数据质控服务管理平台

**过程记录**

**四川久远银海软件股份有限公司**

**2021年04月**

**目录**

[1. 项目管理 1](#_Toc69311042)

[1.1 代码管理 1](#_Toc69311043)

[1.2 文档管理 1](#_Toc69311044)

[2. 客户端开发问题及经验总结 2](#_Toc69311045)

[2.1 准备工作 2](#_Toc69311046)

[2.2 客户端源码启动报错 4](#_Toc69311047)

[2.3 汉化DataCleaner 5](#_Toc69311048)

[2.4 启动后如何新建任务 6](#_Toc69311049)

[2.5 界面区域了解 7](#_Toc69311050)

[2.6 DataCleaner资料介绍 8](#_Toc69311051)

[2.7 源码分析之job区域菜单栏 9](#_Toc69311052)

[2.8 源码分析之开发转换器 10](#_Toc69311053)

[2.9 源码分析之转换器的渲染器 12](#_Toc69311054)

[2.10 源码分析之转换器的面板建设 13](#_Toc69311055)

[2.11 源码分析之转换器的属性小部件 15](#_Toc69311056)

[2.12 源码分析之转换器的整体流程 16](#_Toc69311057)

[2.13 源码分析之转换器的结果显示 17](#_Toc69311058)

[2.14 源码分析之开发分析器 18](#_Toc69311059)

[2.15 源码分析之InputRow 20](#_Toc69311060)

[3. web端开发问题及经验总结 21](#_Toc69311061)

[3.1 准备工作 21](#_Toc69311062)

[3.2 应用框架 23](#_Toc69311063)

[3.3 Windows启动不了项目问题 24](#_Toc69311064)

[3.4 在windows系统编译问题： 25](#_Toc69311065)

[3.5 项目发布问题 25](#_Toc69311066)

[3.6 页面后台拼接 26](#_Toc69311067)

[3.7 数据请求参数 26](#_Toc69311068)

[3.8 多个用户登录 27](#_Toc69311069)

[3.9 数据冒泡排序 27](#_Toc69311070)

[3.10 文件存储 27](#_Toc69311071)

[3.11 首页java.lang.NullPointerException问题查找 27](#_Toc69311072)

[3.12 时间戳的截取 28](#_Toc69311073)

[3.13 使用@requestParam注解接收时，是接收不到 29](#_Toc69311074)

[3.14 JDBC访问数据库问题 29](#_Toc69311075)

[3.15 List去重 29](#_Toc69311076)

[3.16 网页报403错误 30](#_Toc69311077)

[3.17 数值-数学公式报错。 30](#_Toc69311078)

[3.18 简单搜索/替换预览报错 30](#_Toc69311079)

[3.19 记录遇到的迫切需要解决的问题 31](#_Toc69311080)

[3.20 项目启动 整体界面布局及功能 35](#_Toc69311081)

[3.21 整体界面布局及功能 37](#_Toc69311082)

[3.22 整体界面布局及功能web端源码分析 41](#_Toc69311083)

[3.22.1 代码注释讲解 41](#_Toc69311084)

[3.22.2 登录代码检测 41](#_Toc69311085)

[3.22.3 前后端数据传输 43](#_Toc69311086)

[3.22.4 项目启动源码分析 44](#_Toc69311087)

[3.23 项目打包发布 45](#_Toc69311088)

[3. 开发问题web端 17](#_Toc17044)

[3.1 准备工作 17](#_Toc14698)

[3.2 应用框架 19](#_Toc14698)

[3.3 项目启动 20](#_Toc14698)

[3.4 整体界面布局及功能 22](#_Toc14698)

[3.5 web端源码分析 25](#_Toc14698)

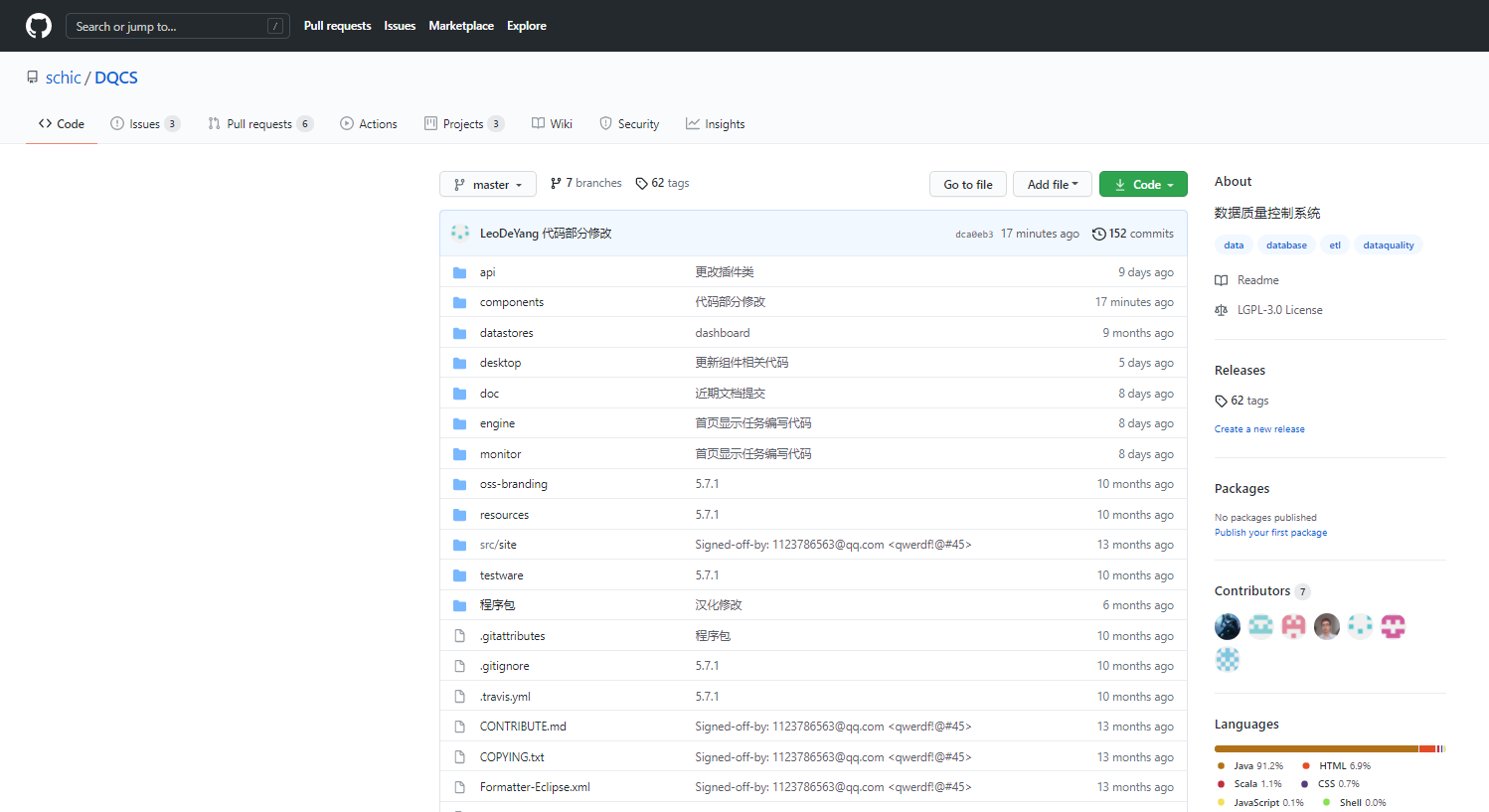
[3.6 项目打包发布 28](#_Toc14698)

# 

# 项目管理

## 代码管理

本项目应甲方要求可见可查共享协作的开发方式，将代码都集中上传至要求的github项目小组。如图：



如图所示，本项目可在github网站中搜索DQCS直接搜索到，方便大家一同探索该技术方向。项目在基于一流的数据质控解决方案DataCleaner基础上扩展打造。DataCleaner是一个帮助我们进行数据质量分析的开源工具，从数据完整性、数据规范性、数据一致性、数据准确性、数据唯一性、数据关联性多维度进行数据质控。

## 文档管理

本项目工程相关文档同代码一起上传至github工程中。在整个开发过程中，形成了DataCleaner功能拓展设计文档、过程文档、DataCleaner中文文档。

# 客户端开发问题及经验总结

## 准备工作

此项目使用的主要框架有springMVC，JWT，scala语言，xhtml，jsp等等。

首先JWT其实是三个英语单词JSON Web Token的缩写。token一般都是用来认证的，比如我们系统中常用的用户登录token可以用来认证该用户是否登录。JWT也是经常作为一种安全的token使用。JSON Web Token（JWT）是一个开放标准（RFC 7519），它定义了一种紧凑且独立的方式，可以在各方之间作为JSON对象安全地传输信息。此信息可以通过数字签名进行验证和信任。JWT可以使用秘密（使用HMAC算法）或使用RSA或ECDSA的公钥/私钥对进行签名。

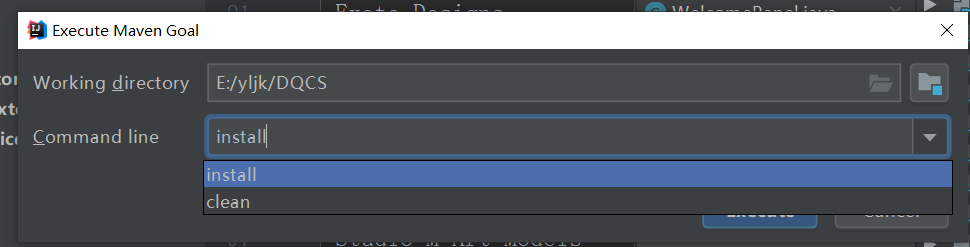
Scala是一门多范式的编程语言，一种类似java的编程语言，设计初衷是实现可伸缩的语言、并集成面向对象编程和函数式编程的各种特性。Scala 是一门多范式（multi-paradigm）的编程语言，设计初衷是要集成面向对象编程和函数式编程的各种特性。Scala 运行在 Java 虚拟机上，并兼容现有的 Java 程序。Scala 源代码被编译成 Java 字节码，所以它可以运行于 JVM 之上，并可以调用现有的 Java 类库。

springMVC通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText和POI。Spring MVC 框架并不知道使用的视图，所以不会强迫开发者只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、过滤器以及处理程序对象的角色，这种分离让它们更容易进行定制。

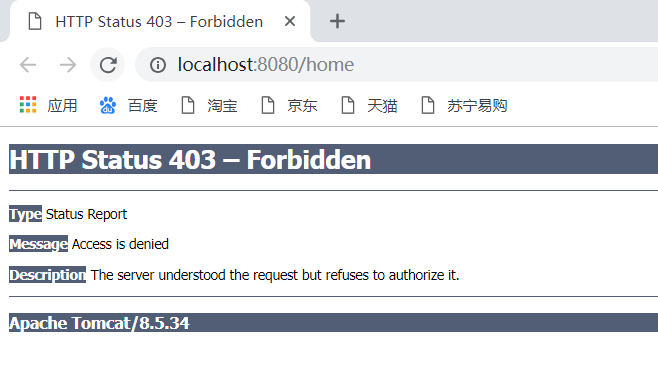
XHTML（Extensible HyperText Markup Language的缩写，意为可扩展超文本标记语言）表现方式与超文本标记语言（HTML）类似，不过语法上更加严格。从继承关系上讲，HTML是一种基于标准通用标记语言（SGML）的应用，是一种非常灵活的置标语言，而XHTML则基于可扩展标记语言（XML），XML是SGML的一个子集。XHTML 1.0在2000年1月26日成为W3C的推荐标准。

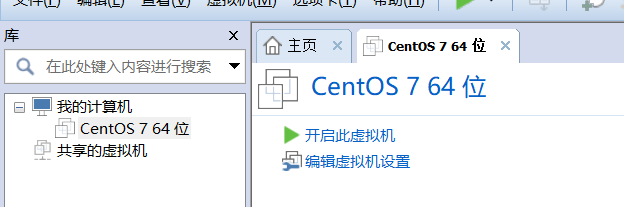
web端源码启动报错

jar包下载安装：github网站在idea工具上下载代码成功，但是install下载jar包的时候网络不通畅，导致下载jar包部分失败。



在windows系统编译问题：



由于在windows系统上，此项目编译头过长，系统启动之后403错误，它是由于xml文件拼接起来的启动头过长，windows不支持。解决方法：安装虚拟机。

我安装的是centOS7，虚拟机上安装idea开发软件，然后安装git项目控制软件，安装tomcat服务器，安装maven，并且配置所有相关的环境变量。导入DataCleaner项目代码之后下载jar包，运行。解决成功。

项目发布问题

解决方法：虚拟机上通过idea成功编译为war文件，使用FileZilla文件导出到本机，然后通过tomcat服务器完成项目发布。

## 客户端源码启动报错

DataCleaner是由Java语言编写的C/S架构软件，因此客户端运行需服务器或电脑必须安装好JDK环境。DataCleaner源码从github官网即可拉取，安装好JDK环境的情况下，直接从desktop目录下的Main类的main方法启动客户端即可。若有报错，检查依赖缺失或工程配置即可。

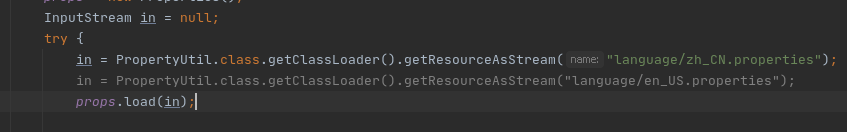
## 汉化DataCleaner

1、客户端汉化

客户端采用了.properties配置文件的重写，把英文改为中文重写一份配置文件。



在我们需要操作汉化版本的时候，只需要修改配置，中英文就可以任意切换，加载中文配置文件，如下：

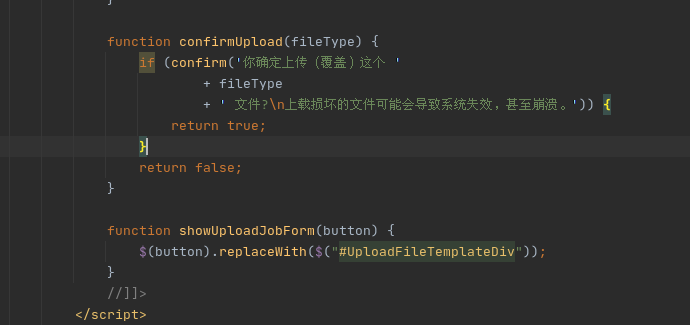


中英文配置文件如下：



1. Web端汉化

通过每个页面专属id找到对应前端代码做修改；通过idea的全局搜索做修改。Web端页面显示问题包括前端页面显示，以及后台提示语和后台代码拼接的前端页面。



文件上传页面就是一个xhtml文件，只需要将英文进行翻译成中文之后替换掉。

## 启动后如何新建任务

客户端成功启动之后，直接左下角新建任务，再新建数据库连接。新建数据库连接完成后，该数据库连接以后都会存储到记录中方便使用。由此即可进入界面进行数据操作。



## 界面区域了解

官方给出资料有部分介绍，更多的是需要自己熟练使用即可。整个工作界面中，左侧菜单栏中上面为数据库连接相关，下面为组件库，包含了转换器、分析器、数据写入相关等等。界面的正上方工具栏为基础工具。右上角执行按钮，在配置好整个流程作业后，即可点击执行按钮执行任务。中间区域为白色画布，画布下方会有操作提示。有效合理的提示用户如何组装组件以达成自定义数据质控目的。



## DataCleaner资料介绍

DataCleaner官网:

<https://datacleaner.org/>

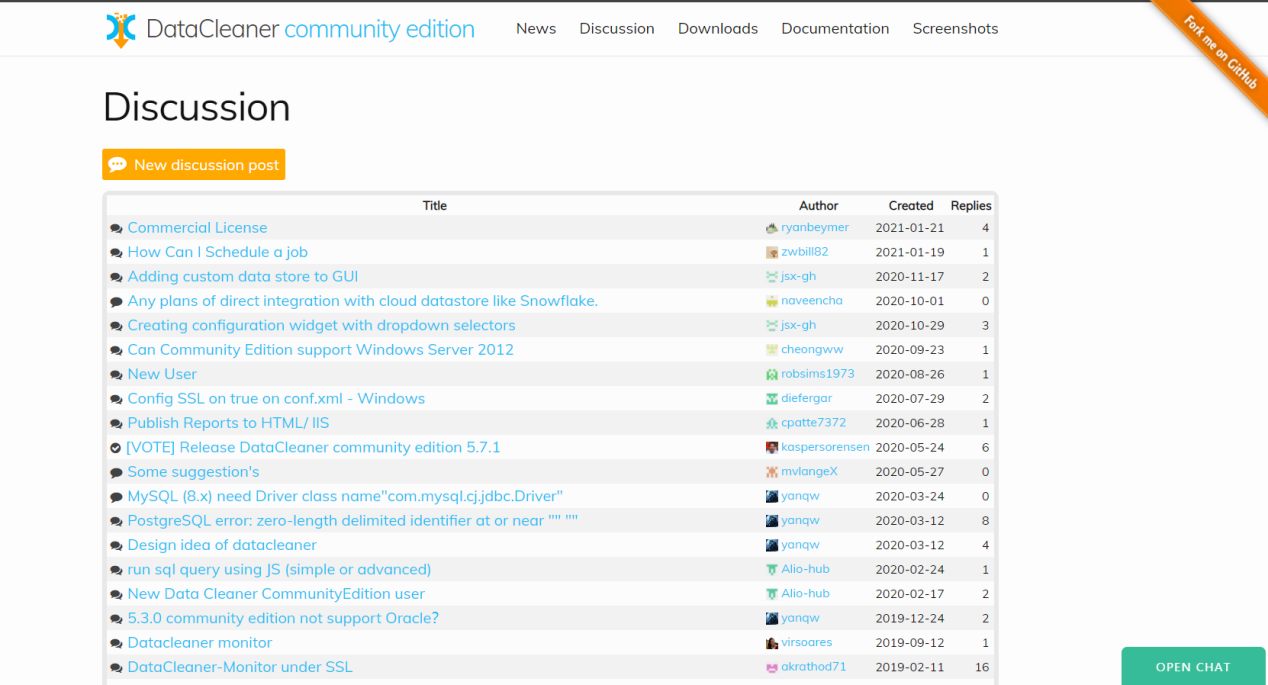
DataCleaner代码社区：

<https://github.com/datacleaner>

DataCleaner官网社区：

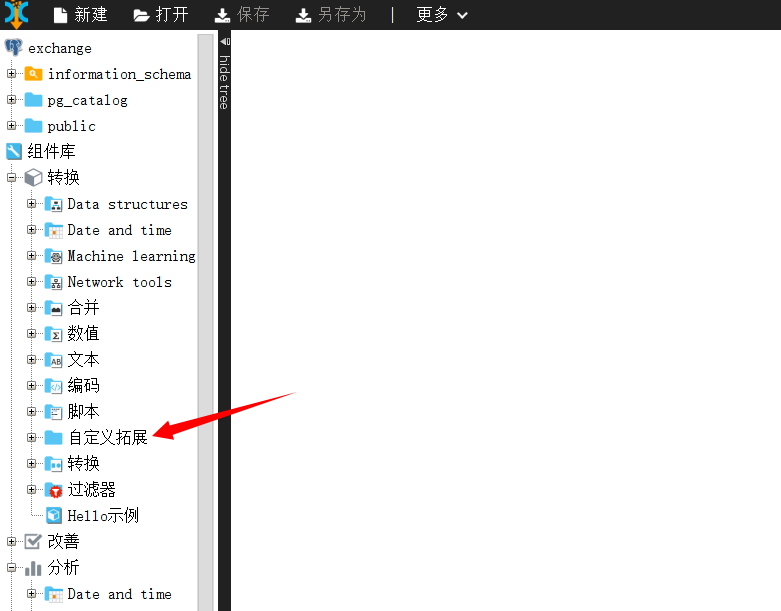
<https://datacleaner.github.io/discuss>

目前DataCleaner资料较少，除官网提供英文文档之外。并未查找到有场景使用或项目利用实例。因此只能从官网上和少有的博客中了解使用资料和使用情况。在本着甲方的开源社区精神下，鉴于目前为止并没有中文文档，因此将DataCleaner的官方文档直译出来，方便反复查阅和理解。

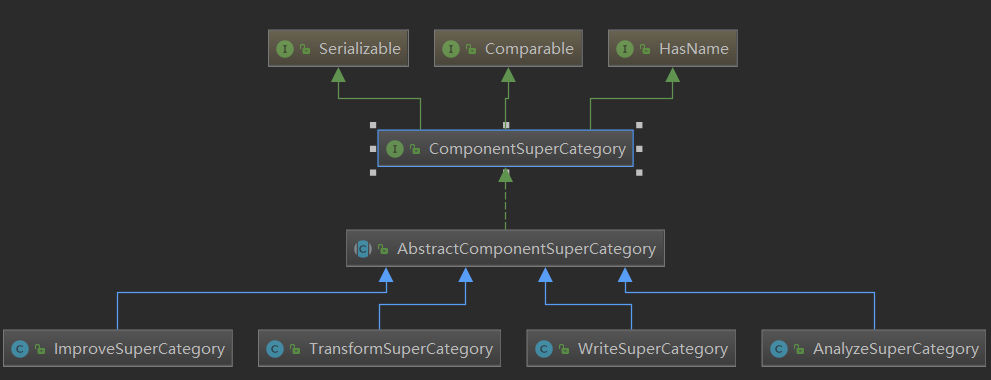


## 源码分析之job区域菜单栏

因为要对DataCleaner进行拓展，根据官网文档指导，可根据自身需求开发出自定义的转换器组件和分析器组件。第一步，便是如何建立自定义的组件目录。



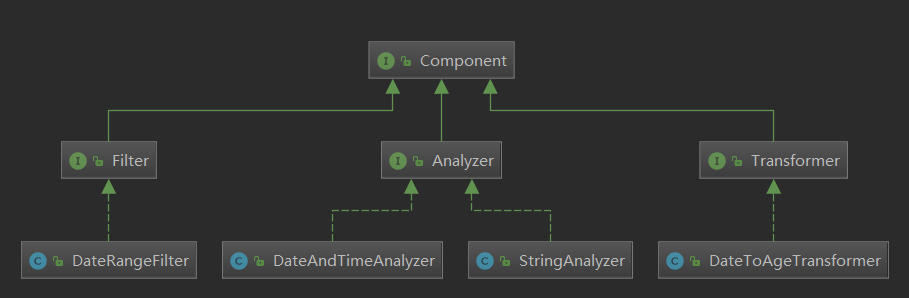
根据源码跟踪发现，整个目录类别的控制是由ComponentSuperCategory组件超级类别该接口控制。



因此，构建目录需要新建目录类继承AbstractComponentSuperCategory类，并通过getName()方法获取目录名字，getDescription()获取目录描述，getSortIndex()获取目录的索引位置。若没有给该目录取名字，则该目录的名字由类名依据驼峰命名规则拆解而来。最后组件只需要通过在组件的类级别打上@Categorized注解便可嵌套自己的父级目录和一级目录，也就是将自己放在什么目录内。

## 源码分析之开发转换器

转换器(Transformer)，顾名思义。将部分特定数据或整体数据据按照某种规则进行过滤提取转换，转换之后可二次利用数据或展示数据。根据源码追踪发现，所有的组件都起源于Component接口。而后继承出过滤器接口、分析器接口、转换器接口等。最后自定义组件再去实现对应的接口。



根据官方文档，以及跟踪源码可发现。自定义开发出一个转换器必须满足：

* 必须实现Transformer接口。
* 类上面必须有@Named注解(组件展示的名字)、@Description注解(组件的描述)、@Categorized注解(组件的分类目录)。
* 通过@Configured注解注入一个 InputColumn<E> 实例(或者一个数组)，其中<E>是传入字段的数据类型。

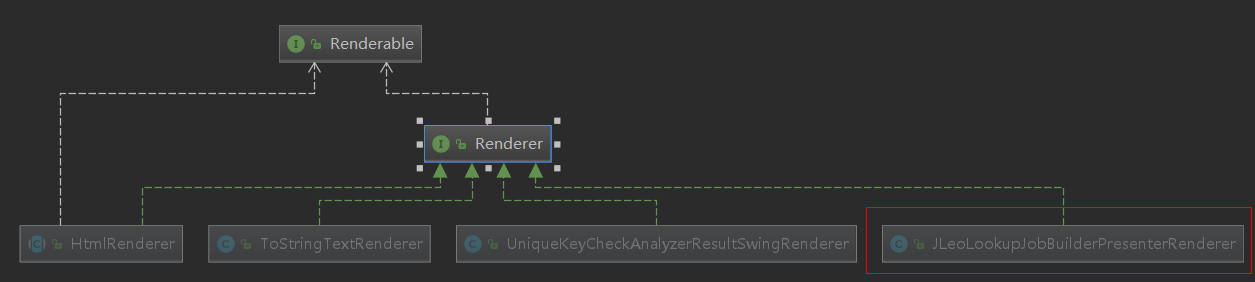


这里可以看到，必须实现两种方法：

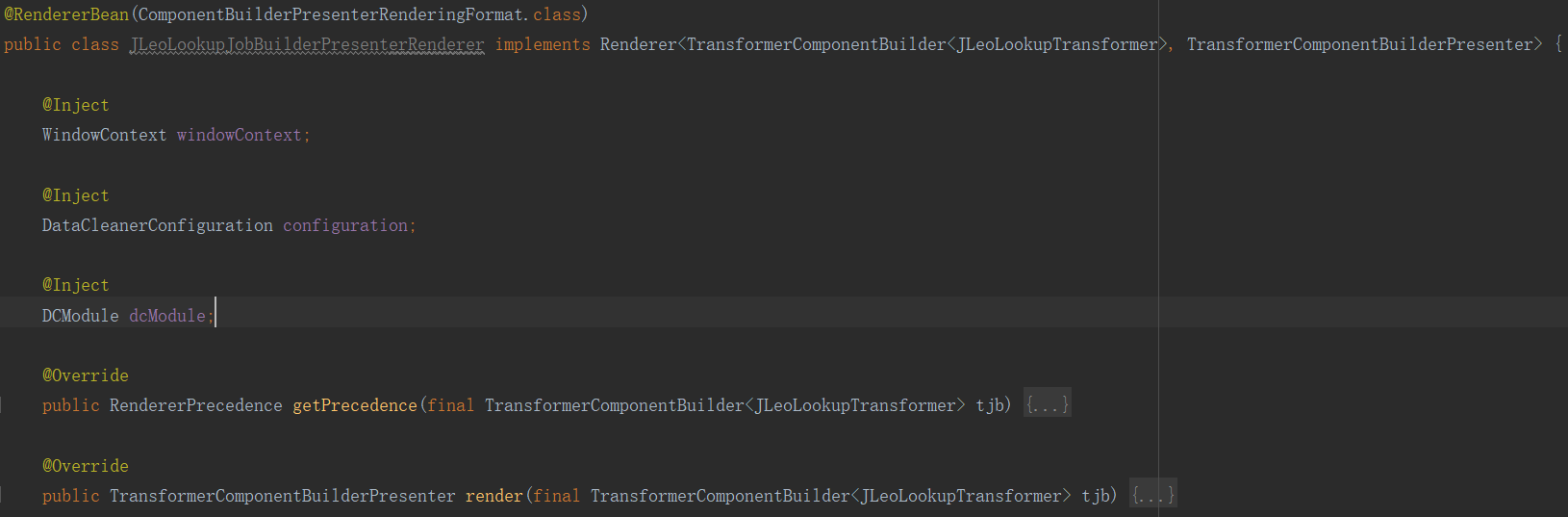
* **getOutputColumns():** 框架调用这个方法生成输出列，这个输出列定义在**OutputColumns的属性中，**最终值显示在执行结果中
* **transform(InputRow):该方法是组件配置的表格数据，该表格数据一行一行的进入该方法，因此该方法会被框架循环调用N次。**

## 源码分析之转换器的渲染器

根据源码跟踪发现，在DataCleaner的组件分类中，每一种组件都有不同的写法。转换器、过滤器、分析器等等，都有各自的实现或继承。而同类型的组件又有不同的渲染器和自定义的小部件以及功能实现。要将组件渲染出来可见，就必须要实现自定义的渲染器。



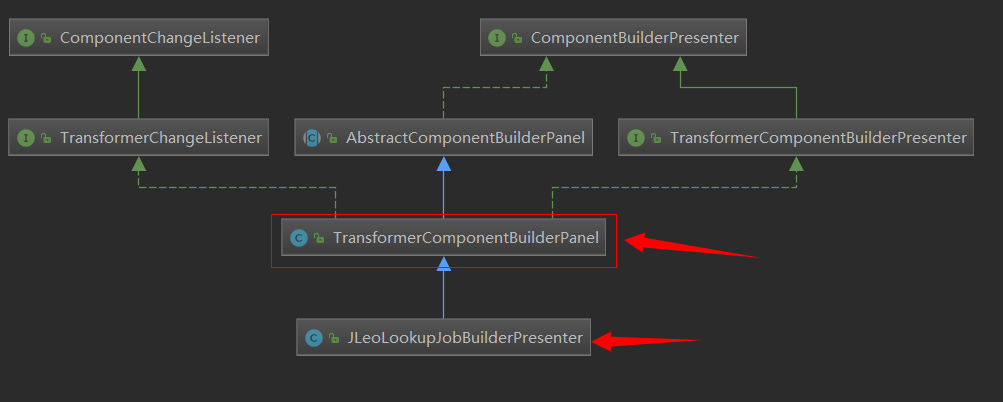
所有的渲染器都要实现Render接口。



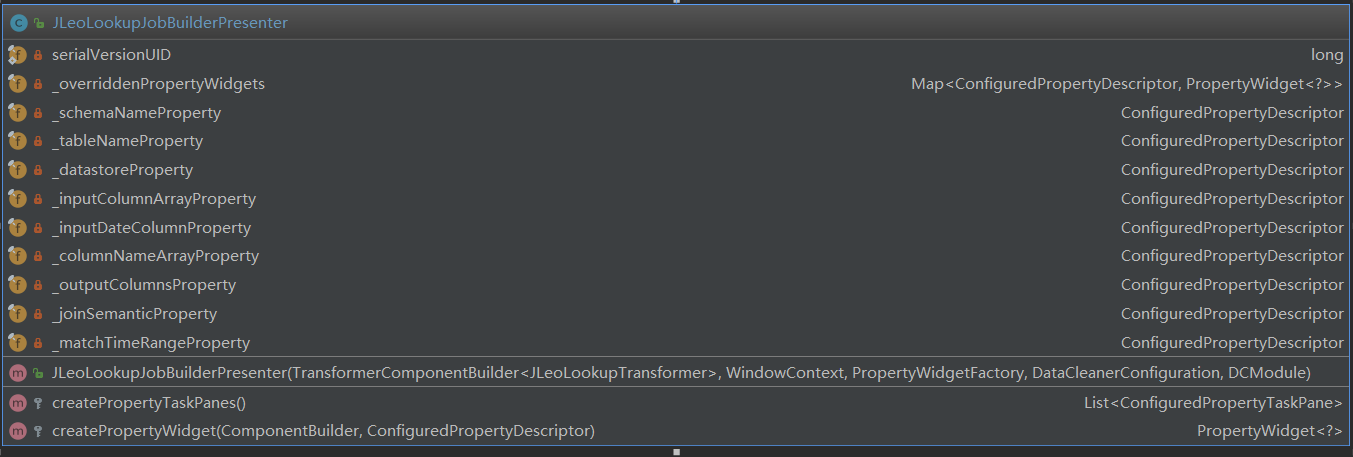
自定义的渲染器实现Render接口后，必须要覆写两个方法。getPrecedence(I renderable)方法，是为了检查组件优先级，检查渲染对象是否可被渲染。render(I renderable)方法，是为了呈现出渲染结果，如整个组件的样子。这里就设计到自定义的BuilderPresenter(面板构造器)。

## 源码分析之转换器的面板建设

这里的面板只是指转换器组件的面板，因为整个DataCleaner的面板各种各样。为了构造出我们自定义的面板，所以也是需要新建类构建。



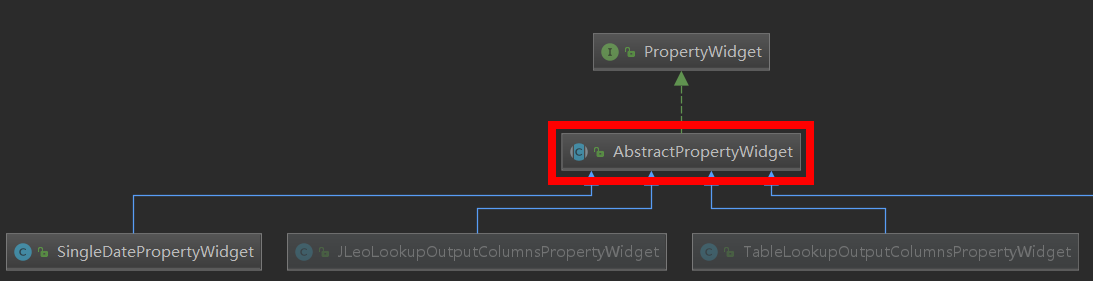
新建的类以Presenter单词结尾，继承TransformerComponentBuilderPanel类，然后以此构建自己的整个面板。同样开发Transformer，并非一定是继承的TransformerComponentBuilderPanel，也可能是其他的面板构造器。多种多样，需根据API和自身需求去决定。



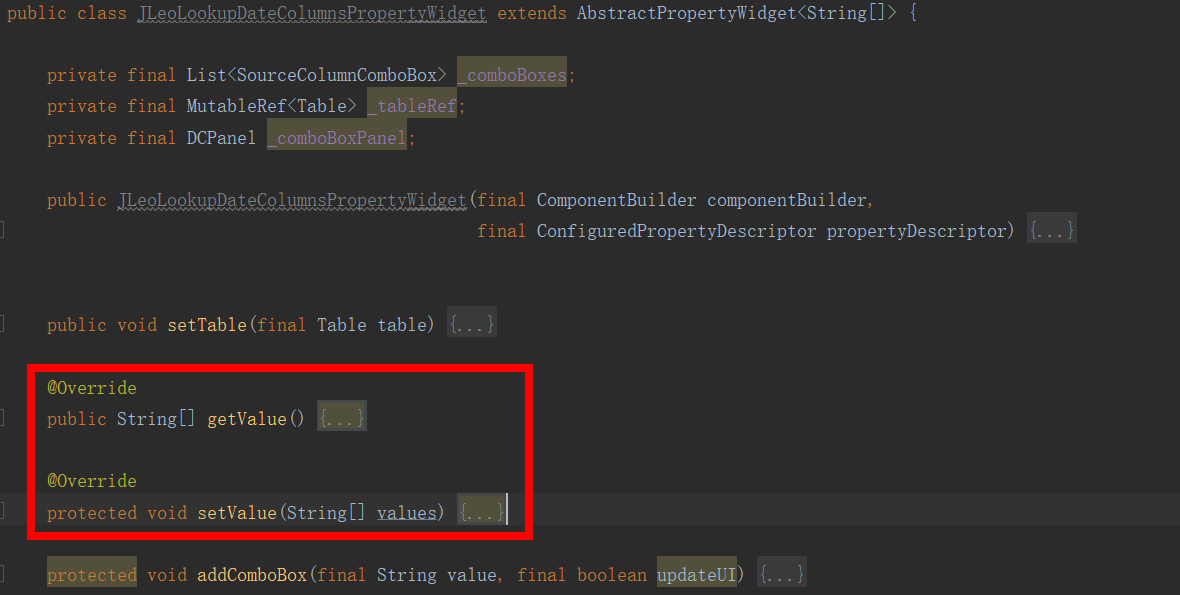
整个自定义的面板建设BuilderPresenter相对还是简单。定义了一些属性，这些属性即是自定义的Transformer中的属性，这些属性同时也对应着一些小部件。一个有参数的构造方法，为了将小部件的值传递给自定义的Transformer中去。剩下就是创建整个属性面板的方法和创建属性小部件的方法。这里的属性小部件指\_overriddenPropertyWidgets，该属性小部件是一个Map类型，其中装着其他的多个属性小部件，如输入列的复选框、输出列的表格、数据库的下拉框、表名的下拉框等等。也就是一个大的属性控件生成在一个面板上，而该属性控件中还嵌套着各种各样的小部件，这些小部件获取到值才可以传递给Transformer中对应的属性。

## 源码分析之转换器的属性小部件

针对不同的组件属性和开发需求，可能需要在组件的配置面板上加上一些特殊的小部件。如输入列属性、输出列属性、时间字段属性、而这些小部件各式各样。小部件的类名大多以Widget单词结尾。



在这样的自定义xxxWidget类，也就是小部件中。需要继承AbstractPropertyWidget，实现其中的setValue和getValue方法。当然，实际需求中要更加的复杂多变。在该xxxWidget类中，根据自身需求去构造复选框、单选框、下拉框、时间插件、事件等等。这需要对源码非常的熟悉和对DataCleanerAPI非常的了解。这也是组件开发中难点之一。



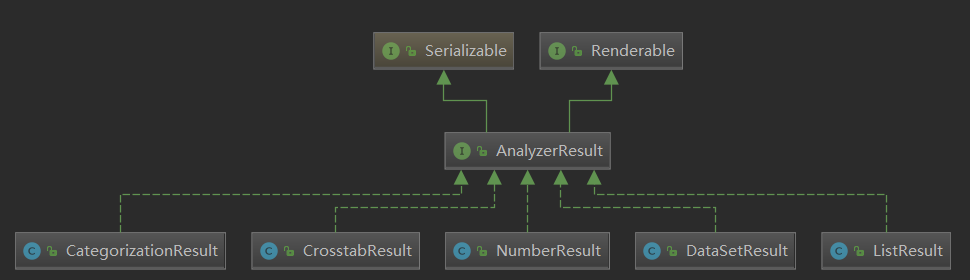
## 源码分析之转换器的整体流程

首先呢是先建自定义的转换器类以xxxTransformer结尾，实现其中的getOutputColumns方法和transform方法。然后是建立该转换器对应的xxxRenderer渲染器，渲染器中因为返回的是面板建设。因此，接着创建对应的xxxBuilderPresenter，将转换器中的属性绑定到面板建设中。最终再创建对应着各个属性的xxxWidgets小部件。监听事件大部分由框架完成，也可自定义更改。以上，只是转换器运转的一个简短流程，实际上要复杂得多。组件的配置面板任何一个小部件的值被更改时，都会传递到组件中去，保证最后的正常执行。

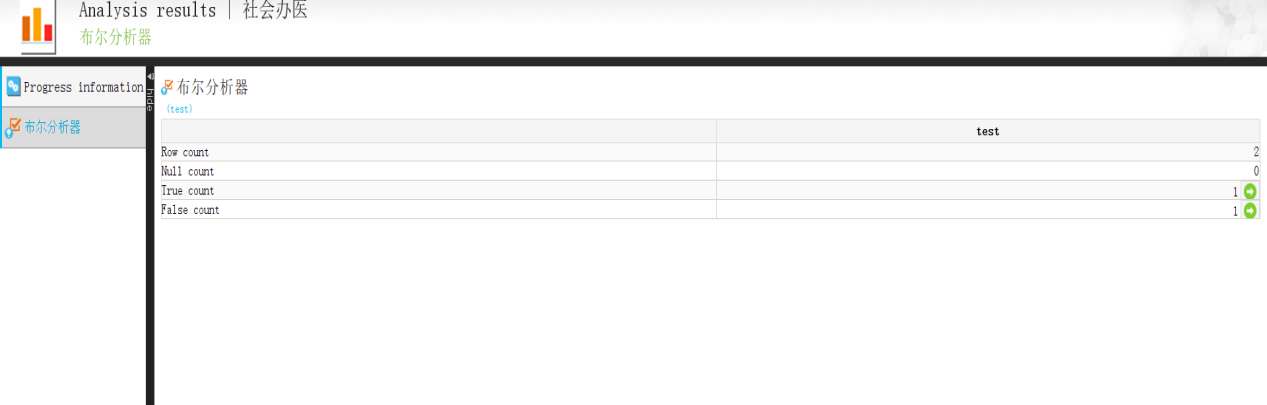


## 源码分析之转换器的结果显示

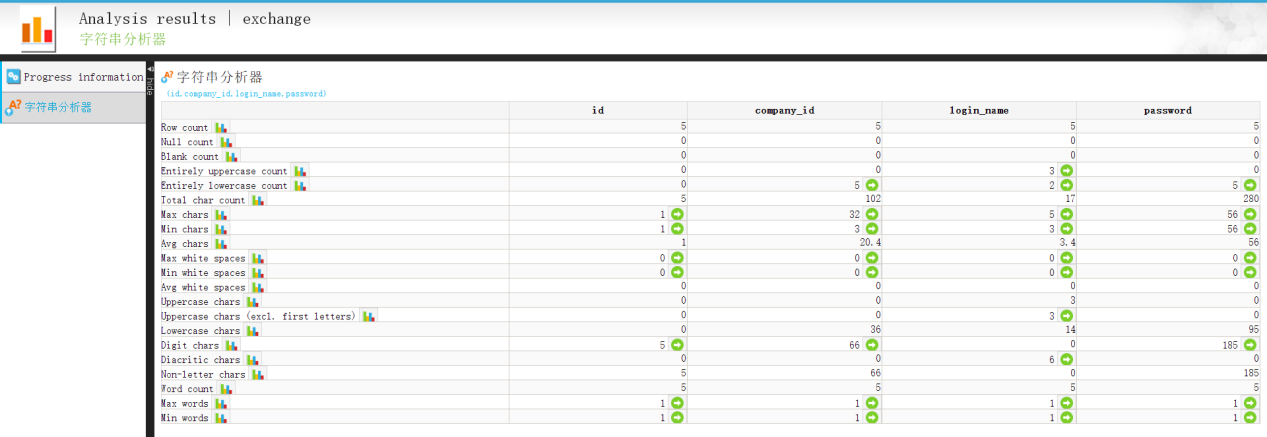
在DataCleaner中有各种各样的结果显示。目的在于不同组件执行后的效果不同。而不同的结果都需要实现AnalyzerResult接口，可选取DataCleaner中已经封装好的结果类型来表示组件执行结果，也可自定义结果类型来表示组件执行结果。如果没有在组建中规定好自己的组件执行结果类型，那么该组件最后只能配合分析组件或写入组件来执行呈现出结果。换而言之，组件自身规定好执行结果类型的，也就是可以单独配置好后直接执行。



布尔分析器的结果类型:BooleanAnalyzerResult



字符串分析器的结果类型:StringAnalyzerResult

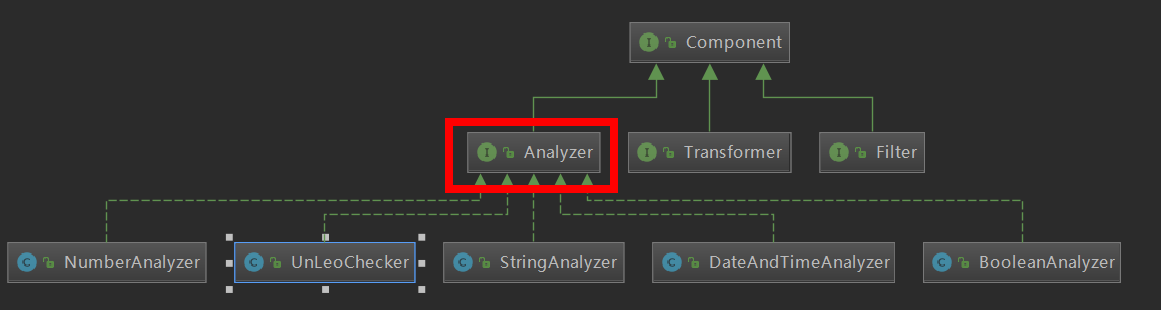


除此之外，还有很多不同的结果类型，如CategorizationResult、ListResult、

CharacterSetDistributionResult、DateAndTimeAnalyzerResult等等。不同的结果类型都是为了处理不同的组件执行结果。因此开发自定义组件可选取已有的结果类型或者新建结果类型。

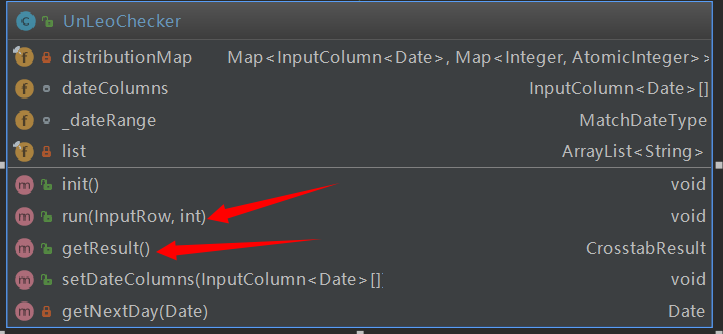
## 源码分析之开发分析器

分析器(Analyzer)，指针对数据进行质量分析或性能分析以便对数据有更深的了解。如，数字分析器针对数字字段的统计分析，字符分析器针对字符串字段的统计分析等等。



根据官方文档介绍，以及跟踪源码发现。开发分析器必须满足：

* 实现Analyzer<R>接口，其中泛型R则是定义该分析器的结果类型。
* 该类需要用@AnalyzerBean注解。此注解接受一个参数：分析器的显示名称。
* 需要使用@Configured注释注入一个或多个InputColumn<E>，以便使用传入的数据。<E>参数定义感兴趣的数据类型。



新建分析器类之后，必须要实现两个方法：

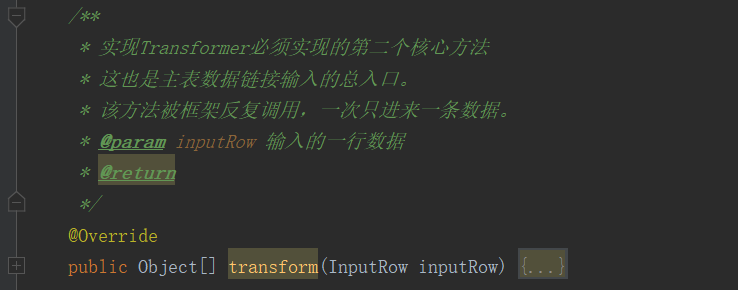
* run(InputRow row, int distinctCount)方法，对单行数据执行分析。同转换器一样，该数据是组件配置表格时的表格数据，一行一行进入的该方法。
* getResult()方法，获取分析器执行的分析的结果。该方法的返回类型也就是定义类时，泛型中规定好的执行结果类型。

同样的道理。若需要在配置面板上添加特定的小部件，仍需自己新建类新增加小部件。和转换器不同的是，分析器的结果类型还可能需要对应定义出一个结果类型简化器，结果类型以Result结尾的类，结果类型简化器以ResultReducer结尾的类。

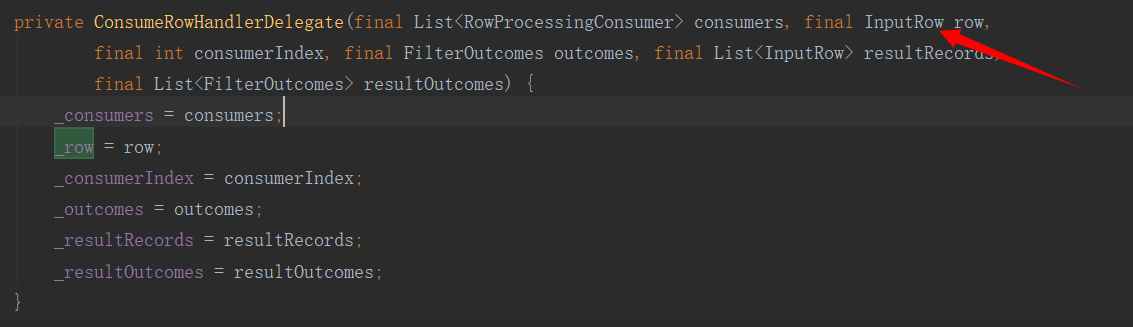
## 源码分析之InputRow

无论是开发分析器还是转换器，其中必须要实现的方法run(InputRow row, int distinctCount)方法、**transform(InputRow** row**)方法**都包含了同样一个参数InputRow。该参数表示连接组件配置的表格的一行数据。因此包含此参数的方法都会被循环调用，循环调用次数看数据量大小。那么，该InputRow从何而来？经过源码跟踪并未发现找到最初来源。

这是transform具体实现，其中包含InputRow参数。



这是分析器和转换器追踪InputRow殊途同归的一个类。



这是最终追踪到的RowProcessingFunction类中的call方法。行数据的源头。



至此，全局没有调用call方法的地方，不知道Iterator<InputRow>从何而来，也并未查找到InputRow从何而来。

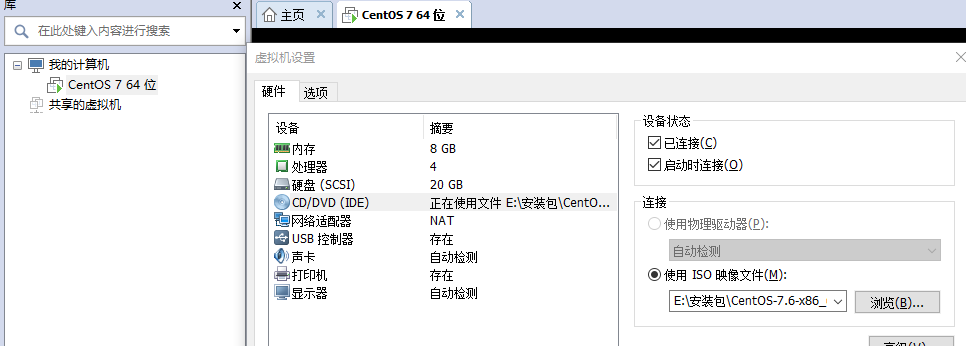
# web端开发问题及经验总结

## 准备工作

此项目是个开源项目，代码发布在github官网（https://github.com/）上面，开发之前需从github官网下载代码。

此项目在windows系统开发，编译会报错，原因是因为windows命令长度有限，编译报错，所以需要在linux系统下进行开发。

准备工作1：安装虚拟机VMware，使用CentOS 7 64位作为镜像新建虚拟机。

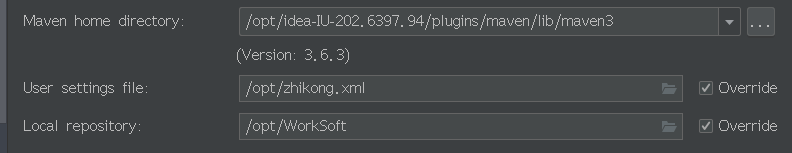
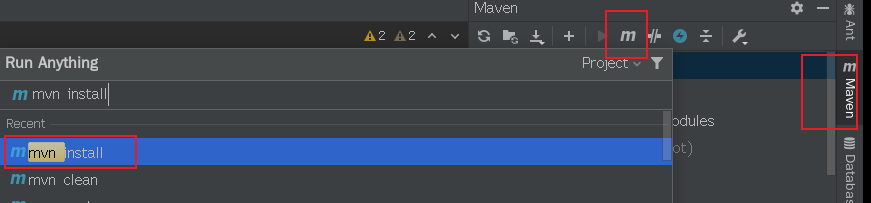
准备工作2：虚拟机下安装tomcat，idea开发工具，java（1.8.0），git，maven等开发环境



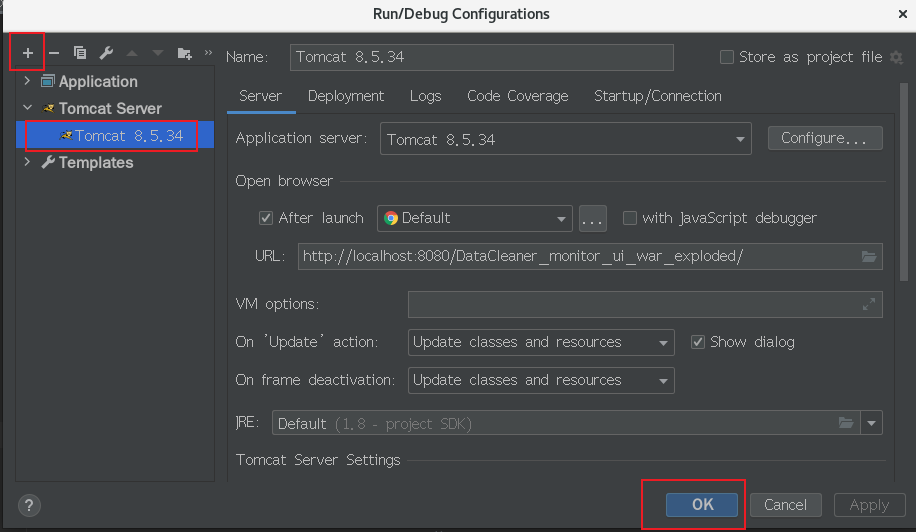
准备工作3：

代码下载，使用git命令git clone git clone <https://github.com/schic/DQCS.git>下载到指定目录；

Idea编译，通过idea打开下载好的代码，修改maven配置，使用maven的install下载项目jar包

添加Tomcat，在idea中添加tomcat启动，准备工作完成。



## 应用框架

此项目使用的主要框架有springMVC，JWT，scala语言，xhtml，jsp等等。

首先JWT其实是三个英语单词JSON Web Token的缩写。token一般都是用来认证的，比如我们系统中常用的用户登录token可以用来认证该用户是否登录。JWT也是经常作为一种安全的token使用。JSON Web Token（JWT）是一个开放标准（RFC 7519），它定义了一种紧凑且独立的方式，可以在各方之间作为JSON对象安全地传输信息。此信息可以通过数字签名进行验证和信任。JWT可以使用秘密（使用HMAC算法）或使用RSA或ECDSA的公钥/私钥对进行签名。

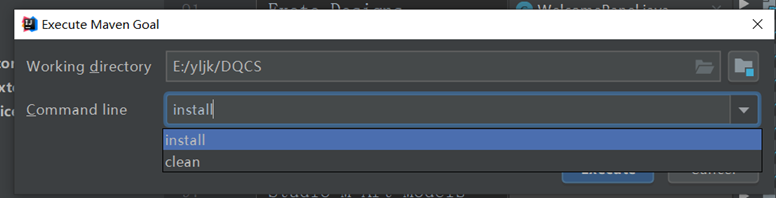
Scala是一门多范式的编程语言，一种类似java的编程语言，设计初衷是实现可伸缩的语言、并集成面向对象编程和函数式编程的各种特性。Scala 是一门多范式（multi-paradigm）的编程语言，设计初衷是要集成面向对象编程和函数式编程的各种特性。Scala 运行在 Java 虚拟机上，并兼容现有的 Java 程序。Scala 源代码被编译成 Java 字节码，所以它可以运行于 JVM 之上，并可以调用现有的 Java 类库。

springMVC通过策略接口，Spring 框架是高度可配置的，而且包含多种视图技术，例如 JavaServer Pages（JSP）技术、Velocity、Tiles、iText和POI。Spring MVC 框架并不知道使用的视图，所以不会强迫开发者只使用 JSP 技术。Spring MVC 分离了控制器、模型对象、过滤器以及处理程序对象的角色，这种分离让它们更容易进行定制。

XHTML（Extensible HyperText Markup Language的缩写，意为可扩展超文本标记语言）表现方式与超文本标记语言（HTML）类似，不过语法上更加严格。从继承关系上讲，HTML是一种基于标准通用标记语言（SGML）的应用，是一种非常灵活的置标语言，而XHTML则基于可扩展标记语言（XML），XML是SGML的一个子集。XHTML 1.0在2000年1月26日成为W3C的推荐标准。

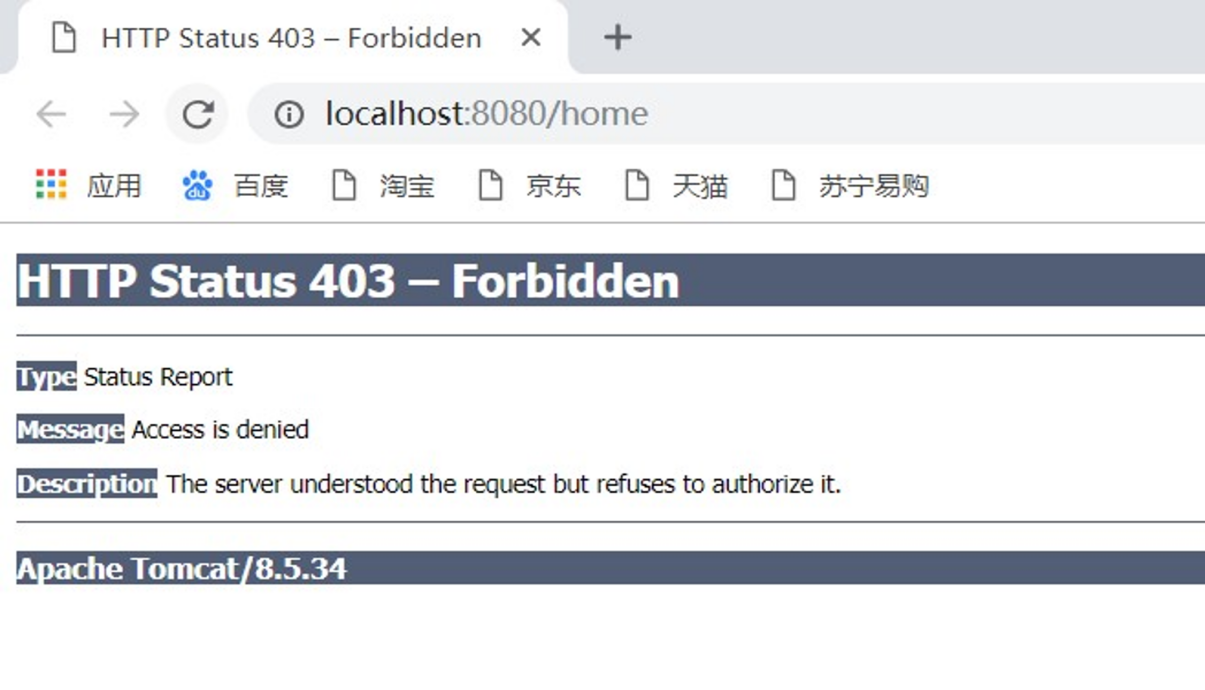
## Windows启动不了项目问题

① ：jar包下载安装：github网站在idea工具上下载代码成功，但是install下载jar包的时候网络不通畅，导致下载jar包部分失败。



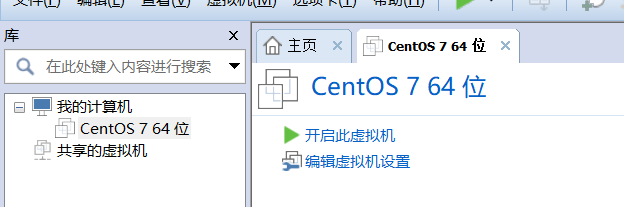
打开maven管理，install开始下载jar包。循环多次，jar包完全下载。

## 在windows系统编译问题：



由于在windows系统上，此项目编译头过长，系统启动之后403错误，它是由于xml文件拼接起来的启动头过长，windows不支持。

解决方法：安装虚拟机



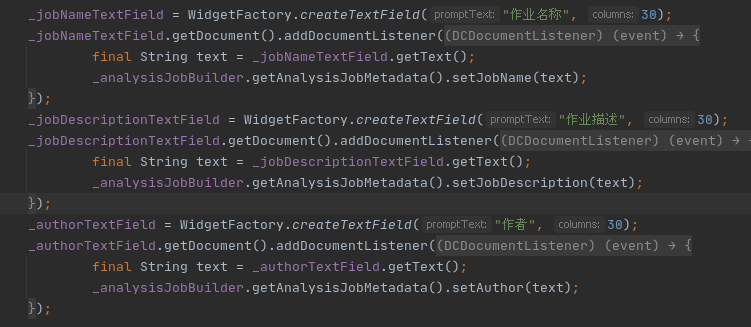
我安装的是centOS7，虚拟机上安装idea开发软件，然后安装gie项目控制软件，安装tomcat服务器，安装maven，并且配置所有相关的环境变量。导入datacleaner项目代码之后下载jar包，运行。解决成功。

## 项目发布问题

解决方法：虚拟机上通过idea成功编译为war文件，使用FileZilla文件导出到本机，然后通过tomcat服务器完成项目发布。

## 页面后台拼接

Datacleaner开源项目与以往所接触到的java项目有很大一部分不同点在于，此项目前端页面或者窗口有部分是由后台代码拼接而成，如下：

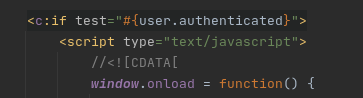


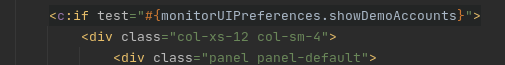
此图就是拼接的前端显示tab，分为作业名称，作业描述，作者，每个tab所占比例为30%，这与以往所接触的静态页面不同，它涉及到参数传递必须要准确，页面大小设计必须合理，其中开发的注释很少，必须先去了解源码，进一步理解其中的意思，导致开发的难度加大。

解决：先找一个小功能点从页面的按钮开始，找到后台对应的接口，从接口深入，一步步的读懂源码，然后在开发的时候模仿datacleaner的代码去写。

## 数据请求参数

页面当中有很多参数名，刚接触的时候并不知道代表的是什么意思，必须去全局搜索出代码来理解。如下：





一个页面有多个参数需要处理，所以开发的时候必须注意。

解决：使用浏览器自带调试工具，打印出各个参数，联系后台代码做理解。

## 多个用户登录

该项目分可以用两个用户登录系统， admin/admin: “演示”工作的管理员以及viewer/viewer: “演示”工作的普通用户，其中每个用户对他自己的账户所操作的工程需要分类，不能混合。所以采用了对项目加一个过滤器，在登录的时候进行权限控制或者文件的查找。从而两个用户能单独操作。

## 数据冒泡排序

在首页列表显示数据的时候需要时间的顺序排列，我采用了冒泡算法进行排列



## 文件存储

此项目没一次执行一个工程都会生成对应的文件，包括执行文件、操作日志文件还有结果文件，存放在/root/IdeaProjects/DQCS/monitor/ui/src/main/webapp/repository/demo/jobs/目录下，每次调用就需要获取文件进行解析数据。

## 首页java.lang.NullPointerException问题查找

每一次工程会有几个结果，成功、失败或是执行中，其中执行中没有结束时间，所以开发过程中就遇到了没结束时间导致计算不出执行时间，导致系统出错，输出日志显示空指针异常。

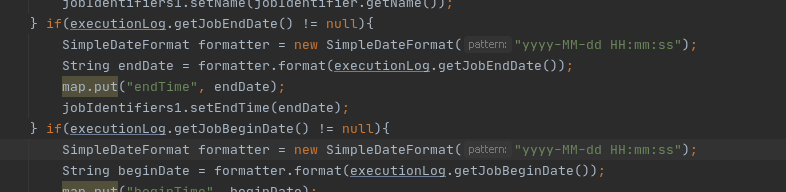
处理：



首先使用debug打断点进行程序调试，找出问题所在的地方。然后进行代码的整改。对每一项都进行一个判断为是否为空，若为空给他则给他一个默认的标志，若不为空就继续执行。

## 时间戳的截取

Copy employees to customer table1-1606804050236.analysis.result.dat这是一个工程的结果，他包括了工程名Copy employees to customer table1，时间戳1606804050236和结尾.dat组成。其中时间戳需要在后台进行转换，其中使用的是SimpleDateFormat工具，把他转换成熟悉的年月日时分秒（yyyy-MM-dd HH:mm:ss）格式，也可以方便后面做时间的计算。代码如下：



## 使用@requestParam注解接收时，是接收不到

在spring mvc应用中，对于前台form表单提交的数组或json字符串，后端在使用@requestParam注解接收时，是接收不到的，需要配置对应的转换器，比如配置实现ConditionalGenericConverter的转换器，然后再配置到spring mvc框架中即可，我使用的是spring boot，使用配置类实现WebMvcConfigurer，然后覆盖方法，配置形如此即可。

@Override

public void addFormatters(FormatterRegistry registry) {

WebMvcConfigurer.super.addFormatters(registry);

registry.addConverter(new StringToLongListConverter());

}

## JDBC访问数据库问题

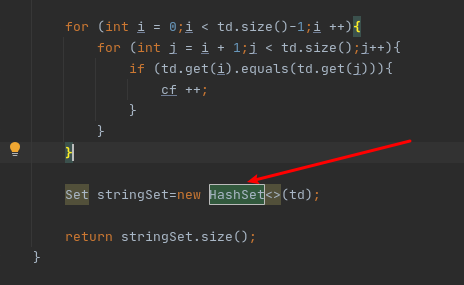
JDBC在机器不连接VPN的时候可以访问到数据库，但是在当机器连接了VPN之后，JDBC访问数据库失败，报连接被拒绝的错误。这是在Java的网络编程中，涉及到了IPv4和IPv6的问题，只需要设置Java虚拟机的属性 Java java.net.preferIPv4Stack=true即可。默认是false。

## List去重

方式一，利用HashSet不能添加重复数据的特性 由于HashSet不能保证添加顺序，所以只能作为判断条件；

方式二，利用LinkedHashSet不能添加重复数据并能保证添加顺序的特性 ；

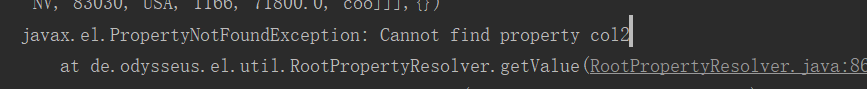
我采用的是HashSet去重方法，效率高。



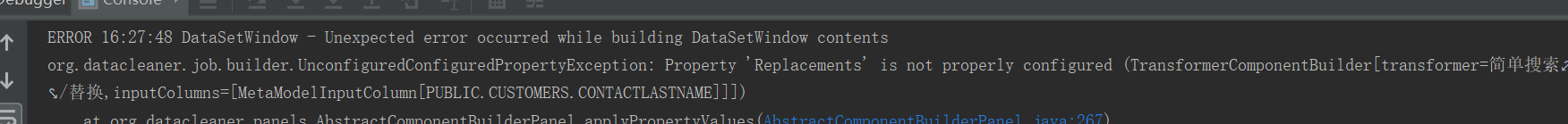
## 网页报403错误

原因：浏览器缓存问题。解决：只需要清理浏览器缓存刷新页面就可以。

## 数值-数学公式报错。



## 简单搜索/替换预览报错

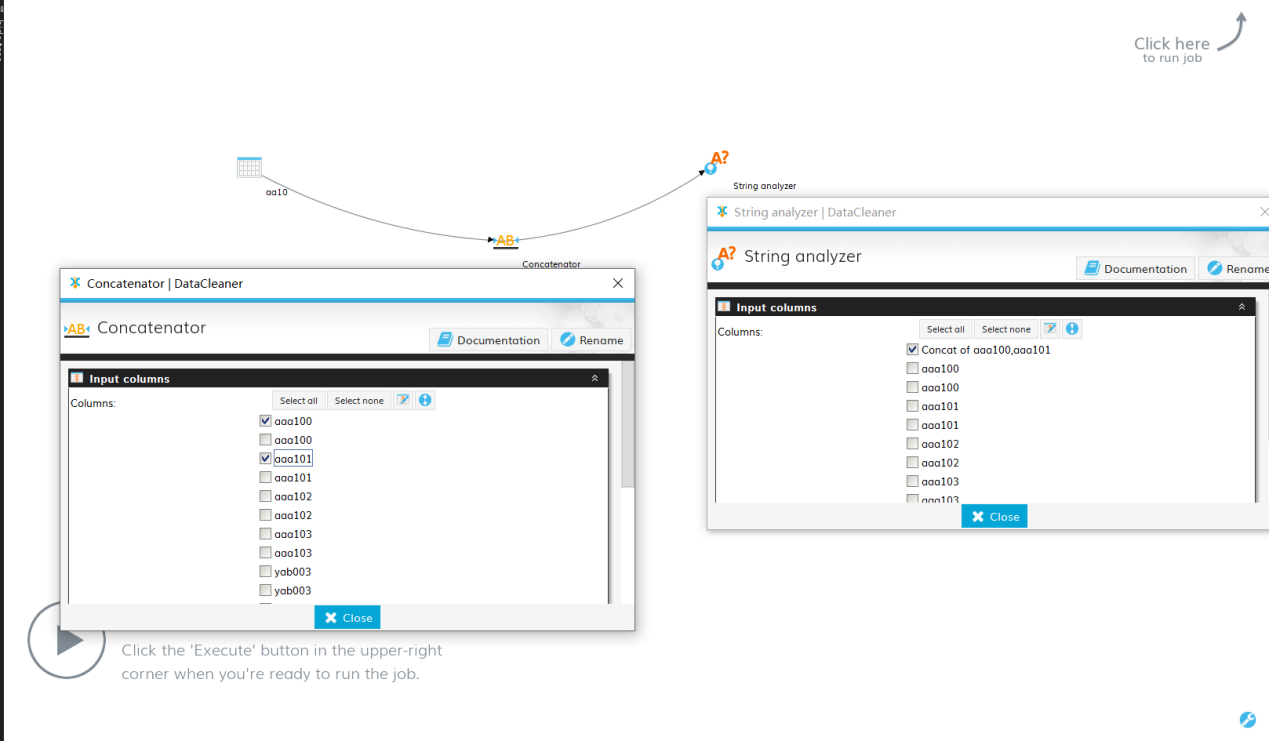


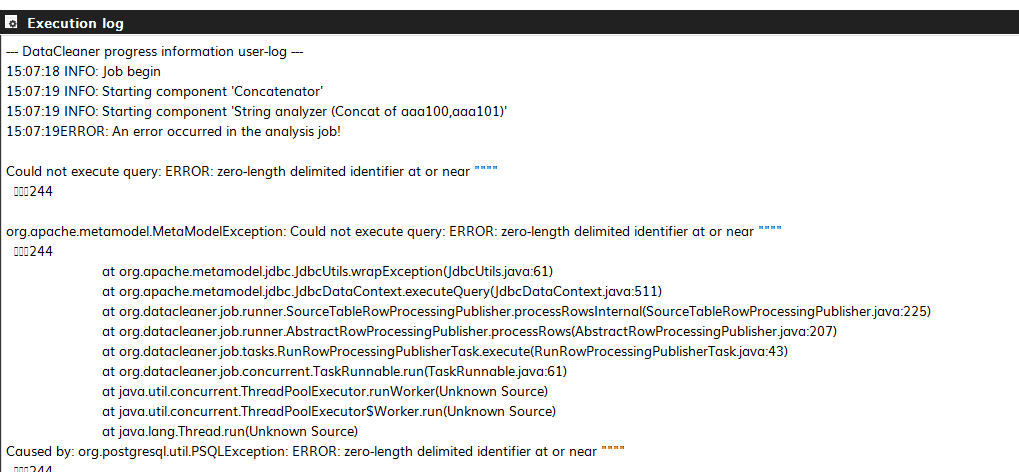
创建csv文件时，文件名有.csv后缀，执行失败



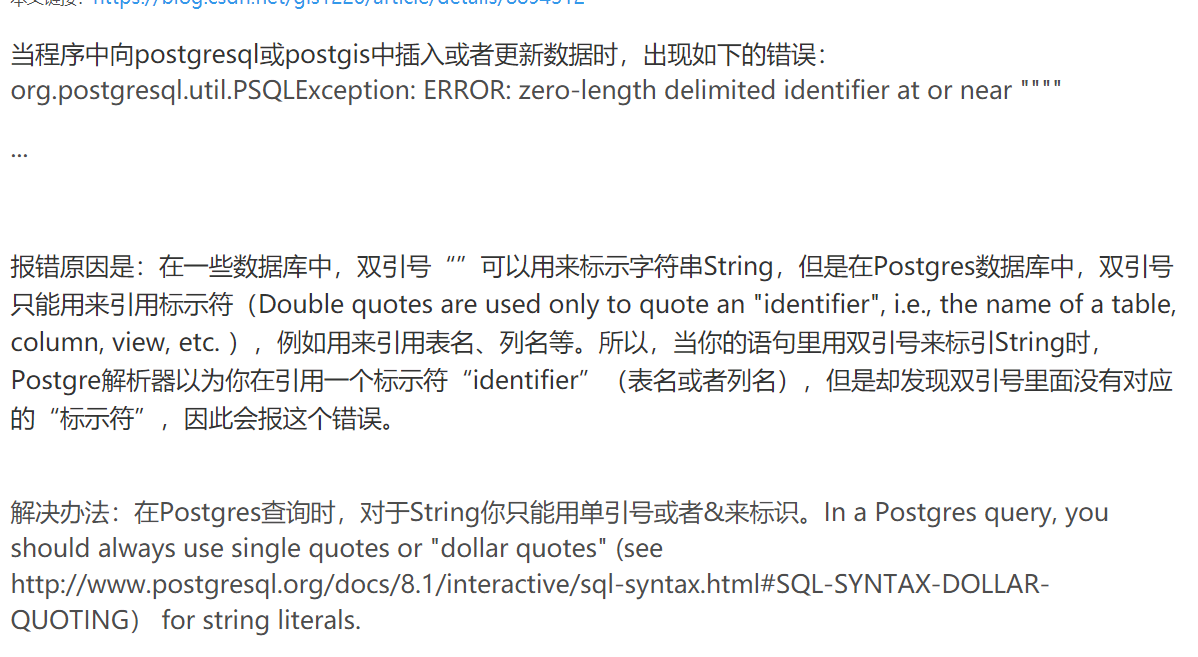
## 记录遇到的迫切需要解决的问题

对aa10表aaa100、aaa101字段数据合并，并对合并后的数据进行分析

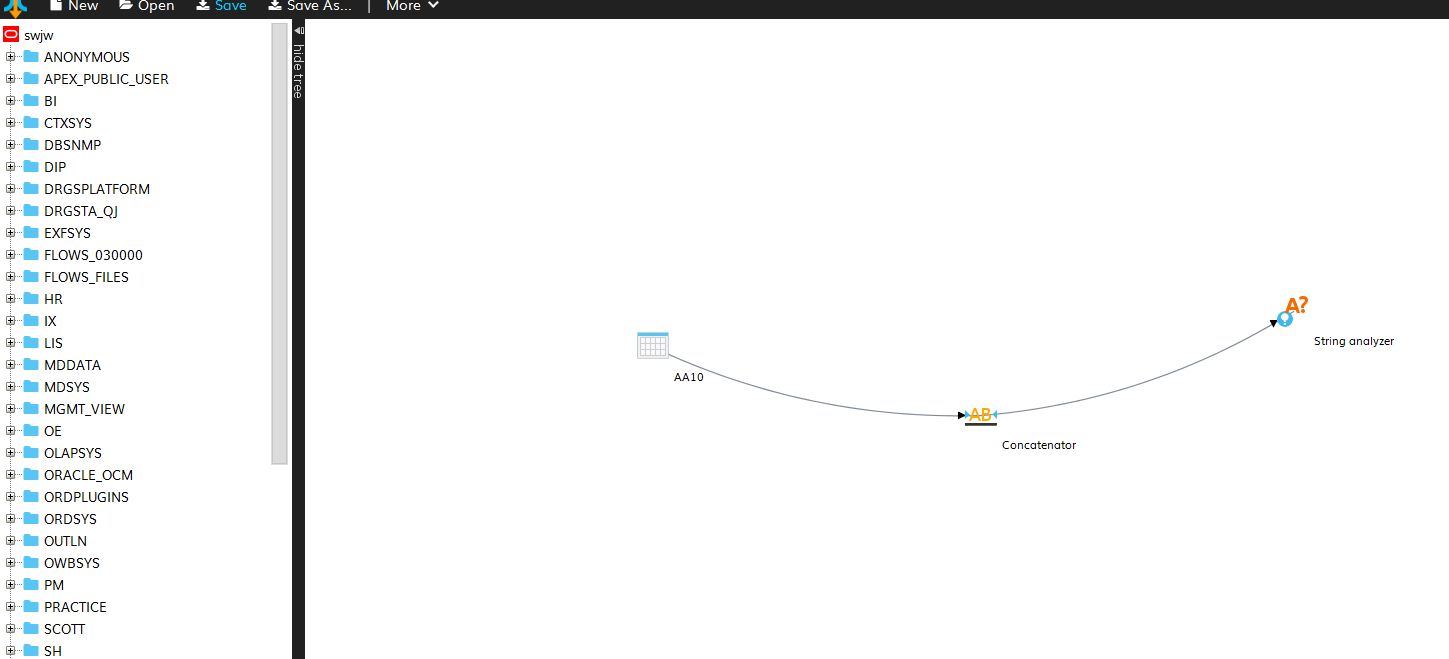


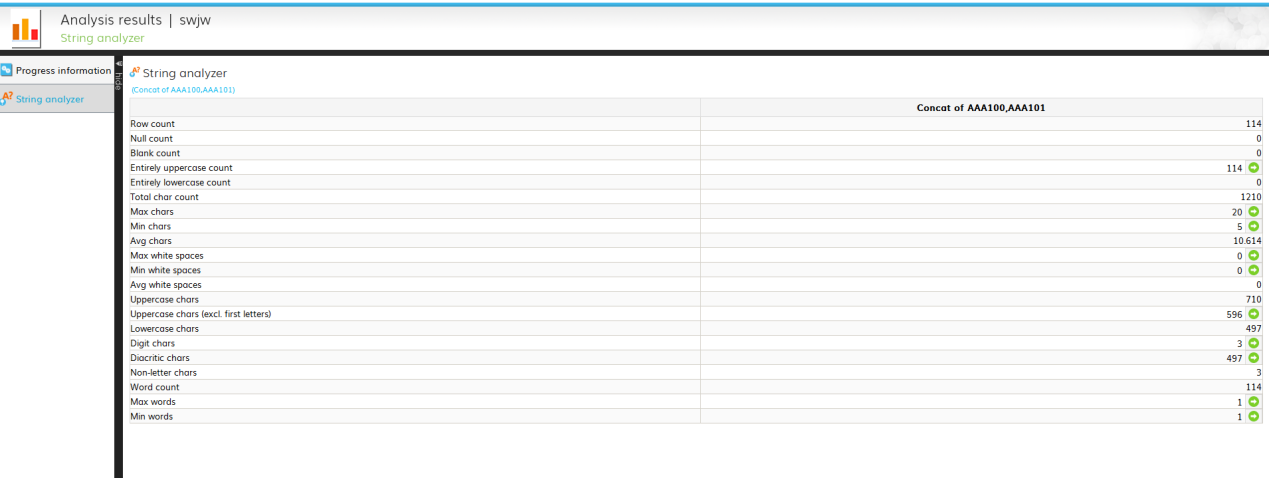


后经过多次不容分析任务测试，run job 时均会报这个错误，无法看到数据分析报告。解决思路如下



用Oracle数据库测试一下，查看是否为数据库原因造成此问题。



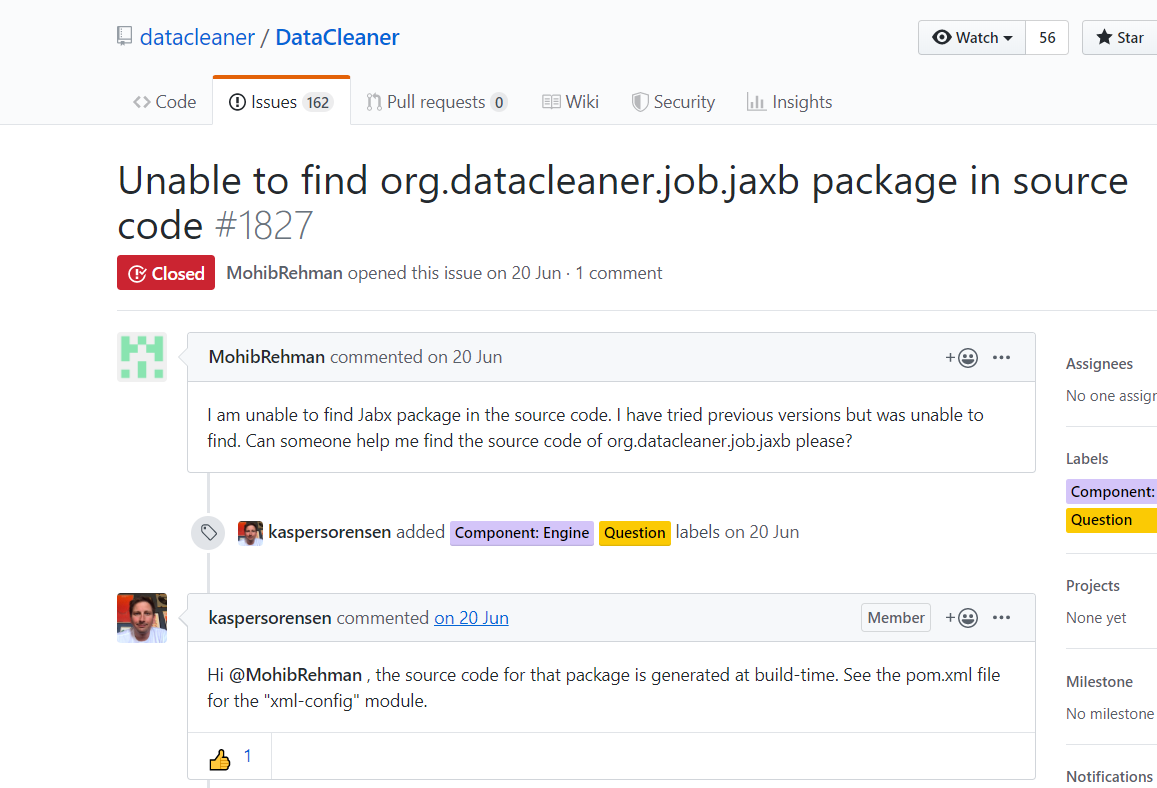


Oracle数据库经测试可以正常输出分析结果

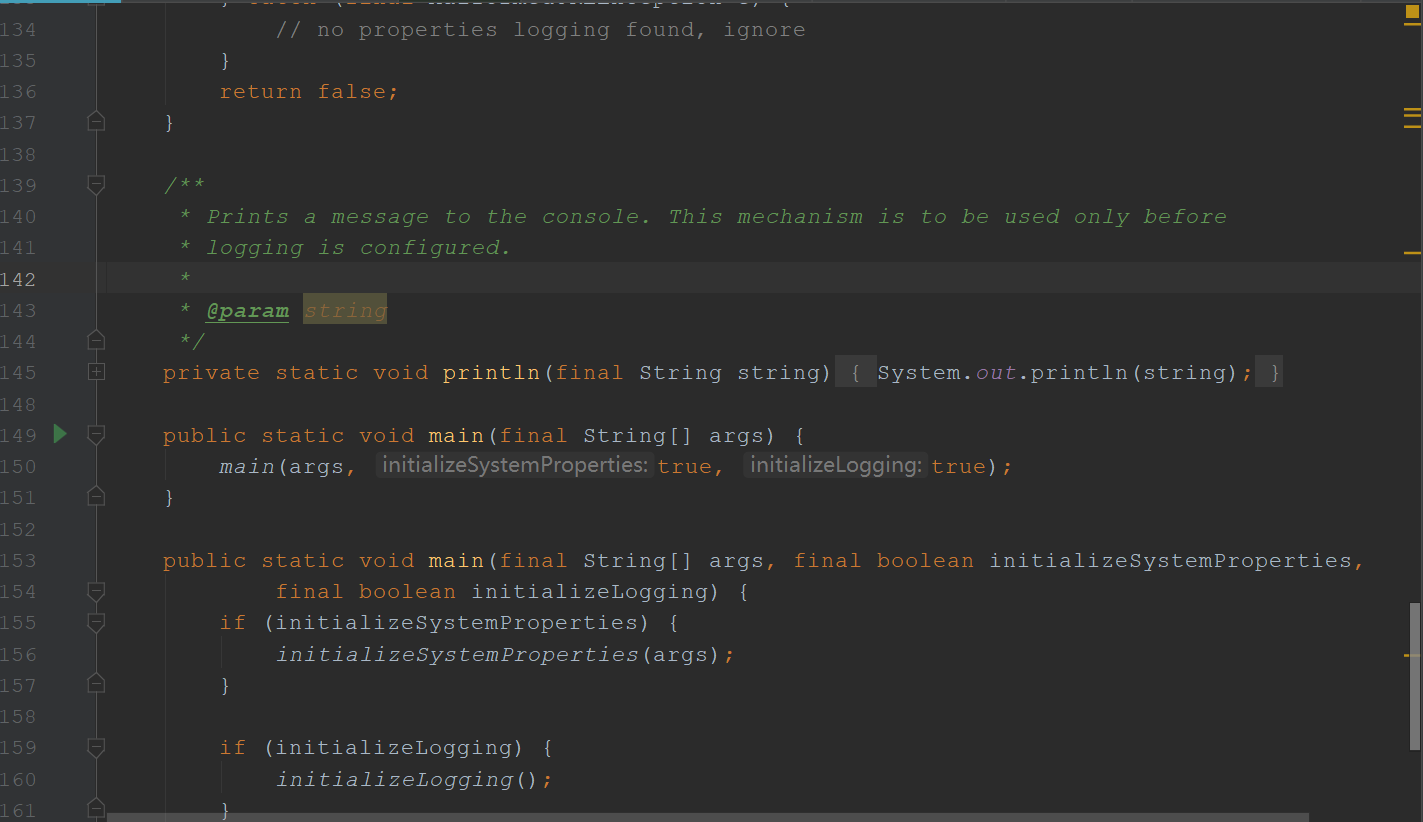
解决思路：

这里报错是直接在运行桌面程序时报的错，无法查看对应的一些错误日志信息。准备从源码部分着手，通过控制台、断点调试等方式来查看问题。

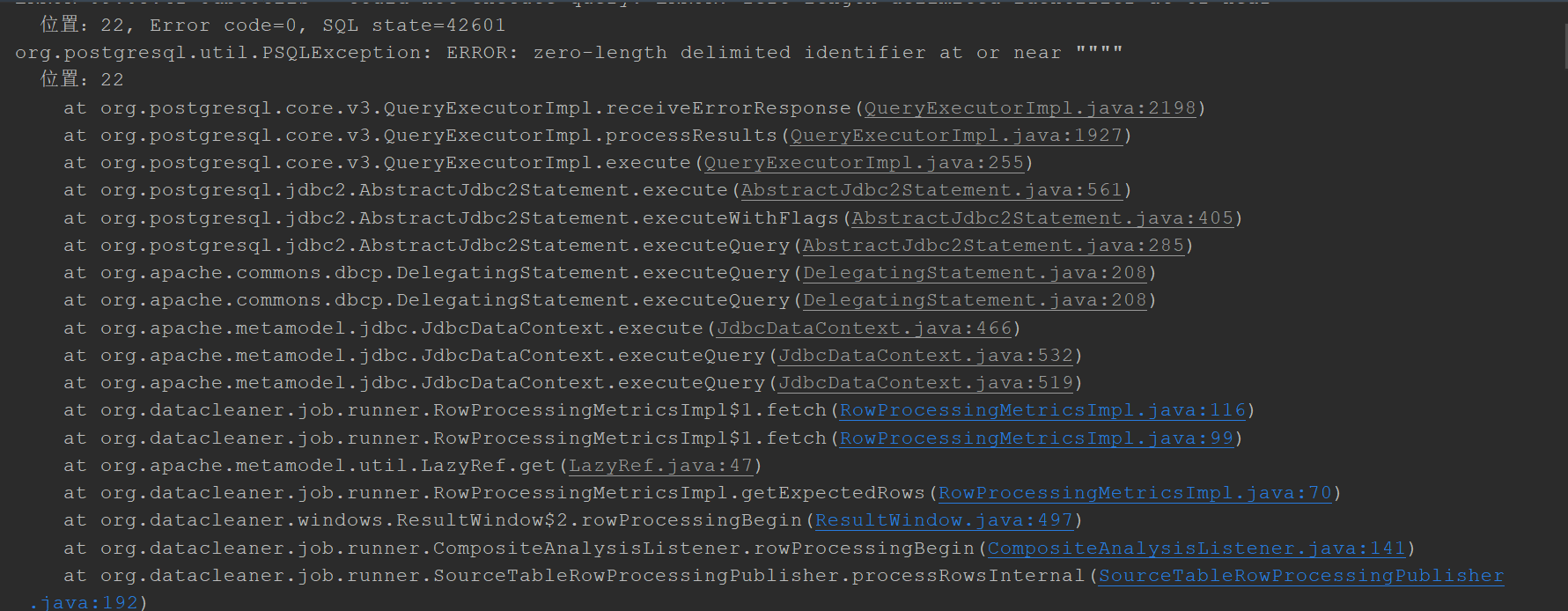
将datacleaner源码导入idea，根据pom.xml导入所有的依赖之后，找不到org.datacleaner.job.jaxb这个包，对应有几个类依然报错，一开始也找不到解决办法，几经周折最后在github上面找到相同问题，得到解决方法。



在代码里找到桌面程序的启动入口，运行main()方法，启动桌面程序



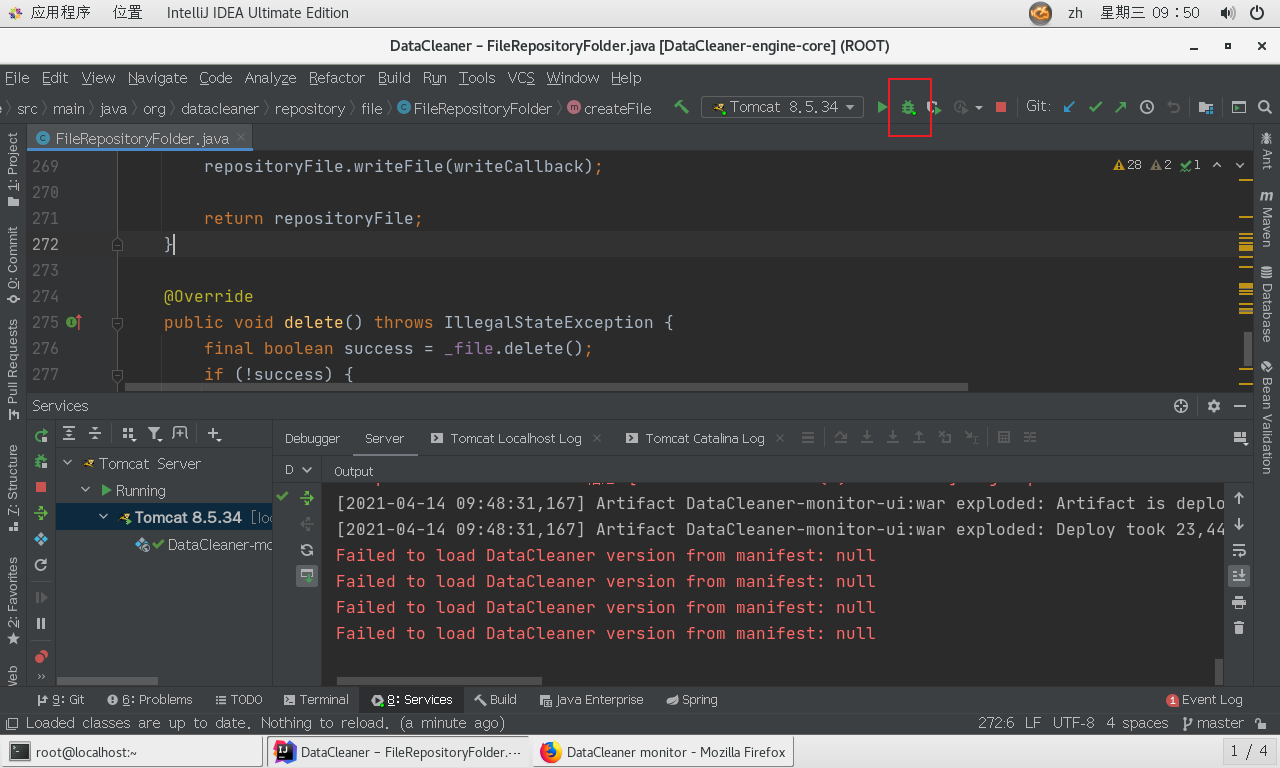
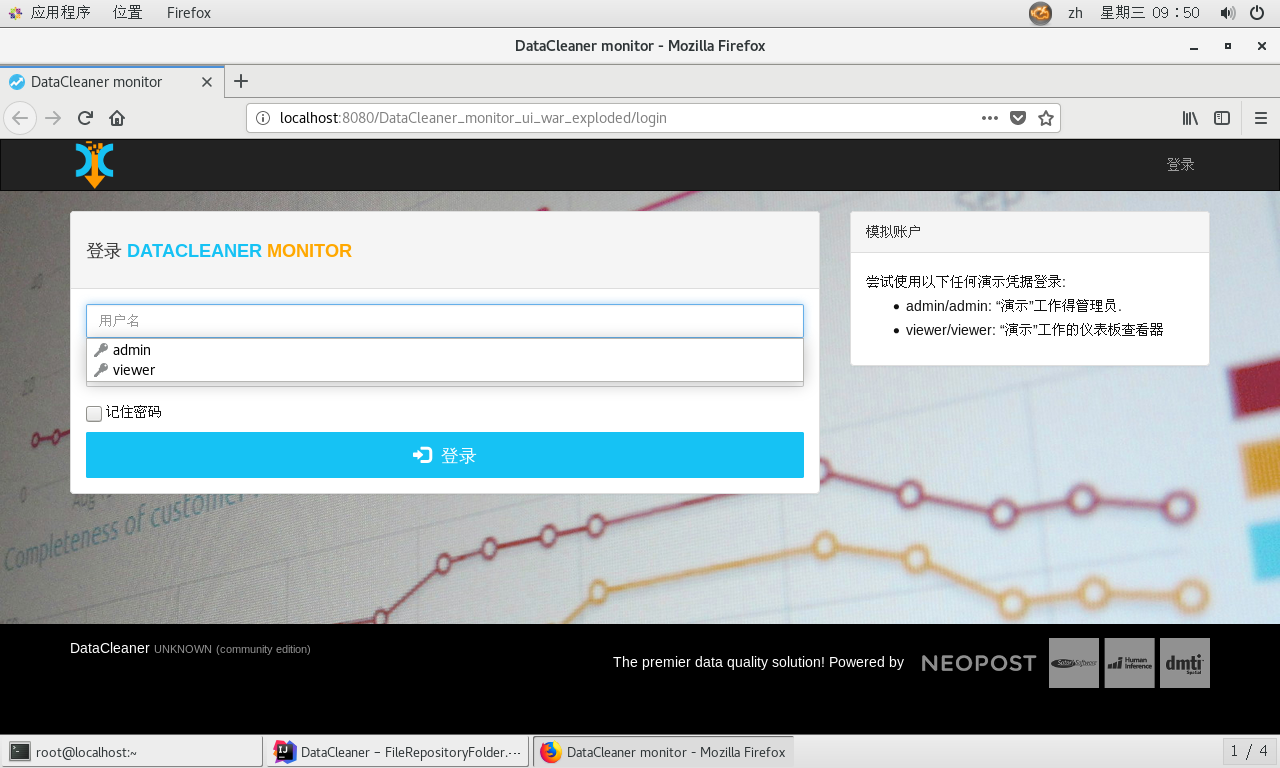
桌面程序启动后，配好任务，执行任务时依旧报同样的错，控制台输出:



接下来依次对蓝色标示的Java文件进行检查，以期查找到问题原因。

## 项目启动 整体界面布局及功能

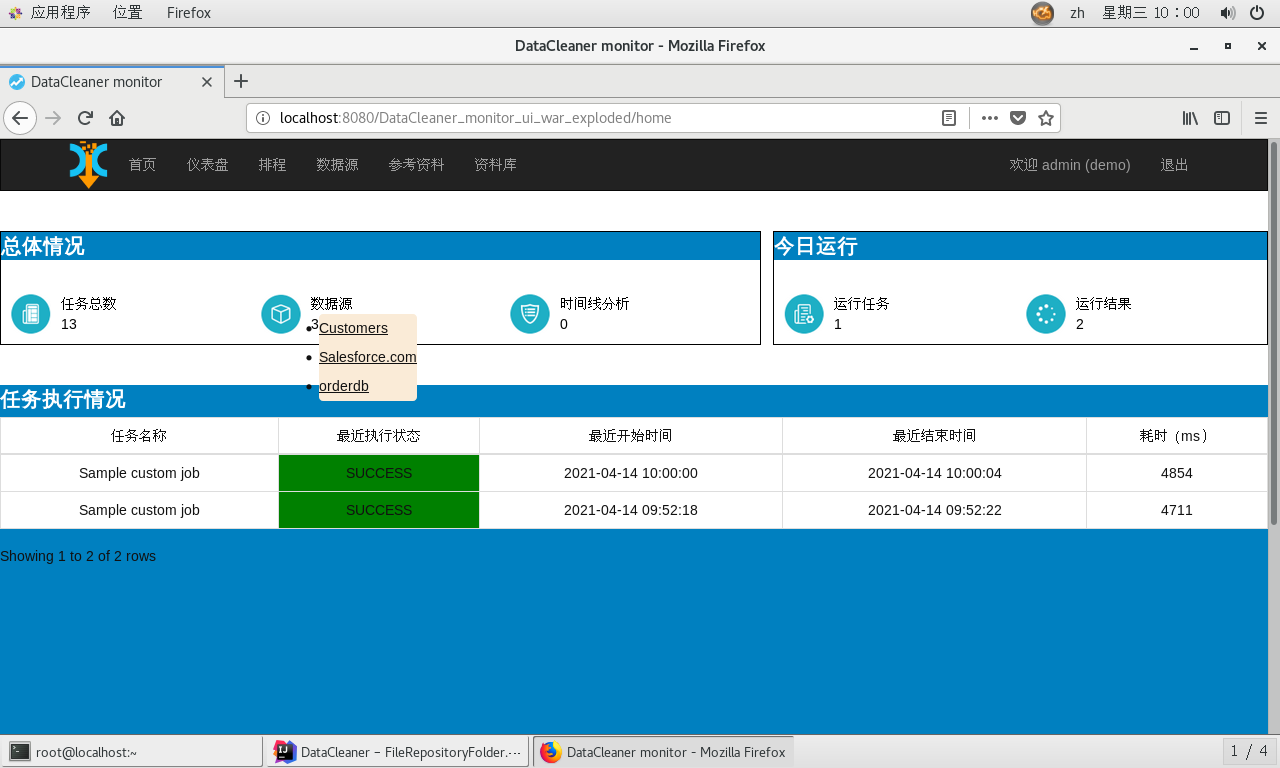
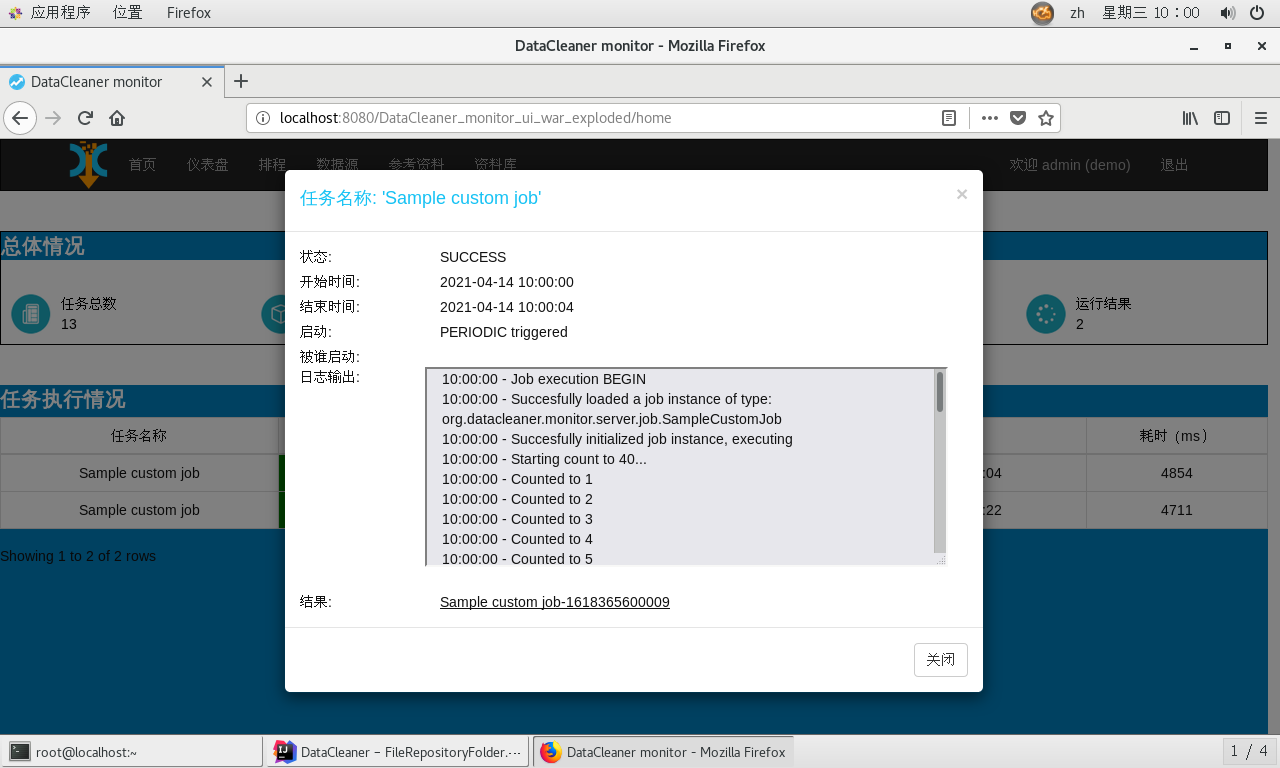
为了开发方便调试，我们通过debug启动项目，点击idea右上角tomcat启动键，等待项目运行，启动成功之后浏览器自动跳转到质控项目登录页面

## 整体界面布局及功能

该项目分成6个模块，分别是首页、仪表盘、排程、数据源、参考资料、资料库。

首页展示的是整个情况总结，包括任务总数，数据源总数，时间线分析数，还有今日运行数、云心结果、任务执行情况的详细信息，首页具有联动性，点击具体的一条信息会跳转到该任务的运行结果展示该条任务基础信息，以及运行结果。

仪表盘页面功能是管理时间线图，会展示所有任务运行的时间线结果。

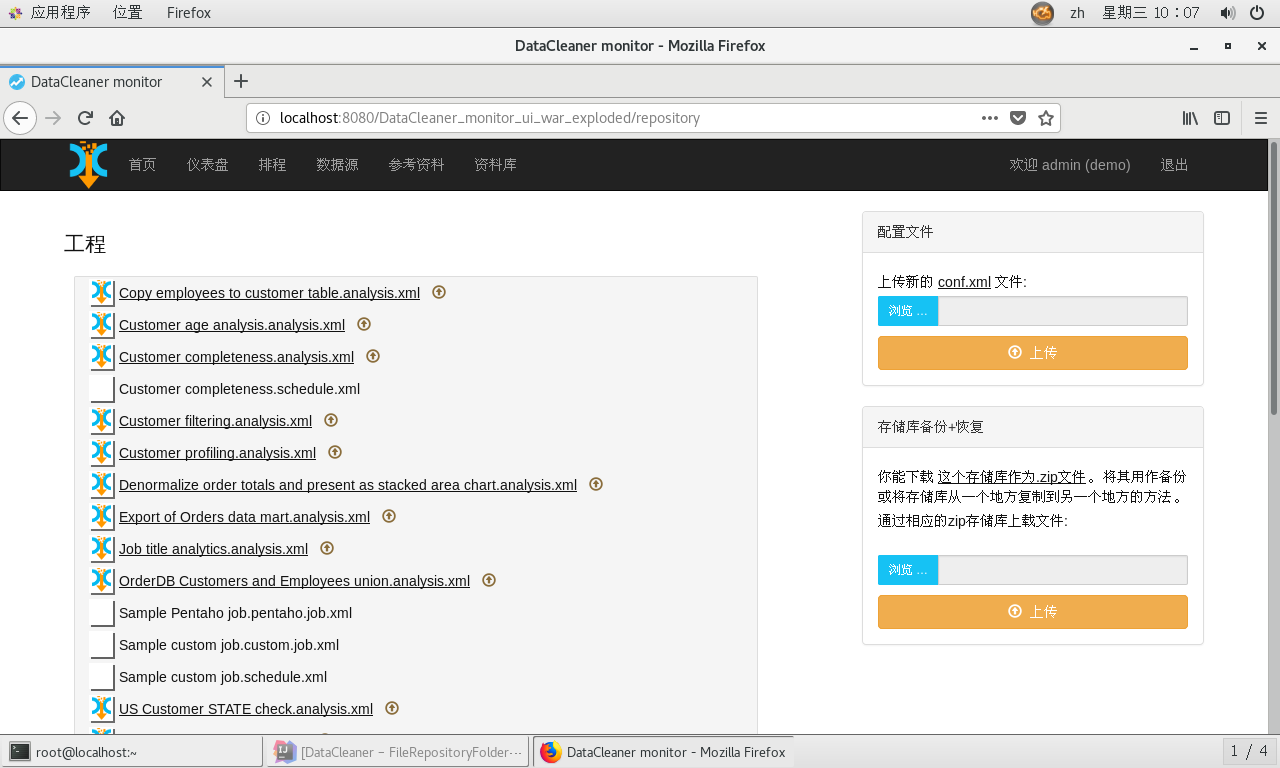
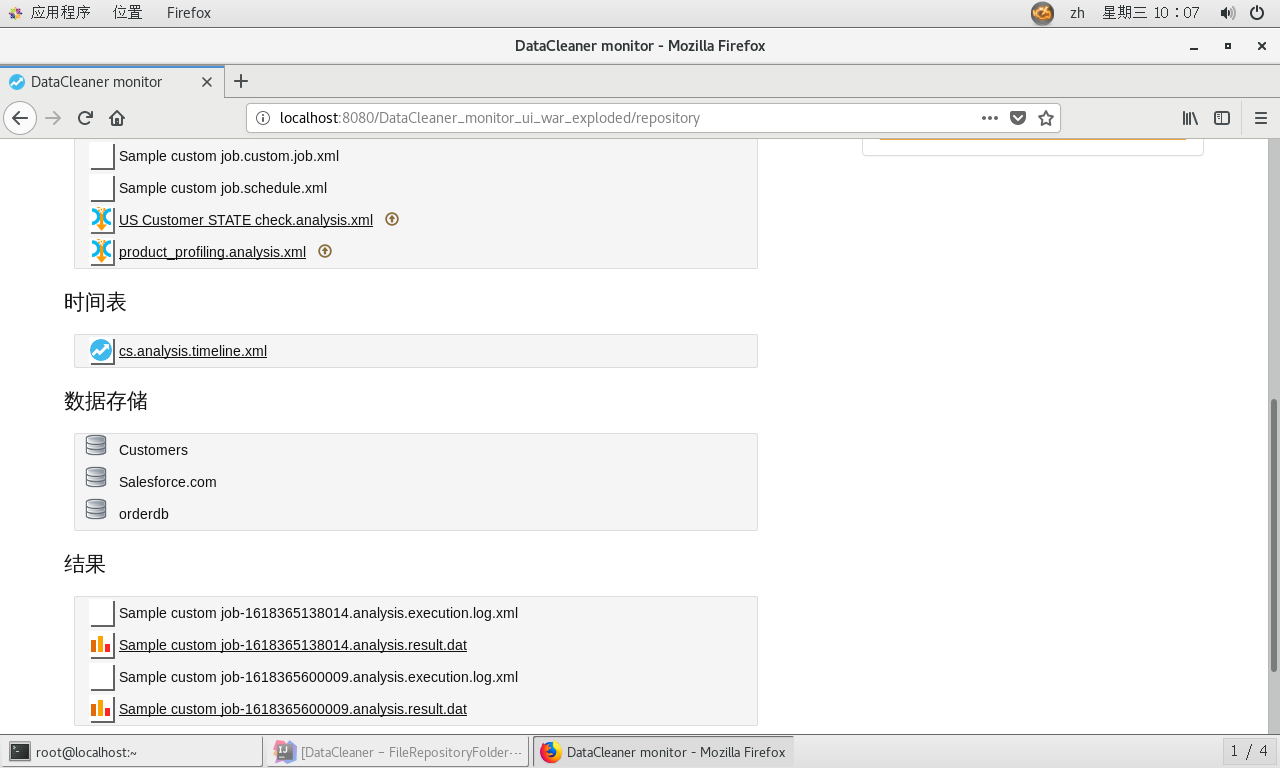
排程页面是用来管理任务的，新建任务，执行之后，结果会在首页展示出来，该页面可以查看任务的所有执行记录和结果。



数据源则可以配置数据连接源，选择数据库，编辑连接地址，用户名，密码，可以连接几乎所有数据库



参考资料和资料库则是总结展示了上传的所有任务，配置的所有数据源时间表、运行结果等

## 整体界面布局及功能web端源码分析

### 代码注释讲解

* **@\*\*SpringBootConfiguration\*\***

Spring Boot的配置类；标注在某个类上，表示这是一个Spring Boot的配置类；

* **@\*\*Configuration\*\***

配置类上来标注这个注解；配置类 ----- 配置文件；配置类也是容器中的一个组件；

* **@\*\*EnableAutoConfiguration\*\***

开启自动配置功能，自动配置生效；

* **@\*\*AutoConfigurationPackage\*\***

自动配置包

* **@\*\*Import\*\*(AutoConfigurationPackages.Registrar.class)**

Spring的底层注解@Import，给容器中导入一个组件；导入的组件由AutoConfigurationPackages.Registrar.class；==将主配置类（@SpringBootApplication标注的类）的所在包及下面所有子包里面的所有组件扫描到Spring容器；==

* **@\*\*Import\*\*(EnableAutoConfigurationImportSelector.class)**

给容器中导入组件。

* **@\*\*PropertySource\*\***

加载指定的配置文件；

* **@\*\*ImportResource\*\***

导入Spring的配置文件，让配置文件里面的内容生效；

### 登录代码检测

/\*\*

\* 登陆检查，

\*/

public class LoginHandlerInterceptor implements HandlerInterceptor {

//目标方法执行之前

@Override

public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {

Object user = request.getSession().getAttribute("loginUser");

if(user == null){

//未登陆，返回登陆页面

request.setAttribute("msg","没有权限请先登陆");

request.getRequestDispatcher("/index.html").forward(request,response);

return false;

}else{

//已登陆，放行请求

return true;

}

}

@Override

public void postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) throws Exception {

}

@Override

public void afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) throws Exception {

}

}

该系统有两个登录用户，并且权限不一样，源码采用的是拦截器进行登陆检查。

注册拦截器

//所有的WebMvcConfigurerAdapter组件都会一起起作用

@Bean //将组件注册在容器

public WebMvcConfigurerAdapter webMvcConfigurerAdapter(){

WebMvcConfigurerAdapter adapter = new WebMvcConfigurerAdapter() {

@Override

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/").setViewName("login");

registry.addViewController("/index.html").setViewName("login");

registry.addViewController("/main.html").setViewName("dashboard");

}

//注册拦截器

@Override

public void addInterceptors(InterceptorRegistry registry) {

//super.addInterceptors(registry);

//SpringBoot已经做好了静态资源映射

registry.addInterceptor(new LoginHandlerInterceptor()).addPathPatterns("/\*\*")

.excludePathPatterns("/index.html","/","/user/login");

}

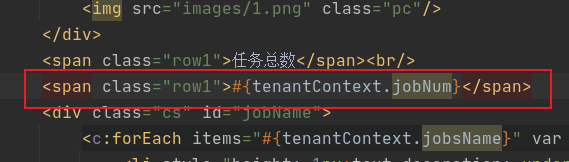
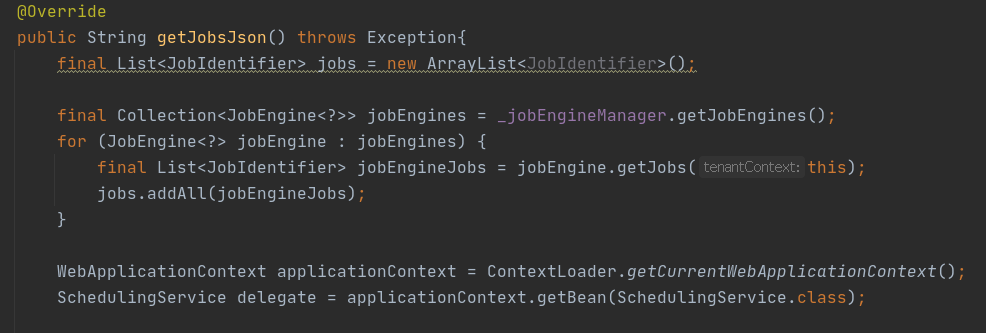
};

return adapter;

}

### 前后端数据传输

用户发送一个URL请求到前端控制器DispatcherServlet；前端控制器将请求发给处理器映射器HandlerMapping，它会根据xml配置、注解等进行查找hander；处理器映射器返回执行链HandlerExecutionChain(里面有handler对象，在它之前有多个interceptor拦截器)；前端控制器通过处理器适配器HandlerAdapter去执行Handler，不同的Handler由不同的适配器执行；通过Handler处理器，即我们熟悉的Controller来处理业务逻辑；处理完之后，返回ModelAndView对象，其中有视图名称，模型数据等； HandlerAdapter将ModelAndView返回给DispatcherServlet； DispatcherServlet将得到的ModelAndView传递给视图解析器ViewResolver进行解析； ViewResolver解析后返回具体的视图View；前端控制器对视图View和数据进行渲染，将模型数据等填充到request域中；将最终的视图返回给客户，产生response响应。

前端控制器DispatcherServlet：接收请求响应结果，相当于转发器、中央处理器，减少了其他组件之间的耦合度；

处理器映射器HandlerMapping：根据请求URL查找handler；

处理器适配器HandlerAdapter：按特定规则去执行handler，故编写handler时按HandlerAdapter要求去做，这样适配器才可正确执行handler；

视图解析器ViewResolver：根据逻辑视图解析成真正的视图（View对象）

视图View：View是一个接口，实现类支持不同的View类型（jsp，PDF，Excel...）

### 项目启动源码分析

* **原理**

jar包：执行SpringBoot主类的main方法，启动ioc容器，创建嵌入式的Servlet容器；

war包：启动服务器，\*\*服务器启动SpringBoot应用SpringBootServletInitializer，启动ioc容器。

* **规则**

1、服务器启动（web应用启动）会创建当前web应用里面每一个jar包里面ServletContainerInitializer实例：

2、ServletContainerInitializer的实现放在jar包的META-INF/services文件夹下，有一个名为javax.servlet.ServletContainerInitializer的文件，内容就是ServletContainerInitializer的实现类的全类名。

3、还可以使用@HandlesTypes，在应用启动的时候加载项目中的类。

* **流程**

1、启动Tomcat

2、找到下面路径中的文件：org\springframework\spring-web\4.3.14.RELEASE\spring-web-4.3.14.RELEASE.jar!\META-INF\services\javax.servlet.ServletContainerInitializer：

Spring的web模块里面有这个文件：\*\*org.springframework.web.SpringServletContainerInitializer\*\*

3、SpringServletContainerInitializer将@HandlesTypes(WebApplicationInitializer.class)标注的所有这个类型的类都传入到onStartup方法的Set<Class<?>>；为这些WebApplicationInitializer类型的类创建实例；

4、每一个WebApplicationInitializer都调用自己的onStartup；

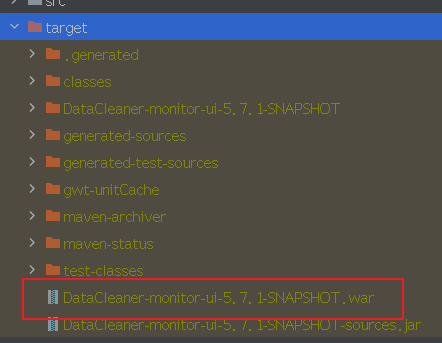
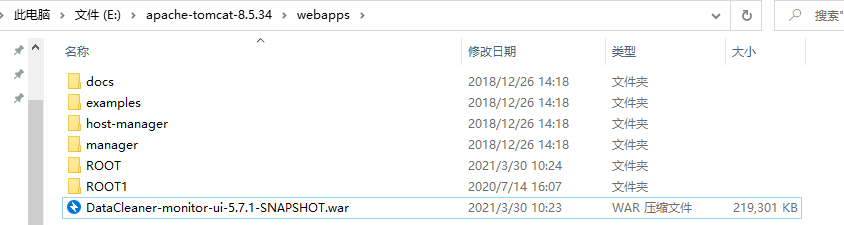
5、相当于我们的SpringBootServletInitializer的类会被创建对象，并执行onStartup方法

6、SpringBootServletInitializer实例执行onStartup的时候会createRootApplicationContext；创建容器；

7、Spring的应用就启动并且创建IOC容器。

## 项目打包发布

使用maven的package功能打包项目，成功之后会在项目中生成一个war包，具体位置是自己在xml文件中配置的。使用该war包，放在服务器上的tomcat中webapps文件夹下。执行bin文件下的start.bat文件，等待启动。

将打包好的项目war包放在webapps启动。

需要注意的是，windows系统下，通过tomcat启动项目，可能会出现乱码问题。

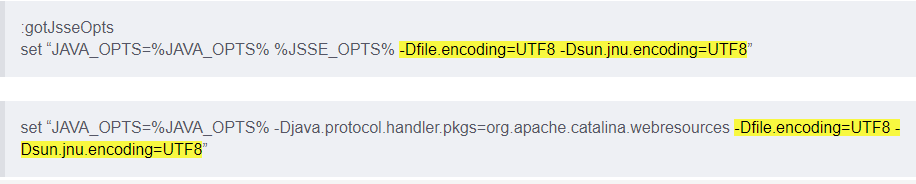


解决思路：

这需要改一下tomcat配置文件。tomcat目录 —— bin文件夹 —— catalina.bat ：:gotJsseOpts

set “JAVA\_OPTS=%JAVA\_OPTS% %JSSE\_OPTS% -Dfile.encoding=UTF8 -Dsun.jnu.encoding=UTF8”

set “JAVA\_OPTS=%JAVA\_OPTS% -Djava.protocol.handler.pkgs=org.apache.catalina.webresources -Dfile.encoding=UTF8 -Dsun.jnu.encoding=UTF8”



修改之后关闭服务，重新启动，乱码问题成功解决。