## dada-report报表

## 创建工程

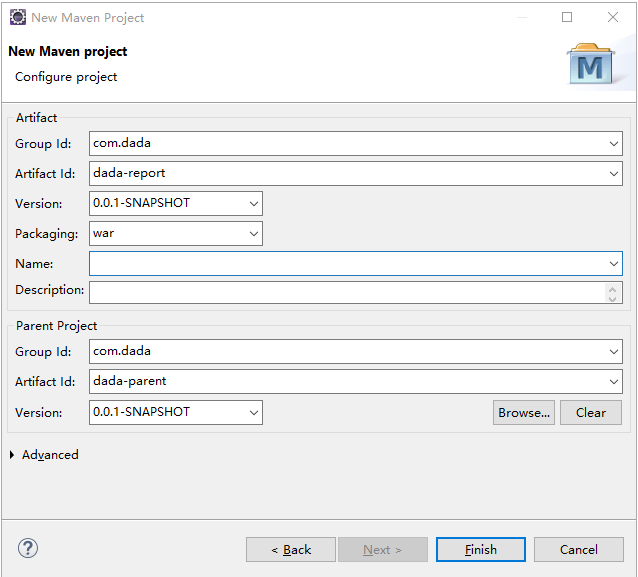
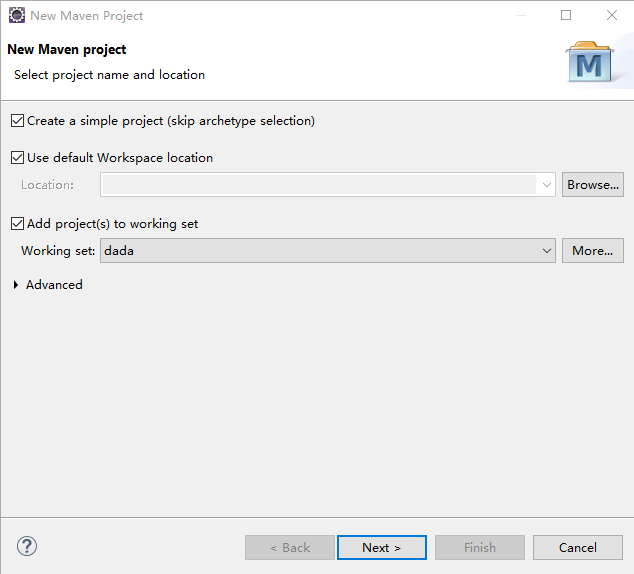
### 服务层dada-report整合UReport报表设计器工具

使用maven管理工程：War包。

#### 使用的技术

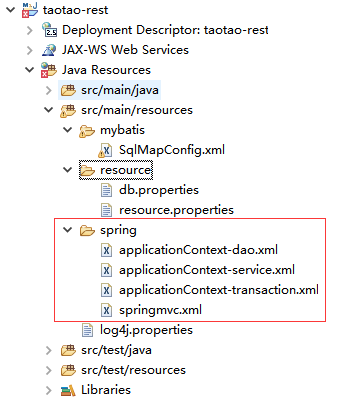
* Mybatis
* Spring
* 发布服务：cxf、springmvc

#### 创建工程



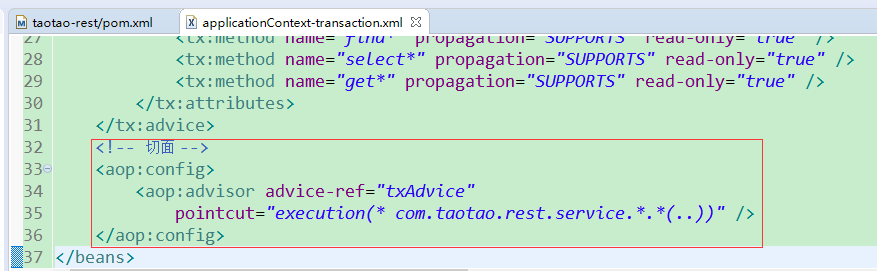
#### 框架整合

整合ssm框架（可以参考dada-manager、dada-rest工程）



修改说明：





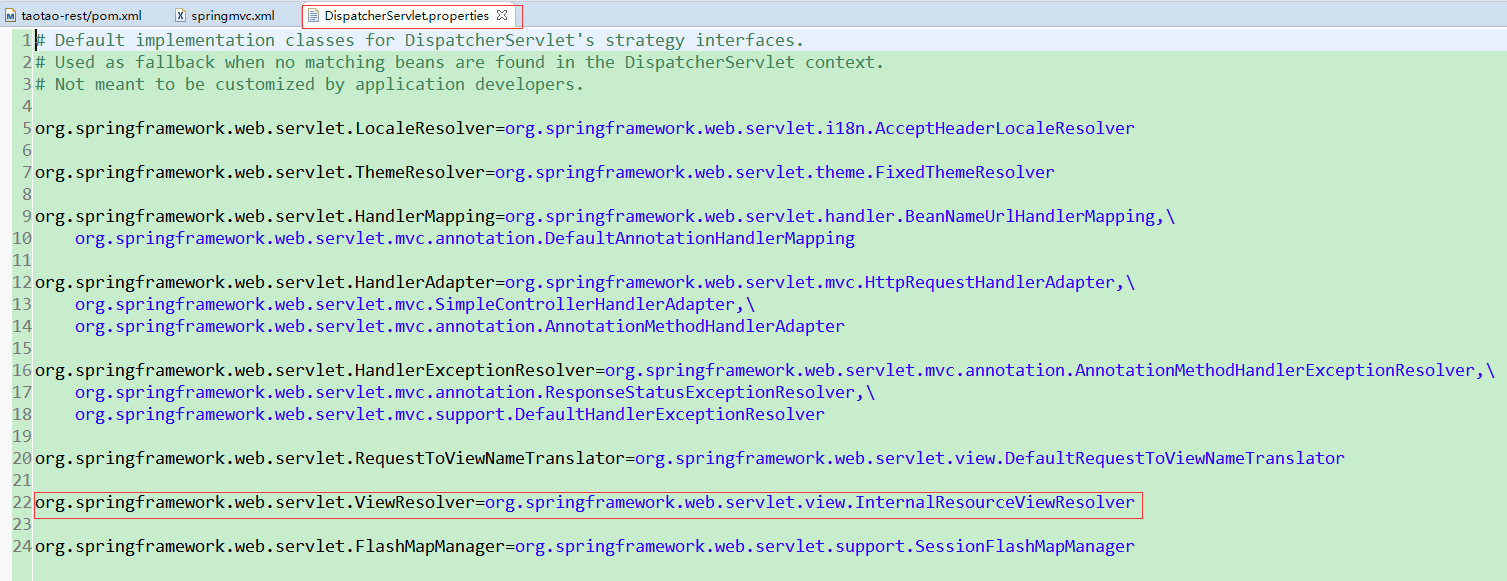


Spring默认配置了视图解析器，如果springmvc.xml中没有配置，默认也可以使用jsp

在对应springmvc的依赖jar中查看相关的信息

org\springframework\spring-webmvc\4.1.3.RELEASE\spring-webmvc-4.1.3.RELEASE.jar

org.springframework.web.servlet包下有DispatcherServlet.properties，默认配置了视图解析器



Web.xml文件复制：修改两个内容--项目名称、拦截路径、控制器

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

#### Pom文件

添加对taotao-manager-dao的依赖

添加spring的依赖

配置tomcat插件

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*> |

#### SSM框架整合UReport

参考链接：

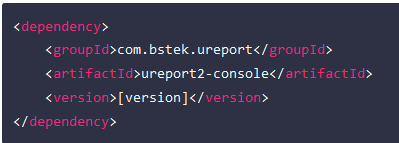
<https://blog.csdn.net/dongjak/article/details/83211795>

<https://www.w3cschool.cn/ureport/ureport-5j7b2h8l.html>



##### 整合过程分析

**1.在pom.xml文件中引入相关依赖**



<dependency>

<groupId>com.bstek.ureport</groupId>

<artifactId>ureport2-console</artifactId>

<version>[version]</version>

</dependency>

完整pom.xml参考配置：



<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.bstek.ureport</groupId>

<artifactId>ureport2-demo</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>com.bstek.ureport</groupId>

<artifactId>ureport2-console</artifactId>

<version>2.2.2</version>

</dependency>

</dependencies>

</project>

**2.在web.xml中配置UReport的Servlet**

<servlet>

<servlet-name>ureportServlet</servlet-name>

<servlet-class>com.bstek.ureport.console.UReportServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>ureportServlet</servlet-name>

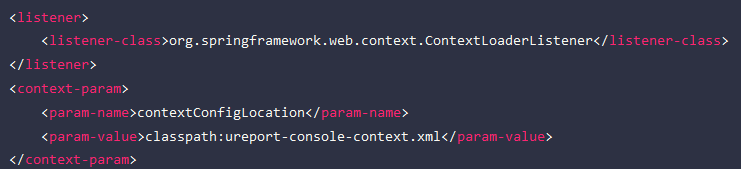
<url-pattern>/ureport/\*</url-pattern>

</servlet-mapping>

**3.引入UReport相关的配置文件**

有三种方式实现配置文件引入：

方式1：直接在web.xml中加载ureport-console-context.xml文件（针对普通的Maven工程配置，没有采用spring文件）



<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:ureport-console-context.xml</param-value>

</context-param>

方式2：在已有的spring配置文件中导入数据

<import resource="classpath:ureport-console-context.xml" />

方式3：如果没有spring配置文件，直接创建一个context.xml中



<?xml version="1.0" encoding="UTF-8"?><beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd

">

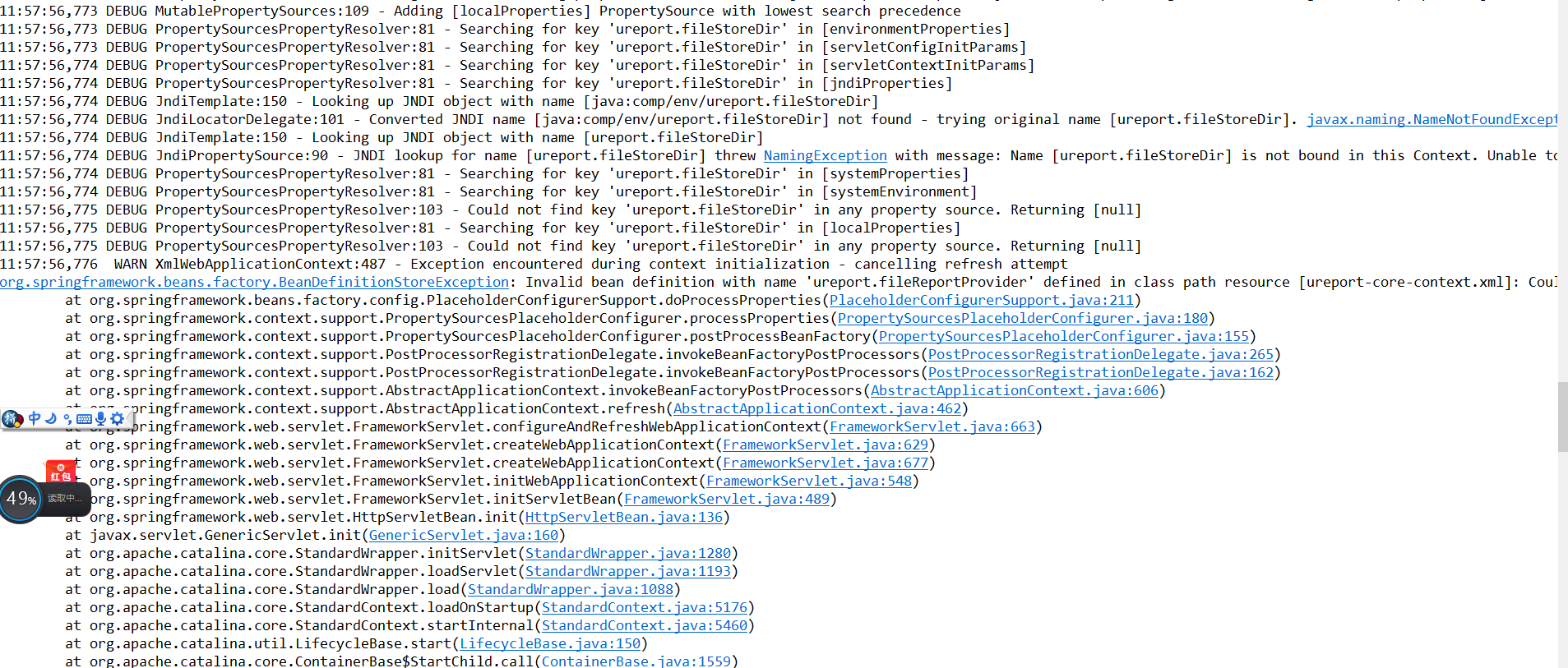
<import resource="classpath:ureport-console-context.xml" /></beans>

此处对已有的SSM项目整合做说明：

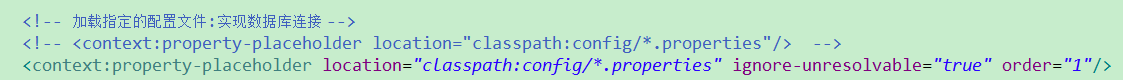
无论是以上哪种方式实现，必须保证在web.xml需要加载相应的spring配置文件,否则项目在启动的时候就会报视图解析失败，没有相应的designer

问题1：指定属性无法找到

此外，启动的时候如果报找不到指定的属性：则需要在属性配置的时候需要不存在（未配置）的属性（在applicationContext-dao.xml、springmvc.xml下加载了属性文件，添加属性忽略掉没有定义的属性查找），报错内容如下所示



针对spring配置文件处理：

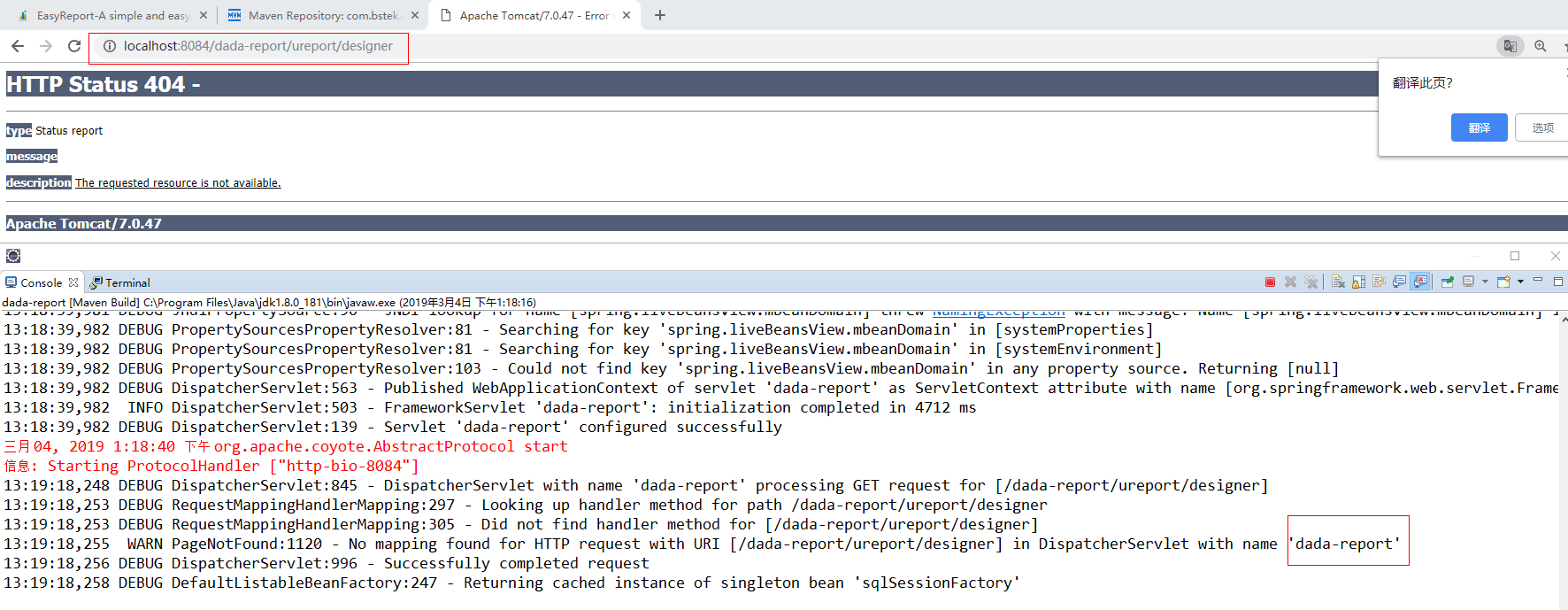


针对普通maven工程处理：

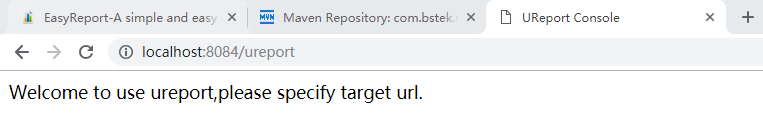


问题2：404或指定designer设计器不存在

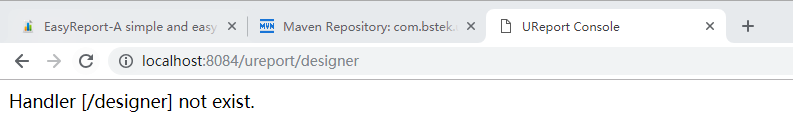
在整合的ssm中配置ureport出错，通过链接访问数据的时候无法找到指定的页面，初步考虑路径配置，之后查看web.xml配置，发现在之前的前端控制器配置中，默认拦截了所有的请求，因此在请求的时候直接被dada-report这个前端控制器拦截，但dada-report这个控制器下并没有指定的ureport相关的内容存在，因此会报404访问路径错误或者是designer报表设计器不存在的问题，此处需要对dada-report拦截的内容做限制，参考前端控制器拦截（拦截说明、拦截顺序、执行顺序）

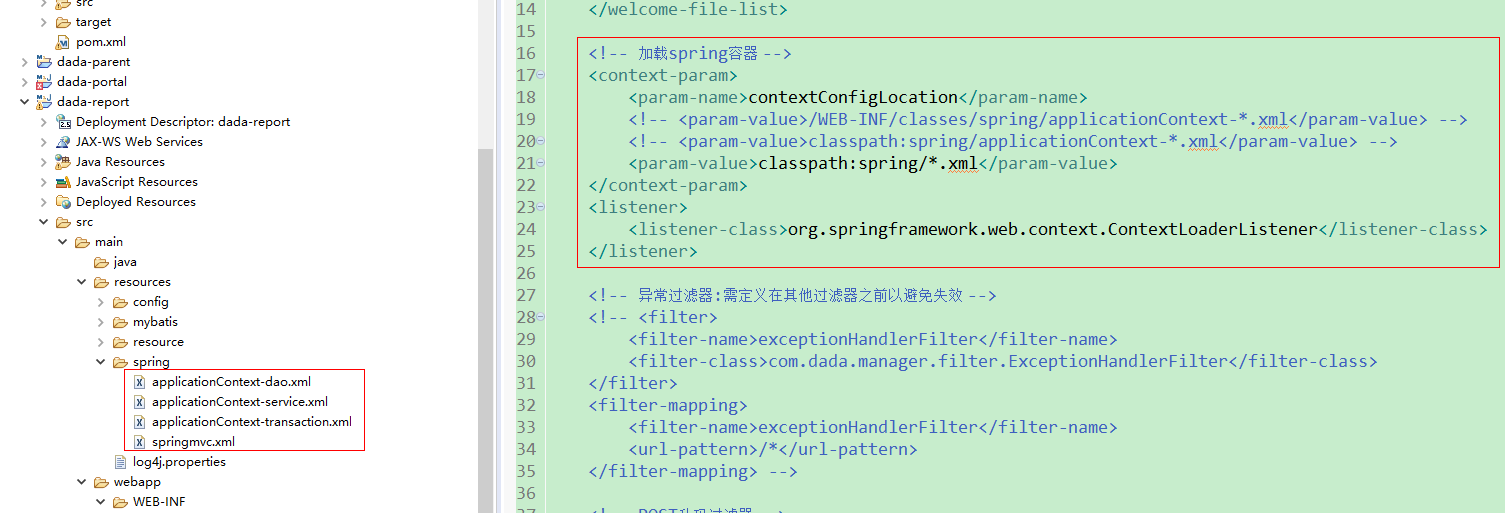


原始配置：

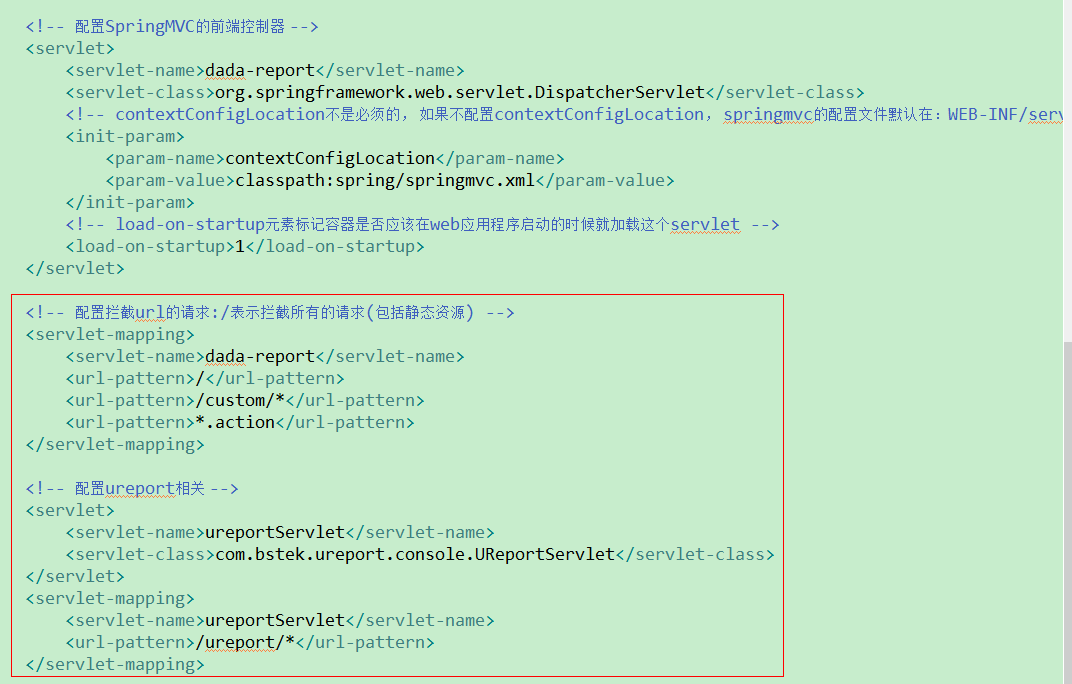


直接配置如下所示，但如果出现designer视图不存在，考虑是相应的文件没有加载进去，因为在springmvc.xml中加载了有关的ureport2配置文件，因此要在初始化加载spring容器的时候将对应的springmvl.xml配置文件进行加载，否则无法找到相应内容，出现如下所示错误





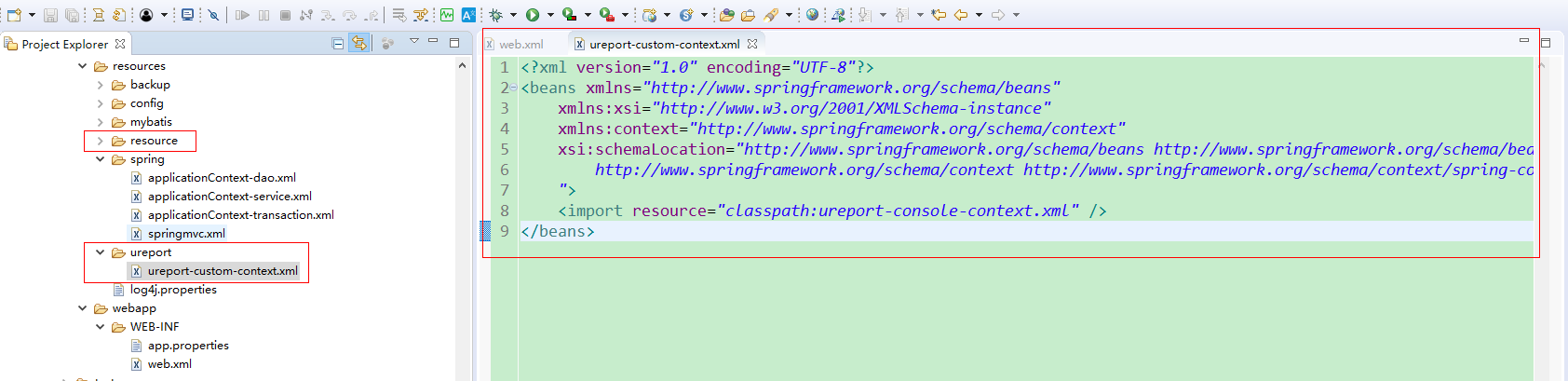
要么就设置Servlet默认启动的时候加载配置文件

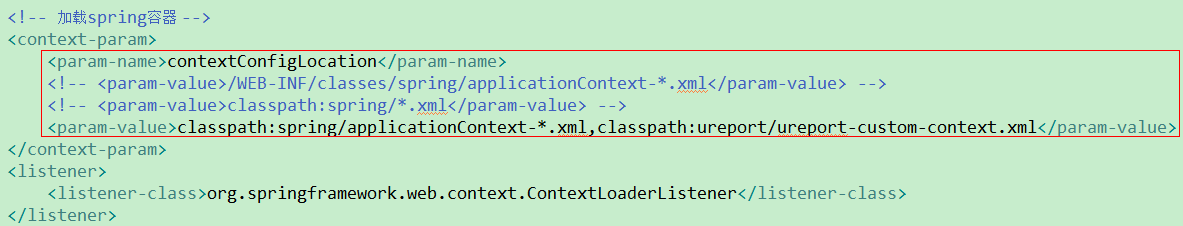


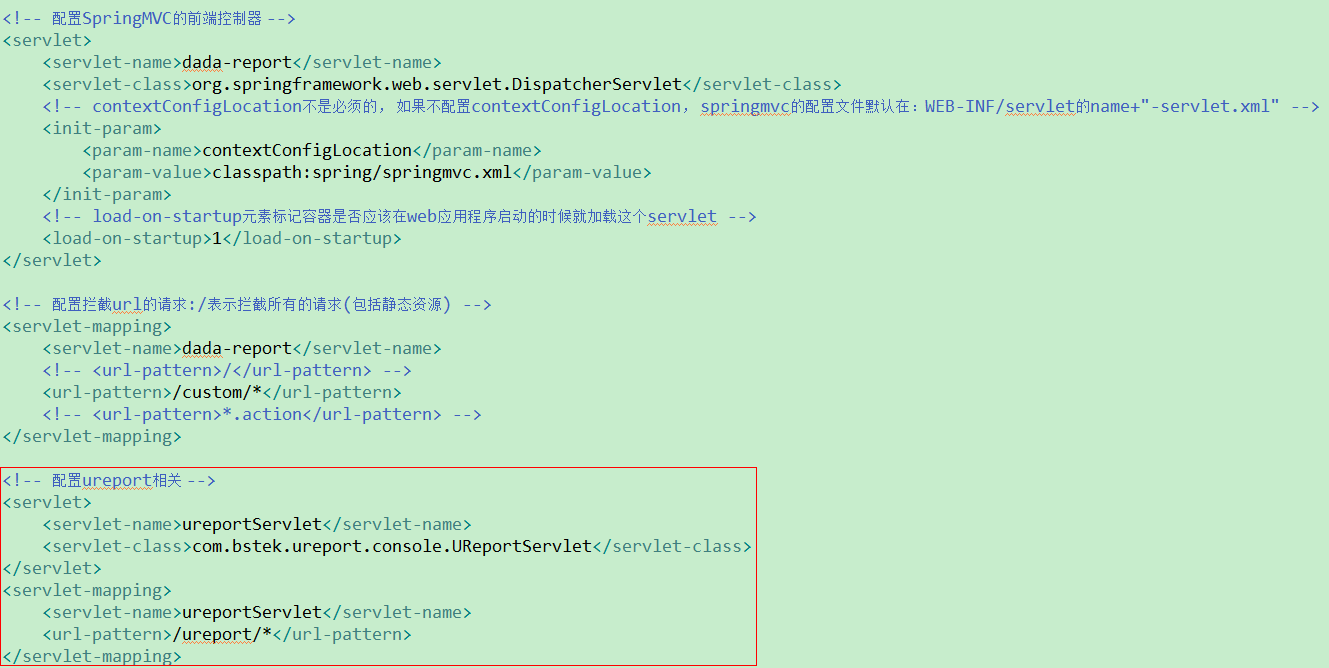
实际SSM整合说明如下：

依赖配置：添加ureport相关的依赖jar包

为了避免配置文件混乱，也避免改动原有ssm配置，此处单独在resources目录下创建一个文件夹ureport存放ureport-custom-context.xml配置文件以编辑相关的ureport配置，并在web.xml中配置相应的全局参数（必须在context-param中配置加载文件）

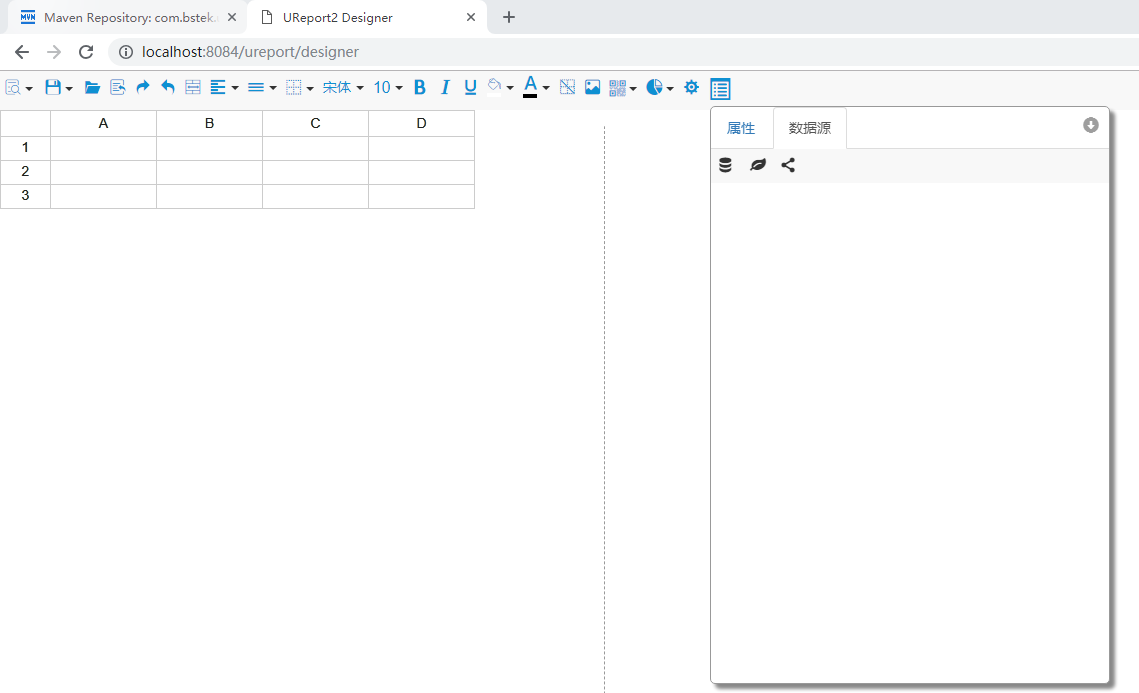






配置完成，启动项目，访问链接：<http://localhost:8084/ureport/designer>

Ureport报表设计器显示如下：



可以适当了解web.xml各个参数配置的含义：



完成配置，根据W3CSchoool等相关教程完成业务和报表工具的整合，以下简单根据教程完成业务流程简单实现

<https://www.w3cschool.cn/ureport/ureport-y4op2han.html>

测试的时候如果出现按钮点击无反应，则可能是弹出式窗口被浏览器拦截，只需要允许弹出即可，或可直接通过url直接访问

##### 存在问题分析

在整合启动的过程中可能由于maven相关jar包没有正常导入，导致ureport依赖的相关jar包非正常导入，项目启动报错（删除maven仓库的内容，重新导入jar包）

## Ureport报表工具学习

报表设计器访问路径：http://localhost:8084/ureport/designer

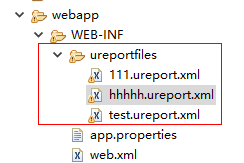
http://host[:port][/context-path]/ureport/designer

### 1.报表存储存储目录配置

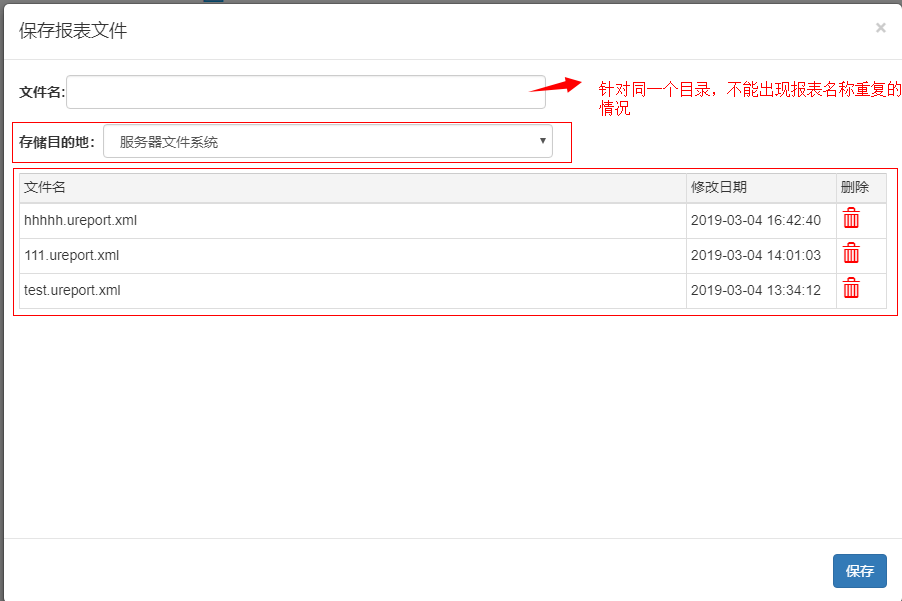
#### a.默认的报表存储目录配置

**默认情况下**：在项目的WEB-INF目录有会自动生成了一个名为“ureportfiles”目录，这个目录是UReport2提供的默认的报表文件存储目录，也就是说默认情况下，UReport将在项目的WEB-INF/ureportfiles目录下存储设计好的报表文件。

**默认报表存储目录** 如果项目在Eclipse的开发环境运行时，采用的是jetty(比如run-jetty-run插件)，那么就可以在项目的WEB-INF目录下发现一个名为“ureportfiles”目录。  
 但如果采用tomcat运行项目，那么在WEB-INF目录下就没有一个名为“ureportfiles”目录，原因是在Eclipse中运行tomcat，tomcat需要创建一个临时的工作目录(该目录一般位于workspace\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps\下)，所以采用tomcat运行项目，则需要到这个临时的工作目录下找到对应的项目，再到这个项目的WEB-INF目录下找到对应的“ureportfiles”目录



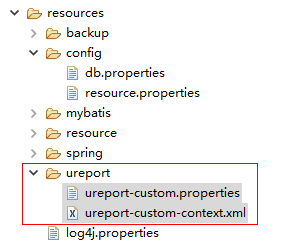
运行搭建好的项目，打开报表设计器，点击工具栏的保存按钮，弹出如下所示窗口：



完成数据保存，刷新工程目录则可看到对应生成的数据信息

#### b.自定义报表存储目录配置

在项目对应配置文件（参考整合过程，此处ureport独立的spring配置文件在resource/ureport/ureport-custom-context.xml），相应在其ureport目录下创建一个ureport-custom.properties配置文件存放有关ureport配置的基本属性

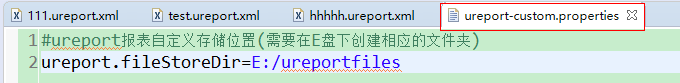




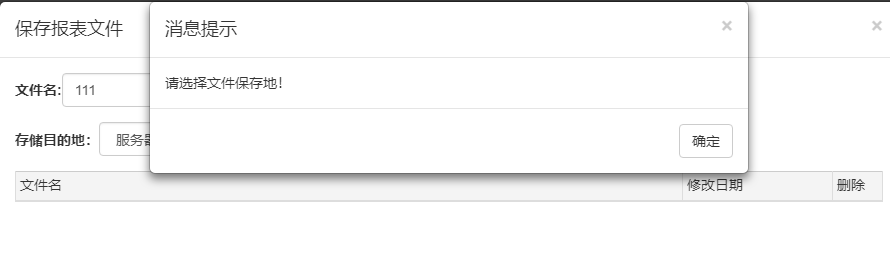
上面的Bean配置，实际上就是一个标准的

org.springframework.beans.factory.config.PropertyPlaceholderConfigurer类配置，通过配置PropertyPlaceholderConfigurer，可以实现加载Spring外部properties的作用，所以项目中已配置了PropertyPlaceholderConfigurer类，那么直接在这个类对应的properties文件中添加对应的UReport2属性即可，而不需要额外配置config.properties文件

完成文件加载配置，在相应的配置文件中配置属性即可：



表示在E盘根下名为ureportfiles的目录中存储报表文件，需要注意的是，这里指定特定目录时，一定要保证这个目录已存在，否则将不会被采用，比如上面的E盘下名为ureportfiles的目录，就需要预先创建好。（指定文件目录不存在，则提示如下）



#### c.自定义报表存储器

UReport2默认提供的名为“服务器文件系统”的报表存储机制，实际上是实现了UReport2提供的com.bstek.ureport.provider.report.ReportProvider接口，该接口源码如下：

**package** com.bstek.ureport.provider.report;

**import** java.io.InputStream;

**import** java.util.List;

/\*\*

\* **@author** Jacky.gao

\* **@since** 2016年12月4日

\*/

**public** **interface** ReportProvider {

/\*\*

\* 根据报表名加载报表文件

\* **@param** file 报表名称

\* **@return** 返回的InputStream

\*/

InputStream loadReport(String file);

/\*\*

\* 根据报表名，删除指定的报表文件

\* **@param** file 报表名称

\*/

**void** deleteReport(String file);

/\*\*

\* 获取所有的报表文件

\* **@return** 返回报表文件列表

\*/

List<ReportFile> getReportFiles();

/\*\*

\* 保存报表文件

\* **@param** file 报表名称

\* **@param** content 报表的XML内容

\*/

**void** saveReport(String file,String content);

/\*\*

\* **@return** 返回存储器名称

\*/

String getName();

/\*\*

\* **@return** 返回是否禁用

\*/

**boolean** disabled();

/\*\*

\* **@return** 返回报表文件名前缀

\*/

String getPrefix();

}

实现了ReportProvider接口后，只需要将实现类配置到Spring中，让其成为一个标准的Spring Bean，这样UReport2就会检测到它而将其加载。下面是UReport2默认提供的名为“服务器文件系统”的报表存储器源码：

**package** com.bstek.ureport.provider.report.file;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.util.ArrayList;

**import** java.util.Calendar;

**import** java.util.Collections;

**import** java.util.Comparator;

**import** java.util.List;

**import** javax.servlet.ServletContext;

**import** org.apache.commons.io.IOUtils;

**import** org.springframework.beans.BeansException;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.ApplicationContextAware;

**import** org.springframework.web.context.WebApplicationContext;

**import** com.bstek.ureport.exception.ReportException;

**import** com.bstek.ureport.provider.report.ReportFile;

**import** com.bstek.ureport.provider.report.ReportProvider;

/\*\*

\* **@author** Jacky.gao

\* **@since** 2017年2月11日

\*/

**public** **class** FileReportProvider **implements** ReportProvider,ApplicationContextAware{

**private** String prefix="file:";

**private** String fileStoreDir;

**private** **boolean** disabled;

@Override

**public** InputStream loadReport(String file) {

**if**(file.startsWith(prefix)){

file=file.substring(prefix.length(),file.length());

}

String fullPath=fileStoreDir+"/"+file;

**try** {

**return** **new** FileInputStream(fullPath);

} **catch** (FileNotFoundException e) {

**throw** **new** ReportException(e);

}

}

@Override

**public** **void** deleteReport(String file) {

**if**(file.startsWith(prefix)){

file=file.substring(prefix.length(),file.length());

}

String fullPath=fileStoreDir+"/"+file;

File f=**new** File(fullPath);

**if**(f.exists()){

f.delete();

}

}

@Override

**public** List<ReportFile> getReportFiles() {

File file=**new** File(fileStoreDir);

List<ReportFile> list=**new** ArrayList<ReportFile>();

**for**(File f:file.listFiles()){

Calendar calendar=Calendar.*getInstance*();

calendar.setTimeInMillis(f.lastModified());

list.add(**new** ReportFile(f.getName(),calendar.getTime()));

}

Collections.*sort*(list, **new** Comparator<ReportFile>(){

@Override

**public** **int** compare(ReportFile f1, ReportFile f2) {

**return** f2.getUpdateDate().compareTo(f1.getUpdateDate());

}

});

**return** list;

}

@Override

**public** String getName() {

**return** "服务器文件系统";

}

@Override

**public** **void** saveReport(String file,String content) {

**if**(file.startsWith(prefix)){

file=file.substring(prefix.length(),file.length());

}

String fullPath=fileStoreDir+"/"+file;

FileOutputStream outStream=**null**;

**try**{

outStream=**new** FileOutputStream(**new** File(fullPath));

IOUtils.*write*(content, outStream,"utf-8");

}**catch**(Exception ex){

**throw** **new** ReportException(ex);

}**finally**{

**if**(outStream!=**null**){

**try** {

outStream.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

@Override

**public** **boolean** disabled() {

**return** disabled;

}

**public** **void** setDisabled(**boolean** disabled) {

**this**.disabled = disabled;

}

**public** **void** setFileStoreDir(String fileStoreDir) {

**this**.fileStoreDir = fileStoreDir;

}

@Override

**public** **void** setApplicationContext(ApplicationContext applicationContext) **throws** BeansException {

File file=**new** File(fileStoreDir);

**if**(file.exists()){

**return**;

}

**if**(applicationContext **instanceof** WebApplicationContext){

WebApplicationContext context=(WebApplicationContext)applicationContext;

ServletContext servletContext=context.getServletContext();

String basePath=servletContext.getRealPath("/");

fileStoreDir=basePath+fileStoreDir;

file=**new** File(fileStoreDir);

**if**(!file.exists()){

file.mkdirs();

}

}

}

@Override

**public** String getPrefix() {

**return** prefix;

}

}

**禁用系统提供的默认报表存储器**  
  如果想使用自定义的报表存储器，同时又不想再使用系统默认提供的”服务器文件系统“的报表存储器，那么只需要在相应的报表配置文件(举例：config.properties)文件中添加一个名为ureport.disableFileProvider属性，将其值设置成true即可。

  通过实现ReportProvider接口，可以开发出其它类型的报表存储器，比如开发一个新的报表存储器将报表文件存储到数据库、FTP等。

参考链接：

存储报表至fastdfs：

https://blog.csdn.net/qq\_35170213/article/details/80290797

存储报表至数据库：

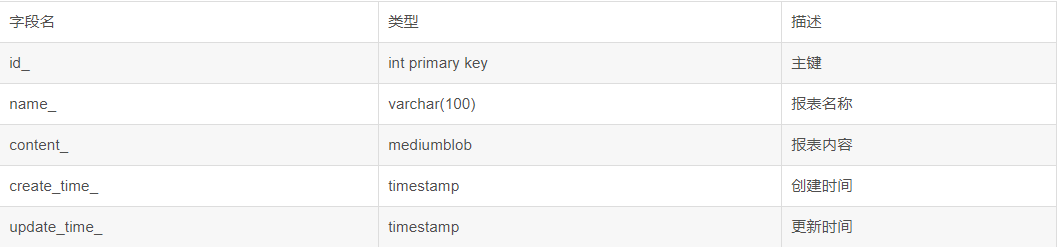
https://blog.csdn.net/qq\_35170213/article/details/80290425

##### 自定义报表存储器（数据库）

###### 创建存储报表信息的数据表

考虑实际需求，此处分别定义了CustomProvider、BackupProvider两个报表存储器，分别适用于用户自定义和平台备份，两者的数据表设计区别在于前者加入了报表所属机构限制（关联外键，用于表示该报表所属机构），而后者是属于平台备份数据数据库，为避免前者测试出现错误的情况而提供备用的平台报表存储器（公共）

基本设计：



/\*\* 建表 sql \*/

create table ureport\_file\_tbl (

id\_ int primary key auto\_increment,

name\_ varchar(100) not null,

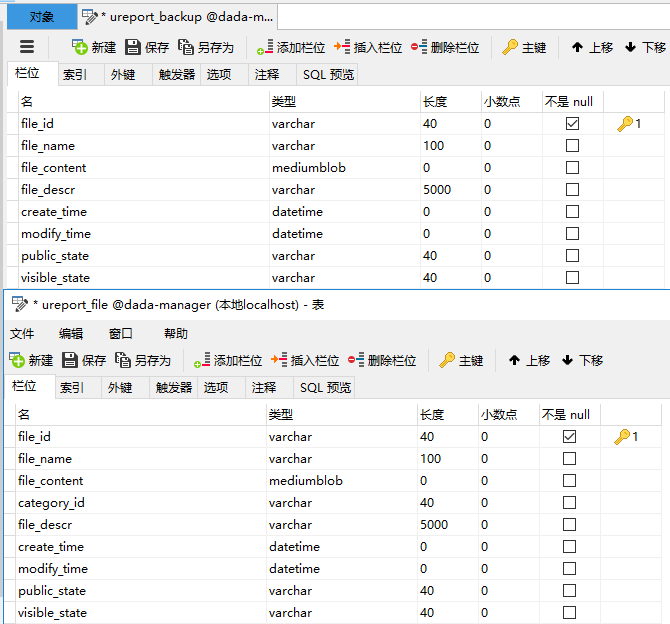
content\_ mediumblob,

create\_time\_ timestamp default now(),

update\_time\_ timestamp default now()

)

初步考虑数据表设计如下，具体的内容根据实际需求做调整



###### dao层/Mapper实现方法（基本的增删改查）

此处提供一个通用的接口模板，具体实现的逻辑则需要根据实际的业务需求去设计

例如整合报表平台的时候需要考虑实际报表权限的问题，因此需要考虑根据当前用户登录的权限，所属的机构等进行条件筛选

@Mapper

public interface UreportFileMapper {

/\*\*

\* 根据报表名称检查报表是否存在

\* @param name 报表名称

\* @return

\*/

public int checkExistByName(String name);

/\*\*

\* 根据报表名称查询报表

\* @param name 报表名称

\* @return

\*/

public UreportFileEntity queryUreportFileEntityByName(String name);

/\*\*

\* 查询全部报表

\* @return

\*/

public List<UreportFileEntity> queryReportFileList();

/\*\*

\* 根据报表名称删除报表

\* @param name

\* @return

\*/

public int deleteReportFileByName(String name);

/\*\*

\* 保存报表

\*/

public int insertReportFile(UreportFileEntity entity);

/\*\*

\* 更新报表

\* @param entity

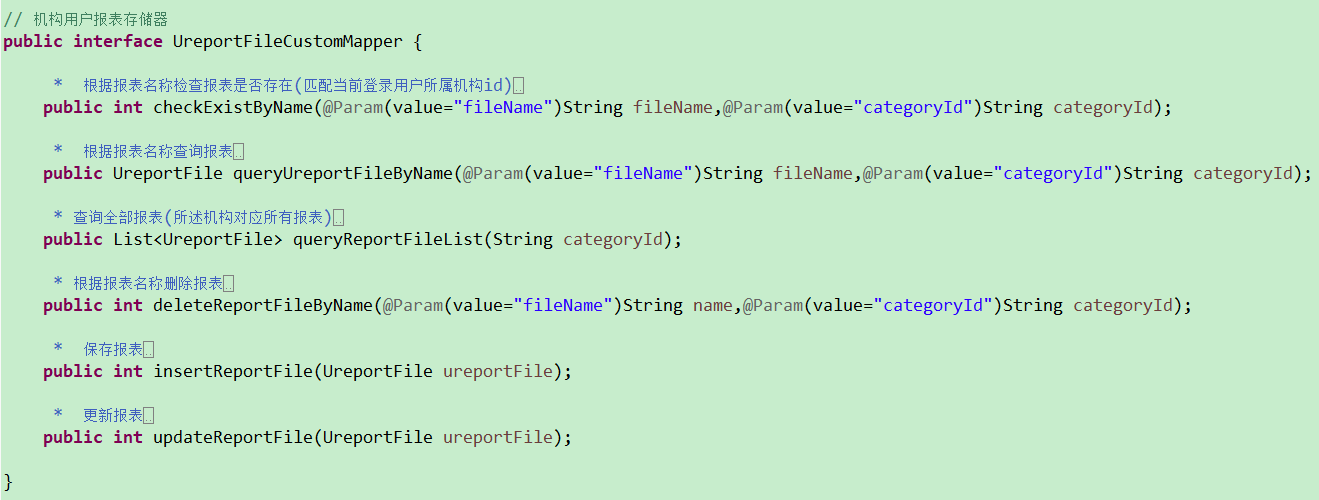
\* @return

\*/

public int updateReportFile(UreportFileEntity entity);

}





具体的mapper.xml文件则需要参考实际表设计完成相应配置，需要注意的是name、content这两个属性，这两个属性是报表设计的必要属性，以下提供基本模板，实际内容参考实际需求进行调整

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >

<mapper namespace=*"indi.qiaolin.test.ureport.mapper.UreportFileMapper"*>

<resultMap type=*"UreportFileEntity"* id=*"ureportFile"*>

<id column=*"id\_"* jdbcType=*"BIGINT"* property=*"id"*/>

<result column=*"name\_"* jdbcType=*"VARCHAR"* property=*"name"*/>

<result column=*"content\_"* jdbcType=*"BLOB"* property=*"content"*/>

<result column=*"create\_time\_"* jdbcType=*"TIMESTAMP"* property=*"createTime"*/>

<result column=*"update\_time\_"* jdbcType=*"TIMESTAMP"* property=*"updateTime"*/>

</resultMap>

<select id=*"checkExistByName"* parameterType=*"java.lang.String"* resultType=*"java.lang.Integer"*>

select count(\*) from ureport\_file\_tbl where name\_ = #{name}

</select>

<select id=*"queryReportFileList"* resultMap=*"ureportFile"*>

select \* from ureport\_file\_tbl

</select>

<select id=*"queryUreportFileEntityByName"* resultMap=*"ureportFile"*>

select \* from ureport\_file\_tbl where name\_ = #{name}

</select>

<delete id=*"deleteReportFileByName"* parameterType=*"java.lang.String"*>

delete from ureport\_file\_tbl where name\_ = #{name}

</delete>

<insert id=*"insertReportFile"* parameterType=*"UreportFileEntity"*>

insert into ureport\_file\_tbl (name\_, content\_, create\_time\_, update\_time\_) values

(#{name}, #{content}, #{createTime}, #{updateTime})

</insert>

<update id=*"updateReportFile"* parameterType=*"UreportFileEntity"*>

update ureport\_file\_tbl set

name\_ = #{name} ,

content\_ = #{content},

create\_time\_ = #{createTime},

update\_time\_ = #{updateTime}

where id\_ = #{id}

</update>

</mapper>

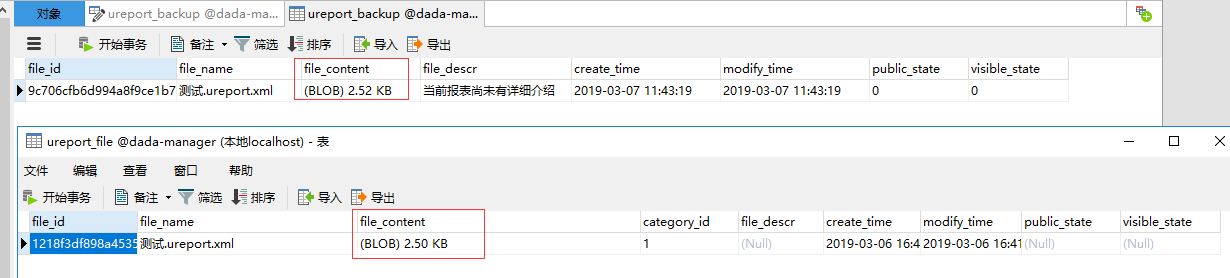
报表内容存储的类型：

Mysql数据库中表字段设置为‘mediumblob’类型

在mapper配置的数据类型为‘BLOB’、‘LONGARBINARY’



两种类型在数据库中存储的形式：



###### 报表存储器设计

基于springboot的参考

**import** java.io.ByteArrayInputStream;

**import** java.io.InputStream;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.context.properties.ConfigurationProperties;

**import** org.springframework.stereotype.Component;

**import** com.bstek.ureport.provider.report.ReportFile;

**import** com.bstek.ureport.provider.report.ReportProvider;

/\*\*

\* Mysql 报表存储器

\*/

@Setter

@Component

// 该注解可以利用其 prefix属性值 + 类的属性名 在yml中配置属性值

@ConfigurationProperties(prefix = "ureport.mysql.provider")

**public** **class** MySQLProvider **implements** ReportProvider{

**private** **static** **final** String NAME = "mysql-provider";

// 特定前缀，ureport底层会调用 getPrefix 方法来获取报表操作的Provier类

**private** String prefix = "mysql:";

        // 是否禁用

**private** **boolean** disabled;

@Autowired

**private** UreportFileMapper ureportFileMapper;

@Override

**public** InputStream loadReport(String file) {

UreportFileEntity ureportFileEntity = ureportFileMapper.queryUreportFileEntityByName(getCorrectName(file));

**byte**[] content = ureportFileEntity.getContent();

ByteArrayInputStream inputStream = **new** ByteArrayInputStream(content);

**return** inputStream;

}

@Override

**public** **void** deleteReport(String file) {

ureportFileMapper.deleteReportFileByName(getCorrectName(file));

}

@Override

**public** List<ReportFile> getReportFiles() {

List<UreportFileEntity> list = ureportFileMapper.queryReportFileList();

List<ReportFile> reportList = **new** ArrayList<>();

**for** (UreportFileEntity ureportFileEntity : list) {

reportList.add(**new** ReportFile(ureportFileEntity.getName(), ureportFileEntity.getUpdateTime()));

}

**return** reportList ;

}

@Override

**public** **void** saveReport(String file, String content) {

file = getCorrectName(file);

UreportFileEntity ureportFileEntity = ureportFileMapper.queryUreportFileEntityByName(file);

Date currentDate = **new** Date();

**if**(ureportFileEntity == **null**){

ureportFileEntity = **new** UreportFileEntity();

ureportFileEntity.setName(file);

ureportFileEntity.setContent(content.getBytes());

ureportFileEntity.setCreateTime(currentDate);

ureportFileEntity.setUpdateTime(currentDate);

ureportFileMapper.insertReportFile(ureportFileEntity);

}**else**{

ureportFileEntity.setContent(content.getBytes());

ureportFileEntity.setUpdateTime(currentDate);

ureportFileMapper.updateReportFile(ureportFileEntity);

}

}

@Override

**public** String getName() {

**return** NAME;

}

@Override

**public** **boolean** disabled() {

**return** disabled;

}

@Override

**public** String getPrefix() {

**return** prefix;

}

/\*\*

\* 获取没有前缀的文件名

\* **@param** name

\* **@return**

\*/

**private** String getCorrectName(String name){

**if**(name.startsWith(prefix)){

name = name.substring(prefix.length(), name.length());

}

**return** name;

}

}

平台备份存储器设计参考

**package** com.dada.report.provider;

**import** java.io.ByteArrayInputStream;

**import** java.io.InputStream;

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** com.bstek.ureport.provider.report.ReportFile;

**import** com.bstek.ureport.provider.report.ReportProvider;

**import** com.dada.common.constant.ConstantUtils;

**import** com.dada.common.utils.IDUtils;

**import** com.dada.manager.custom.mapper.UreportBackupCustomMapper;

**import** com.dada.manager.custom.mapper.UreportFileCustomMapper;

**import** com.dada.pojo.UreportBackup;

// 平台备用报表文件存储器

**public** **class** BackupProvider **implements** ReportProvider{

**private** **static** **final** String ***NAME*** = "哒哒公共报表存储器";

// 特定前缀，ureport底层会调用 getPrefix 方法来获取报表操作的Provier类

**private** String prefix = "dada-public:";

// 是否禁用

**private** **boolean** disabled;

@Autowired

**private** UreportBackupCustomMapper ureportBackupCustomMapper;

@Autowired

**private** UreportFileCustomMapper ureportFileCustomMapper;

@Override

**public** InputStream loadReport(String file) {

// 根据报表名称获取报表信息(dada公共仓库)

UreportBackup ureportBackup = ureportBackupCustomMapper.queryUreportBackupByName(getCorrectName(file));

**byte**[] content = ureportBackup.getFileContent();

ByteArrayInputStream inputStream = **new** ByteArrayInputStream(content);

**return** inputStream;

}

@Override

**public** **void** deleteReport(String file) {

ureportBackupCustomMapper.deleteReportBackupByName(getCorrectName(file));

}

@Override

**public** List<ReportFile> getReportFiles() {

List<UreportBackup> list = ureportBackupCustomMapper.queryReportBackupList();

List<ReportFile> reportList = **new** ArrayList<>();

**for** (UreportBackup ureportBackup : list) {

reportList.add(**new** ReportFile(ureportBackup.getFileName(), ureportBackup.getModifyTime()));

}

**return** reportList;

}

@Override

**public** **void** saveReport(String file, String content) {

file = getCorrectName(file);

UreportBackup ureportBackup = ureportBackupCustomMapper.queryUreportBackupByName(file);

Date currentDate = **new** Date();

**if**(ureportBackup == **null**){

ureportBackup = **new** UreportBackup();

// 设置默认属性

ureportBackup.setFileId(IDUtils.*genRandomUUId*());

ureportBackup.setFileDescr(ConstantUtils.***REPORT\_FILE\_DEFAULT\_DESCR***);

ureportBackup.setCreateTime(currentDate);

ureportBackup.setModifyTime(currentDate);

ureportBackup.setPublicState(ConstantUtils.***REPORT\_FILE\_PUBLIC\_STATUS\_PRIVATE***);

ureportBackup.setVisibleState(ConstantUtils.***REPORT\_FILE\_VISABLE\_STATUS\_FALSE***);

// 设置报表名称、报表内容

ureportBackup.setFileName(file);

ureportBackup.setFileContent(content.getBytes());

ureportBackupCustomMapper.insertReportBackup(ureportBackup);

}**else**{

ureportBackup.setFileContent(content.getBytes());

ureportBackup.setModifyTime(currentDate);

ureportBackupCustomMapper.updateReportBackup(ureportBackup);

}

}

@Override

**public** String getName() {

**return** ***NAME***;

}

@Override

**public** **boolean** disabled() {

**return** disabled;

}

@Override

**public** String getPrefix() {

**return** prefix;

}

/\*\*

\* 获取没有前缀的文件名

\* **@param** name

\* **@return**

\*/

**private** String getCorrectName(String name){

**if**(name.startsWith(prefix)){

name = name.substring(prefix.length(), name.length());

}

**return** name;

}

}

###### 测试问题分析

创建xxxProvider继承ureport的ReportProvider方法，完成报表存储相关的操作

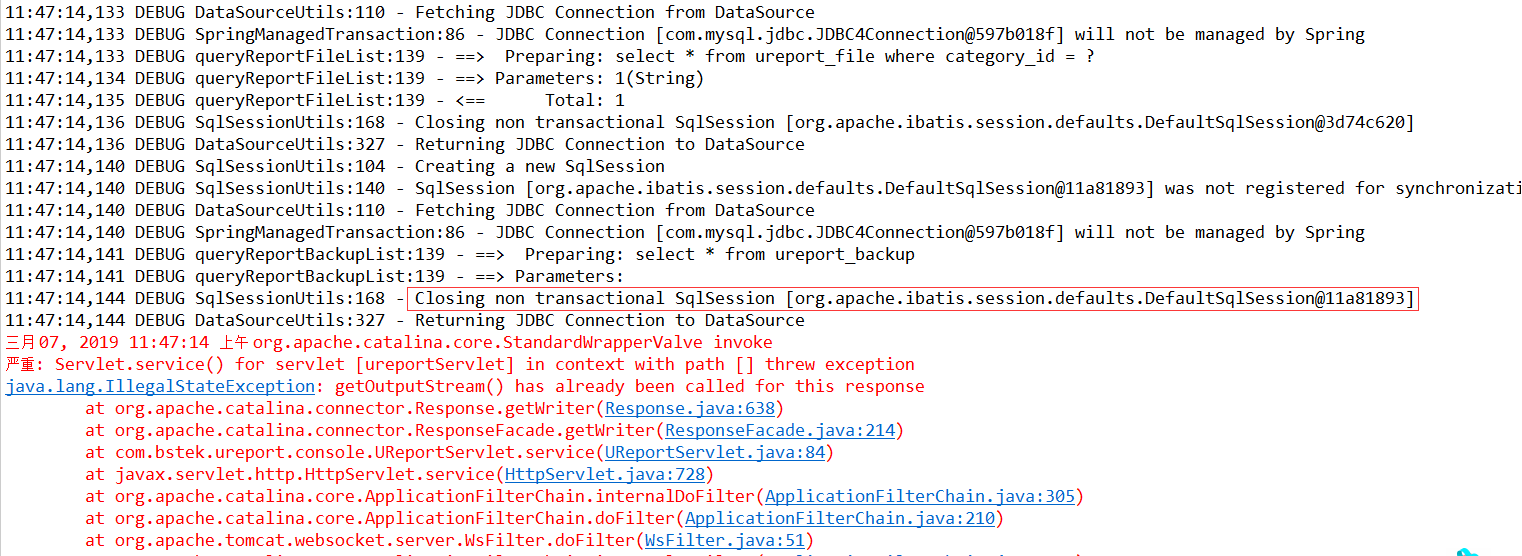
测试：数据能够正常保存，但是却无法正常访问

分析，在保存报表文件信息，或者是打开报表文件的时候请求接口，报错。

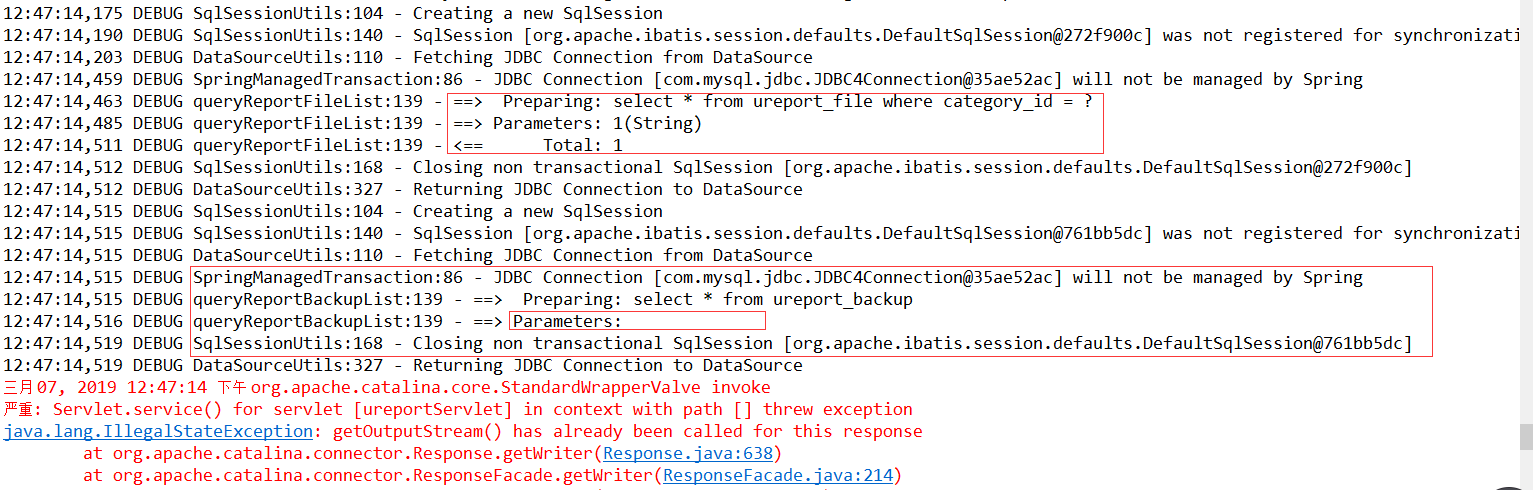
理想结果：自动封装相应存储的数据信息

实际结果：只封装了第一个定义的存储器，随后报如下错误

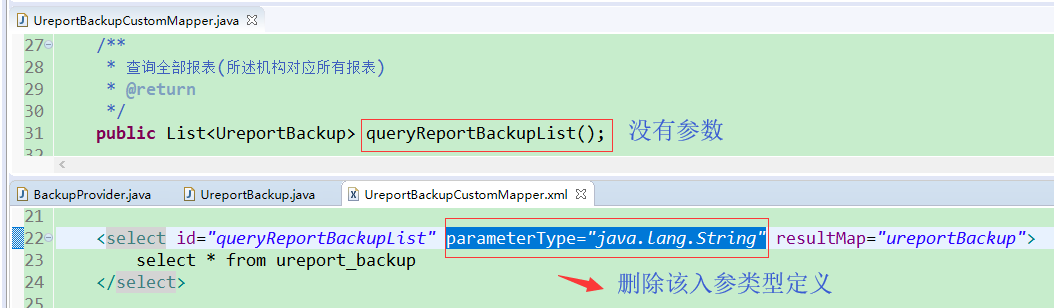
**（查看数据库mapper配置！！）**



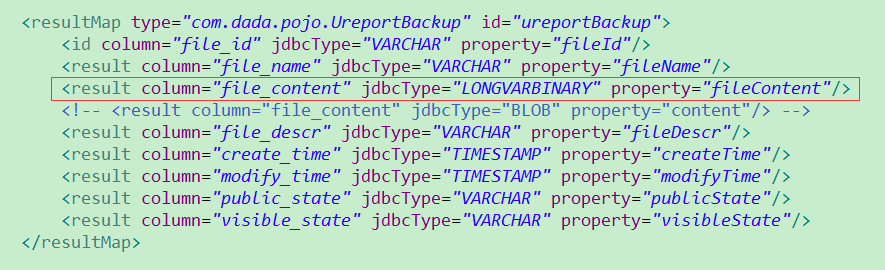
一开始考虑是多个报表存储器冲突导致的问题，之后测试无论是启用或禁用服务默认报表存储器还是出现同样的问题，数据库访问也没有看见明显的错误，随后查阅代码发现问题所在，由于一开始考虑机构报表存储器和平台备份报表存储器是类似的，直接复制粘贴代码，后期改动没有注意到一些细节性的问题，设置断点也无法找到明显的错误点（点击的时候加载就报错，无法通过设置断点查找出错问题），仔细查看报错信息，发现问题所在



可以看到保存信息显示两条sql语句查询的结果不同，前者正常查找数据，而后者查找数据到“Parameters：”（参数没有正常传递）则异常结束关闭数据库连接，并没有正常的返回结果。原因分析CustomProvide查找报表列表的时候是通过所属机构id查找的，而此处BackupProvider直接查找对应数据库下所有的信息不需要传递参数，在复制粘贴修改的过程中忽略了一些隐藏的bug，从而导致数据库查询语句没有正常执行导致报错，无关报表存储器个数限制问题、无关缓存问题，是个人疏忽导致出错（修改完成，更新maven仓库，重新测试）



再次测试，还是出现同样的问题，网上查阅输出流格式冲突，具体原因并不明确，考虑是报表内容存储格式问题，由mybatis逆向工程生成的mediumblob对应的是LONGVARBINARY，而网上参考是BLOB，修正后再次测试数据方能正常显示！！由此考虑报表内容存储格式导致出错！！（具体可以参考相关链接了解LONGVARBINARY与BLOB的区别）



以上描述的问题均为mapper配置文件出现问题导致，在配置mapper文件时需要注意mapper层接口和相应配置文件的名称、方法名意义对应，需要注意方法的定义（入参、出参），需要注意参数定义的类型（文本型字符串需要考虑数据定义的类型要相对应）

##### 自定以报表存储器（ftp-待定）

参考链接：https://blog.csdn.net/qq\_35170213/article/details/80290797

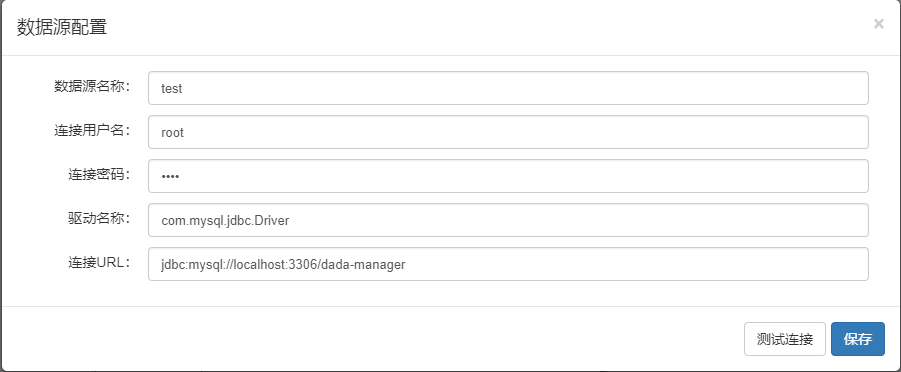
### 2.数据源配置

打开UReport2的报表设计器，可以看到UReport2提供了三种类型的报表数据源，如下图所示：三种类型的数据源分别是直接连接数据库，Spring Bean以及通过实现com.bstek.ureport.definition.datasource.BuildinDatasource接口提供的内置数据源。



#### 配置1：直连数据库

直接连接数据库比较简单，就是在项目的classpath中添加好相应数据库的驱动Jar包后，在弹出的窗口中配置数据源连接信息即可，如下图所示：**在页面用数据源配置链接数据库注意，不要刷新浏览器，刷新之后链接会清空**



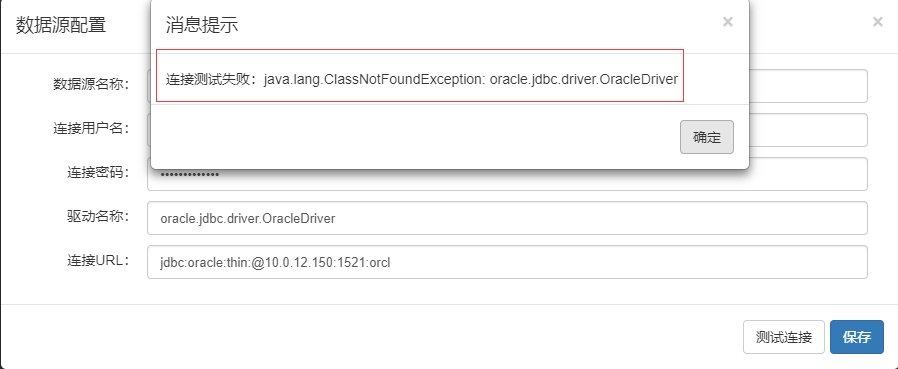
点击测试连接，可以看到对应目录下生成了相应的数据源，右键选择需要操作的数据源可进行“添加数据集、编辑、删除操作”



根据相应的数据集配置相关的数据：

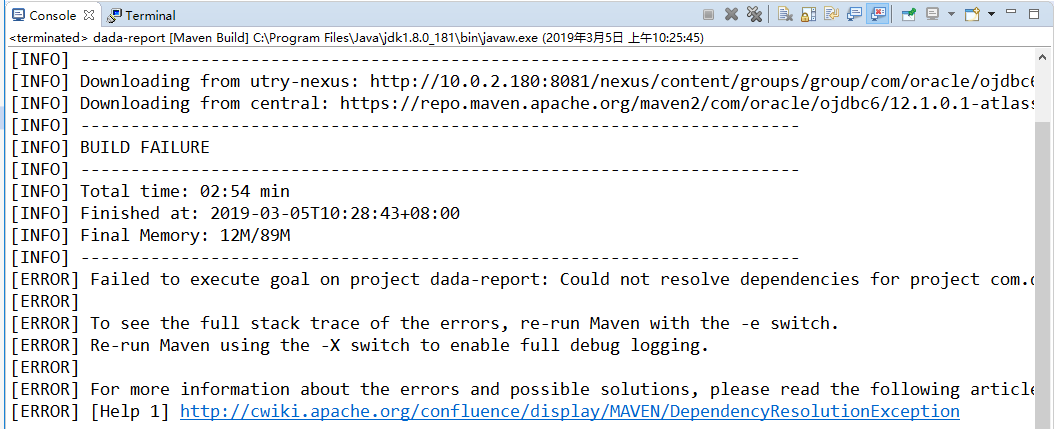


如果连接测试失败：查看失败原因，依次进行查找



##### Maven引入Oracle驱动依赖

（在pom.xml文件中添加oracle驱动maven依赖存在一定的问题，可参考链接了解相关内容https://blog.csdn.net/erlian1992/article/details/74279106）



由于Oracle数据库JDBC驱动包的使用的JDK版本有所区别，一定要下载对应的JDK版本的Oracle数据库JDBC驱动，JDK1.5的选择ojdbc.jar驱动包，如果是JDK1.6及以上版本的选择ojdbc6.jar驱动包，如果JDK环境是JDK1.8，那么选择ojdbc6.jar驱动包。

方式1：可以通过在官网下载相应的jar包：

<https://www.oracle.com/technetwork/database/application-development/jdbc/downloads/index.html>

方式2：如果本地安装了oracle数据库，则可在指定目录查找到相应的ojdbc6.jar(其路径为对应安装目录）E:\Oracle\app\Administrator\product\11.2.0\dbhome\_1\jdbc\lib\ojdbc6.jar)

接下来就是手动向Maven的本地仓库中添加Oracle数据库的JDBC驱动依赖。安装的前提是你的电脑必须安装了Maven，并且配置了Maven环境变量（参考maven配置）通过cmd窗口（或Eclipse的Maven插件、IDEA的Terminal）执行命令（虽然使用的工具不同，但是指定的文件位置和Maven坐标都是一样的）

需要注意的是如果是使用方式2，需要将相关的ojdb6.jar包放到其他文件夹中，不要在oracle安装目录下直接执行指令，可能会出现Oracle拒绝访问的情况

此处用eclipse的maven插件实现（Terminal）

$ mvn install:install -file -Dfile=F:\jar\ojdbc6.jar -DgroupId=com.oralce -DartifactId=ojdbc6 -Dversion=11.2.0.1.0 -Dpackaging=jar（根据实际调整路径）

进入到指定的目录下直接执行：

$ mvn install:install-file -Dfile=ojdbc6.jar -DgroupId=com.oralce -DartifactId=ojdbc6 -Dversion=11.2.0.1.0 -Dpackaging=jar



执行完成之后可以看到对应的mvn安装仓库有相应的资源生成：



完成jar导入到本地仓库之后，则可在项目中添加使用oracle相关依赖（添加的依赖版本需要与指定的数据（导入到maven本地仓库时的依赖定义）一致）

<!-- 添加oracle驱动依赖 -->

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.1.0</version>

</dependency>

##### 待测试

#### 配置2：SpringBean数据源

Spring Bean类型的数据源可以选择Spring上下文中定义好的一个Bean来作为数据源，点击IMG_256图标，在弹出的窗口中输入数据源名称及要采用的Bean的ID，如下图所示：

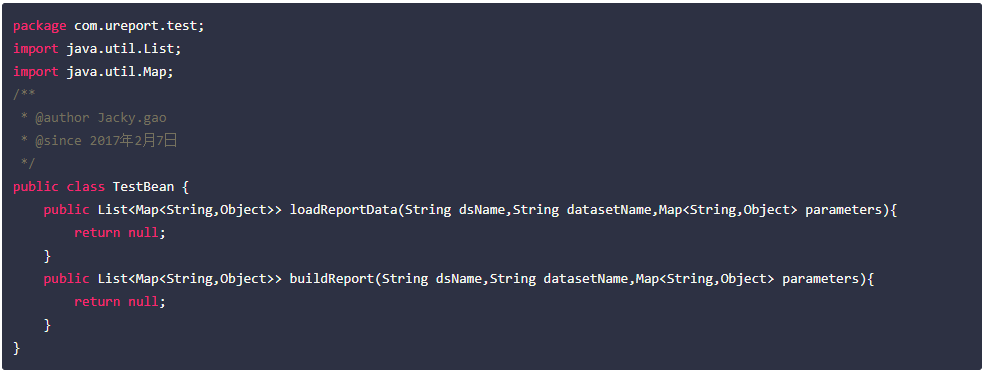


保存后，就可以在这个数据源下添加具体的数据集，添加方法就是在这个数据源下右键，在弹出的菜单中选择添加数据集，在弹出的窗口中定义数据集名称、对应的方法名以及返回对象类型，如下图所示：

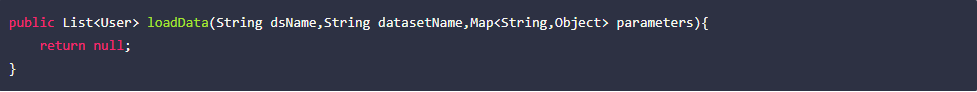


（如果springBean中没有指定id的bean兑现个，此处在添加数据集，选择方法的时候相应报错-空指针异常，指定的bean对象不存在）

在Spring bean数据集配置中，方法名我们可以点击右侧的“选择方法”按钮来选择当前Bean对应的类中定义的方法，但这里对方法的要求是：方法必须要有三个参数，依次是String,String,Map，比如我们上面定义的testBean里就包含两个合法的方法，如下所示：

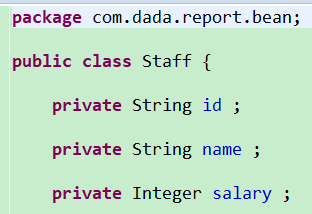


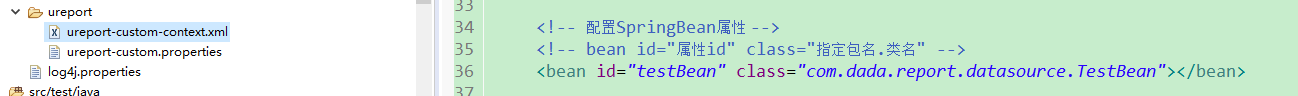
  对于一个合法的Bean数据集方法要有三个参数，分别是String,String,Map，依次对应数据源名称、数据集名称以及外部传入的参数Map，Bean的方法只有是这种结构才可以进行选择。对于数据集方法的返回值，目前来说可以支持两种类型，一种是我们TestBean中返回的Map<String,Object>类型的List集合；另一种就是返回一个POJO类型的List集合，比如像下面的方法：  在上面的示例方法中，返回的就是User对象集合，这里的User对象，就是一个普通的POJO对象。（提供示例可以通过SpringBean自定义相关数据实现）



参考上述介绍，设置了如下数据配置：

在dada-report下创建com.dada.report.datasource包，创建相应的xxxBean类，并在指定的ureport配置文件中配置bean对象（或者是直接通过@Component注解进行定义）





**package** com.dada.report.datasource;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

**import** org.apache.commons.lang.RandomStringUtils;

**import** org.apache.commons.lang.math.RandomUtils;

**import** com.dada.common.utils.IDUtils;

**import** com.dada.report.bean.Staff;

**public** **class** TestBean {

**public** List<Map<String,Object>> loadReportData(String dsName,String datasetName,Map<String,Object> parameters){

List<Map<String,Object>> list=**new** ArrayList<Map<String,Object>>();

**for**(**int** i=0;i<10000;i++){

Map<String,Object> m=**new** HashMap<String,Object> ();

m.put("id", i);

m.put("name",RandomStringUtils.*random*(10,**true**,**false**));

m.put("salray",RandomUtils.*nextInt*(10000)+i );

list.add(m);

}

**return** list;

}

// Staff.java类(简单测试类实现)

**public** List<Staff> builderReport(String dsName,String datasetName,Map<String,Object> parameters){

List<Staff> list=**new** ArrayList<Staff>();

**for**(**int** i=0;i<10000;i++){

Staff staff = **new** Staff();

staff.setId(IDUtils.*genRandomUUId*());

staff.setName(RandomStringUtils.*random*(10,**true**,**false**));

staff.setSalary(RandomUtils.*nextInt*(10000)+i);

list.add(staff);

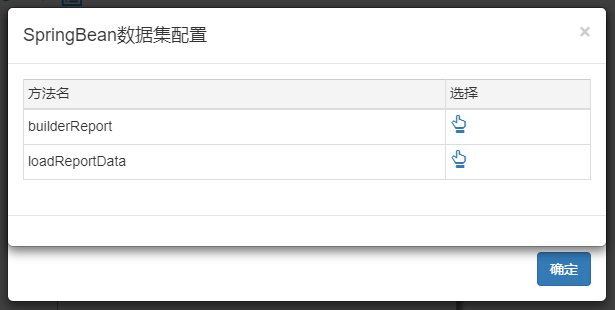
}

**return** list;

}

}

配置完成，启动项目，再次填写数据并点击保存按钮，能够看到相应的SpringBean数据配置完成，相应的会返回相关的方法

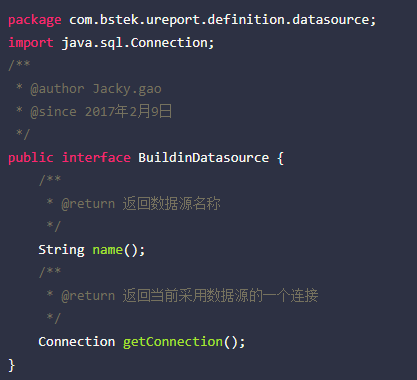


选择相应数据源配置，返回相应为数据类型（**如果不配置则需要手动添加字段**）



#### 配置3：内置数据源

内置数据源要求实现BuildinDatasource接口，同时将BuildinDatasource接口实现类配置到Spring即可，BuildinDatasource接口源码如下：



此处测试参考代码如下：

##### 方式1：借助JDBCUtils工具类（自定义）获取数据源

**package** com.dada.report.datasource;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** javax.sql.DataSource;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** com.bstek.ureport.definition.datasource.BuildinDatasource;

**import** com.dada.report.utils.JDBCUtils;

**public** **class** TestDataSource **implements** BuildinDatasource {

**private** **static** **final** String ***NAME*** = "平台测试内置数据源";

**private** Logger log = LoggerFactory.*getLogger*(TestDataSource.**class**);

//@Autowired

//private DataSource dataSource;

/\*\*

\* 获取数据源连接

\*/

@Override

**public** Connection getConnection() {

// 自定义创建数据源连接(借助JDBCUtils获取数据源)

**return** JDBCUtils.*getConnection*();

/\*

try {

// 通过spring配置的dataSource对象获取数据库连接

return dataSource.getConnection();

} catch (SQLException e) {

log.error("Ureport 数据源 获取连接失败！");

e.printStackTrace();

}

return null;

\*/

}

/\*\*

\* 获取数据源名称

\*/

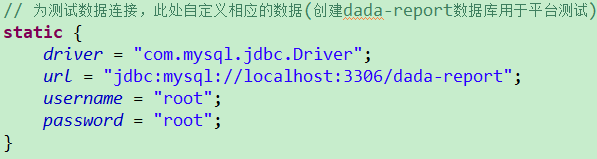
@Override

**public** String name() {

**return** ***NAME***;

}

}



在ureport配置文件中配置dataSource对象：



##### 方式2：通过spring管理DataSource数据源（以注解方式获取数据源连接）

**package** com.dada.report.datasource;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** javax.annotation.Resource;

**import** javax.sql.DataSource;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** com.bstek.ureport.definition.datasource.BuildinDatasource;

**public** **class** TestBackupDataSource **implements** BuildinDatasource {

**private** **static** **final** String ***NAME*** = "平台测试内置数据源(备份)";

**private** Logger log = LoggerFactory.*getLogger*(TestDataSource.**class**);

// @Autowired(byType:配置多个相同数据源连接采用该注解会报错)

@Resource(name="backupDataSource")

**private** DataSource dataSource;

/\*\*

\* 获取数据源连接

\*/

@Override

**public** Connection getConnection() {

// 自定义创建数据源连接(借助JDBCUtils获取数据源)

// return JDBCUtils.getConnection();

**try** {

// 通过spring配置的dataSource对象获取数据库连接

**return** dataSource.getConnection();

} **catch** (SQLException e) {

log.error("Ureport 数据源 获取连接失败！");

e.printStackTrace();

}

**return** **null**;

}

/\*\*

\* 获取数据源名称

\*/

@Override

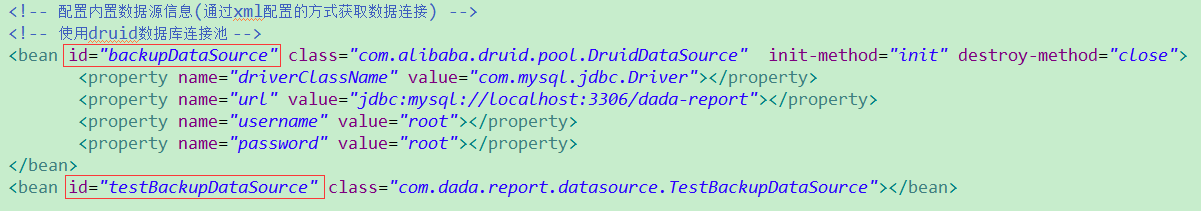
**public** String name() {

**return** ***NAME***;

}

}

在ureport配置文件中配置dataSource对象：



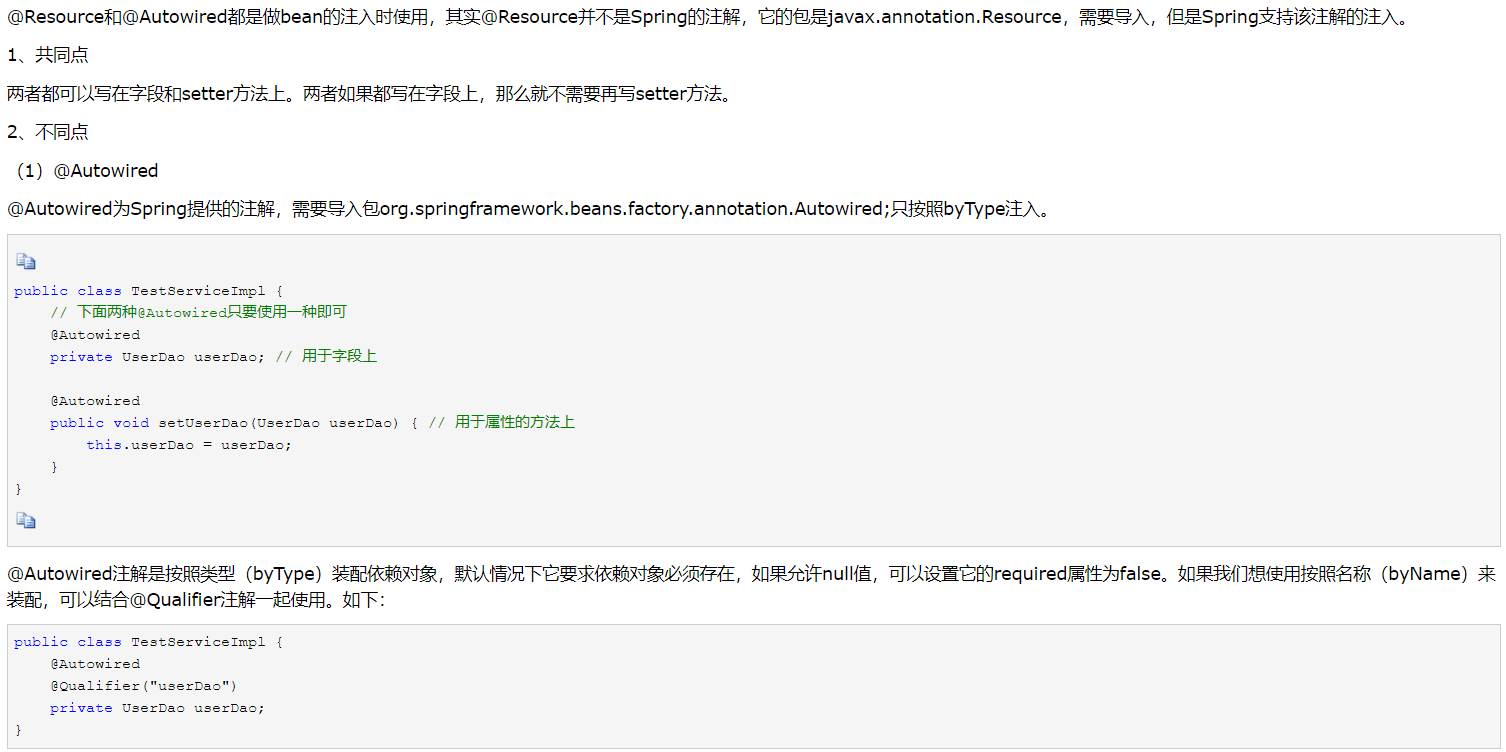
##### 多数据源配置问题：

在通过xml配置文件配置的时候，使用@Autowired注解，其按住奥byType自动植入，但如果文中配置两个相同类型的dataSource，则在项目启动加载spring自动注入的时候会报错

org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type [javax.sql.DataSource] is defined: expected single matching bean but found 2: dataSource,backupDataSource

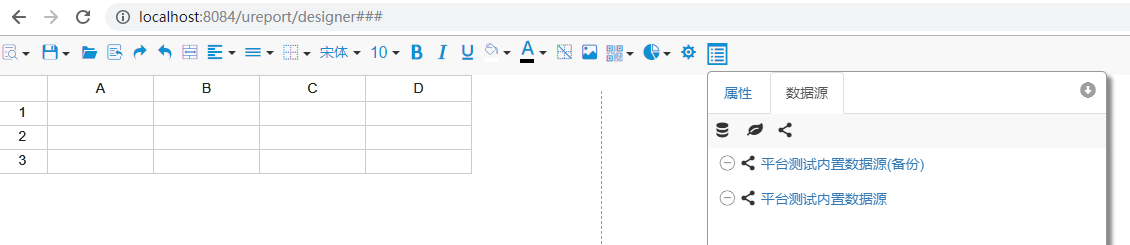
因此在配置注解的时候需要明确注入的是哪个对象（可以参考spring注解说明，了解每个注解之间的异同，参考链接：https://www.cnblogs.com/think-in-java/p/5474740.html）

@Resource的作用相当于@Autowired，只不过@Autowired按照byType自动注入





BuildinDatasource接口实现类配置到Spring中后，UReport2会自动检测到，完成配置启动测试，在报表设计器中，点击数据源页签中的IMG_256按钮，在弹出的窗口中就可以选择定义好的内置数据源，如下图所示：



查看配置是否正确，可通过右键选择添加数据集，如果数据正常显示则连接正常，但如果出现“数据源不存在”或“后台空指针异常”则说明数据源配置存在问题，需要进行修改调整



#### 数据源配置方法说明

在数据源中有三个选择：

服务器启动期间数据源配置是有效，当服务器重启后数据源配置并不保存，除却在spring配置完成的springBean、内置数据源数据是在服务器启动的时候便加载，直接连接数据源信息不会保存，三种不同的数据源连接方式创建数据保存到指定的xml，重启服务器后再次加载该模板数据也不会丢失（基本的配置在xml文件中已做存储）

数据源直接连接：在报表数据多的情况下如果环境替换则难以维护

spring bean数据源：需要指定spring 中 datasource的bean Id，其会在IOC容器中获取！

内置数据源：在spring配置该数据源对象，即需要创建一个类，其实现

com.bstek.ureport.definition.datasource.BuildinDatasource接口，封装数据源信息

### 3.报表的基本使用

### 4.报表设计器与业务结合

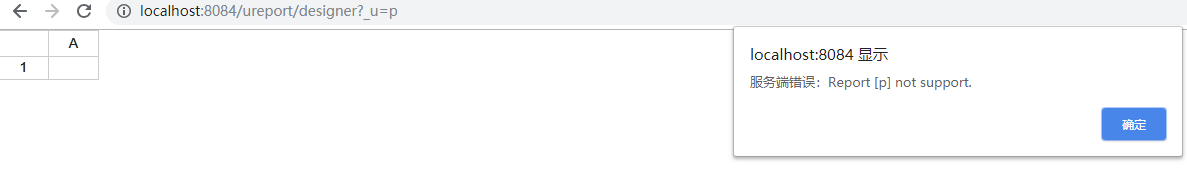
#### 报表相关url介绍

报表设计器：http://host[:port][/context-path]/ureport/designer

报表预览：http://host[:port][/context-path]/ureport/preview?\_u=报表名称

**预览正在设计中的报表注意事项**：在预览正在设计器中设计的报表时（也就是\_u=p），UReport2会在点击设计器工具栏IMG_256图标时，将设计中的报表模版信息提交到服务器，存放于当前用户的session中（用户会话中），所以一旦用户session失效，如果我们再次直接预览\_u=p类型报表时，会看到“com.bstek.ureport.console.exception.ReportDesignException: Report data has expired,can not do preview.”这样的错误提示，表示存放于session中的正在设计器中设计的报表模版已经失效，不能预览。但如果\_u的值是一个具体的报表名称，则不存在这种问题，只有正在设计器中设计的报表模版在预览中采用了这种机制，这点需要注意。

如果预览的不是正在设计器中设计的模版，那么只需要给出具体的报表名称即可，需要注意的是，这里的报表名称要以其ReportProvider中要求的前缀开始，比如http://localhost:8080/ureport2-demo/ureport/preview?\_u=file:test.ureport.xml等。





**在线打印：**在UReport2的报表预览页面中，可以看到UReport2提供了三种在线打印方式，分别是直接HTML打印、直接PDF打印以及PDF在线预览打印。这其中直接HTML打印是利用浏览器的HTML打印功能实现，能用于普通的A4纸类型报表页面的打印，同时它不能打印出报表中定义的页眉页脚，如果有定义的话。对于直接PDF打印以及PDF在线预览打印是服务端向浏览器中写入PDF流，利用Chrome、Firefox、Edge这些浏览器可以在线显示PDF功能实现的打印，这种是直接打印的PDF，所以可以应付各种复杂报表纸张类型的打印输出。

**Firefox预览PDF：**在预览PDF时，Firefox不允许当前frame外通过javascript调用其打印功能（目前来看可能是Firefox的Bug），所以我们需要手工点击其frame内部自带的打印按钮实现打印。

#### 在代码中使用报表

在UReport2当中，我们可以使用ExportManager实现在业务代码中导出各种类型的报表，ExportManager接口源码如下：

**package** com.bstek.ureport.export;

**import** java.util.Map;

**import** com.bstek.ureport.export.html.HtmlReport;

/\*\*

\* **@author** Jacky.gao

\* **@since** 2016年12月4日

\*/

**public** **interface** ExportManager {

**public** **static** **final** String ***BEAN\_ID***="ureport.exportManager";

/\*\*

\* 导出Html报表

\* **@param** file 报表模版文件名

\* **@param** contextPath 当前项目的context path

\* **@param** parameters 参数

\* **@return** 返回一个HtmlReport对象，里面有报表产生的HTML及相关CSS

\*/

HtmlReport exportHtml(String file,String contextPath,Map<String, Object> parameters);

/\*\*

\* 导出指定页码的Html报表

\* **@param** file 报表模版文件名

\* **@param** contextPath 当前项目的context path

\* **@param** parameters 参数

\* **@param** pageIndex 页码

\* **@return** 返回一个HtmlReport对象，里面有报表产生的HTML及相关CSS

\*/

HtmlReport exportHtml(String file,String contextPath,Map<String, Object> parameters,**int** pageIndex);

/\*\*

\* 导出PDF报表

\* **@param** config 包含报表模版文件名、参数等信息的配置对象

\*/

**void** exportPdf(ExportConfigure config);

/\*\*

\* 不分页导出Excel

\* **@param** config 包含报表模版文件名、参数等信息的配置对象

\*/

**void** exportExcel(ExportConfigure config);

/\*\*

\* 分页导出Excel

\* **@param** config 包含报表模版文件名、参数等信息的配置对象

\*/

**void** exportExcelWithPaging(ExportConfigure config);

/\*\*

\* 分页分Sheet导出Excel

\* **@param** config 包含报表模版文件名、参数等信息的配置对象

\*/

**void** exportExcelWithPagingSheet(ExportConfigure config);

/\*\*

\* 导出Word

\* **@param** config 包含报表模版文件名、参数等信息的配置对象

\*/

**void** exportWord(ExportConfigure config);

}

因为ExportManager接口实现是配置在Spring当中，所以要使用ExportManager接口，我们需要首先通过Spring的ApplicationContext取到ExportManager实例对象，我们可以通过ExportManager.BEAN\_ID来取到对应的Bean实例。

以下测试如何采用ExportManager导出HTML报表，并将报表内容嵌入到一个JSP中。 首先需要采用报表设计器设计好一个报表模版文件，将其保存，比如报表模版保存后的文件名为demo.ureport.xml，然后在项目中创建一个JSP（这里之所以选择JSP，是因为它最为简单，可以保证所有的J2EE开发者都能看懂，实际使用时可能是MVC框架、Ajax等，但如果了解了在JSP中用法，其它的就简单了），在JSP中输入相应代码，导出Html报表，并将其写入到JSP中，完整的JSP源码如下：



预览：http://localhost:8084/ureport/preview?\_u=file:hhhhh.ureport.xml

在SSM配置基础上添加相应的jsp文件，完成配置启动服务器访问测试：

<http://localhost:8084/custom/report/page/other?url=report/testReport>



#### 考虑实际业务整合问题：

##### 数据源配置问题

获取的数据源配置来源：

**方式1：**可根据模板保存需要的数据源信息（数据源配置方式沿用urport的三种数据源配置方式，分别为直接连接数据源、springbean数据源、内置数据源）

直接连接数据源：用户可以通过连接配置本地数据源完成配置

Springbean数据源：由相应的spring配置文件中确定数据来源（在源代码中限定了数据来源）

内置数据源：由平台提供数据库，相应的数据都是来源于平台数据库（供用户测试使用）

**方式2：**是否可以考虑通过从数据库中获取相应用户配置的数据源信息，然后整合到报表设计器中。类似自定义报表存储器配置，在相应的数据源实现类中查找当前用户默认启用的数据库配置，随后通过spring配置文件在加载的时候便会将该数据库连接默认配置到相应的位置（待验证）

解决方案：

参考两种数据源配置方式，考虑到两者的优缺点结合，前者是数据源来源过于局限，后者则是数据源配置智能访问到用户自身默认的数据源信息（无法直接加载数据源列表，除非考虑平台加限制，设置“首选数据源、备用数据源”概念），因此此处考虑提供两种解决方案，贴合用户的需求

方案1：采用方式1，在数据库中添加“数据源配置文件”（相当于新增一个文件存储器），用于存储用户数据源配置信息，实现用户数据源配置的多样性（但如果用户数据源发生变动则需要相应调整，无法实现动态更新）,为预防测试出现问题，在内置数据源配置中配置了两个平台测试的数据源用以备用

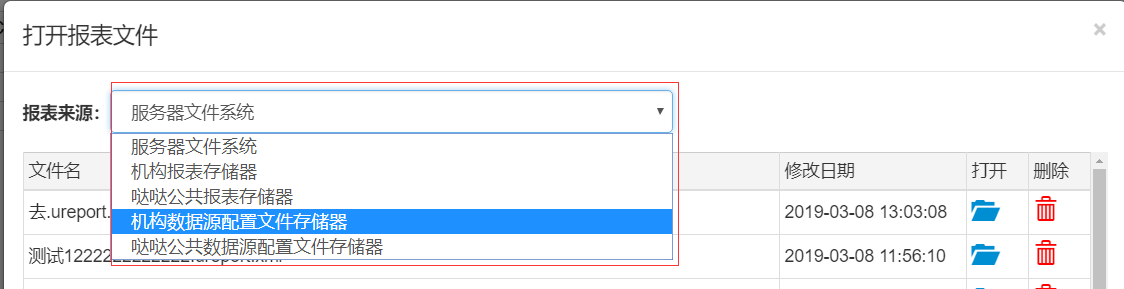
方案2：采用方式2，提供“数据源配置”概念，用户可以根据自身需求完成数据源配置，选定“首选数据源、备用数据源”，随后在加载报表设计器的时候则根据当前登录的用户信息获取到相应配置的数据源，加载用户自身的数据源信息

两种方案结合使用，供用户选择

##### 方案实施

###### 新增“数据源配置文件”存储器

相当于为数据源配置文件额外开辟一个存储空间，用于存储相应的数据源配置文件，此处可参考“自定义文件存储器”设计和实现完成，实现提供了两个数据源存储器，分别为机构适用、平台备用



###### 新增“数据源配置信息管理”概念

现存问题：数据源配置数据表如何设计（实际需求是什么？）

用户报表存储管理到机构层面（机构管理人员-涉及数据库操作）：

数据库操作考虑是给报表的维护人员使用，

而对外预览供普通的用户使用

提供服務：

报表文件和数据源文件的管理要分开！！！

数据源配置？

针对的对象是谁？是这个机构？还是个人？

报表权限则根据实际的分类进行相应的权限管理

文件保存路径问题：

数据存储器配置：自定义报表存储器

如何根据不同的机构进行分类获取相应的服务器路径

考虑利用fastdfs（待定）

自定义报表存储器：

将报表存储到数据库中，考虑如何进行分类处理：

处理器是定死的，，，如果要划分报表分类，则根据报表分类区分报表存储仓库的信息

同一类的报表方能???

前端业务控制报表设计器入口,针对普通用户，只提供报表预览的入口，

机构根据实际需求分配角色权限，提供报表设计功能

管理员管理报表设置报表属性可见、隐藏

平台用户访问报表数据

根据不同的分类显示报表数据：

报表设计器入口

分类（所有报表、当前分类报表）

用户密码加密解密通过工具类实现

调用服务？？？？

当用户操作的时候使用拦截器修改默认保存的名称，，解析的时候拦截请求再进行解析？

数据源配置：

（如何自动加载用户已保存的数据源信息）

可自定义提供用户报表数据信息

Designer相关url分析：

文件保存至服务器路径：

<http://localhost:8084/ureport/designer/saveReportFile>

自定义报表存储器：

能否 根据当前登录用户自定义报表存储的仓库信息：

报表存储的时候：

报表返回的时候：

拦截器：

拦截、设置查找参数属性：

报表添加之后：保存数据到指定的数据库（添加报表记录-考虑当前登录用户所属机构）

加载报表之前，拦截报表数据

拦截器？

在不影响基本功能实现的 基础上完成报表设计：

额外扩展功能：

数据保存：

添加按钮

通过ThreadLocal保存登录用户信息，从而在dao层即可调用登录用户数据

#### 用户登录信息保存在ThreadLocal中

ThreadLocal<Session> 跨工程调用问题：