔多个系统属性值。

- ✓ **_vg4+attribute name**：通过"_vg4" 前缀 叠加上 对象引用的业务属性字

段来完成生成中台对当前对象在视图上的内容分组

```
<merchant
  name="双链科技有限公司|[1,100]"
  owner="$(user)"
  merchant_type="$(merchant_type)"
  merchant_status = "$(merchant_status)"
  create_time="createTime()"
  last_update_time="updateTime()"
  platform="$(platform)"

  _vg4platform="商户管理"
  _vg4owner="商户管理"
/>
<!--

定义从平台的角度上看 业务模型对象 <merchant /> 是
-->

<merchant_warehouse
  name="天府二街东仓库|[1,100]"
  address="四川省成都市高新区南华路 100 号|[1,100]"
  merchant="$(merchant)"
  latitude="30.31251|[-180.00,180.00]"
  longitude="104.02931|[-180.00,180.00]"
  create_time="createTime()"
  last_update_time="updateTime()"

  _vg4merchant="商户管理"
  _features="log"
/>

<user
  name="张三|[1, 100]"
  mobile="13812341234"
  avatar_image="avatar.jpg"
  user_status="$(user_status)"
  weixin_openid="wx123456789abcdefghijklmn|[0,128]"
  weixin_appid="wxapp12098410239840|[0,128]"
  access_token="jwt_token_12345678|[0,128]"
  create_time="createTime()"
  last_update_time="updateTime()"
  platform="$(platform)"
```

```
  _vg4platform="员工管理"
/>

<employee_assignment
  user="$(user)"
  working_merchant="$(merchant)"
  working_location="$(merchant_warehouse)"
  start_date="2099-06-16"
  end_date="2099-09-18"
  role="$(role)"
  create_time="createTime()"
  last_update_time="updateTime()"

  _vg4working_merchant="员工管理"
  _vg4user="员工管理"
  _vg4working_location="员工管理"
  _features="setting"
/>
```

例如 _vg4platorm 针对 功能属性字段 platform="$(platform)" 定义从平台(platform)的角度上看 业务模型对象 <merchant /> 将会被划分到 “商户管理”当中;

_vg4owner 针对 功能属性字段 owner = "$(user)" 定义从 用户(user)的角度上看业务模型对象 <merchant /> 将会被划分到 “商户管理”当中;

2.4.2.3 元素属性值 vs 业务属性值/系统属性

属性值是使用带有语义描述的文本对属性字段进行描述的字符串字面量，通过属性值内容的字符串字面量详细描述属性字段、建立业务对象与业务对象之间的关系。

对于业务对象的每个业务属性字段，模型构建人员在定义模型的时候，可以用实例化语义来描述各业务属性字段的样例值、字段约束、业务对象关联关系。当模型文件提交代码生成云以后，代码生成系统会根据业务属性字段的字面量来推测该字段在数据库中的数据类型。

对于系统属性，根据属性值的字面量来构建该业务对象在系统中的功能约束。

当前支持推测的业务属性值字面量类型有：

业务属性值字面量类型	例子	说明
普通字符串字面量	"成都双链科技有限公司|双链科技有限公司"	非空的中英文字符串文本，表示该字面量修饰的业务字段的类型为普通字符串类型； 默认普通字符串字面量中的提供的字符串样本的长度将会作为该业务属性字段的长度检查约束条件； 在当前 Code Generator 系统中，普通字符串类型的字段最小长度为 2 个字符，最长长度为 20 个字符；
整型数字字符串字面量	"3|100|1000"	不包含中英文字符、不包含小数点的数字 0~9 的数字字符串，表示该字面量所修饰的业务字段的类型为整型字符串类型； 整型数字字符串样例中提供的数字的最大值和最小值将会作为该业务字段最小值和最大值的约束检查条件；
浮点型数字字符串字面量	"1.2|2.7|3.6"	带小数点的浮点数字字符串，表示该字面量所修饰的业务字段的类型为浮点型字符串；

		浮点型字符串样例中提供的浮点精度将会约束该业务字段所接受的数值精度；
货币字符字面量	"$8.56"	英文"$"符号 与 数字 或 浮点数字字符串的组合表示货币字符串字面量，表示该字面量修饰的业务属性字段是货币字段；
对象引用字符串字面量	"$(platform)" "$(user_status)" "$(merchant)"	通过英文"$"符号，英文"()" 包裹业务对象名称 来表示对象引用字面量； 对象引用字符串字面量表示该字面量修饰的业务字段的类型为当前所包裹的对象引用类型； 请确保，所引用的对象的存在性； 通过业务字段的对象引用，建立起了当前业务对象与被引用业务对象的多对一的关系；
日期时间字符串字面量	"20991212T18:25:59"	匹配以下范式的日期时间字符串： "YYYYMMDDTHH:MM:SS" 日期时间字符串字面量表示该字面量修饰的业务字段的类型为日期时间类型；
手机号码字	"13812341234"	当字面量的内容匹配符合中国大陆手

符串字面量		机号码规范的 11 位手机号码的数字：13，15，17，18，19 开头的 11 位数字认为该字面量为手机号码字面量； 手机号码字面量标识该字面量修饰的业务字段的类型为手机号码类型； 手机号码类型的数据在系统中默认对手机号码的处理是中间四位数字隐藏展示；
身份证号码字符串字面量	"510124198508281128"	当字面量的值符合中国大陆身份证号码规则的 18 位数字/英文字符串时，认为该字面量为身份证号码字面量；
图片字符串字面量	"banner_200_200_red.jpg" "https://xubai-test.oss-cn-beijing.aliyuncs.com/app/test/person_1.jpg"	以 ".jpg", ".png", ".gif", ".jpeg", ".tiff", ".bmp" 结尾的字符串 表示所修饰的字段的类型为图片字段； xxx_width_height_color.imageAppendix：将可以辅助在代码生成的时候，生成所定义尺寸、颜色的默认图片； 图片字符字面量修饰的业务字段类型为图片类型； 图片类型字段存放的信息为图片存放

		的路径，最长可以存放 1024 字节的字符；
系统函数字符串字面量	"createTime()" "updateTime()" "text()"	Code Generator 平台内置的系统函数，用于在代码生成时为该字段生成相应的处理函数或者为相应字段的类型进行定义。 当前支持的系统函数有： createTime() updateTime() future() past() now() text()
系统属性字符串字面量	"setting", "log", "status", "uiform"	详见 2.4.2.2 中的系统属性及其属性值的描述
URL 地址字符串字面量	"https://www.baidu.com"	以 "https:", "http:" 开头的，并且不满足图片字符串字面量规则字符串表示该字段的类型为 URL 字段； URL 地址字符串字面修饰的字段长度为 1024 字节；
营业执照字符串字面量	"91510100MA6C66P74M"	符合中国大陆三证合一证件号码的字符串；

布尔字符串字面量	"false" "true"	布尔值字面量；
性别字符串字面量	"男" "女" "male" "female"	中英文性别字符串；
邮件地址支字符串	"liyaqing@doublechaintech.com"	带有"@"符号，且符合电子邮件地址规范的字符串；
图像列表（注：尚未发布）	Images()	DaaS 标准图像列表类型。 在数据库内部，以 json 格式存储；在 DaaS 生成的标准界面上，自动支持。 提供多张（可以通过[m,n]来限定图片数量）图片的展示，删除，上传。

字面量修饰符

修饰符	例子	说明
$	price = "$100.00" platform = "$(platform)"	"$" 和数字组合在一起用来表示该业务字段表示的信息是价格信息 "$" 和 对象字符串 组合在一起，表示对对象的引用
[]	introduce = "简介|[2,100]"	[] 用于文本型字符串业务字段的长度进行约束，将会根据约束规则，自动生成约束检查方法

		[min, max] 声明相应字段值最小长度以及最长长度； min, max 为整数数字，请确保 min 和 max 大小关系；
?	platform = "$(platform?)	"?"和对象引用在一起使用，用于表示在当前对象中，"?"问号修饰的对象引用业务字段在进行对象完整性检查时，该业务字段是可选字段。
|	name="纸类|金属|塑料|木材|橡胶|[1,30]"	"|" 在构造多个用例数据的时候，表示用例数据值之间 或 的关系

系统将会根据属性值字面量的类型、引用关系、约束关系、系统配置生成业务关系代码。

2.5 实战：构建一个简单的系统

接下来，我们将通过构建一个医院医生排班系统的案例来展示如何创建业务模型、定义业务字段，展示自动代码生成过程。

2.5.1 新建模型文件

用 VS Code 新建一个 XML 文件 hospital.xml ，用来存放业务模型信息。

2.5.2 添加根节点对象

```
<?xml version='1.0'?>
<root
chinese_name = "医生排班系统"
english_name = "Doctor Scheduling System"
org = "doublechaintech"
>
</root>
```

```
<!--
模型根节点: root
属性名称: chinese_name, english_name, org
属性值: "医生排班系统", "Doctor Scheduling System", "doublechaintech"
-->
```

属性名称	描述	参考
chinese_name	root 节点的中文名称	显示在中文版的中台界面上
english_name	root 节点的英文名称	显示在英文版的中台界面上
org	开发公司的名称	后台代码的包名域名，doublechaintech 建辉显示为 com.doublechaintech

2.5.3 添加基础对象节点对象

```
<?xml version='1.0'?>
<root
chinese_name = "医生排班系统"
english_name = "Doctor Scheduling System"
org = "doublechaintech"
>
<hospital
name = "上和医院|[1,200]
address = "毕升路 22 号"
telephone = "028-91231234"
/>
<doctor
name = "白居易|[2,10]"
avatar = "user_400_300_red.jpg"
hospital = "$(hospital)"
update_time = "updateTime()"
/>
</root>
<!--
模型根节点: root
属性名称: chinese_name, english_name, org
属性值: "医生排班系统", "Doctor Scheduling System", "doublechaintech"
```

-->

最佳实践：把不重要的属性放后面，这样显示过长的表格的时候就可能隐藏起来。

标签和属性名最好使用无拼写错误的英文，这样很容易翻译为中文。拼写错误的单词，系统会给出警告。后续章节会指示如何发现拼写错误。

有了医院以后，我们希望管理医院里医生信息。以下的例子代码医院里面可以管理很多医生，医生在医院里面工作(这里没有考虑医生在多个医院工作的情况)。

黑体部分建立了医生和医院的关联，其中$(hospital)中的 hospital 必须是已经定义的对象。名称可以改为其他。

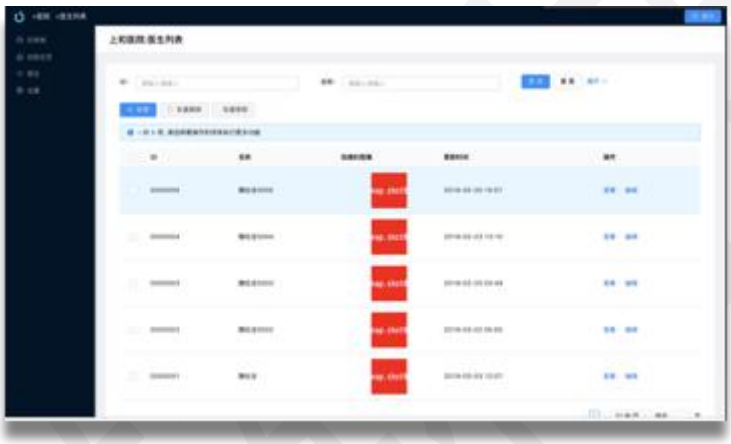
有了这个文本文件，我们可以根据指令上传文件，执行相关命令，就可以生成系统了。

2.5.4 系统基本功能中台页面

登录界面

App 选择界面

趋势统计分析界面



医生列表

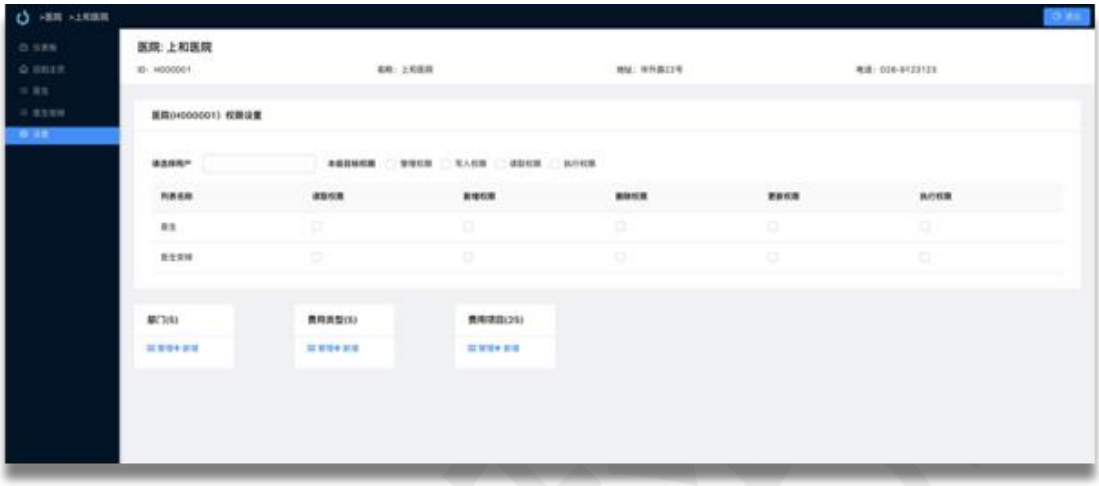
权限访问控制设置

每个字段按照定义顺序生成，ID 是系统生成，除了文本字段以为，如果产品经理输入了 shot_image="snap.shot-400-300-red.jpg"，系统就自动推断为是一幅图片，这样在展示的时候，系统就自动展示为图片，自动生成上传下载的代码，400-300-red 标明了尺寸和底色。更多规则请参考附表。

2.5.5 添加排班管理

如下面内容所示，添加排班管理的业务对象以及属性字段信息

权限管理增加了“医生安排“ 相应项目



我们会在后面解释为何能生成这些界面。

分析

医生排班系统的核心是这么一句话:

某个医生在某个科室在上午，下午或者晚上值班，每个排班可能有若干个号，用完为止。

记下这个事，我们需要如下的结构

```
<?xml version='1.0'?>
<root
chinese_name = "医生排班系统"
english_name = "Doctor Scheduling System"
org = "doublechaintech"
>
<hospital
name = "上和医院|[1,200]
address = "毕升路 22 号"
telephone = "028-91231234"
/>
<doctor_schedul
  name="2019 年 3 月 11 日魏松全在内分泌科坐班收诊疗费,每个 10"
  doctor="$(doctor)"
  schedule_date="2019-3-11"
```

```
  period="上午|下午|夜班"
  department="$(department)"
  available="20"
  price="$123.99"
  expense_type="$(expense_type)"
  create_time="createTime()"
  update_time="updateTime()"
  hospital="$(hospital)"
/>
<expense_type
  name="诊疗费|治疗费|检查费"
  helper_chars="zlf|zlf|jcf"
  status="正常|停用"
  hospital="$(hospital)"
  description="text()"
  update_time="updateTime()"
  _features="setting"
/>
<expense_item
  name="专家诊疗费|血常规|煎药费"
  price="$99.00"
  expense_type="$(expense_type)"
  hospital="$(hospital)"
  update_time="updateTime()"
  _features="setting"
/>
<doctor
  name="魏松全"
  shot_image="snap.shot-400-300-red.jpg"
  hospital="$(hospital)"
  update_time="updateTime()"
/>
<department
  name="放射科"
  hospital="$(hospital)"
  update_time="updateTime()"
  _features="setting"
/>
<doctor_assignment
  name="魏松全在内分泌科室上"
  doctor="$(doctor)"
  department="$(department)"
  update_time="updateTime()"
  _features="setting"
```

```
/>
</root>
<!--
模型根节点: root
属性名称: chinese_name, english_name, org
属性值: "医生排班系统", "Doctor Scheduling System", "doublechaintech"
-->
```

里面提及了 doctor, department, expense_type，没有建立，我们让这些对象都存在于一个医院下。

在上述例子中，如果我们不需要，expense_type 等对象显示到主仪表盘上，我们加上了一个特性 _features="setting"。加入了 setting 后，expense_type 就会出现在系统设置区域内，主面板不再出现该对象列表。

2.6 高级应用：添加 Change Request

随着业务系统复杂度的增加，数据成为了 IT 系统中最宝贵的资产。为进一步提高 IT 系统的可靠性，实现数据变化的追溯性、业务事件回放，我们参考了许多单一事件源驱动的软件系统，为 daas 系统添加了基于 Change Request 的 Event Source 方案，业务系统的所有数据变化，都需要产生一个 Event Source，然后经由 Change Request Gateway 处理，再讲处理完后的事件派发到系统的各个节点完成变更。

2.6.1 Change Request 的 Event Source 方案组成结构

Change Request 的 Event Source 方案由以下三个部分组成：

- Change Request
- Change Request Type
- Change Event

2.6.2 Change Request

```
<change_request
```

```
  name="数据变更|[1,50]"
  create_time="createTime()"
  remote_ip="remoteIp()"
  request_type="$(change_request_type)"
  platform="$(platform)"
  _vg4platform="数据管理"
  _features="change_request"
/>
```

Change Request 数据模型，如上表所示。

您可以在您的 XML 模型中，复制粘贴，贴入 Change Request 的模型，将 platform 属性挂载到您的项目模型文件 root 根节点下的第一个子节点上。

对于 request_type，参考 2.6.3 Change Request Type。

2.6.3 Change Request Type

```
<change_request_type
  name="商户入驻|商户注销|用户注册|发布招标|提交投标|签署合同|发起合同变更|创建订单|支付订单|产品入库|产品出库|申请发票|登记收票|登记损耗|过磅称重"
  code="ADD_MERCHANT|WITHDRAW_MERCHANT|USER_REGISTRATION|CREATE_TENDERING|CREATE_TENDER|CREATE_CONTRACT|CREATE_CONTRACT_CHANGE|CREATE_ORDER|CREATE_PAYMENT|STOCK_CHECKIN|STOCK_CHECHOUT|CREATE_INVOICE_APPLICATION|CREATE_INVOICE_REGISTRATION|CREATE_DESTROYED|CREATE_WEIGHT_RECORD"
  icon="shop|poweroff|user-add|file|file-zip|file-done|interaction|shopping-cart|transaction|export|import|account-book|audit|file-markdown|dashboard"
  display_order="1|2|3|4|5|6|7|8|9|10|11|12|13|14|15"
  bind_types="text()"
  step_configuration="text()"
  platform="$(platform)"
  _features="cr_type"
  _identified_by="code"
  create_time="createTime()"
/>
```

Change Request Type 的数据结构，如上表所示，在项目中使用的时候，请原样复制照搬，配对修改&添加 name，code，icon，display_order 四个属性即可。

Change Request Type 中聚合了系统的所有 Event Source 类型，即，当有新的 Change Event 增加时，需要将标识相应新增 Event 的 cr_type 添加到

Change Request Type 中。

2.6.4 Change Event

```
<weigh_record_event_create
   name="进行称重"
   change_request="$(change_request)"
   create_time="createTime()"
   _features="event"
   _bind_with_cr_type="CREATE_WEIGHT_RECORD"
   weigh_record="$(weigh_record)"
/>
<stock_change_event_checkin
   name="商品入库"
   change_request="$(change_request)"
   create_time="createTime()"
   _features="event"
   _bind_with_cr_type="STOCK_CHECKIN"
   stock_change="$(stock_change)"
/>
<stock_change_event_checkout
   name="商品出库"
   change_request="$(change_request)"
   create_time="createTime()"
   _features="event"
   _bind_with_cr_type="STOCK_CHECHOUT"
   stock_change="$(stock_change)"
/>
```

Change Event，即业务变更事件，每个业务变更事件就是一个 Change Event，需要为其添加一个 Event 模型。

每个 Event 模型至少包含以下属性：

name：事件名称

change_request：关联到 change_request 模型

create_time：事件创建的时间，默认使用系统函数 createTime()

_features="event"：通过 features 标识当前模型是 Event 类型

_bind_with_cr_type：定义标识当前 Event 的唯一标识符，定义完成后，将该标识符添加到 Change Event Type 中。

另外，将当前 Change Event 修改所涉及到的模型对象，都添加作为该 Change Event 模型的属性。

3. 后台业务定制

3.1 概述

SYSX 因为以下三点使得二次开发变得容易，也实现了自动生成的代码和手写代码隔离

- ✓ 吸收了大型可定制产品设计思想
- ✓ 经过若干个迭代调整
- ✓ 数个实际上线的项目中调整

3.2 工具

后端开发使用到如下工具：

- ✓ Oracle JDK1.8 或者 OpenJDK1.8
- ✓ 开发 IDE 环境 Eclipse 或者 Idea 较新版本
- ✓ 编译环境 Gradle5.1+

3.3 运行时环境

- ✓ 实时类装载和调试服务器 Resin-3.1.16
- ✓ MySQL 5.7+
- ✓ Redis 3.2+
- ✓ Kafka, ZooKeeper, Arangodb 可选，后期运维人员即可处理

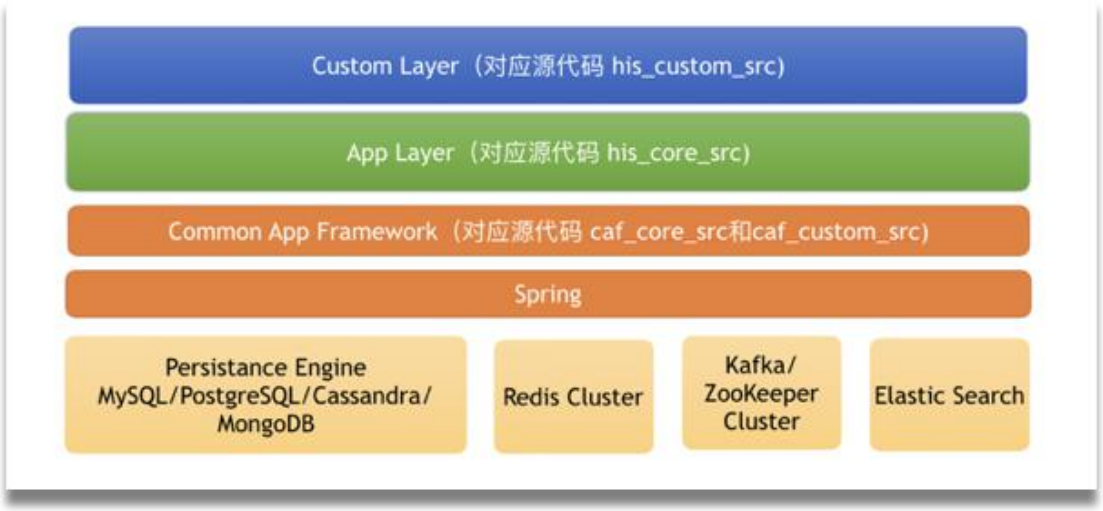
3.4 方法

3.4.1 逻辑架构

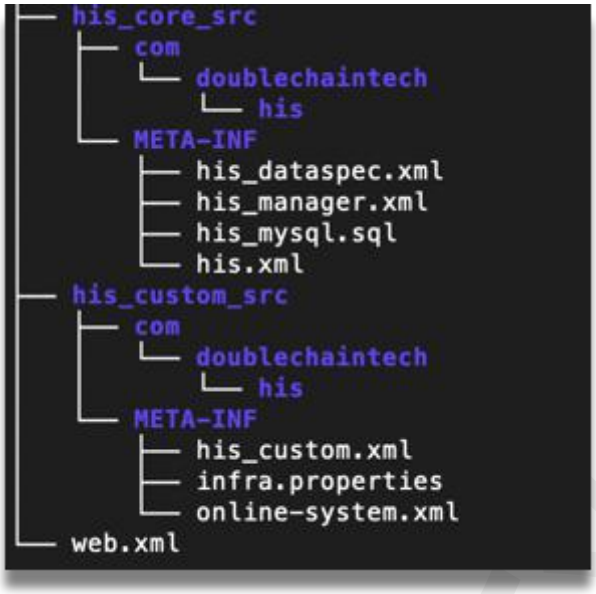
下图就是软件系统的逻辑架构，层和层之前使用目录进行了完全的隔离。在定制的时候，core 里面的代码只是作为参考，因为每次模型变更后生成代码都会被覆盖。后端代码的位置在 bizcore/WEB-INF/里面，和 war 目录结构保持一致的原因是开发人员更容易找到 java 文件编译成的 class，也利于在 Resin 中重新装载类以免去系统启动的时间。

本章节假设读者了解如何利用重载，调用父类方法，来增加或者修改已经存在的 Java 对象行为。

bizcore 下面有 gradle 文件，通常情况下运行 gradle classes 来编译并同步目录即可。



3.4.2 项目目录介绍



以下是以 his 系统的为例项目介绍

这个目录显示了高层的包结构，其中

his_core_src 中

- ✓ his_dataspec.xml: 规格文件，在开发时候可以作为参考
- ✓ hist_manager.xml: 所有的 manager bean 的定义文件
- ✓ his_mysql: 该系统的 mysql 脚本和数据初始化脚本
- ✓ his.xml, 基础的 DAO BEAN 定义文件

his_custom_src 中

- ✓ his_custom.xml: 定制化的 bean 定义文件, 通常通过覆盖相同 beanid 的文件，已有的 bean 定义
- ✓ infra.properties: 基础设施参数文件，里面定义了数据库连接和 redis 连接

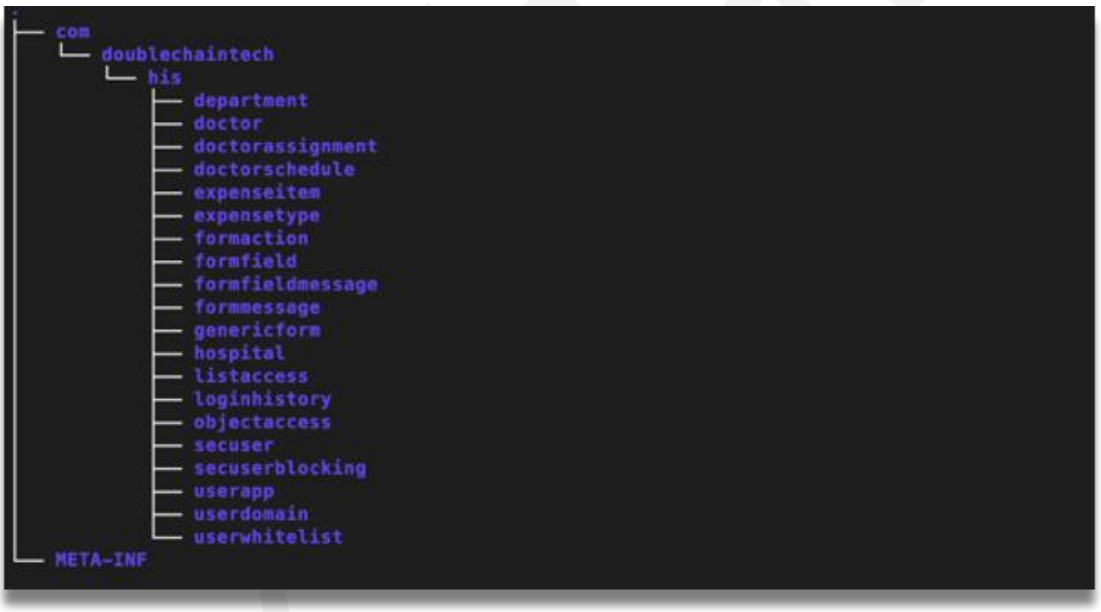
等参数

- ✓ online-system.xml: xml bean 定义系统的入口文件，可以增加其他定义文件来扩展系统功能

注意：在 his_custom.xml 中，系统自动推断某些 manager 具有极大可能重写，这些类无需在 his_custom.xml 定义。

3.4.3 JAVA 源代码目录介绍

这是 core 的源代码组织，custom 按照同样目录进行源代码组织



每个对象都对应了一个源代码包，在 his 下面，有很多基础类，这些类大大减少了源代码的行数。

以下是源代码文件定义，后缀统一为类型，前面为对象类型，里面涉及 DAO 和 Manager 接口定义等。

文件	描述
CandidateDoctor.java	产品一个选择列表的对象
DoctorDAO.java	DAO的接口定义，可以实现该接口支持新的数据类型
DoctorJDBCTemplateDAO.java	JDBC Template的实现
DoctorManagerException.java	从Manager里面抛出的异常定义
DoctorManagerImpl.java	默认的Manager实现，里面包含了很多可能用到的方法
DoctorManager.java	Manager接口定义
DoctorMapper.java	JDBC Mapper
DoctorNotFoundException.java	找不到的时候抛出的异常
DoctorSerializer.java	JSON的序列化类，可以重写改方法
DoctorTable.java	JDBC Template的Table实现
DoctorTokens.java	搜索，排序，分析，分页，保存令牌
DoctorVersionChangedException.java	当对象发生版本变更时候抛出的异常

开发人员需要在一般情况下已经集成已经存在的类，这样能够最大量的减少源代码的编写和调试工作。

3.5 过程

在一个实际项目开发中，可能按照顺序考虑如下情况

✓ 需要在不同 Manager 里面增加方法

✓ 定义新的 POJO 对象，以支持接口

✓ 在 DAO 中增加方法来支持新的查询

根据以往经验，应该尽量按照以上顺序来获取解决方案。

4. 中台前端界面定制

4.1 概述

SYSX 因为以下三点使得二次开发变得容易，也实现了自动生成的代码和手写代码隔离

✓ 吸收了大型可定制产品设计思想

- ✓ 经过若干个迭代调整
- ✓ 数个实际上线的项目中调整

4.2 工具

前端开发使用到如下工具：

- ✓ 基础环境 Nodejs
- ✓ 开发 IDE 环境 VS Code
- ✓ 编译工具 yarn 1.13+

4.3 前端

在浏览器总运行
生成的静态文件部署到服务器上

4.4 逻辑架构

前端代码在源代码目录的 bizui 中，下图是软件系统的逻辑架构，有如下特性：

- ✓ 后端代码的位置在 bizui/src/里面
- ✓ 层和层之前使用目录进行了完全的隔离在定制的时候
- ✓ bizcomponent 里面的代码只是作为参考，因为每次模型变更后生成代码都会被覆盖
- ✓ customcomp 是手写代码的位置，用于注册，重写已经存在的组件或者增加新的组件

通过这个架构，开发人员无需修改生成的代码，就可以：

- ✓ 定制表格
- ✓ 定制弹出框
- ✓ 定制搜索表单
- ✓ 定制更新界面
- ✓ 定制新增界面

```
├── axios
├── bizcomponents(自动生成的代码，可以查看，但不能修改)
|     ├── department（每个概念对应一个目录）
|     ├── doctor
|     ├── doctorassignment
|     ├── doctorschedule
|     ├── expenseitem
|     ├── expensetype
|     ├── hospital
|     ├── listaccess
|     ├── loginhistory
|     ├── objectaccess
|     ├── platform
|     ├── profile
|     ├── registration
|     ├── secuser
|     ├── secuserblocking
|     ├── userapp
|     ├── userdomain
|     └── userwhitelist
├── common（基础函数）
├── components（通用组件，大部分为 Ant Design Pro 组件，少部分是双链提供）
├── custcomponents（定制组件的位置)
|     ├── customindex.js(注册定制组件）
|     ├── customrouter.js（定会 router）
```

4.5 项目目录介绍：

以下是项目目录以及说明

如果要定制某个组件，可以在 _features=“custom”，则可以自动生成要定制的组件的代码，可以在其上进行修改。

生成的文件说明

文件描述 Doctor.app.jsApp 文件，用于组织基础的界面

Doctor.associateform.js 关联的弹出界面 Doctor.base.js 该包共享的数据

Doctor.createform.js 创建该对象用的表单 Doctor.dashboard.js 仪表板

Doctor.modaltable.js 弹出的表格，目前用于确认删除的对话框

Doctor.model.jsdvajs 风格的模型 Doctor.preference.js 设置界面

Doctor.search.js 搜索界面，包含表单和结果表格 Doctor.searchform.js 搜索表单 Doctor.table.js 显示结果的表格 Doctor.updateform.js 更新表单

文件	描述
Doctor.app.js	App文件，用于组织基础的界面
Doctor.associateform.js	关联的弹出界面
Doctor.base.js	该包共享的数据
Doctor.createform.js	创建该对象用的表单
Doctor.dashboard.js	仪表板
Doctor.modaltable.js	弹出的表格，目前用于确认删除的对话框
Doctor.model.js	dvajs风格的模型
Doctor.preference.js	设置界面
Doctor.search.js	搜索界面，包含表单和结果表格
Doctor.searchform.js	搜索表单
Doctor.table.js	显示结果的表格
Doctor.updateform.js	更新表单

每个界面元素还有一个 less 文件，这个文件可以直接修改用于更改生成数据的规格。

完成后的系统 https://demo.doublechaintech.com/admin/his/

管理员 用户名/密码：13900000001/DoubleChain!y1

医生 用户名/密码：13900000003/DoubleChain!y1