RestCloud

二次开发手册

谷云科技(广州)有限责任公司

最后更新:2020.10

目 录

1. 引言

1.1 编写目的

本手册是 “RestCloud API 开发平台”开发人员的开发指南，针对本系统业务而定制开发的 API 的使用方法、技巧和一些相关的注意事项进行介绍，帮助用户更好的掌握 API 的开发特性。

2. 开发环境

2.1 硬件环境

IT 服务管理系统对用户的运行环境,推荐的 PC 配置如下：
CPU：Intel 2.4 G 或以上
内存：4G 或以上
空闲硬盘空间：50G 或以上

2.2 软件环境

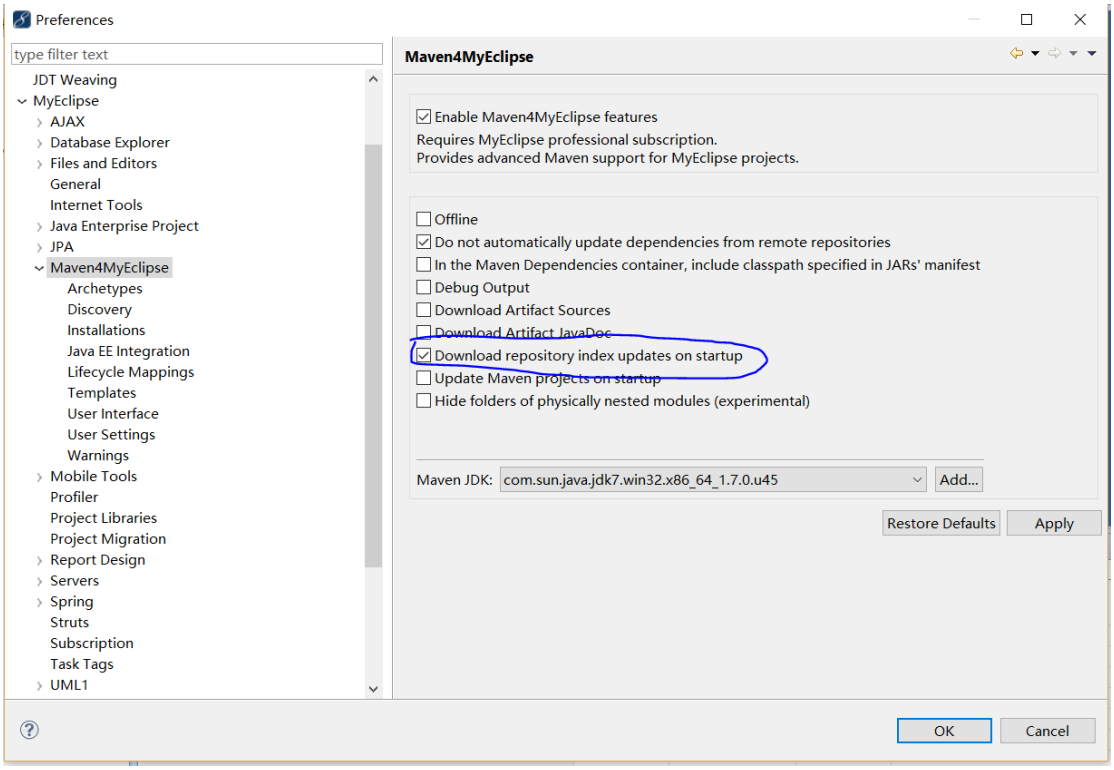
开发 IDE：Eclipse/MyEclipse/IDEA
操作系统：Windows10/Windows7/Windows2012
浏览器：chorme
数据库: MongoDB3.2 或更高
Web 服务器：Tomcat8.5.3
Springboot:2.2.0
JDK:1.8 以上版本

3. Maven 工程导入

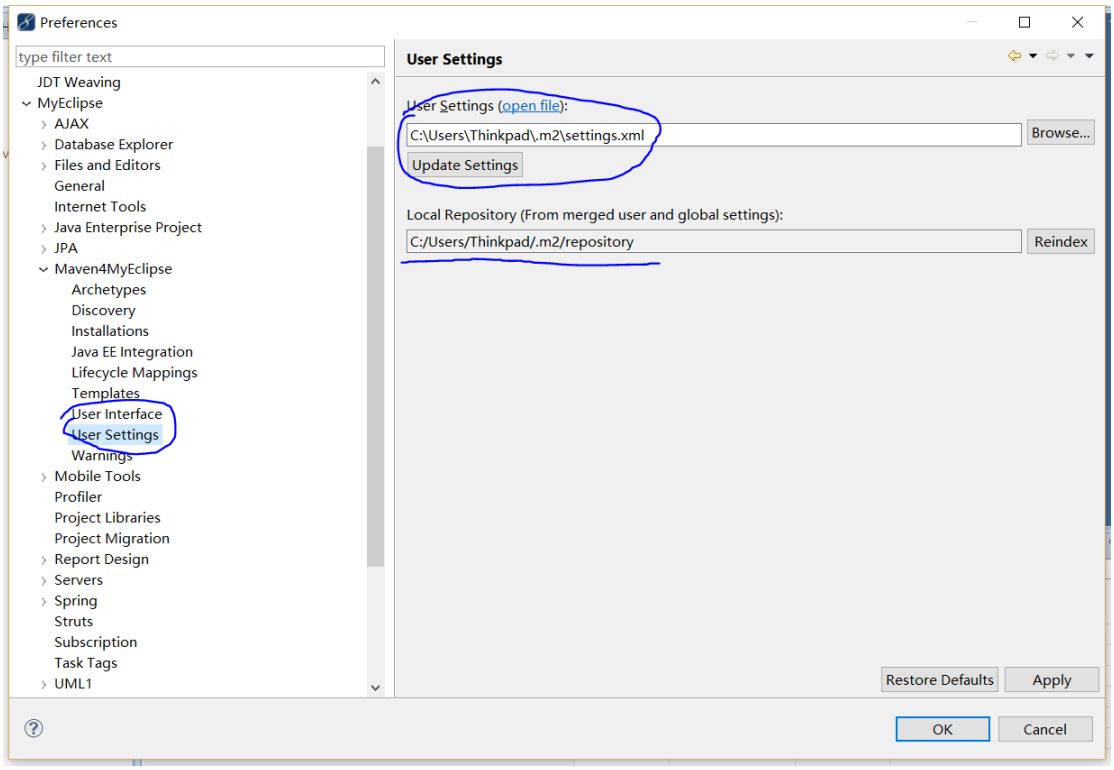
这里以 MyEclipse（Eclipse 则需要先安装 Maven 插件）为例导入说明几个注意事项：

1. Maven 需要去掉 index updates 选项如下图：

Window => Preferences => Myeclipse Enterprise Workbench => Maven4Myeclipse => Maven=>禁用 Download repository index updates on startup



（去掉 updates 选项）



（首先要指定自定义的 settings.xml 文件，然后指定本地仓库地址）

Settings.xml 主要修改以下两处：

指定远程仓库如下：

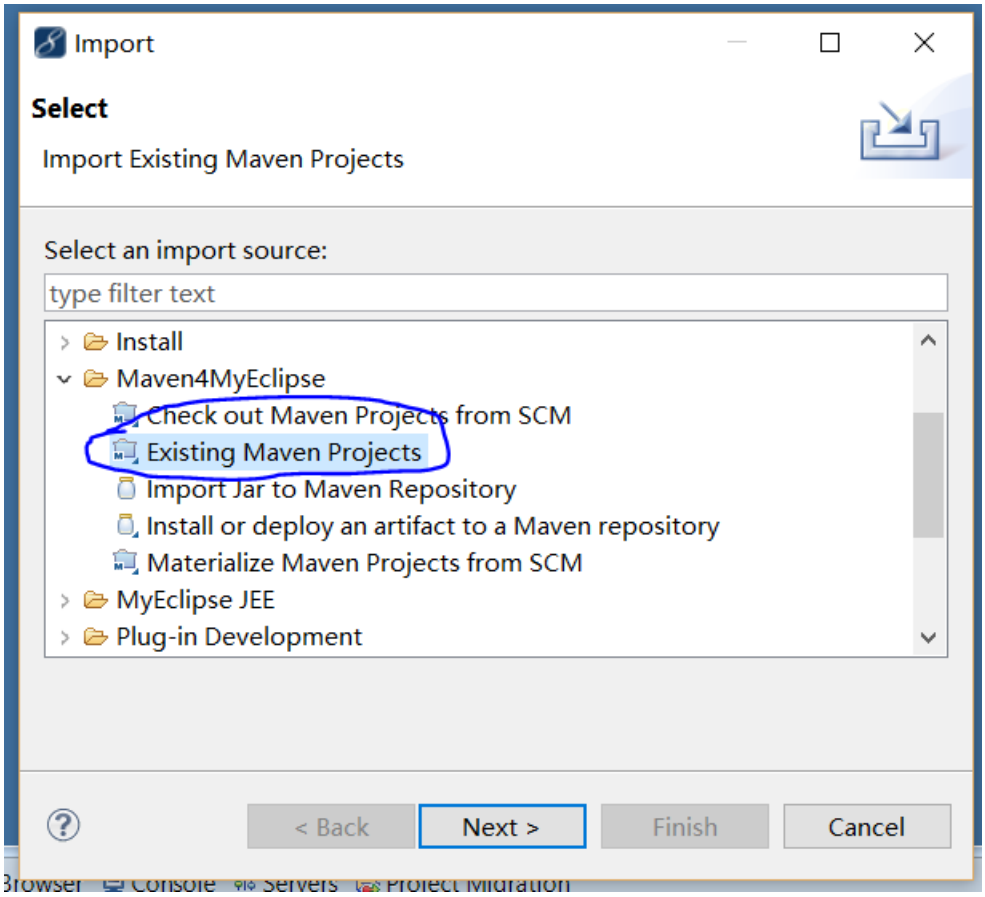
<mirrors>

```
        <!-- 阿里云仓库 -->
        <mirror>
            <id>alimaven</id>
            <mirrorOf>central</mirrorOf>
            <name>aliyun maven</name>
            <url>http://maven.aliyun.com/nexus/content/repositories/central/</url>
        </mirror>
    </mirrors>
```

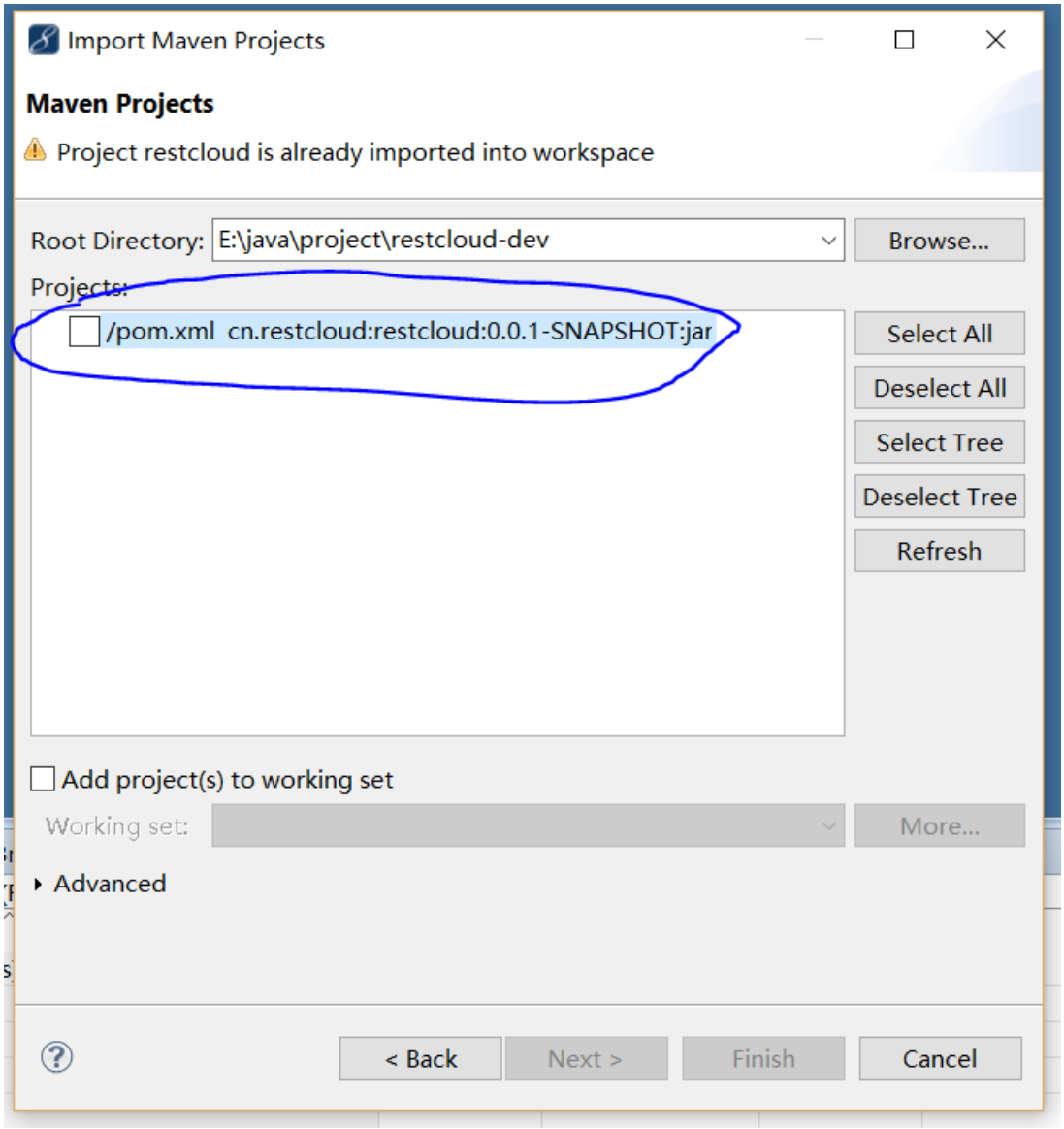
如果不指定则下载会非常慢。

在最后增加一行指定本地仓库的地址：
<localRepository>C:/Users/administartor/.m2/repository</localRepository>

上述都设定好后点击：

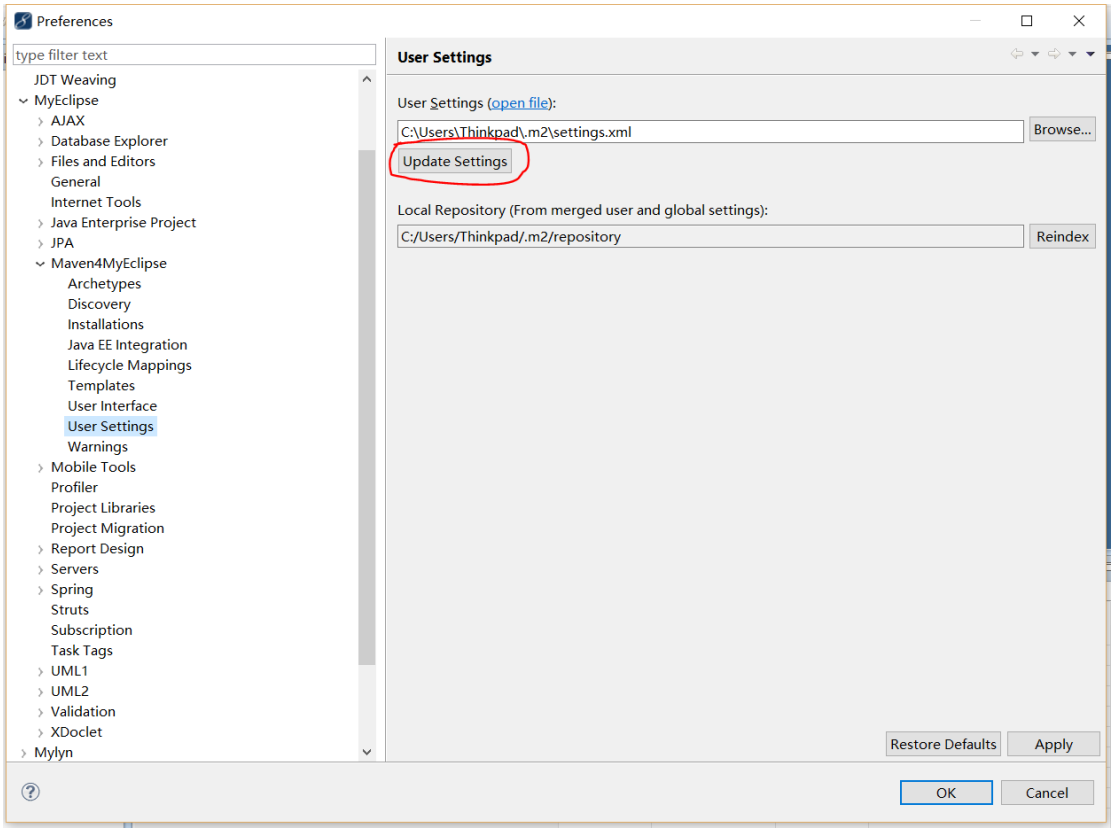


（导入已存在的 Maven 项目）

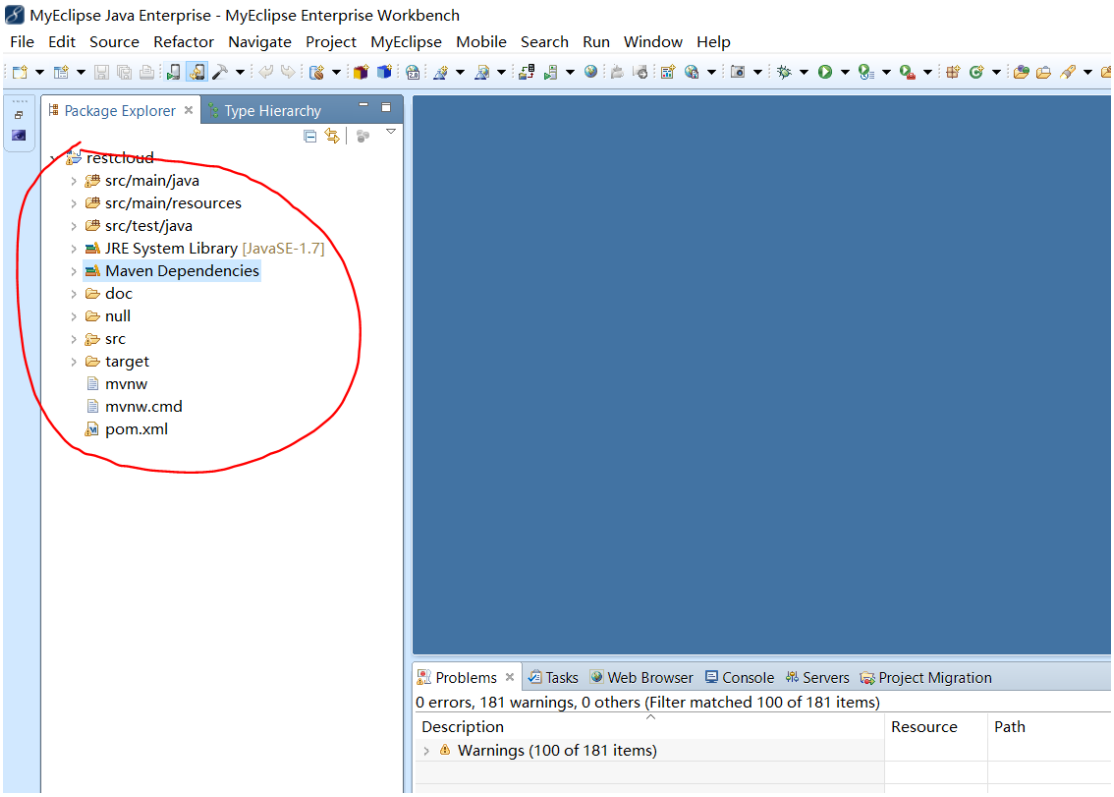


（选中提供的 RestCloud 项目包）

导入后系统会自动分析 pom.xml 文件并从远程仓库中下载依赖的 jar 包，如果有问题可以再打开 Maven 配置界面点击“update setting”进行下载。

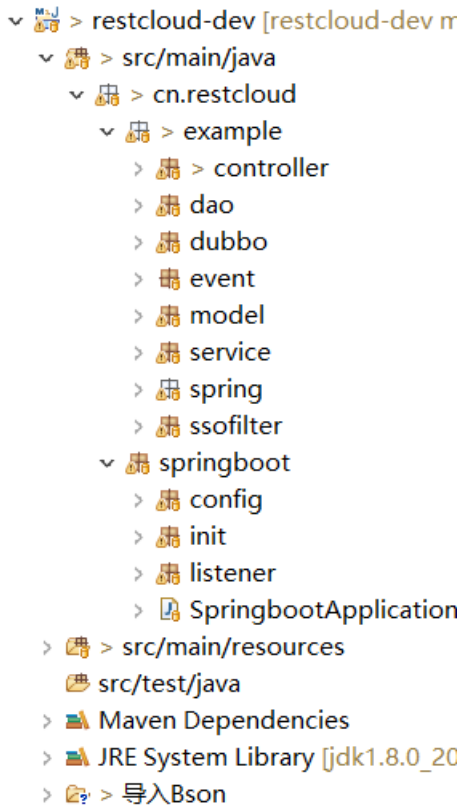


（点击 update settings 下载依赖的 jar 包）



（导入成功后显示如上界面）

4. 工程目录结构



(1) 平台主要的 Class 类全部包含在 lib 目录下的 RestCloud.jar 文件中
(2) Example 是基于 Eclipse 下的开发示例代码，有各种开发模式的代码示例
(3) Springboot 是 springboot 的一些特别配置的类，springbootApplication.java 文件是工程的启动文件.
(4) DruidConfiguration 配置类是用来配置 druid 数据库链接池监控的用户名和密码的
(5) SpringbootUtil 类是用来和 spring 原生的 java bean 进行交付的工具类

技巧：如何替换 restcloud.jar 文件中的类？
只需要在 cn.restcloud 目录下建一个相同的 java 文件的类即可以替换掉 jar 包中的类，因为 tomcat 在加载类时工程中的类是优先于 jar 包中的类加载的。

5. AppContext

功能介绍

AppContext 是全局线程类，在所有的类中都可以通过这个类来进行全局变量的获取，如获取数据库链接对像、获取 Request 对像，获取 Response 对像，获取 API 传入的参数对像、获取当前登录的用户名、当前用户所具有权限等。
注意：AppContext 中的方法全部为 static 的方法，所以可以直接调用即可

获取 HttpServletRequest，HttpServletResponse

HttpServletRequest request=AppContext. getRequest()
HttpServletRequest response=AppContext. getResponse ()

获取当前登录用户 id

String userId=AppContext.getUserId();
String username=AppContext.getUserName(); 登录用户的中文名

获取用户相关权限信息

AppContext.getUserContext() 可以获取用户登录后的相关信息

获取当前用户的权限列表
List<String> permissionIds=AppContext.getUserContext().getPermissionId();
获取当前用户的所有角色列表
List<String> rolesCodes=AppContext.getUserContext().getRolesCode();
判断当前用户是否是超级管理员
boolean isSystemSuperAdmin= AppContext.getUserContext().isSystemSuperAdmin();

把当前线程设为调试状态

在任务代码中调用
AppContext.setDebug(true); 即可以把当前线程设置为调试并输出相应的 SQL 或 API 调用的信息
AppContext.setDebug(false);则可以关闭调试状态

6. RequestUtil

功能介绍

RequestUtil 主要是用来接收 http 请求的参数和变理的工具类,可以获取 reqeust,response 以及 body 请求参数等。

获取请求参数

不管是 get 的还是 post 的表单参数，路径参数，只要不是 body 方式提交的都可以用 RequestUtil.getString(“变量 id”)来进行获取到.

String userid=RequestUtil.getString(“userid”);

获取请求 Body

如果请求参数是 body 的 json 提交方式提交的则用以下方式获取

```
String body=RequestUtil.getRequestBody();
```

获取所有请求的 Headers

```
HashMap<String, String> headers=RequestUtil.getHeaders();
```

获取所有请求参数并转为 Document 对像

```
Document doc=RequestUtil.getContextDocument();
```

系统会把所有请求的 url 或者 post 的参数全部转换到 Document 对像中

HTTP 请求参数直接转为 JavaBean 对像

UserModel userObj=RequestUtil.getBean(UserModel.class);

系统会把所有 get 或 post 进来的参数设置到 UserModel 对像中，参数名要和字段名一一对应即可。

获取用户的 IP 地址

String ip=RequestUtil.getRemoteAddr();

获取请求的 URL 地址

String url=RequestUtil.getRequestURL();

获取所有上传的附件对像

List<FileItem> files=RequestUtil.getUploadFileItems();
其中 FileIem 是 org.apache.commons.fileupload.FileItem 对像，要真正写到硬盘中才能成为一个文件

写入的方法如下：

```
for(FileItem fileItem:fileItems){
   File newFile=new File("d:/"+fileItem.getName());
   fileItem.write(newFile);
}
```

7. RdbUtil 关系数据库工具类

功能介绍

RdbUtil 为本平台操作关系数据库的工具类，通过本工具类可以执行 SQL 语句，获取数据库链接等操作。
注意：本类的所有方法均为 static 所以可以直接使用，无需 new 对像

获取数据库链接

RdbUtil.getConnection()方法可以获取 default 所指向的数据源链接对像
RdbUtil.getConnection(configId); 方法可以获取指定数据源链接的 Connection 对像

代码示例
Connection conn=RdbUtil.getConnection("myoracle");
RdbUtil.closeConn(conn); //关闭数据库链接

关闭数据库链接

系统在线程结束时会自动检测所有没有关闭的Connection对像并进行自动关闭，如果想在代码中提前关闭可以手动关闭链接对像

```
Connection conn=RdbUtil.getConnection();
RdbUtil.closeConn(conn);
```

执行 SQL 语句

String sql="update maindata set subject='test' where WF_OrUnid=?";
int i=RdbUtil.executeUpdateSql(sql,"123456 "); //传入参数

执行分页查询 SQL

以下语句执行 SQL 语句进行分页，但是不会返回总页数，总页数需要再执行另一个 count(*)的 sql 去获取.

```
String sql="select * from mytable ";
int pageNo=1;
int pageSize=10;
List<Document> docs=RdbUtil.listDocsByPage(sql, pageNo, pageSize);
//根据 sql 语句进行分页
```

以下方法通过 pageBean 对像来进行分页可以返回分页后的总数

```
//初始化分页数据
IBasePageBean<Document> pageBean=RdbUtil.getPageBean(Document.class);
pageBean.setPageNo(1);  //当前第几页
pageBean.setPageSize(20);  //每页显示数
pageBean.setSqlTableName("数据库表名");
pageBean.setSqlFields("userid,username,state,age");
pageBean.setSqlOrderby("age");
pageBean.setSqlWhere("where age=12 and state=1");
```

```
RdbUtil.listDocsByPage(RdbUtil.getConnection("mysql"),pageBean);
```

pageBean 对像可以直接转为 json 或者通过 ViewUtil.getJsonView(pageBean);进行返回

开启事务并提交或回滚数据

```
AppContext.getConnection("mysql").setAutoCommit(false);  //开启事务
RdbUtil.executeUpdateSql("update  table ...."); //执行 sql
If(true){
    RdbUtil.commit(); //提交数据
}else{
    RdbUtil.rollback(); //回滚数据
}
```

保存数据到数据库表中

保存一个 Document 对像，Document 实质上是 HashMap 的扩展对像

```
Document doc=new Document();
doc.put("Subject",  "test subject 0012");
doc.put("WF_OrUnid",  DocumentUtil.getNewDocumentId()); //获取一个 22 位的唯一 id 号
doc.put("WF_Processid",  "100001");
int i=RdbUtil.saveDoc("数据库表名", doc, "WF_OrUnid"); //已存在更新，不存在插入
```

注意：WF_OrUnid 是数据库表名的主键，只支持一个主键字段，如果主键已存在则系统自动更新数据，如果不存在则系统自动插入一条数据

保存一个 JavaBean 对像到数据库表中

```
UserModel userObj=new UserModel();
userObj.setUserId("test001");
userObj.setPassword("pass");
userObj.setJobDesc("经理");
int i=RdbUtil.saveBean("数据库表名  ", userObj);
```

注意：UserModel 对像的主键字段必须要用 @KeyId 注解进行标名表示为主键

如果 UserModel 不想用@KeyId 进行注解则可以用以下方法保存

```
UserModel userObj=new UserModel();
userObj.setUserId("test001");
```

```
userObj.setPassword("pass");
userObj.setJobDesc("经理");
Document doc=DocumentUtil.bean2Doc(userObj);
int i=RdbUtil.saveDoc("数据库表名", doc, "数据库表名主键"); //已存在更新，不存在插入
```

从数据库表中读取数据

读取数据到 List<Document>对像中
List<Document>实质上是 List<HashMap<String,Object>对像的扩展

```
String sql="select * from 数据库表名";
List<Document> docs=RdbUtil.listDocs("mysql",sql);
```

读取数据到 JavaBean 对像中 List<Object>类型

```
String sql="select * from 数据库表名 where UserId=?";
List<UserModel> userObjs=RdbUtil.listBeans(UserModel.class, sql, "admin");
```

其中 UserModel 是一个自定义的 JavaBean 对像
注意：使用?号的优势是可以防止参数被 sql 注入，可以用多个？号组成

```
String sql="select * from 数据库表名 where UserId=? And age=?";
List<UserModel> userObjs=RdbUtil.listBeans(UserModel.class, sql, "admin",”18”);
```

8. DocumentUtil

DocumentUtil 已经继承了 JsonUtil 工具类，所以 Json Util 工具类中的方法也可以直接用 DocumentUtil 来实现，DocumentUtil 可以实现 Document 对像的操作也可以实现 Json 与 Document 对像等的互转

功能介绍

DocumentUtil 是对 Document 对像的一些常用功能和取值，设置值的二次封装，例如取字符串用 Document 取为 doc.getString(“userid”); 如果 doc 中 userid 不存在则会返回 null 容易报空指针异常而用 DocumentUtil.getString(doc,”userid”); 如果 doc 中没有 userid 字段时返回 “” 空字符串 DocumentUtil.getString(doc,”userid”,”张三”); 还可以在 userid 为空时返回“张三”默认值。

注意：DocumentUtil 工具类继承了 JsonUtil 工具类，所以 DocuentUtil 也有 Json 转换功能的方法.

Json 字符串与 Document 对像互转

单个对像转换

```
String jsonStr="{a:1,b:2}";
Document doc=DocumentUtil.json2doc(jsonStr);
```

数组转换

```
String jsonStr="[{a:1,b:2}]";
List<Document> docs=DocumentUtil.jsonArray2docs(jsonStr);
```

单个 doc 文档对像转换为 json

```
Document doc=new Document();
doc.put("a", 1);
doc.put("b", 2);
String jsonStr=DocumentUtil.doc2Json(doc);
```

数组转换

```
List<Document> docs=new ArrayList<Document>();
Document doc=new Document();
doc.put("a", 1);
doc.put("b", 2);
docs.add(doc);
String jsonStr=DocumentUtil.docs2Json(docs);
```

JavaBean 与 Document 对像互转

```
Document doc=DocumentUtil.bean2Doc(Object);
```

Object 为 JavaBean 实例对像

Document 对像转为 JavaBean 对像

```
Document doc=new Document();
Doc.put("userid","admin");
Doc.put("age",1);
UserModel userObj=DocumentUtil.doc2Bean(doc, UserModel.class);
```

UserModel 对像中必须要用 set 和 get 方法才可以

使用 JsonPath 取 Document 对像中的值

如果 Document 对像中的层次很多如下：

```
{
   userId:”admin”,
   data:{age:18,address:”广州”}
}
```

要取 data 中的 age 字面则可以用

```
String age=DocumentUtil.readPathValue(doc, "$.data.age");
```

注意：具体 JsonPath 的用法可以搜索 fastjson 的 jsonpath 语法即可

使用 JsonPath 修改 Document 对像中的值

如果 Document 对像中的层次很多如下：

```
{
   userId:”admin”,
   data:{age:18,address:”广州”}
}
```

要修改 data 中的 age 字面则可以用

```
String a=DocumentUtil.setPathValue(doc, “$.data.age”,30);
```

注意：具体 JsonPath 的用法可以搜索 fastjson 的 jsonpath 语法即可

Document 转 JSON 时输出 null 字段

通常情况下如果 Document 中某一个字段的值为 null 时在转为 json 字符串后是不会在 json 字符串中看到的

```
Document doc=new Document();
Doc.put(“userid”,”admin”);
Doc.put(“username”,null);
String jsonStr=DocumentUtil.doc2json(doc);
```

输出的 json 字符串为: {userid:”admin”}并没有 username 字段

如果要想输出 username 字段则要使用如下方法：

```
String jsonStr=DocumentUtil.doc2json(doc,true);
```

后面增加一个 true 参数即可输出如下格式：

```
{userid:”admin”,username:null}
```

9. Document 类

功能介绍

Document 是 org.bson.Document 类的使用，实质上是对 LinkedHashMap 对像的二次封装，因为本平台的数据是存储在 MongoDB 中的，所以本平台就直接使用了 org.bson.Document 类作为数据的存储对像，这个类是 MongoDB 的驱动自带的一种 HashMap 的封装。

常用方法

Document 类的说明文档请直接访问：
http://mongodb.github.io/mongo-java-driver/3.12/javadoc/org/bson/Document.html

Document的常用方法的部分源码如下：

```
public Integer getInteger(final Object key) {
    return (Integer) get(key);
}
public int getInteger(final Object key, final int defaultValue) {
    return get(key, defaultValue);
}
public Long getLong(final Object key) {
    return (Long) get(key);
}
public Double getDouble(final Object key) {
    return (Double) get(key);
}
public String getString(final Object key) {
    return (String) get(key);
}
public Boolean getBoolean(final Object key) {
    return (Boolean) get(key);
}

public boolean getBoolean(final Object key, final boolean defaultValue) {
    return get(key, defaultValue);
}
```

获取 Document 中的所有字段的值

```
Document doc=new Document();
doc.put("a", 1);
doc.put("userid","admin");
doc.put("b",true);
for(String key:doc.keySet()) {
    System.out.print(key+"="+DocumentUtil.getString(doc, key));
}
```

获取 Document 中的字段值不区分大小写

Doc.put(“userid”,”admin”);
String userid=DocumentUtil.getStringByKeysIgnoreCase(doc, "UsErId");
均可以成功获取到 userid 的值 admin

一次获取多个字段的值

```
Document doc=new Document();
doc.put("a", 1);
doc.put("userid","admin");
doc.put("b",true);
String v=DocumentUtil.getStringByKeys(doc, "a,userid");
```

多个字段用逗号分隔则 v 的值返回为: 1,admin

10. RestClient API 调用工具类

功能介绍

RestClient 对是调用 Http API 接口的二次封装，通过 RestClient 类可以很方便的在代码或者规则中调用任意 API 接口。

发起 GET 请求

```
String url="http://ip/getUserInfo";
Document doc=RestClient.getInstance().setUrl(url).get().getDocument();
```

Get 请求的结果如果是 json 可以直接转为 Document 对像，如果不想转则用以下方法

```
String body=RestClient.getInstance().setUrl(url).get().getResponseBody();
```

如果想转为 JavaBean 对像可以用以下方法

```
UserModel
userObj=RestClient.getInstance().setUrl(url).get().getBean(UserModel.class);
```

发起 POST 请求

```
String url="http://ip/rest/example/server/info";
String body=RestClient.getInstance().setUrl(url).addPostParams("userid",
"admin").post().getResponseBody();
```

使用:addPostParams() 方法可以增加 post 参数

如果 body 是一个 json 字符串可以直接转为 Document 对像

```
Document doc=RestClient.getInstance().setUrl(url).post().getDocument();
```

如果 Post 请求接收的参数为 json 字符串时(非 form 提交)，请用如下方法调用:

```
String
body=RestClient.getInstance().setUrl(url).setRequestBody(true).setRequestBodyStr("
{a:1,b:2}").post().getResponseBody();
```

先调用 setRequestBody(true)表示要用 json 传送数据
setRequestBodyStr()表示要传送的 json body 字符串

增加自定义的 Header 头信息

```
String url="http://ip/rest/example/server/info";
String body=RestClient.getInstance().addHeaders("token",
"1234556").setUrl(url).get().getResponseBody();
```

自动追加 token 头认证

如果想在规则或者代码中调用本平台的发布的 API 接口，但是因为本平台的 API 接口都需要 token 头才能进行调用，可以用以下方法自动生成当前用户的 token 并进行 API 的请求。

```
String url="http://ip/rest/example/server/info";
String
body=RestClient.getInstance().addToken(true).setUrl(url).get().getResponseBody();
```

只需要增加 addToken(true)即可

传送附件

```
String url="http://ip/rest/example/server/info";
List<File> files=new ArrayList<File>();
files.add(new File("d:/test.doc"));
String                    body=RestClient.getInstance().setUrl(url).addFile("file",
files).post().getResponseBody();
```

调用 addFile 即可进行文件发送，注意只能使用 post 方法

11. ConfigUtil 获取配置变量

功能介绍

不管是在平台的首页中的平台变量配置中的变量，还是在应用开发中配置的应用级别的变量，还是在 application.perpties 文件中配置的变量都可以统一使用 ConfigUtil 工具类来统一获取变量。

如果有些值不想在代码中编码写死，则可以使用 ConfigUtil 工具类来获取配置值

获取变量值

```
String value=ConfigUtil.getConfig("变量配置的唯一 id","没有配置时的缺省值");
```

12. SpringbootUtil 获取 spring 对像

功能介绍

如果想在规则或者平台的 Java 类中获取 Spring 中生成的 Java Bean 对像则可以使用 SpringbootUtil 工具类来进行获取

获取 Spring 的 JavaBean 对像

Spring 中的所有生成的 java 实例对像都存储在 spring 的容器中，通过 SpringbootUtil 工具类就可以获取到
Object obj=SpringbootUtil.getBean(beanId); //从 spring 容器中按 beanId 进行获取对像
其中 beanId 为 spring 容器中生成 javabean 对像的唯一 id

可以在 API 监控中心中的缓存及容器监控=》SpringBean 监控中查找到所有 spring 中已经初始化的 beanId 值。

13. 其他工具类说明

1. MongoDBUtil 为操作 Mongodb 数据库的专用二次封装工具类
2. RedisUtil 为操作 Redis 数据库的专用二次封装类
3. HBaseService 为操作 HBase 的专用工具类
4. ElasticsearchUtil 为操作 ES 的专用工具类
5. DateTimeUtil 为时间日期的专用工具类
6. LogUtil 为用来打印日志的工具类
7. PrintUtil 是用来输出调试变量的工具类
8. XmlUtil 是用来处理 xml 文件的工具类
9. EasyExcelUtil 是用来处理 excel 文件的工具类
10. ExceptionUtil 是用来处理异常信息并把异常打印到控制台日志的工具类