灵活通用的操作日志系统设计

一次写成，所有系统直接使用

作者：加伊 liukaixuan@gmail.com
欢迎关注我：http://weibo.com/guzzframework

目录

通用应用日志

这里的日志是指管理操作日志，如“XXX 什么时间删除了一篇文章”等，不是程序往日志文件输出的调试日志。调试日志已经有了很多成熟的工具，如各类 log4j，数不胜数，不再研究。

对于操作日志，一般都存储在数据库中以方便查询；但由于每个系统可能记录的字段属性不同，一般都需要单独设计，造成每套系统都需要单独设计一个日志模块，增加了工作量。

通用应用日志服务是指，通过一种足够灵活的设计方式，使得几乎所有系统都可以直接使用的日志模块。应用只需要少量的工作，即可把操作日志转给日志服务处理，节省日志模块开发工作量。

架构设计

通用应用日志的架构是客户端+服务器模式。服务器端运行日志服务，实现日志的存储和检索，每个应用的日志存储在单独的一张表中。客户端通过远程 API 调用，实现日志的存储和查询。由于每个应用需要存储的日志信息不同，通用日志只提供最简单的基本日志属性，其他属性由应用自己定义。

应用可以自定义多个属性，每个属性包含将在程序中使用的属性名，数据库存储使用的字段名，对外显示的名称，以及字段数据类型。数据类型可以为 string, int, bigint 等各种类型，根据应用需要而定。日志表不设计冗余字段，需要什么字段，应用直接添加直接使用。

客户端不包含复杂的业务逻辑操作，允许不同的编程语言调用。

服务器端-表结构

服务器端设计 2+n 张表，gs_log_app 记录使用此服务的应用系统信息，gs_log_custom_property 记录每个应用自定义的属性。N 表是指每个应用单独创建的详细日志记录表。

gs_log_app 表结构：

```
CREATE TABLE gs_log_app (
  id int(11) NOT NULL auto_increment,
  appName varchar(64) NOT NULL,
  secureCode varchar(64) NOT NULL,
  description varchar(255),
  recordsCount int(11) default 0,
  createdTime datetime,
  PRIMARY KEY  (id),
```

```
  KEY idx_scode (secureCode)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

id，自增 ID，用于分表，让每个应用的日志存储在单独的表中；appName 为应用名称，用于显示；secureCode 为接入密码，64 位随机字符串，客户端接入时使用，用于验证客户端身份；recordsCount 为日志记录数，以后用来做同一个应用日志的分表之类的。

gs_log_custom_property 表结构：

```
CREATE TABLE gs_log_custom_property (
  id int(11) NOT NULL auto_increment,
  appId int(11) not null,
  propName varchar(32) NOT NULL,
  colName varchar(32) NOT NULL,
  displayName varchar(32) NOT NULL,
  dataType varchar(32) NOT NULL,
  createdTime datetime,
  PRIMARY KEY  (id),
  KEY idx_appId (appId)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

gs_log_custom_property 每条记录为一个应用的一个自定义属性。

appId 为应用的编号， foreign key 到 gs_log_app 的 id。 propName java 程序里使用的属性名； colName 数据库字段名； displayName 在网页上显示的名称 dataType 数据类型，和 hbm.xml 的 type 属性类似。 createdTime 创建时间。

对于详细的操作日志记录，java 域对象定义如下：

```
public class LogRecord {

	/**
	 * 自增 Id
	 */
	private long id ;

	/**
	 * 用户编号
	 */
	private int userId ;

	/**
	 * 应用编号
	 */
	private int appId ;
```

```
    /**
     * 请求插入此日志的服务器 IP
     */
    private String appIP ;

    private Date createdTime ;

    /**
     * 自定义属性的值。
     */
    private Map<String, Object> otherProps = new HashMap<String, Object>() ;

    //get&set 方法
```

可以看到，详细的日志记录，我们只设计了 id, userId, appId, appIP, createdTime 5 个基本属性；另外还设计了一个 otherProps 用于存储每个应用自定义的属性。

对于 LogRecord 的表结构，需要根据自定义属性决定。假设我们添加了一个应用，编号为 5，有 2 个属性 string userIP 和 float moveSpeed；则应该在数据库中创建日志记录表：

```
CREATE TABLE gs_log_record_5 (
  id bigint(20) NOT NULL auto_increment,
  userId int(11) not null,
  appId int(11) not null,
  appIP varchar(32) not null,

  userIP varchar(32) not null,
  moveSpeed float(15,3) not null,

  createdTime datetime not null,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

create index idx_ms on gs_log_record_5(moveSpeed) ;
```

每个自定义属性在数据库中按照各自的类型创建成单独的字段。我们还给 moveSpeed 加了索引，用于检索。

客户端-API 设计

从使用者的角度，客户端 API 越简单越好。对于自定义属性的管理和应用配置，全部放到服务器端通过 web 方式管理。客户端 API 只提供 3 个接口，用于：插入日志，查询日志，查询每个属性的对外显示名称。

接口如下：

```
public interface AppLogService {

        /**
         * 根据配置的日志组，插入一条日志。
         *
         * @param userId 操作用户
         * @param customProps 日志自定义属性
         */
        public void insertLog(int userId, Map<String, Object> customProps) throws Exception ;

        /**
         *
         * 根据配置的日志组查询日志。用户编号的属性名为 userId；日志记录时间的属性名为 createdTime，
传入查询条件的格式为：yyyy-MM-dd HH:mm:ss
         *
         * @param conditions 条件列表。每条一个条件，如：userId=1，如：title~=读书
         * @param orderBy
         * @param pageNo
         * @param pageSize
         *
         * @return 如果条件不足，可能返回 null；如果条件错误，可能抛出异常。
         */
        public PageFlip queryLogs(List<String> conditions, String orderBy, int pageNo, int
pageSize) throws Exception ;

        /**
         * 根据配置的日志组，查询配置的自定义属性的元数据。
         *
         * <p/>返回的是数据库中记录的数据，如果自定义属性正在调整，可能会和{@link #queryLogs(List,
String, int, int)} 返回的数据列对不上。
         *
         * @return 返回自定义属性的元数据 Map。key 为 java 属性名，value 为用于对网友显示的
displayName。
         */
        public Map<String, String> queryCustomPropsMetaInfo() throws Exception ;

        /**
         * 插入一条日志到给定的日志组。
         *
         * @param secureCode 日志组密码
         * @param userId 操作用户
         * @param customProps 日志自定义属性
         */
```

```
        public void insertLog(String secureCode, int userId, Map<String, Object>
customProps) throws Exception ;

        /**
         *
         * 查询给定日志组的日志。用户编号的属性名为 userId；日志记录时间的属性名为 createdTime，传入
查询条件的格式为：yyyy-MM-dd HH:mm:ss
         *
         * @param secureCode 日志组密码
         * @param conditions 条件列表。每条一个条件，如：userId=1，如：title~=读书
         * @param orderBy
         * @param pageNo
         * @param pageSize
         *
         * @return 如果条件不足，可能返回 null；如果条件错误，可能抛出异常。
         */
        public PageFlip queryLogs(String secureCode, List<String> conditions, String
orderBy, int pageNo, int pageSize) throws Exception ;

        /**
         * 查询给定日志组配置的自定义属性的元数据。
         *
         * <p/>返回的是数据库中记录的数据，如果自定义属性正在调整，可能会和{@link #queryLogs(String,
List, String, int, int)} 返回的数据列对不上。
         *
         * @param secureCode 日志组密码
         * @return 返回自定义属性的元数据 Map。key 为 java 属性名，value 为用于对网友显示的
displayName。
         */
        public Map<String, String> queryCustomPropsMetaInfo(String secureCode) throws
Exception ;

}
```

接口提供了 6 个方法，分为 2 组。一组按照默认配置的 secureCode，也就是密码插入查询日志；一组按照给定的 secureCode 查询插入，由应用自己选择使用。因为有些系统可能 1 组日志即可；而有些系统虽然是一个应用，但需要记录不同的日志，需要记录到不同的日志组中（在服务器端就是不同的日志应用）。

在这些接口中，customProps 为每个应用自定义的属性 Map，key 为属性名，也就是服务器端自定义属性管理中的 propName，value 为属性值。

List<String> conditions 为查询条件。因为是远程调用，因此查询条件需要简化成 String 类型，按照”属性+操作符+值“的方式传递，服务器负责解析。其中”属性“为 java 属性值，不是数据库字段名，这样开发者只需要记忆一个，不用和数据库直接打交道。

String 类型查询条件的格式，使用 guzz 的标签查询语法，请参看：http://code.google.com/p/guzz/wiki/TutorialTaglib?wl=zh-Hans

架构实现

客户端 API 实现

使用 CommandService，实现代码：

```
public class AppLogServiceImpl extends AbstractService implements AppLogService {

	public static final String COMMAND_NEW_LOG = "gs.alog.new.l" ;

	public static final String COMMAND_QUERY_LOG = "gs.qlog.q.l" ;

	public static final String COMMAND_QUERY_META = "gs.qlog.q.m" ;

	public static final String KEY_APP_SECURE_CODE = "__key_app_scode" ;

	public static final String KEY_APP_USER_ID = "__key_app_uid" ;

	private CommandService commandService ;

	private String secureCode ;

	public void insertLog(int userId, Map<String, Object> customProps) throws
Exception{
		this.insertLog(this.secureCode, userId, customProps) ;
	}

	public PageFlip queryLogs(List<String> conditions, String orderBy, int pageNo, int
pageSize) throws Exception{
		return this.queryLogs(this.secureCode, conditions, orderBy, pageNo,
pageSize) ;
	}

	public Map<String, String> queryCustomPropsMetaInfo() throws Exception {
		return this.queryCustomPropsMetaInfo(this.secureCode) ;
	}

	public void insertLog(String secureCode, int userId, Map<String, Object>
customProps) throws Exception {
		Assert.assertNotNull(secureCode, "secureCode 不能为空! ") ;
```

```
                customProps.put(KEY_APP_SECURE_CODE, secureCode) ;
                customProps.put(KEY_APP_USER_ID, userId) ;

                this.commandService.executeCommand(COMMAND_NEW_LOG,
JsonUtil.toJson(customProps)) ;
        }

        public PageFlip queryLogs(String secureCode, List<String> conditions, String
orderBy, int pageNo, int pageSize) throws Exception {
                Assert.assertNotNull(secureCode, "secureCode 不能为空! ") ;

                AppLogQueryRequest r = new AppLogQueryRequest() ;
                r.setSecureCode(secureCode) ;
                r.setConditions(conditions) ;
                r.setOrderBy(orderBy) ;
                r.setPageNo(pageNo) ;
                r.setPageSize(pageSize) ;

                String json = this.commandService.executeCommand(COMMAND_QUERY_LOG,
JsonUtil.toJson(r)) ;

                if(json == null) return null ;

                return JsonPageFlip.fromJson(json, LogRecord.class).toPageFlip() ;
        }

        public Map<String, String> queryCustomPropsMetaInfo(String secureCode) throws
Exception {
                Assert.assertNotNull(secureCode, "secureCode 不能为空! ") ;

                String json = this.commandService.executeCommand(COMMAND_QUERY_META,
secureCode) ;

                return JsonUtil.fromJson(json, HashMap.class) ;
        }

        public boolean configure(ServiceConfig[] scs) {
                if(scs.length == 1){
                        String secureCode = scs[0].getProps().getProperty("secureCode") ;
                        Assert.assertNotEmpty(secureCode, "secureCode is a must!") ;

                        this.secureCode = secureCode ;
                }
```

```
        return true ;
    }

    public boolean isAvailable() {
        return true ;
    }

    public void shutdown() {
    }

    public void startup() {
    }

    public CommandService getCommandService() {
        return commandService;
    }

    public void setCommandService(CommandService commandService) {
        this.commandService = commandService;
    }

    public static class AppLogQueryRequest{

        private String secureCode ;

        private int pageNo ;

        private int pageSize ;

        private String orderBy ;

        private List<String> conditions ;

        public String getSecureCode() {
            return secureCode;
        }

        public void setSecureCode(String secureCode) {
            this.secureCode = secureCode;
        }

        public List<String> getConditions() {
            return conditions;
```

```
		}

		public void setConditions(List<String> conditions) {
			this.conditions = conditions;
		}

		public int getPageNo() {
			return pageNo;
		}

		public void setPageNo(int pageNo) {
			this.pageNo = pageNo;
		}

		public int getPageSize() {
			return pageSize;
		}

		public void setPageSize(int pageSize) {
			this.pageSize = pageSize;
		}

		public String getOrderBy() {
			return orderBy;
		}

		public void setOrderBy(String orderBy) {
			this.orderBy = orderBy;
		}

	}

}
```

通过通信信道，将请求发送到服务器端，执行完毕后返回结果。查询返回结果 PageFlip 内的对象转换成本地 LogRecord pojo 对象，与领域对象结构相同：

```
public class LogRecord {

	private long id ;

	private int userId ;

	private int appId ;
```

```
    /**
     * 请求插入此日志的服务器 IP
     */
    private String appIP ;

    private Date createdTime ;

    /**
     * 自定义属性的值。
     */
    private Map<String, Object> otherProps = new HashMap<String, Object>() ;

    public long getId() {
            return id;
    }

    public void setId(long id) {
            this.id = id;
    }

    public int getUserId() {
            return userId;
    }

    public void setUserId(int userId) {
            this.userId = userId;
    }

    public int getAppId() {
            return appId;
    }

    public void setAppId(int appId) {
            this.appId = appId;
    }

    public Date getCreatedTime() {
            return createdTime;
    }

    public void setCreatedTime(Date createdTime) {
            this.createdTime = createdTime;
    }
```

```
	public Map<String, Object> getOtherProps() {
		return otherProps;
	}

	public void setOtherProps(Map<String, Object> otherProps) {
		this.otherProps = otherProps;
	}

	public String getAppIP() {
		return appIP;
	}

	public void setAppIP(String appIP) {
		this.appIP = appIP;
	}

}
```

使用者使用方式如下(查询 springMVC):

```
	public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse
response) throws Exception {
		//yyyy-MM-dd HH:mm:ss
		String startTime = request.getParameter("startTime") ;
		String endTime = request.getParameter("endTime") ;
		int userId = RequestUtil.getParameterAsInt(request, "userId", -1) ;
		int pageNo = RequestUtil.getParameterAsInt(request, "pageNo", 1) ;

		LinkedList<String> conditions = new LinkedList<String>() ;
		if(userId > 0){
			conditions.addLast("userId=" + userId) ;
		}

		if(StringUtil.notEmpty(startTime)){
			conditions.addLast("createdTime>=" + startTime) ;
		}

		if(StringUtil.notEmpty(endTime)){
			conditions.addLast("createdTime<=" + endTime) ;
		}

		HashMap<String, Object> params = new HashMap<String, Object>() ;
		params.put("appId", appId) ;

		PageFlip logs = this.appLogService.queryLogs(conditions, "id asc", pageNo,
```

```
20) ;

                if(logs != null){
                        logs.setFlipURL(request, "pageNo") ;
                        params.put("logs", logs) ;
                }

                Map<String, String> customProperties =
this.appLogService.queryCustomPropsMetaInfo() ;
                Set<String> customPropNames = customProperties.keySet() ;

                params.put("customProperties", customProperties) ;
                params.put("customPropNames", customPropNames) ;

                return new ModelAndView("/console/log/logRecordList", params);
        }
```

获取数据，然后传给 jsp 做显示，jsp 如下：

```
        <hr>
        <form>
                用户编号：<input name="userId" type="input" value="${param.userId}" />
                开始时间：<input name="startTime" type="input" value="${param.startTime}" />
                结束时间：<input name="endTime" type="input" value="${param.endTime}" />

                  <input type="submit" value="检索" />
        </form>
    <hr>

    <c:if test="${not empty logs}">
    <table border="1" width="96%">
                <tr>
                        <th>序号</th>
                        <th>用户编号</th>
                        <c:forEach items="${customPropNames}" var="m_propName">
                        <th>${customProperties[m_propName]}</th>
                        </c:forEach>
                        <th>记录时间</th>
                </tr>
        <c:forEach items="${logs.elements}" var="m_log">
                <tr>
                        <td><c:out value="${logs.index}" /></td>
                        <td><c:out value="${m_log.userId}" /></td>

                        <c:forEach items="${customPropNames}" var="m_propName">
```

```
                    <td><c:out value="${m_log.otherProps[m_propName]}" /></td>
                    </c:forEach>

                    <td><fmt:formatDate value="${m_log.createdTime}"
pattern="yyyy-MM-dd HH:mm:ss" /></td>
                </tr>
            </c:forEach>
        </table>

        <table border="1" width="96%">
            <tr>
                    <c:import url="/WEB-INF/jsp/include/console_flip.jsp" />
                </tr>
        </table>
        </c:if>
```

插入日志，可以定义一个自己的接口，传入需要的参数。在接口实现处，将自定义的属性存放到 Map 中，和查询一样，调用客户端 API。

服务端-API 实现

对于应用和应用自定义属性的管理，只是普通的增删改查操作，不再赘述。实现服务器端的难点有两个，一是实现自定义属性存储按照字段存储到表中，在存储的过程中需要完成属性名到数据库字段名的映射，分表，以及数据类型的转换。另外一个是实现基于”属性 + 操作符 + 值“到 sql 语句的转换查询，并避免 SQL 注入的风险。

对于自定义属性到表字段的映射和切分，我们采用 guzz CustomTableView。

编写 LogRecord 详细日志记录的 CustomTableView：

```
/**
 * 每个日志应用一张表。通过 Manager 的版本系统，控制 ORM 缓存。
 */
public class LogRecordCustomTableView extends AbstractCustomTableView implements
ExtendedBeanFactoryAware {

        private ILogAppManager logAppManager ;

        private Map<Integer, MappingHolder> cachedMapping = new HashMap<Integer,
MappingHolder>() ;

        protected int getTableCondition(Object tableCondition){
                return ((Integer) tableCondition).intValue() ;
        }
```

```
	protected void initCustomTableColumn(POJOBasedObjectMapping mapping, Object
tableCondition) {
		int appId = getTableCondition(tableCondition) ;

		List<LogCustomProperty> properties =
logAppManager.listLogCustomProperties(appId) ;

		for(LogCustomProperty p : properties){
			TableColumn tc = super.createTableColumn(mapping, p.getPropName(),
p.getColName(), p.getDataType(), null) ;
			super.addTableColumn(mapping, tc) ;
		}
	}

	public POJOBasedObjectMapping getRuntimeObjectMapping(Object tableCondition) {
		int appId = getTableCondition(tableCondition) ;

		MappingHolder holder = this.cachedMapping.get(appId) ;
		int newVersion = this.logAppManager.getLastestVersion(appId) ;

		//不需要非常严格的版本和读取一致性，基本一致就能达到要求。
		if(holder == null || holder.version != newVersion){
			POJOBasedObjectMapping newMap =
super.createRuntimeObjectMapping(tableCondition) ;
			holder = new MappingHolder(newMap, newVersion) ;

			this.cachedMapping.put(appId, holder) ;
		}

		return holder.mapping ;
	}

	public Object getCustomPropertyValue(Object beanInstance, String propName) {
		LogRecord record = (LogRecord) beanInstance ;

		return record.getOtherProps().get(propName) ;
	}

	public void setCustomPropertyValue(Object beanInstance, String propName, Object
value) {
		LogRecord record = (LogRecord) beanInstance ;

		record.getOtherProps().put(propName, value) ;
```

```
        }

        public String toTableName(Object tableCondition) {
                return super.getConfiguredTableName() + "_" +
getTableCondition(tableCondition) ;
        }

        public void setExtendedBeanFactory(ExtendedBeanFactory extendedBeanFactory) {
                this.logAppManager = (ILogAppManager)
extendedBeanFactory.getBean("logAppManager") ;
        }

        static class MappingHolder{

                public MappingHolder(POJOBasedObjectMapping mapping, int version){
                        this.mapping = mapping ;
                        this.version = version ;
                }

                public final POJOBasedObjectMapping mapping ;

                public final int version ;

        }

}
```

通过自定义属性表，动态生成 ORM，以及表分切。表分切的条件为 appId，也就是应用的编号。我们通过 Manager 提供的版本控制，确保集群内所有机器同步刷新自定义属性。

将 LogRecordCustomTableView 配置给 LogRecord 完成自定义属性的持久层支持。配置：

```
@javax.persistence.Entity
@Table(name="gs_log_record", shadow=LogRecordCustomTableView.class)
public class LogRecord {

        .....
```

这样，日志模块就完成了根据传入的 Map，将数据分切存储到数据库单独的表中，并保持 Map 的每项值在一个单独的数据库字段中。

现在我们来处理 List<String>的表达式查询。

表达式的解析一直是一件很麻烦的事情，这里我们直接使用 guzz 标签的表达式定义和解析代码来处理。首先获取日志对象的 BusinessInterpreter 以及 ORM 定义，然后直接逐条翻译即可：

```
	public PageFlip queryLogs(String appIP, String secureCode, List<String> conditions,
int pageNo, int pageSize, String orderBy){
		int appId = this.logAppManager.getAppIdBySecureCode(secureCode) ;

		//not exist.
		if(appId < 1){
			throw new ServiceExecutionException("unknown secure code:[" +
secureCode + "] from app server:" + appIP) ;
		}

		if(pageSize > maxPageSize){
			pageSize = maxPageSize ;
		}

		LinkedList<Object> terms = new LinkedList<Object>() ;
		BusinessInterpreter gi
=  super.getGuzzContext().getBusiness(LogRecord.class.getName()).getInterpret() ;
		ObjectMapping mapping =
super.getGuzzContext().getObjectMappingManager().getObjectMapping(LogRecord.class.getN
ame(), appId) ;

		if(conditions != null && !conditions.isEmpty()){
			for(int i = 0 ; i < conditions.size() ; i++){
				Object condition = conditions.get(i) ;

				try {
					if(condition != null){
						Object mc = gi.explainCondition(mapping,
condition) ;

						if(mc != null){
							terms.addLast(mc) ;
						}
					}
				} catch (Exception e) {
					throw new ServiceExecutionException("error to
translate condition:[" + condition + "], msg:" + e.getMessage()) ;
				}
			}
		}

		if(terms.isEmpty()){
```

```
                        return null ;
                }else{
                        //query
                        SearchExpression se = SearchExpression.forClass(LogRecord.class,
pageNo, pageSize) ;
                        se.setTableCondition(appId) ;
                        se.and(terms) ;
                        if(StringUtil.notEmpty(orderBy)){
                                se.setOrderBy(orderBy) ;
                        }

                        return super.page(se) ;
                }
        }
```

大功告成。 我们使用 guzzservices 的 RPC 服务，将服务器端注册到远程调用的接收接口中，这样就可以从客户端调用了：

```
        public void setCommandServerService(CommandServerService css){
                css.addCommandHandler(AppLogServiceImpl.COMMAND_NEW_LOG, handler) ;
                css.addCommandHandler(AppLogServiceImpl.COMMAND_QUERY_LOG, handler) ;
                css.addCommandHandler(AppLogServiceImpl.COMMAND_QUERY_META, handler) ;
        }

        private final CommandHandler handler = new CommandHandlerAdapter(){

                public String executeCommand(ClientInfo client, String command, String
param) throws Exception {
                        if(AppLogServiceImpl.COMMAND_NEW_LOG.equals(command)){
                                Map<String, Object> params = JsonUtil.fromJson(param,
HashMap.class) ;
                                String scode = (String)
params.remove(AppLogServiceImpl.KEY_APP_SECURE_CODE) ;
                                Integer userId = (Integer)
params.remove(AppLogServiceImpl.KEY_APP_USER_ID) ;

                                LogRecord r = new LogRecord() ;
                                r.setOtherProps(params) ;

                                insert(client.getIP(), scode, userId.intValue(), r) ;

                                return null ;

                        }else if(AppLogServiceImpl.COMMAND_QUERY_LOG.equals(command)){
```

```
                        AppLogQueryRequest request = JsonUtil.fromJson(param,
AppLogQueryRequest.class) ;

                        PageFlip data = queryLogs(client.getIP(),
request.getSecureCode(), request.getConditions(), request.getPageNo(),
request.getPageSize(), request.getOrderBy()) ;

                        if(data == null) return null ;

                        return JsonPageFlip.fromPageFlip(data,
LogRecord.class).toJson() ;
                    }else if(AppLogServiceImpl.COMMAND_QUERY_META.equals(command)){
                        String secureCode = param ;
                        List<LogCustomProperty> props =
queryCustomProperties(client.getIP(), secureCode) ;

                        HashMap<String, String> mis = new HashMap<String, String>() ;
                        for(LogCustomProperty p : props){
                            mis.put(p.getPropName(), p.getDisplayName()) ;
                        }

                        return JsonUtil.toJson(mis) ;
                    }

                    return null ;
                }

        } ;
```

客户端使用示例

客户端调用实例。 当然一般情况下，建议对 AppLogService 客户端 API 做一层封装，调用者像是用普通的 Log 服务一样传入所有参数，封装的地方再装换成 Map 等进行调用。示例：

```
AppLogService appLogService = (AppLogService)
GuzzWebApplicationContextUtil.getGuzzContext(session.getServletContext()).getService("
appLogService") ;

java.util.HashMap<String, Object> props = new java.util.HashMap<String, Object>() ;
props.put("userIP", "11.22.33.44") ;
props.put("moveSpeed", 1.45f) ;

appLogService.insertLog(12345, props) ;
```

```
java.util.LinkedList<String> cs = new java.util.LinkedList<String>() ;
cs.add("userId=12345") ;

org.guzz.dao.PageFlip data = appLogService.queryLogs(cs, "id asc", 1, 20) ;
request.setAttribute("data", data) ;

out.println("<hr/>") ;

%>
<table border="1" width="96%">
	<tr>
		<th>序号</th>
		<th>userIP</th>
		<th>moveSpeed</th>
		<th>createdTime</th>
	</tr>
	<c:forEach items="${data.elements}" var="m_log">
		<tr>
			<td><c:out value="${data.index}" /></td>
			<td><c:out value="${m_log.otherProps.userIP}" /></td>
			<td><c:out value="${m_log.otherProps.moveSpeed}" /></td>
			<td><c:out value="${m_log.createdTime}" /></td>
		</tr>
	</c:forEach>
</t
```