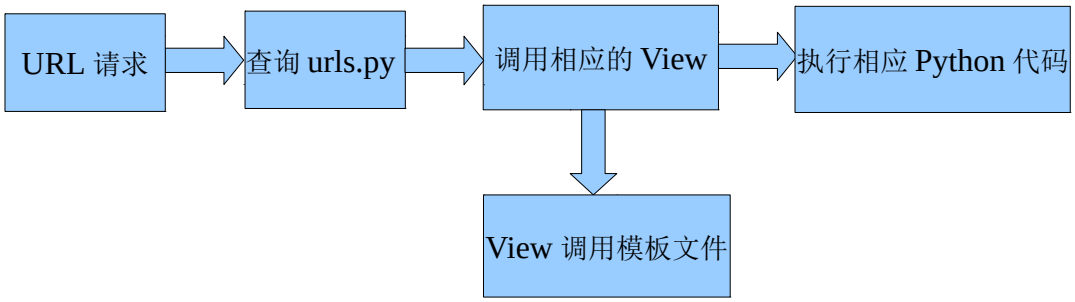
# 基于 Web 的故障诊断系统

| 编写 | 王宁远 | 编写时间 | 2013.4.22 |
|---|---|---|---|
| 审批 | | 审批时间 | |
| 文档版本 | V0.01 | | |

## 1.系统概述

### 1.1 系统结构概述

该故障诊断系统基于 Web 技术，采用 Django 框架进行设计。一般情形下的系统运行会按照如下流程进行：




具体而言，当浏览器给出用户请求的 URL 时，系统查询 urls.py，urls.py 将正则表达式同 view(视图)进行匹配，当 url 匹配 urls.py 中的某个正则表达式，系统即调用一个 view。view 在 Django 中以一个函数的形式存在。通常而言，系统要求 view 返回一个具体的 html 页面，直接在 view 函数中组织 html 代码并不是方便的选择，所以通常选择使用模板系统。

模板是一个 html 文件，可以向其中嵌入 python 代码，也可以由 view 传递参数进去。也就是说，view 向模板文件传入参数和嵌入代码，当参数被实例化，以及 python 代码被执行后。模板成为了一个真正的 html 文件，作为参数被 view 返回，显示在用户面前。

此外，在 Django 的框架中，系统的运作是以 App 为单位进行的，每个 App 有着自己独立的 views.py,urls.py,models.py.他们被组织在一起，以完成特定的功能为目的。

### 1.2 系统功能设计

该故障诊断系统目前实现的功能如下：

(1).可以直接查询，或者进行模糊搜索故障模式。
(2).系统会找出指定故障模式的子模式，通过数据库中的特定属性判断是否需要进行人工检测，并给出与检测相关的信息。
(3).提供了调用外部测试函数的接口。

## 2.前端与 UI 设计

### 2.1 URL 设计

如前文所述，URL 设计即是使特定的 URL 匹配特定的正则表达式，并由预留的映射关系调用相应的 View 函数。

在本系统中，存在两个 urls.py 文件，其中一个位于 project 根目录下，另一个存在于名为 fault_data 的 app 的目录下。

他们的内容被设计如下：

1.
/mysite/urls.py
```
    url(r'^fault_data/', include('fault_data.urls')),
    url(r'^$', homepage_view),
    url(r'^test/', test_view),
    url(r'^media/(?P<path>.*)$', 'django.views.static.serve',{'document_root':
settings.MEDIA_ROOT}),
```

2./fault_data/urls.py
```
    (r'^databrowse/(.*)', databrowse.site.root),
    url(r'^$', views.databrose, name='databrose'),
    url(r'^search-form/$', views.search_form),
    url(r'^search/$', views.search),
    url(r'^(?P<mode_id>\d+)/$', views.detail, name='detail'),
    url(r'^diagnose/$', views.diagnose, name='diagnose'),
    url(r'^diagnose/(?P<mode_id>\d+)/$', views.dgdetail, name='dgdetail'),
    url(r'^dfunction/(?P<function_id>\w+)/$', views.dfunction,
name='function_id'),
    url(r'^manage/$', views.datamanage),
```

由此可见，url 后的括号内，前半部分为正则表达式，后半部分为 view 的具体路径。

由前文可见，任何一个 url 被解析后都指向具体的 view，这为调用图片，视频等非 html 的文件造成了困难。Django 提供了一种功能，可以指定一个特殊的路径，通常称之为/media，可以让它指向项目目录中具体的文件，而不是 view 函数。在本项目中，实现这个功能的语句是这一句：

```
url(r'^media/(?P<path>.*)$', 'django.views.static.serve',{'document_root':
settings.MEDIA_ROOT}),
```

可见/media 目录被指向一个一个称为 MEDIA_ROOT 的常量，这个常量在 settings.py 中定义，定义如下：

```
    MEDIA_ROOT = '/home/wny/fd_sys/mysite/media'
```

这里要求用绝对路径给出，而项目目录即为'/home/wny/fd_sys/mysite/',所以从相对路径来看，以/media 开始的 URL 请求，被指向了项目目录下的/media 目录及其中的文件。

## 2.2 模板设计

模板是一些包含参数与嵌入 python 代码的 html 文件，我们可以利用 Django 提供的模板继承功能实现一系列的模板。

我们的模板文件，都存于项目目录或者 app 目录下的/templates 目录。

其中最根本的模板文件是/templates/frame.html

其中关键的语句如下：

```
    {% block content %}{% endblock %}
```

这是一个为继承预留的 block，凡是继承 frame.html 的任何模板文件，都需要用具体的 html 代码实现，代替这一 block。

```
    <link rel = "stylesheet" type = "text/css" href = "/media/css/main.css">
```

这一句是为了调用指定的 css 文件，在 css 文件中规定了 html 中的各种样式，例如分块大小，字体样式，背景颜色等信息。此外由于该项目使用了第三方提供的纯 CSS 图标(按钮)，所以在 man.css 文件中也包含了 CSS 图标的原始定义。

在project中还有以下的模板文件：

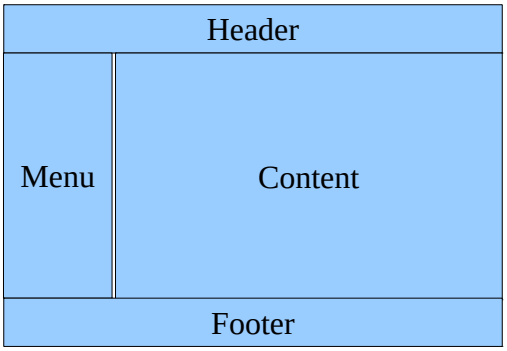
/templates/homepage.html
/templates/test.html
/fault_data/templates/details/details.html
/fault_data/templates/details/dgdetails.html
/fault_data/templates/diagnose/diagnose.html
/fault_data/templates/diagnose/dfunction.html
/fault_data/templates/diagnose/search_list.html
/fault_data/templates/manage/data_manage.html
/fault_data/templates/modes/databrose.html

这其中的大多数模板文件都继承了frame.html.

## 2.3 UI与CSS设计

关于UI，该系统通过把整个页面分为4个div。分别是 header,menu.content,footer。我们还在main.css中对这四个区域定义了相关的样式。

其中header是最顶上的标题区域，在这里是”Web Based Fault Diagnose System”，menu是左边的导航栏，通过左边的一列按钮可以连接到指定的功能页面。content是页面右端的待填充具体内容的区域，footer是最下面的类似于页脚的区域，这里暂时只写上了”University of Science and Technology of China”。实际的布局可以参照以下图示。




利用CSS可以简化对于html的设计。同时CSS提供的按钮或图标可以避免图片的堆积，提高了页面的调用效率。

(内容待补充)

# 3.后端设计

### 3.1 View(视图)设计

视图是Django框架中最为核心的部分，该系统的View目前分为如下几个部分：

在/fault_data/views.py中定义了以下的view：

```
def databrose(request):
def detail(request,mode_id):
def dgdetail(request,mode_id):
def search_form(request):
def search(request):
def diagnose(request):
def datamanage(request):
def dfunction(request,function_id):
```

databrose提供了对故障模式数据的浏览与检索功能。

detail 通常被 databrose 调用，用以显示特定故障模式的具体信息。

diagnose 和 databrose 类似，不过它调用的是 dgdetail 而不是 detail，这里除了可以显示特定故障模式的具体信息，还可以显示子故障模式列表，以及检测手段等相关信息。

dfunction 是预留的用来调用外部检测代码(函数)的视图。

search 一般用来显示搜索的结构

search_form 用来传递搜索的参数

data_manage 用来提供给用户管理数据库的界面。

## 3.2 数据库设计

Django 系统采用的是采用 MVC 设计模式。 M：Model V；View C：Controller。数据库中的每一个 Table 对应着一个 models.py 中的一个 Class。一般情况下并不用具体的 SQL 代码来操作数据库中的数据，而是采用经过 Python 封装后的方法进行操作。

按照下面的数据库定义：(来自文档《FAST 智能诊断》)

### 故障模式表

表名称：FaultMode

表描述：该表存储故障模式的相关信息

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| FaultModeID | CHAR(15) | 故障原因 ID：故障模式的 ID 号 |
| FaultMode | VARCHAR | 故障模式：故障模式或故障原因，即故障树中的节点名称 |
| FaultDescription | TEXT | 故障描述：对于故障模式的具体描述 |
| HighLevelFaultModeID | CHAR(15) | 高层故障模式 ID：当前故障模式的父节点的 ID |
| DetectionMethod | BOOL | 故障模式检测方式：手动方式或通过状态来检测（0 是自动检测，非零是手动检测） |
| ManualDetectionMethodID | CHAR(15) | 手动检测方法 ID |
| FunctionID | CHAR(15) | 相关函数 ID |
| Priority | ENUM | 优先级 |
| LogicalRelationship | BOOL | 逻辑关系：当前故障模式和其高层故障模式的关系（与、或） |

### 故障原因表

表名称：FaultCause

表描述：该表存储故障原因的相关信息

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| FaultCauseID | CHAR(15) | 故障原因 ID：故障原因的 ID 号 |
| FaultCause | VARCHAR | 故障原因：故障原因名称 |
| FaultCauseDescription | TEXT | 故障原因描述：对于故障原因的具体描述 |
| DetectionMethod | BOOL | 故障原因检测方式：手动方式或通过状态来检测（0 是自动检测，非零是手动检测） |
| ManualDetectionMethodID | CHAR(15) | 手动检测方法 ID |
| FunctionID | CHAR(15) | 相关函数 ID |
| Priority | ENUM | 优先级 |
| LogicalRelationship | BOOL | 逻辑关系：当前故障模式和其高层故障模式的关系（与、或） |

| | | |
|---|---|---|
| MaintenanceSuggestions | TEXT | 维修建议：故障的维修建议 |

**故障关系表**

表名称：FaultRelation
表描述：该表存储故障模式和故障原因的关系

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| FaultModeID | CHAR(15) | 故障模式 ID：故障模式的 ID 号 |
| FaultCauseID | CHAR(15) | 故障原因 ID：故障原因的 ID 号 |
| LogicalRelationship | BOOL | 逻辑关系：当前故障模式和其高层故障模式的关系（与、或） |

**手动检测方法表**

表名称：ManualDetection
表描述：该表存储手动检测方法的相关信息

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| ManualDetectionMethodID | CHAR(15) | 手动检测方法 ID |
| MethodDescription | TEXT | 手动检测方法描述 |

**自动检测方法表**

表名称：AutoDetection
表描述：该表存储故障模式和故障原因的关系

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| FunctionID | CHAR(15) | 自动检测函数 ID |
| FunctionName | VARCHAR | 自动检测函数名称 |
| LogicalExp | TEXT | 逻辑表达式 |

**函数参数表**

表名称：FunctionArg
表描述：该表存储自动检测函数的参数列表

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| FunctionID | CHAR(15) | 自动检测函数 ID |
| ArgID | CHAR(15) | 函数参数 ID |

**函数参数描述表**

表名称：FunctionArgDes
表描述：该表存储自动检测函数的参数的想换信息描述

| 字段名称 | 字段类型 | 字段描述 |
|---|---|---|
| ArgID | CHAR(15) | 函数参数 ID |
| ArgName | VARCHAR | 函数参数名称 |
| ArgDescription | TEXT | 函数参数描述 |

我们在/fault_data/models.py 中定义的 models 如下：

```
class FaultMode(models.Model):

    FaultModeID = models.CharField(max_length=15)
    FaultMode = models.CharField(max_length=200)
    FaultDescription= models.CharField(max_length=200)
    HighLevelFaultModeID = models.CharField(max_length=15)
    DetectionMethod = models.BooleanField()
    ManualDetectionMethodID = models.CharField(max_length=15)
    FunctionID = models.CharField(max_length=15)
    Priority = models.IntegerField()
    LogicalRelationship = models.BooleanField()

class FaultCause(models.Model):

    FaultCauseID = models.CharField(max_length=15)
    FaultCause = models.CharField(max_length=15)
    FaultCauseDescription= models.CharField(max_length=200)
    HighLevelFaultModeID = models.CharField(max_length=15)
    DetectionMethod = models.BooleanField()
    ManualDetectionMethodID = models.CharField(max_length=15)
    FunctionID = models.CharField(max_length=15)
    Priority = models.IntegerField()
    LogicalRelationship = models.BooleanField()
    MaintenanceSuggestions = models.CharField(max_length=300)

class FaultRelation(models.Model):

    FaultModeID = models.CharField(max_length=15)
    FaultCauseID = models.CharField(max_length=15)
    LogicalRelationship = models.BooleanField()

class ManualDetection(models.Model):

    ManualDetectionMethodID = models.CharField(max_length=15)
    MethodDescription= models.CharField(max_length=200)

class AutoDetection(models.Model):

    FunctionID = models.CharField(max_length=15)
    FunctionName = models.CharField(max_length=15)
    LogicalExp= models.CharField(max_length=200)

class FunctionArg(models.Model):

    FunctionID = models.CharField(max_length=15)
    ArgID = models.CharField(max_length=15)

class FunctionArgDes(models.Model):

    ArgID = models.CharField(max_length=15)
    ArgName = models.CharField(max_length=200)
    ArgDescription = models.CharField(max_length=200)
```