|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Proposer** |
| CRUD | 1. 如果涉及多表查询，查询实体是否命名为${Entity}Record (e.g. ${Entity}=User, 即多表分页查询实体命名为 Page<UserRecord>) ? 2. 如查询一对多或其他非单表实体， 实体是否命名为 ${Entify}Model (e.g. ${Entity}=User, name= UserModel )? 3. 自定义API 实体参数（除标准实体外） 统一命名以 Request 为作后缀。   e.g. 如需要处理客户转转 ClientXfrRequest   1. Service的返回类型是否尽量能够弃用 泛Map<String,Object> 类型（可读性很低） 2. 业务错误代码是否引用 BusinessException ? 3. 多表联查不使用实体Id，如通过 类别/类型 查找实体，不适用类别/类型的id，而是通过类别/类型 名称去查找 （按类别精确查询除外） | Mandatory |
| CRUD | 充分考虑 API 是否需要向外提供，如历史记录表不能提供 create, update,delete 相关API | Mandatory |
| CRUD | 对实体状态的变更提供独立的API，不能通过实体update service 对状态进行更新  e.g.  Draft: /api/${entify}/action/draft  Close: /api/${entify}/action/close | Mandatory |
| CRUD | Persistence以及crud目录下面的代码不能更改，必须由工具生成与维护（具体位置如下）  com.jfeat.am.module.{moduleName}.services.persistence  com.jfeat.am.module.{moduleName}.services.crud | Mandatory |
| CRUD | 更改数据库表字段尽量用code-generator工具通过设定<initialize>true</initialize>使之生效，而且变更的代码(src/main/resources/sql/\*-schema.sql) 必须提交  如手工变更，确保以下文件作相应字段处理：  persistence.model.\*  persistence.dao.\*  persistence.dao.mapping.\* | Mandatory |
| CRUD | 通过CRUD框架 新建一对多实体，无法自动更新实体id， 其新建实体id由CRUD框架保存在CRUDFilterResult 的result()当中 。新建实体后，可通过filter.result()获取实体 id值，然后更新至实体.  e.g.  @PostMapping  void createMaster(@RequestBody EntityModel **entity**){  EntityFilter **filter** = **new** EntityFilter ();  **entityService**.createMaster(entity, filter, **null**, **null**);  Long **entityId** = **filter**.result().containsKey(**"id"**)  ? **filter**.result().getLong(**”id”**): **null**;  **entity**.setId(**entityId**);  } |  |
| CRUD | @Select 的使用, 基于 BaseMapper<T>接口，以id为条件返回单个实例  @Select(**"SELECT \* FROM user WHERE id=#{id}"**) User getUser(Long id);  以其他字段为查询条件返回实例列表  @Select(**"SELECT \* FROM user WHERE account=#{account}"**) List<User> getUsers(String account); |  |
| 代码可读性 | 1. 对于一段功能代码块（20行内），是否有写 comment说明, 是否有留空行分隔离。 2. 变量是否可读（禁止不可读变量，如a, bb, ss）？ 3. 是否每一个实现方法都能够少于 200 行代码? 4. Endpoint 代码行数是否不超过10行（参数查询选项除外），如超过20行功能代码，即需要通过 Service实现并提供 | Mandatory |
| 代码片断 | 如何从myBatis中获取JDBC连接 ?  @Autowired DataSource **dataSource**;  Connection getConnection(){  **try** {  **return dataSource**.getConnection();  } **catch** (SQLException e) {  e.printStackTrace();  }  **return null**; }  具体用法请参考：  [devops@zele.pro:/home/devops/repo/env/env-test-dao.git](mailto:devops@zele.pro:/home/devops/repo/env/env-test-dao.git)  **global search** **“GeneralJDBCService”** | Mandatory |
| 代码片断 | 需要 获取当前用户的ID或account等信息可以使用JWKit来实现: 例如: 获取当前用户的ID Long **userId** = JWTKit.getUserId(getHttpServletRequest());  获取用户登录账号:  String **account** =JWTKit.getAccount (getHttpServletRequest()); | Mandatory |
| 代码片断 | 如何处理search查询  Endpoint:  @RequestParam(name = **"search"**, required = **false**) String search  Dao  **queryStoreDao**.findStorePage(page,record,orderBy,search)  基于Mapper XML  <**if test="search != null and search != '' "**>  AND (  t\_store.name LIKE CONCAT('%',#{search},'%')  OR t\_store.code LIKE CONCAT('%',#{search},'%')  ) </**if**>  基于EntityWrapper  对于要处理多个 () ,使用 andNew/orNew  if(search!=null && search.length>0){  wapper.  .and(Entity.field, field)  .andNew()  .like(Entity.field1, search)  .or().like(Entity.field2, search)  .or().like(Entity.field3, search);  }else{  wapper.  .and(Entity.field, field)  }  对应执行的sql syntax 为  where entity.field=#{field}  and( field1 like #{search} or field2 like #{search} or field3 like #{search})  不使用 andNew/orNew 无法在实现类中实现对多个 () 处理。  不指名 .like() 为 .or()索引会默认使用 and 进行条件的拼接。 |  |
| 代码片断 | 如何进行时间段查询?  实体列表通常通过URL?query=“参数”方式进行查询。前端对时间段的查询为该字段的一个数组，如查询的字段是createTime, 查询url为 createTime=2018-12-01&createTime=2018-12-12  1# Endpoint接收参数  @RequestParam(required = **false**) @DateTimeFormat(pattern = **"yyyy-MM-dd"**)  Date[]createTime,  Date startTime = createTime!=**null**? (createTime.**length** > 0?createTime[0]:**null**) : **null**; Date endTime = createTime!=**null** ? (createTime.**length**==2?createTime[1]:(createTime.**length**==1?createTime[0]:**null**)) : **null**;  page.setRecords(**queryAppointmentDao**.findEntityPage(page, record, orderBy, startTime, endTime));  2# 通过Record定义与createTime相关参数属性  **public interface** QueryEntityDao **extends** BaseMapper<EntityRecord> {  List< EntityRecord > findEntityPage(Page<EntityRecord> page, @Param(**"record"**) EntityRecord record, @Param(**"orderBy"**) String orderBy, @Param(**"startTime"**) Date startTime, @Param(**"endTime"**) Date endTime); }  3# Dao/xml定义日期段xml查询语句  <**if test="startTime != null and endTime != null"**>  <![CDATA[  AND t\_test\_table.create\_time >= date(#{startTime}) AND  t\_test\_table.create\_time < DATE\_ADD(date(#{endTime}),INTERVAL 1 DAY)  ]]> </**if**>  注：不能用Between, 因为Between 包括右边界，将包含不期望的下一天 yyyy-MM-dd 00:00:00  注意： xml文件里，对于时间字段的比较test, 不能和空字符串比较，只需要和null比较即可。  <**if test="createTime != null"**> |  |
| 代码片断 | 如何处理 报表查询中的 CONCAT?  CONCAT遇到其中一个null字符串会返回null，这样并不是期望值，如  SELECT CONCAT(‘a’,null,’b’) 并不会返回期望值 ab,而是返回null  SELECT id,order\_number,CONCAT(COALESCE(t\_order.province,’’), COALESCE(t\_order.city,’’), COALESCE(t\_order.district,’’), COALESCE(t\_order.detail,’’)) AS address |  |
| 代码片断 | 关于Swagger 的查询参数说明  e.g.  @ApiImplicitParams({  @ApiImplicitParam(name=**"type"**,value=**"类型"**, required=**true**, dataType=**"String"**, paramType=**"query"**)})  @BusinessLog(name = "type", value = "query term"  @GetMapping **public** Tip getTermConfigByType(@RequestParam(name = **"type"**, required = **true**) String type) {  **return** SuccessTip.*create*(**termConfigService**.getTermConfigByType(type)); } |  |
| 代码片断 | 关于图片的上传，统一使用数组方式 [ { “url”: “<http://xxx>" } ]  后台接收方式：  **public class** TestModel {  **private** List<com.jfeat.am.common.constant.tips.Image  > **covers**;  } eg: String json = "{ covers: [ { \"url\": \"http:/[xxx.com](http://xxx.com)\" } ] }";  TestModel model = JSON.*parseObject*(json, TestModel.**class**); |  |
| 代码片断 | Java 8 特性的函数式处理列表  List<ConfigGroup> **tuples** = getGroupTuples(**fieldname**, **type**);  **tuples** = **tuples**.stream()  .filter(u->u.getLang().equals(“zh”)  .collect(Collectors.toList()); |  |
| 代码片断 | 基于唯一字段插入实体的方法  INSERT INTO meice\_recommend\_product(`shootid`,`comment`) VALUES(#{shootid}, #{comment}) ON DUPLICATE KEY UPDATE `comment` = #{comment}  CREATE TABLE `meice\_recommend\_product` (  `id` bigint(20) NOT NULL AUTO\_INCREMENT,  `shootid` varchar(50) COLLATE utf8mb4\_unicode\_ci DEFAULT NULL,  `comment` text COLLATE utf8mb4\_unicode\_ci,  PRIMARY KEY (`id`),  UNIQUE KEY `uk\_shootid` (`shootid`)  ) ENGINE=InnoDB AUTO\_INCREMENT=85 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_unicode\_ci |  |
| 代码片段 | ReentrantLock：可重入锁  使用场景：  多线程情况下,对于共享数据，不允许多个线程同时访问,也即访问共享数据时只能有一个线程,这种情况下,可以使用可重入锁。  例如,模拟多线程售票,因为票的总数固定,所以当售出一张票,票数减1时,这个操作需要加锁,保证售出的票数量和票的总数一致。  使用示例：  Lock **lock** = **new** ReentrantLock();  **if**(**lock**.tryLock()) {  **try**{  //处理任务  }**catch**(Exception **e**){    }finally{  lock.unlock();*//释放锁*  }  }**else**{  *//如果不能获取锁，则直接做其他事情*  }  ReentrantReadWriteLock: 读写锁，分为读锁和写锁。  读写锁可以同时进行读操作。  读锁被占用的时候,此时其他线程申请写锁,需要等待读锁的释放。  写锁被占用的时候,此时其他线程申请写锁或者读锁,需要等待写锁的释放。  使用场景：  多线程情况下,对于读操作时不能写入,写操作时不能读取,并且不能同时进行写操作的情况，可以使用读写锁。  例如,操作同一个文件,查看文件内容的时候肯定不能写入内容,写入内容的同时不能查看文件内容,查看的时候要使用读锁,写入的时候要使用写锁。  使用示例：  private ReentrantReadWriteLock rwl = new ReentrantReadWriteLock();  //获取写锁 rwl.writeLock().tryLock()  //释放写锁 rwl.writeLock().unlock()  if(rwl.readLock().tryLock()) {//获取读锁  try{  //处理任务  }catch(Exception e){    }finally{  rwl.readLock().unlock();//释放读锁  }  }else{  //如果不能获取读锁，则直接做其他事情  }  注意：创建Lock对象时不能放在局部变量位置，应该放在类的成员变量位置。 |  |
| 代码 片断 | *//任务执行*  @Component **public class** CreditReset {  @Resource  AccountService **accountService**;  @Scheduled(cron=**"0 0 0 1 1 ?"**) *//每年一月一号凌晨执行* **public void** task() {  List<Account> list = **accountService**.retrieveMasterList();  **for** (Account vipAccount : list) {  **int** credit = vipAccount.getCredit();  *//积分清零* vipAccount.setCredit(0);  **accountService**.updateMaster(vipAccount);  }  } } |  |
| 代码质量 | 不允许字段重复是否通过 设置数据库表相关字段为 unique，并且catch DuplicateKeyException?  **try**{  **affected** = **entifyService**.createMaster(**entity**); }**catch** (DuplicateKeyException **e**){  **throw new** BusinessException(BusinessCode.**DuplicateKey**); } | Mandatory |
| 代码质量 | 功能模块是否通过单元测试？ | Mandatory |
| 代码质量 | 为防止null参数无法转换为java基本类型错误，代码用**Boolean,Char,Byte,Short,Integer,Long,Float,Double** 代替 **bolean,long** etc. | Mandatory |
| 代码质量 | 测试用例涉及本地文件是否使用相对路径，测试文件是否作为代码提交？ | Mandatory |
| 代码质量 | 1. 单元测试不能直接连接真实数据库进行测试，只能使用内存数据库，自已写测试数据 2. 如果需要连接数据测试，则其测试代码不能提交 | Mandatory |
| 代码质量 | 提交代码前确保连接本地数据库并设置**initialize: true** 保证 \*-schema.sql/\*-data.sql 数据库脚本没有错误。 | Mandatory |
| 代码质量 | 需要保存当前用户的ID/Account 等信息时，后台调用获取当前用户ID/Account函数直接插入，不在前端做该值的传递。 |  |
| 代码质量 | 权限问题:   * + - 1. 需要在API层面对当前登录者做权限的认证时，使用@Permission(“访问需要的权限String值”) 注解在API层进行权限校检并实现拦截。       2. 可进入路由，在路由层不做权限而是在方法里面做权限认证时，使用 ShiroKit.hasPermission(“需要认证的权限”)进行权限的校检。 |  |
| 代码质量 | 空指针问题: 使用{entity}Mapper.selectById(id)或使用{entity}Service.retrieveMaster(id)获取对象时，对获取对象需要执行非空判断，如果为空，直接抛出异常  **if** (entity==**null**){  **throw new** BusinessException(BusinessCode.***FileNotFound***); } |  |
| 代码质量 | 所有使用 @RequsetBody注解来提交的JSON都使用实体类来接收，不使用 形如: Map<String, List<Long>> params等其他非实体类接收。 |  |
| 代码质量 | 控制器（Endpoint）中不使用全局实例变量，防止产生并发时的线性安全问题。如下代码不被允许  @RequestMapping(**"/api/staff/staffs"**) **public class** StaffTeamEndpoint **extends** BaseController {  //不允许使用全局变量  **private** Staff **mStaff**;  @GetMapping(**"/{id}"**) **public** Tip getTeamOfStaff(@PathVariable Long id) {  //容易产生线性安全问题  **mStaff** = getStaff(id);    //采用局部变量  Staff staff = getStaff(id);  }  } |  |
| 代码质量 | Integer,Long,Double,BigDecimal等数字类型，相同数据类型间进行大小比较时，尽量使用compareTo(param)方法。 |  |
| 代码质量 | 1. 对于第三方接口数据，必须写日志记录接收数据，以方便追踪及定位问题，以及确定问题责任归属  **logger**.info(**“### Received POST /api/entity/entities Body:”** + JSON.toJSONString(entity));  2.模块的第三方接口代码应该独立于模块实现，并以模块名+openapi命名，并需要通过oauth2授权（授权方案待定）  app-module.git  app-module-openapi.git |  |

Timeline Query SQL Reference

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| ***99:T***,  *///till now全部* | No WHERE CONDITION | MySQL Only |
| ***00:D***, */// current day* | **[Selected]**  WHERE date([column-name])=curdate()  **[Optional]**  WHERE [column-name] >= curdate() and [column-name] < DATE\_ADD(curdate(),INTERVAL 1 DAY) |  |
| ***10:LD***, */// last day昨天* | **[Selected]**  WHERE date([column-name])= DATE( DATE\_SUB(CURRENT\_DATE, INTERVAL 1 DAY) )  **~~[Optional]~~**  ~~WHERE [column-name] >= curdate() and [column-name] < DATE\_ADD(curdate(),INTERVAL 1 DAY)~~ |  |
| ***21:W***, */// current week 本周* | **[Selected]**  WHERE week([column-name]) = week(curdate()) and year([column-name]) = year(curdate())  ~~int days = calendar.get(Calendar.DAY\_OF\_WEEK)~~  ~~WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL~~ **~~[days]~~** ~~DAY)~~ |  |
| ***20:LW,***  */// last week 上周* | **[Selected]**  TODO |  |
| ***41:M***, */// current month 本月* | **[Selected]**  WHERE month([column-name]) = month(curdate()) and year([column-name]) = year(curdate())  ~~int days = calendar.get(Calendar.~~***~~DAY\_OF\_MONTH~~***~~)~~  ~~WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL~~ **~~[days]~~** ~~DAY)~~ |  |
| ***40:LM***, */// last month 上个月* | **[Selected]**  WHERE month([column-name]) =  month(DATE\_SUB(CURDATE(), INTERVAL 1 MONTH)) month(curdate()) and year([column-name]) = year(curdate()) |  |
| ***61:Q,***  */// current quarter 本季度* | **[Selected]**  WHERE quarter([column-name]) = quarter (curdate()) and year([column-name]) = year(curdate())  TODO:  check FROM\_UNIXTIME |  |
| ***60:LQ***  */// Last Quarter 上个季肚* | WHERE QUARTER([column-name])=QUARTER(DATE\_SUB(CURDATE (),interval 1 QUARTER)); |  |
| ***81:Y***, */// current year 今年* | **[Selected]**  WHERE year(FROM\_UNIXTIME([column-name])) = year(curdate())  ~~int days = calendar.get(Calendar.~~***~~DAY\_OF\_YEAR~~***~~)~~  ~~HERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL~~ **~~[days]~~** ~~DAY)~~ |  |
| ***80:LY****/// last year 去年* | **[Selected]**  WHERE year(FROM\_UNIXTIME([column-name])) = year(DATE\_SUB(curdate(), INTERVAL 1 YEAR)) |  |
| ***11:LD3***, */// Latest 3 days (3 days ago)前三天* | WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL 3 DAY)  注：N天内记录  WHERE TO\_DAYS(NOW()) - TO\_DAYS(时间字段) <= N |  |
| ***12:LW1***, */// Latest week, (about 7 days ago)前七天* | **[Selected]**  WHERE DATE\_SUB(CURDATE(), INTERVAL 7 DAY) <= date([column-name]);  [**Optional**]  WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(CURDATE()),INTERVAL 7 DAY) |  |
| ***13:LM1***, */// Latest month (about 30 days ago)前30天* | **[Selected]**  WHERE DATE\_SUB(CURDATE(), INTERVAL 1 MONTH) <= date([column-name]);  [**Optional**]  WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL 1 MONTH) |  |
| ***14:LM3***, */// Latest 3 months (about 90 days ago)前90天* | WHERE [column-name] <= now() AND [column-name] > DATE\_SUB(date(now()),INTERVAL 3 MONTH) |  |
| ***62:Q1***, *//第一季度* ***63:Q2***,//第二季度 ***64:Q3***,//第三季度 ***65:Q4***,//第四季度  ***70:Q12****//上半年*  ***72:Q34*** *//下半年* | **[Selected]**  WHERE year([column-name]) = year(curdate()) and quarter([column-name])=[1,2,3,4];  [Q12]  WHERE year([column-name]) = year(curdate()) and quarter([column-name]) in (1,2);  [Q34]  WHERE year([column-name]) = year(curdate()) and quarter([column-name]) in (3,4);  ~~int year = Calendar.getInstance().get(Calendar.YEAR);~~  ***~~Q1~~***  ~~String start = "'[year]-01-01'"~~  ~~String end = "'[year]-04-01'"~~  ***~~Q2~~***  ~~String start = "'[year]-04-01'"~~  ~~String end = "'[year]-07-01'"~~  ***~~Q3~~***  ~~String start = "'[year]-07-01'"~~  ~~String end = "'[year]-10-01'"~~  ***~~Q4~~***  ~~String start = "'[year]-10-01'"~~  ~~String end = "'[year+1]-01-01'"~~  ***~~Q12~~***  ~~String start = "'[year]-01-01'"~~  ~~String end = "'[year]-07-01'"~~  ***~~Q34~~***  ~~String start = "'[year]-01-01'"~~  ~~String end = "'[year+1]-01-01'"~~  ~~WHERE [column-name] > date(~~**~~[start]~~**~~) AND [column-name] <= date(~~**~~[end]~~**~~)~~ |  |
| ***Timeframe***  *//时间段* | String start = “2018-01-01 00:00:00”;  String end = “2018-06-20 00:00:00”;  [日期]  WHERE [column-name] >= date([start]) AND [column-name] < DATE\_ADD(date([column-name]),INTERVAL 1 DAY)  [精确时间]  WHERE [column-name] BETWEEN [start] AND [end] |  |

Configuration

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| JSON 时间格式配置 | application.yml:  **am:**  *#fastjson* 格式化时间的格式，如不配置，默认用*yyyy-MM-dd HH:mm:ss*  *json-serialize-date-format: MM/dd/yyyy HH:mm:ss* | Advance |
| Account 可选配置 | application.yml:  **account:**  *#*是否启动时重置管理员权限  **reset-admin-perms:** false  #启动验证码  **enable-captcha:** true  #允许平台用户注册  **allow-platform-user-register:** false  #允许前端用户注册  **allow-app-user-register:** true | Advance |

QUERY SQL REFERENCE

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| 列表查询含item参考 | private Item item;   1. 直接通过多表关联映射   [resultMap]  <association property="item" javaType="com.Item>  </association>   1. 给association传入一个SELECT   [resultMap]  <association property="item" javaType="com.Item" select="findItemById" column="id"></association>  <select id=”findItemById” parameterType=”Long” resultType=”com.Item”>  SELECT \* FROM t\_item WHERE t\_item.id=#{id}  </select>  column也可使用key-value的形式传入多个查询条件column=”{stockId=stockId,stockType=stockType}”  当查询列表中含其他列表时，使用collection进行resultMap映射，且只考虑第二种写法,避免查询结果分页错误  private List<Item> item;  [resultMap]  <collection property="item" ofType="com.Item" select="findItemsById" column="id"></collection> | Advance |
| 使用ORDER BY输出排序 | SELECT tb\_name.\*, (@rank := @rank + 1) AS rank  FROM tb\_name, (SELECT @rank := 0) init\_temp  ORDER BY tb\_name.id DESC | Reference |
| 最佳实践 | 最佳实践：  查询实体或实列表，前端需要转换配置项字段的键值为名称，如 字段job\_type=Doctor 转换为 “医生”：  {“id”:0, “name”:, “jobType”:“Doctor” }  {“id”:0, “name”:, “jobType”:“医生” }  用法:  查询实体或实列表中若存在配置项字段，(e.g. column=job\_type, value=Doctor) 需要转换为配置内容为 ‘医生’，同时提供选项值 {“options”:[{}]}，直接应用 crud-feature 标准库：  请参考crud-feature的用法  @Resource ConfigFieldService configService;  CRUDFieldOptionFeature feature = new CRUDFieldOptionFeature(configService);  feature.apply(data. “job\_type”, true);  //true – 除提供options数据，同时替换为option值,即将 doctor替换为 “医生”， false 即不替换，只有options选项值 |  |

FILE UPLOAD REFERENCE

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| 文件上传 | 1. 添加commons-io依赖   <dependency>  <groupId>commons-io</groupId>  <artifactId>commons-io</artifactId>  <version>2.5</version>  </dependency>   1. Endpoint   @RestController  **public** **class** UploadEndpoint {  **private** **static** **final** String ***UPLOAD\_PATH*** = "tmp/";  @PostMapping("/api/pub/fs/uploadfile")  @ResponseBody  **public** Tip FileUpload(@RequestPart("file") MultipartFile file) **throws** IOException {  **if** (file.isEmpty()) {  **throw** **new** RuntimeException("file is empty");  }  String originalFileName = file.getOriginalFilename();  String extensionName= *getExtensionName*(originalFileName);  String fileName = UUID.*randomUUID*().toString() + "." + extensionName;  String path = **null**;  File uploadFile = **new** File(***UPLOAD\_PATH***);  **if** (!uploadFile.exists()) {  uploadFile.mkdirs();  }  **try** {  File target = **new** File(***UPLOAD\_PATH*** + fileName);  path = target.getCanonicalPath();  target.setReadable(**true**);  FileUtils.*copyInputStreamToFile*(file.getInputStream(), target);  } **catch** (Exception e) {  **throw** **new** BusinessException(BusinessCode.UploadFileError);  }  **return** SuccessTip.create(path);  }  **public** **static** String getExtensionName(String filename) {  **if** ((filename != **null**) && (filename.length() > 0)) {  **int** dot = filename.lastIndexOf('.');  **if** ((dot >-1) && (dot < (filename.length() - 1))) {  **return** filename.substring(dot + 1);  }  }  **return** filename;  }  } | Reference |

ATTACHMENT REFERENCE

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| 附件上传及显示 | 1.引入sb-base依赖  <dependency>  <groupId>com.jfeat</groupId>  <artifactId>sb-base</artifactId>  <version>2.2.0</version> </dependency>   1. 上传附件：   在Innovation Oriented群里搜索:上传下载图片文件api说明，参考该文件。  3.配置附件上传路径：  **am:**  **file-upload-path: /attachment**  4.上传附件后访问返回的url即可获取附件  例如：上传图片后返回包含如下url:/static/100000000000000001/596c309d-13ef-4bc4-b66c-b58d2eff6737.jpg  只需要在前面加上<http://ip:port>,然后访问<http://127.0.0.1:>8080/static/100000000000000001/596c309d-13ef-4bc4-b66c-b58d2eff6737.jpg,即可在浏览器显示图片。  /Static是映射路径，是hard code  100000000000000001目录是取当前用户org\_id | Reference |

URL ENCODED ASSIC

注: 可通过 Notepad++ 菜单 –> 插件->MIME-Tools->URL Encode 进行转换

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 空格 | ! | \* | " | # | % | & | ' | ( | ) | ; | : | @ | & |
| %20 | %21 | %2A | %22 | %23 | %25 | %26 | %27 | %28 | %29 | %3B | %3A | %40 | %26 |
| \ | = | + | $ | < | > | | | , | / | ? | % | # | [ | ] |
| %5C | %3D | %2B | %24 | %3C | %3E | %7C | %2C | %2F | %3F | %25 | %23 | %5B | %5D |

ZIP Unpack The REFERENCE

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| Zip解压 | 1. 添加Apache-ant依赖   <dependency>  <groupId>org.apache.ant</groupId>  <artifactId>ant</artifactId>  <version>1.9.7</version> </dependency>   1. ZipUtil   *import java.io.\*; import java.util.Enumeration; import org.apache.tools.zip.ZipEntry; import org.apache.tools.zip.ZipFile;  public class ZipUtil {   /\*\*  \* 解压zip文件  \* @param unZipfileName unZipfileName需要解压的zip文件路径  \*/  public static void unZip(String unZipfileName){   try{  ZipFile zipFile = new ZipFile(unZipfileName);  for(Enumeration entries = zipFile.getEntries(); entries.hasMoreElements();){  ZipEntry entry = (ZipEntry)entries.nextElement();  File file = new File(unZipfileName.substring(0,unZipfileName.lastIndexOf(".")+1)+File.separator+entry.getName());   if(entry.isDirectory()){  file.mkdirs();  } else{  //如果指定文件的目录不存在,则创建之.  File parent = file.getParentFile();  if(!parent.exists()){  parent.mkdirs();  }  InputStream inputStream = zipFile.getInputStream(entry);  FileOutputStream fileOut = new FileOutputStream(file);  int readedBytes=0;  byte[] buf = new byte[4096];  while(( readedBytes = inputStream.read(buf) ) > 0){  fileOut.write(buf , 0 , readedBytes );  }  fileOut.close();  inputStream.close();  }  }  zipFile.close();  }catch(IOException ioe){  ioe.printStackTrace();  }  } }* | Reference |

SPRINGBOOT RABBITMQ REFERENCE

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| 文件上传 | 1. 添加maven依赖   <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-amqp</artifactId>  <version>2.0.4.RELEASE</version>  </dependency>   1. application.yml中spring下添加rabbitmq配置   *spring:*  *profiles: dev*  *rabbitmq:*  *host: localhost*  *port: 5672*  *username: guest*  *password: guest*   1. 添加rabbitmq配置   @Configuration  **public** **class** RabbitConfig {  @Bean  **public** Queue queue() {  **return** **new** Queue("demo-queue");  }  }   1. Sender   @Component  **public** **class** DemoSender {  **private** **static** **final** Logger ***logger*** = LoggerFactory.getLogger(DemoSender.**class**);  @Autowired  **private** RabbitTemplate rabbitTemplate;  **public** **void** send(JSONObject obj) {  String jsonMessage = JsonKit.toJson(obj);  logger.debug("send message to mq {}, msg = {}", "demo-queue", jsonMessage);  rabbitTemplate.convertAndSend("demo-queue", SerializationUtils.serialize(jsonMessage));  }  }   1. Receiver   @Component  **public** **class** DemoReceiver {  **private** **static** **final** Logger ***logger*** = LoggerFactory.getLogger(DemoReceiver.**class**);  @RabbitListener(queues = "demo-queue") @RabbitHandler  **public** **void** process(@Payload **byte**[] body) {  String message = (String) SerializationUtils.deserialize(body);  logger.debug("Receiver : {}", message);  **try** {  JSONObject data = JSON.parseObject(message);  logger.debug("data : {}", data.toString());  }  **catch** (Exception ex) {  ex.printStackTrace();  }  }  }   1. Test   @RunWith(SpringJUnit4ClassRunner.**class**)  @SpringBootTest(classes = AmApplication.**class**)  **public** **class** MqTest {  @Autowired  DemoSender demoSender;  @Test  **public** **void** testMq() **throws** JsonProcessingException {  JSONObject message = **new** JSONObject();  message.put("code", 200);  message.put("message", "mq test");  demoSender.send(message);  }  } | Reference |
| Response输出流ServletOutputStream | 使用response的输出流向浏览器响应文件,可用于下载和预览：  @RestController @RequestMapping({"/api/pub/invoice/invoices"}) public class PrintInvoiceEndpoint {  public PrintInvoiceEndpoint() {  }  @GetMapping({"/getInvoice"})  public void getInvoice(HttpServletRequest request, HttpServletResponse response) {  String invoiceName = request.getParameter("invoiceName");  if (null != invoiceName && !"".equals(invoiceName) && invoiceName.endsWith(".pdf")) {  try {  response.setContentType("application/pdf");  PdfOperate.getPdf(invoiceName, response.getOutputStream());  } catch (IOException var5) {  var5.printStackTrace();  }  } else {  throw new BusinessException(BusinessCode.BadRequest);  }  } }  public static void getPdf(String invoiceName, OutputStream outputStream) {  try {  InputStream fis = new BufferedInputStream(new FileInputStream(GenerateInvoicePdf.fileSavePath + invoiceName));  byte[] buffer = new byte[fis.available()];  fis.read(buffer);  fis.close();  OutputStream toClient = new BufferedOutputStream(outputStream);  toClient.write(buffer);  toClient.flush();  toClient.close();  FileUtils.deleteQuietly(new File(GenerateInvoicePdf.fileSavePath + invoiceName));  } catch (IOException var5) {  var5.printStackTrace();  } }  在需要OutputStream参数的方法中,可以传入response.getOutputStream(),这样会将写入到response输出流的内容直接响应给浏览器。  如果这个文件是pdf文件，浏览器上可以预览,打印和下载。  如果是zip文件，会直接下载。 |  |

|  |  |  |
| --- | --- | --- |
| **Item** | **Details** | **Feature** |
| redis问题 | 报错信息如下：  redis.clients.jedis.exceptions.JedisConnectionException:Could not get a resource from the pool  解决方法：  取消redis配置,设置为false  am:  shiro-cache-redis: false | Reference |
| Bean不能注入问题 | 错误信息：  The bean 'itemMapper' could not be injected as a 'com.jfeat.am.module.invoice.services.gen.persistence.dao.InvoiceItemMapper' because it is a JDK dynamic proxy that implements: com.baomidou.mybatisplus.mapper.BaseMapper  解决方法：  由于bean同名，导致无法注入，修改bean名称，和类名一致，首字母小写 | Reference |
| Upload | 使用 zero-io 上传文件时，由于spring默认限制最大为1MB，无法满足业务的，通过 yml 对文件大小进行配置，具体配置信息如下:  multipart:  max-file-size: 10MB  max-request-size: 10MB  根据业务的需要自行配置size |  |