使用动态查询

在查询应用中有种场景，对一个对象就行多字段查询，这种问题比较繁杂。

**比较笨的做法是：根据不同的参数类型和数量堆积查询方法。**

|  |  |
| --- | --- |
| 1 | public List<Object> findBy(String name); |
| 2 | public List<Object> findBy(String name, Integer num); | |

|  |  |
| --- | --- |
| 3 | public List<Object> findBy(String name, Integer num, String address); |

**改进的做法是：对多个查询条件进行if-else判断，将满足的条件加入到其中进行查询。**

|  |  |  |
| --- | --- | --- |
| 1 | public List<Object> findBy(String name, Integer num, String address) { | |
| 2 | if (condition) { |

|  |  |
| --- | --- |
| 3 | ...... |
| 4 | } else if { |

|  |  |
| --- | --- |
| 5 | ...... |
| 6 | } | |

|  |  |
| --- | --- |
| 7 | } |

**比较完美的做法是：动态组装参数，调用同一方法做到动态查询。**

|  |  |  |
| --- | --- | --- |
| 1 | public List<Object> findByExample(String[] args) { | |
| 2 | ...... |

|  |  |
| --- | --- |
| 3 | } |

以spring data jpa为例：使用CriteriaBuilder包裹组装查询条件

|  |  |  |
| --- | --- | --- |
| 01 | public Predicate toPredicate(Root<T> root, CriteriaQuery<?> query, CriteriaBuilder builder) { | |
| 02 | if (filters != null && !filters.isEmpty()) { |

|  |  |  |
| --- | --- | --- |
| 03 | List<Predicate> predicates = Lists.newArrayList(); | |
| 04 | for (SearchFilter filter : filters) { |

|  |  |
| --- | --- |
| 05 | Path expression = root.get(names[0]); |
| 06 | predicates.add(builder.equal(expression, filter.getValue())); | |

|  |  |  |
| --- | --- | --- |
| 07 | } | |
| 08 | } |

|  |  |
| --- | --- |
| 09 | // 将所有条件用 and 联合起来 |
| 10 | if (predicates.size() > 0) { | |

|  |  |  |  |
| --- | --- | --- | --- |
| 11 | return builder.and(predicates.toArray(new Predicate[predicates.size()])); | | |
| 12 | | } |

|  |  |
| --- | --- |
| 13 | } |
| 14 | return builder.conjunction(); | |

|  |  |
| --- | --- |
| 15 | } |

使用了该策略，并且对Date和Enum进行了自动转换，这样能做到通用的动态查询，只需要在service中声明类似

|  |  |
| --- | --- |
| 1 | List<LogEntity> findByExample(Specification<LogEntity> specification, Page page); |

的方法，根据规则自定义查询参数就行了。使用时，只需

|  |  |
| --- | --- |
| 1 | Specification<LogEntity> spec = DynamicSpecifications.bySearchFilter(request, LogEntity.class); |
| 2 | List<LogEntity> logEntities = logEntityService.findByExample(spec, page); |

啪啪两步，即可解决这一繁琐的问题。很棒吧，赶紧试试吧！

Log的用法

利用springmvc的拦截器开发了log功能，用于跟踪、记录系统用户的操作轨迹，以便日后的认责。  
该功能使用很方便，是可配置的、细粒度的日志记录功能。  
操作用法详见代码：  
1.首先需要在spring-mvc.xml做全局配置。

|  |  |  |
| --- | --- | --- |
| 01 | | <!-- 系统日志跟踪功能 --> |
| 02 | <bean id="log4JDBCImpl" class="com.ketayao.ketacustom.log.impl.Log4JDBCImpl" > | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 03 | <property name="logEntityService" ref="logEntityServiceImpl"/> | | |
| 04 | | <property name="rootLogLevel" value="ERROR"/> |

|  |  |  |
| --- | --- | --- |
| 05 | <property name="customLogLevel"> | |
| 06 | <map> |

|  |  |  |  |
| --- | --- | --- | --- |
| 07 | <entry key="com.ketayao.ketacustom" value="TRACE" /> | | |
| 08 | | <entry key="com.dd" value="TRACE" /> |

|  |  |  |
| --- | --- | --- |
| 09 | </map> | |
| 10 | </property> |

|  |  |
| --- | --- |
| 11 | </bean> |

2.然后在Controller层使用注解。

|  |  |
| --- | --- |
| 001 | /\*\* |
| 002 | \* 莫紧张，仅仅是一个例子。 主要是展示怎么使用LOG。 | |

|  |  |  |
| --- | --- | --- |
|  |  | |
| 004 | \* Version  2.0.0 |

|  |  |  |
| --- | --- | --- |
| 005 | \* @since   2013-4-21 下午8:43:54 | |
| 006 | \*/ |

|  |  |
| --- | --- |
| 007 | @Controller |
| 008 | @RequestMapping("/management/sample/task") | |

|  |  |  |
| --- | --- | --- |
| 009 | public class TaskControllerLogShow { | |
| 010 |  |

|  |  |
| --- | --- |
| 011 | @Autowired |
| 012 | private TaskService taskService; | |

|  |  |
| --- | --- |
| 013 |  |
| 014 | private static final String CREATE = "management/sample/task/create"; | |

|  |  |
| --- | --- |
| 015 | private static final String UPDATE = "management/sample/task/update"; |
| 016 | private static final String LIST = "management/sample/task/list"; |

|  |  |  |
| --- | --- | --- |
| 017 | private static final String VIEW = "management/sample/task/view"; | |
| 018 |  |

|  |  |
| --- | --- |
| 019 | @RequiresPermissions("Task:save") |
| 020 | @RequestMapping(value="/create", method=RequestMethod.GET) | |

|  |  |  |
| --- | --- | --- |
| 021 | public String preCreate(Map<String, Object> map) { | |
| 022 | return CREATE; |

|  |  |
| --- | --- |
| 023 | } |
| 024 |  | |

|  |  |
| --- | --- |
| 025 | /\*\* |
| 026 | \* LogMessageObject的write用法实例。 | |

|  |  |
| --- | --- |
| 027 | \*/ |
| 028 | @Log(message="添加了{0}任务，LogMessageObject的isWritten为true。", level=LogLevel.INFO) | |

|  |  |
| --- | --- |
| 029 | @RequiresPermissions("Task:save") |
| 030 | @RequestMapping(value="/create", method=RequestMethod.POST) | |

|  |  |  |
| --- | --- | --- |
| 031 | public @ResponseBody String create(@Valid Task task) { | |
| 032 | taskService.save(task); |

|  |  |
| --- | --- |
| 033 |  |
| 034 | // 加入一个LogMessageObject，该对象的isWritten为true，会记录日志。 | |

|  |  |  |
| --- | --- | --- |
| 035 | LogUitl.putArgs(LogMessageObject.newWrite().setObjects(new Object[]{task.getTitle()})); | |
| 036 | | return AjaxObject.newOk("任务添加成功！").toString(); |

|  |  |
| --- | --- |
| 037 | } |
| 038 |  | |

|  |  |
| --- | --- |
| 039 | /\*\* |
| 040 | \* LogMessageObject的ignore用法实例，ignore不会记录日志。 | |

|  |  |
| --- | --- |
| 041 | \*/ |
| 042 | @Log(message="你永远不会看见该日志，LogMessageObject的isWritten为false。", level=LogLevel.INFO) | |

|  |  |
| --- | --- |
| 043 | @RequiresPermissions("Task:edit") |
| 044 | @RequestMapping(value="/update/{id}", method=RequestMethod.GET) | |

|  |  |  |
| --- | --- | --- |
| 045 | public String preUpdate(@PathVariable Long id, Map<String, Object> map) { | |
| 046 | Task task = taskService.get(id); |

|  |  |
| --- | --- |
| 047 |  |
| 048 | map.put("task", task); | |

|  |  |
| --- | --- |
| 049 |  |
| 050 | // 加入一个LogMessageObject，该对象的isWritten为false，不会记录日志。 | |

|  |  |  |
| --- | --- | --- |
| 051 | LogUitl.putArgs(LogMessageObject.newIgnore()); | |
| 052 | return UPDATE; |

|  |  |
| --- | --- |
| 053 | } |
| 054 |  | |

|  |  |
| --- | --- |
| 055 | /\*\* |
| 056 | \* Log的level用法实例 | |

|  |  |
| --- | --- |
| 057 | \* 1.level分为三层，默认包层（rootLogLevel默认值TRACE），自定义包层（customLogLevel），具体方法层（@Log默认值TRACE） |
| 058 | \* 2.参考顺序：默认包层->自定义包层->具体方法层->LogMessageObject |

|  |  |
| --- | --- |
| 059 | \* 3.有自定义包层的level等级会忽略默认包层 |
| 060 | \* 4.@Log的level大于等于自定义包层或者默认的level会输出日志；小于则不会。 | |

|  |  |
| --- | --- |
| 061 | \*/ |
| 062 | @Log(message="Log的level用法实例，LogLevel.TRACE小于自定义包层LogLevel.INFO，不会输出日志。", level=LogLevel.TRACE) | |

|  |  |
| --- | --- |
| 063 | @RequiresPermissions("Task:edit") |
| 064 | @RequestMapping(value="/update", method=RequestMethod.POST) | |

|  |  |  |
| --- | --- | --- |
| 065 | public @ResponseBody String update(@Valid Task task) { | |
| 066 | taskService.update(task); |

|  |  |
| --- | --- |
| 067 |  |
| 068 | return AjaxObject.newOk("任务修改成功！").toString(); | |

|  |  |  |
| --- | --- | --- |
| 069 | } | |
| 070 |  |

|  |  |
| --- | --- |
| 071 | @RequiresPermissions("Task:delete") |
| 072 | @RequestMapping(value="/delete/{id}", method=RequestMethod.POST) | |

|  |  |  |
| --- | --- | --- |
| 073 | public @ResponseBody String delete(@PathVariable Long id) { | |
| 074 | taskService.delete(id); |

|  |  |
| --- | --- |
| 075 |  |
| 076 | return AjaxObject.newOk("任务删除成功！").setCallbackType("").toString(); | |

|  |  |
| --- | --- |
| 077 | } |
| 078 |  | |

|  |  |
| --- | --- |
| 079 |  |
| 080 | /\*\* | |

|  |  |
| --- | --- |
| 081 | \* Log的override用法实例 |
| 082 | \* 假如override为true，会忽略掉level | |

|  |  |
| --- | --- |
| 083 | \* |
| 084 | \* 批量删除展示 | |

|  |  |
| --- | --- |
| 085 | \*/ |
| 086 | @Log(message="Log的override用法实例，override为true，会忽略掉level。删除了{0}任务。", level=LogLevel.TRACE, override=true) | |

|  |  |
| --- | --- |
| 087 | @RequiresPermissions("Task:delete") |
| 088 | @RequestMapping(value="/delete", method=RequestMethod.POST) | |

|  |  |  |
| --- | --- | --- |
| 089 | public @ResponseBody String deleteMany(Long[] ids) { | |
| 090 | String[] titles = new String[ids.length]; |

|  |  |
| --- | --- |
| 091 | for (int i = 0; i < ids.length; i++) { |
| 092 | Task task = taskService.get(ids[i]); | |

|  |  |  |
| --- | --- | --- |
| 093 | taskService.delete(task.getId()); | |
| 094 |  |

|  |  |  |
| --- | --- | --- |
| 095 | titles[i] = task.getTitle(); | |
| 096 | } |

|  |  |  |
| --- | --- | --- |
| 097 | |  |
| 098 | LogUitl.putArgs(LogMessageObject.newWrite().setObjects(new Object[]{Arrays.toString(titles)})); | | |

|  |  |  |
| --- | --- | --- |
| 099 | return AjaxObject.newOk("任务删除成功！").setCallbackType("").toString(); | |
| 100 | } |

|  |  |
| --- | --- |
| 101 |  |
| 102 | @RequiresPermissions("Task:view") | |

|  |  |
| --- | --- |
| 103 | @RequestMapping(value="/list", method={RequestMethod.GET, RequestMethod.POST}) |
| 104 | public String list(Page page, String keywords, Map<String, Object> map) { |

|  |  |
| --- | --- |
| 105 | List<Task> tasks = null; |
| 106 | if (StringUtils.isNotBlank(keywords)) { | |

|  |  |  |
| --- | --- | --- |
| 107 | tasks = taskService.find(page, keywords); | |
| 108 | } else { |

|  |  |  |
| --- | --- | --- |
| 109 | tasks = taskService.findAll(page); | |
| 110 | } |

|  |  |
| --- | --- |
| 111 |  |
| 112 | map.put("page", page); | |

|  |  |
| --- | --- |
| 113 | map.put("tasks", tasks); |
| 114 | map.put("keywords", keywords); | |

|  |  |
| --- | --- |
| 115 |  |
| 116 | return LIST; | |

|  |  |
| --- | --- |
| 117 | } |
| 118 |  | |

|  |  |
| --- | --- |
| 119 | /\*\* |
| 120 | \* 自定look权限，实例。 | |

|  |  |
| --- | --- |
| 121 | \* 描述 |
| 122 | \* @param id | |

|  |  |  |
| --- | --- | --- |
| 123 | \* @param map | |
| 124 | \* @return |

|  |  |
| --- | --- |
| 125 | \*/ |
| 126 | @RequiresPermissions("Task:look") | |

|  |  |
| --- | --- |
| 127 | @RequestMapping(value="/view/{id}", method={RequestMethod.GET}) |
| 128 | public String view(@PathVariable Long id, Map<String, Object> map) { |

|  |  |  |
| --- | --- | --- |
| 129 | Task task = taskService.get(id); | |
| 130 | map.put("task", task); |

|  |  |  |
| --- | --- | --- |
| 131 | return VIEW; | |
| 132 | } |