火龙果•整理
uml.org.cn

基于SSHI架构的开发平台技术方案

概 述

平台的技术概述

- 基于 J2EE 平台软件的终级解决方案，采用最新的 J2EE 技术
- 平台特点说明：技术先进、架构安全、稳定、高效、易快速上手
- 底层为成熟的 J2EE 开源技术框架 Spring3、Struts2、Hibernate3.6
- 数据访问层采用 Hibernate 结合 Ibatis，整合这两种 Orm 框架的优点，高效稳定
- 安全控制框架采用成熟稳定的 Spring Security 3
- 前台基于 Js、Jquery 开源组件、josn、Ajax、Dhtml 等技术
- 采用标准的 J2EE 规范，支持全部流行的商业应用服务器和其它 Web 容器比如：WebSphere、WebLogic、Tomcat、Jboss、等
- 注释配置采用 JSR-250 规范定义的注释标记
- 支持流行的关系型数据库：Oracle、Db2、MySQL 等
- 提供 CXF WebService 接口，面向 SOA，可与其它系统完美整合
- 集成 JBOSS RULE 5 规则引擎集成，灵活业务处理
- 后台大量封装好的基础类，提供简单、易用的调用方式
- 前台提供完整的管理系统开发框架，基于 JQuery 技术
- 包括：表格控件，树形菜单，树形表格，弹出框，对话框，选项卡，日期组件、日历组件、图表组件、报表组件、表单验证框架等
- 平台集成系统管理模块，包括：(1)服务器系统信息监控 (2)组织机构管理 (3)用户管理 (4)角色管理 (5)资源菜单管理 (6)数据字典管理 (7)在线用户管理(8)应用级缓存管理

总结：基于本开发平台可快速开发基于 J2EE 轻量级架构的企业应用软件、企业门户网站、电子商务网站，以及各种管理系统。

平台的定位与扩展性

- 面向中小型软件公司、企业 IT 部门研发团队
- 完整的资源管理平台
- 支持规则引擎
- 完整的 Demo 实例，开发人员可快速上手
- 技术开放
- 支持多语言
- 支持多数据库
- 支持各种 web 容器、商业中间件

- 支持多操作系统
- 支持工作流自定义
- 支持可视化表单自定义
- 支持工作流二次开发
- 支持企业快速成立开发团队
- 集成权限资源控制
- 支持缓存管理
- 支持分布式处理
- 完全知识产权，提供技术支持

目录

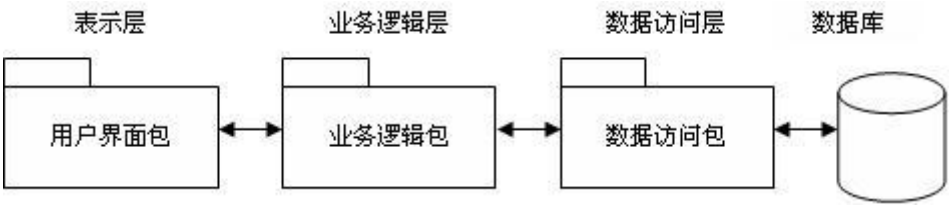
基于SSHI架构的开发平台技术方案 0
简介 1
目录 3
一、 基于Spring Framework 的分层管理 6
1. 分层例图 6
2. Service 实现类继承方式 6
3. Action 控制类实现 6
1) 实现Grid分页控制,继承PagingSupportAction 6
2) 非Grid式分页，继承PaginationAction 7
3) 一般操作，继承BaseAction 7
二、 基于Spring Security3 的安全权限管理 7
1. 自定义登陆认证 7
2. 默认的Url访问认证 7
1) 自定义UserDetailsServiceImpl 8
2) 自定义退出认证 8
三、 基于Spring Framework 的事务管理 8
1. 事务管理配置 8
2. 事务声明 8
四、 基于Spring IOC实现的annotations配置 9
1. ServiceBean的声明 9
2. ServiceBean的调用 9
3. Controller Bean的声明 9
4. Controller Bean的调用 9
五、 基于Hibernate Entity的annotations配置 10
1. Hibernate Entity的声明 10
2. Entity的使用 10
六、 基于Ibatis的联合查询配置 10
1. 命名空间 10
2. Vo的声明（亦可称为领域驱动模型） 11
3. 实例 1 查询记录数，返回Intger 11
4. 实例 2 查询列表,返回List数据集 11
七、 基于EHCache的缓存管理机制 12
1. 主要的特性 12
2. 通过cacheContext.xml对EHCache的配置 13
八、 基于Quartz的任务调度管理 14
1. Service Bean实现任务调度的声明 14
2. 定义调度时间 14
3. 管理调度器的总设置 14
4. 符调度时间表达式定义的说明 14
九、 基于Log4j日志管理 16
1. 日志管理配置 16

2. 程序中调用.....16
十、 基于Junit4.7 的测试平台.....17
1. 平台提供独立的test包结构.....17
2. 一个简单的测试类.....17
3. 重点说明.....17
十一、 基于Ftp4j的ftp文件传输接口.....18
1. Ftp Server 配置信息.....18
2. 程序中的调用实例.....18
十二、 基于Cxf的Web Service实现.....19
1. 配置.....19
2. 实例.....19
3. 引用.....19
十三、 应用服务的启动机制.....19
1. 设计理念.....19
2. 扩展性.....20
十四、 异常、错误信息处理机制.....20
1. 所有的异常都在Acton里捕获并抛出提示信息如下.....20
2. 统一的提示信息处理.....20
十五、 统一的Ctx上下文管理.....21
1. 应用方式.....21
2. 设计理念.....21
十六、 基于Session级别的用户登陆信息管理.....21
1. 用户信息的定义及存贮.....21
2. 用户信息的获取.....22
十七、 基于ServletContext的多系统Session共享方案.....22
1. 应用 1：.....22
2. 应用 2：.....22
十八、 基于Struts2 的国际化管理.....22
1. 采用标准的Struts2 国际化管理定义.....22
2. 使用方式.....23
十九、 基于JbossRules 5 的规则引擎集成.....23
1. Rules 脚本实例.....23
2. 在Java程序中引用实例.....24
3. 总结.....24
二十、 基于Apache Lucene全文检索引擎集成.....24
1. Lucene分析图.....25
2. 实例.....26
二十一、 基于Jquery 的Web前端组件.....27
1. 基于JS的Web界面框架.....27
2. 基于Jquery的Grid实现.....27
3. 其它页面组件.....28
4) 树形菜单演示.....28
5) 树形表格演示.....29
6) 弹出信息框组件.....29

7） 弹出式操作框.....30
8） 日期控件.....30
9） 日历控件.....31
10） 自定义桌面组件.....31
11） 基于Open Flash Chart 的图表控件.....32

一、 基于Spring Framework 的分层管理

1. 分层例图



平台已把数据库访问层封装起来，一般程序员不用关心这一层，程序员所有做的事情是在表示层接收数据在业务逻辑层（Service 层）实现具体的业务逻辑编码。

2. Service 实现类继承方式

Service 接口只需继承 **BaseService** 所提供的数据访问接口可以实现所有的数据操作。
例：

```
@Service
@SuppressWarnings("unchecked")
public class RoleServiceImpl extends BaseService implements IRoleService
{
    public void addRoleModel(RoleModel entity) {
        this.save_Hibernate_Object(entity);
    }
    public void updateRoleModel(RoleModel entity) {
        this.update_Hibernate_Object(entity);
    }
    //…
}
```

3. Action 控制类实现

基类 Action 提供了三个 Base 的实现

1） 实现Grid分页控制，继承PagingSupportAction

```
/**
 *订单管理Action
 */
@Controller("orderBlankAction_Id")
@Scope("prototype")
public class OrderBlankAction extends PagingSupportAction {
```

2） 非Grid式分页，继承PaginationAction

```
/**
 * 产品搜索Action
 */
@Controller("productAction_Id")
@Scope("prototype")
public class ProductAction extends PaginationAction{
```

3） 一般操作，继承**BaseAction**

```
/**
 * 图表分析 Action
 */
@Controller("openFlashChartAction _Id")
@Scope("prototype")
public class OpenFlashChartAction extends BaseAction {
```

二、 基于Spring Security3 的安全权限管理

1. 自定义登陆认证

```
<!-- 自定义登陆认证 -->
<custom-filter before="FORM_LOGIN_FILTER"
ref="usernamePasswordAuthenticationFilter" />
```

继承于 UsernamePasswordAuthenticationFilter,增加了对验证码的支持、未授于角色的用户认证、未启用用户的认证、以及登陆成功后登陆信息的记录和用户 Session 信息的记录

2. 默认的Url访问认证

```
<!-- 默认的认证-->
<custom-filter after="FILTER_SECURITY_INTERCEPTOR" ref="filterSecurity"
/>
```

根据角色代码从缓存资源类中取得对应的资源列表，来控制对未授权 Url 的控制

1） 自定义UserDetailsServiceImpl

用以取得登陆用户信息，载入用户授权的资源标记（resLable）装入 GrantedAuthority[]
用以控制用户未授权的资源访问

2） 自定义退出认证

```
<!-- 自定义退出认证 -->
<custom-filter position="LOGOUT_FILTER" ref="logoutFilter" />
```

在用户退出系统时销毁其 Session 并在 sessionRegistry 中移除其 SessionID

三、 基于Spring Framework 的事务管理

1. 事务管理配置

```
    <bean id="txManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager
">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <!-- 基于@Transactional注解方式的事务管理 -->
    <tx:annotation-driven transaction-manager="txManager" />
```

2. 事务声明

只需要在 Service 实现类的方法前加`@Transactional`注解，在 Service 里控制所有异常信息要抛出 ServiceException 异常类，在 Action 用以捕获异常并处理

```
    /**
     * 提交审批
     */
    @Transactional
    public void submitApprove(Map map)
```

四、 基于Spring IOC实现的annotations配置

1. ServiceBean的声明

上面说到所有的Serive实现类都继承于BaseService,注解此service只需要类的上面加上@Service标签

```
@Service
@SuppressWarnings("unchecked")
public class RoleServiceImpl extends BaseService implements IRoleService
```

2. ServiceBean的调用

Action 调用只需在接口定义的地方加下@Resource 标签

```
    @Resource
    private IRoleService roleService;
```

Service 调用其它 serice 亦是如此

哎：声明调用如此简单

3. Controller Bean的声明

以下在Action类头部加上@Controller标签已完成对该控制类的注解，需要说明的是Action的声明必需要加上作用域：@Scope("prototype")，为非单例模式

```
/**
 * Example Action
 */
@Controller("exampleAction_Id")
@Scope("prototype")
public class ExampleAction extends BaseAction
```

4. Controller Bean的调用

控制类的调用就需要到 Struts2 的配置文件中了,如此：

```
<action name="exampleAction_*" class="exampleAction_Id" method="{1}">
            <result
name="exampleIndex">/WEB-INF/jsp/example/ExampleIndex.jsp</result>
        </action>
```

其中 class="exampleAction_Id" 已完成对 Action 的调用

哎：声明调用也很简单

五、 基于Hibernate Entity的annotations配置

1. Hibernate Entity的声明

在 Spring 追求零配置的时代已来临，Hibernate 也是紧随其后，其实体类的声明也变的如此简单：

```
/**
 * 供能描述：订单表实体
 */
@Entity
@Table(name = "T_EXAMPLE_ORDERBLANK")
public class OrderBlankModel extends BaseModel {
    @Id
    @GeneratedValue(generator = "generator")
    @GenericGenerator(name = "generator", strategy = "assigned")
    @Column(name = "orderBlankID", unique = true, nullable = false,
precision = 36, scale = 0)
    private String orderBlankID;// 订单ID
    @Column(name = "orderBlankNO")
    private String orderBlankNO;// 订单编号
    @Column(name = "orderBlankName")
    private String orderBlankName;// 订单名称
//以下自动生成 set get 方法
```

2. Entity的使用

使用和采用.hbm.xml 配置文件的形式一样

注：Hibernate Entity 的命名以 Model 结尾

六、 基于Ibatis的联合查询配置

1. 命名空间

<sqlMap namespace="example">

2. Vo的声明（亦可称为领域驱动模型）

<typeAlias alias="productVo" type="com.ckm.example.vo.ProductVo" /><!-- 产品 VO -->

3. 实例 1 查询记录数，返回Intger

```
<select id="queryProductListCounctSql" parameterClass="hashmap"
resultClass="integer">
         select count(0)
             from t_example_products t, t_example_insurer i
            where t.insurer_id = i.insurer_id
              and t.disabled = '0'
       <isNotEmpty prepend="AND" property="productName">
           t.product_name like '%$productName$%'
       </isNotEmpty>
       <isNotEmpty prepend="AND" property="insurerName">
           i.insurer_name like '%$insurerName$%'
       </isNotEmpty>
    </select>
```

4. 实例 2 查询列表,返回List数据集

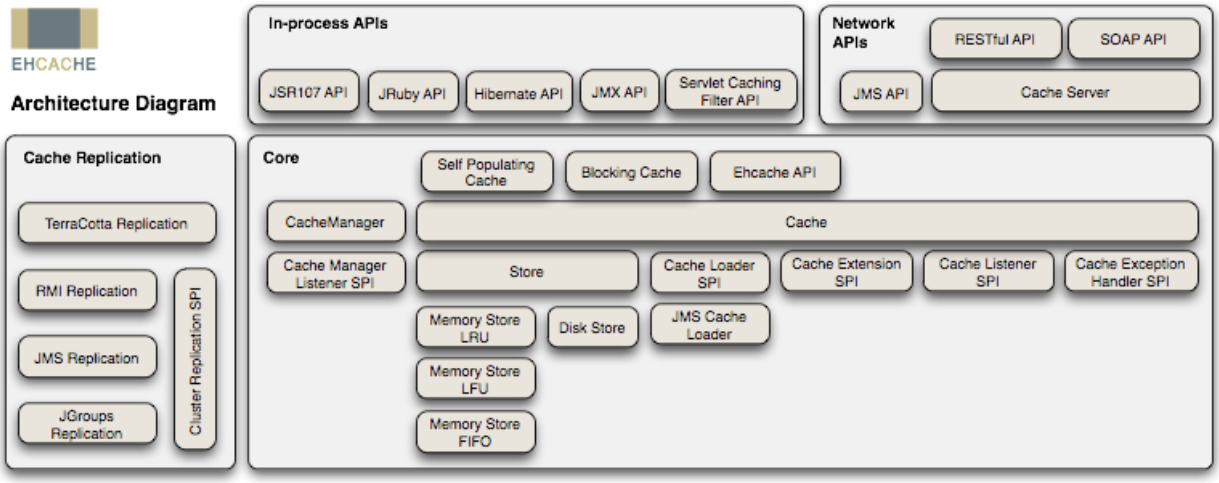
```
<select id="queryProductListSql" parameterClass="hashmap"
resultClass="productVo">
       select t.product_id as productid,
             t.insurer_id as insurerid,
             t.product_name as productname,
             t.prod_name_en as prodnameen,
             t.shortname as shortname,
             t.product_code as productcode,
             t.inner_code as innercode
         from t_example_products t, t_example_insurer i
        where t.insurer_id = i.insurer_id
          and t.disabled = '0'
       <isNotEmpty prepend="AND" property="productName">
           t.product_name like '%$productName$%'
       </isNotEmpty>
       <isNotEmpty prepend="AND" property="insurerName">
           i.insurer_name like '%$insurerName$%'
       </isNotEmpty>
    </select>
```

注：Vo 的命名方式以 Vo 结尾，记录数 Sql name 的命名方式以 ListCounctSql 对尾，列表 Sql name 的命名方式以 ListSql 结尾，参数传递以 hashmap 做为对象

七、 基于EHCache的缓存管理机制

EhCache 是一个纯 Java 的进程内缓存框架，具有快速、精干等特点，是 Hibernate 中默认的 CacheProvider。

下图是 Ehcache 在应用程序中的位置



1. 主要的特性

1. 快速.
2. 简单.
3. 多种缓存策略
4. 缓存数据有两级：内存和磁盘，因此无需担心容量问题
5. 缓存数据会在虚拟机重启的过程中写入磁盘
6. 可以通过 RMI、可插入 API 等方式进行分布式缓存
7. 具有缓存和缓存管理器的侦听接口
8. 支持多缓存管理器实例，以及一个实例的多个缓存区域
9. 提供 Hibernate 的缓存实现
10. 等等

在使用完 Ehcache 后，必须要 shutdown 缓存。Ehcache 中有自己的关闭机制，不过最好在你的代码中显示调用

CacheManager.getInstance().shutdown();

1.EhCache 是什么

EhCache 是 Hibernate 的二级缓存技术之一，可以把查询出来的数据存储在内存或者磁盘，节省下次同样查询语句再次查询数据库，大幅减轻数据库压力；

2.EhCache 的使用注意点

当用 Hibernate 的方式修改表数据(save,update,delete 等等)，这时 EhCache 会自动把缓存中关于此表的所有缓存全部删除掉(这样能达到同步)。但对于数据经常修改的表来说，可能就失去缓存的意义了(不能减轻数据库压力)；

2. 通过cacheContext.xml对EHCache的配置

```
<bean id="methodCachePointCut"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
     <property name="advice">
       <ref local="methodCacheInterceptor"/>
     </property>
       <property name="patterns">
           <list>
               <value>.*ServiceImpl\.*getChannelList.*</value>
               <value>.*ServiceImpl\.*getMenuList.*</value>
               <value>.*ServiceImpl\.*load.*</value>
           </list>
       </property>
   </bean>
   <bean id="methodCachePointCutAdvice"
class="org.springframework.aop.support.RegexpMethodPointcutAdvisor">
     <property name="advice">
       <ref local="methodCacheAfterAdvice"/>
     </property>
     <property name="patterns">
       <list>
           <value>.*ServiceImpl\.*add.*</value>
           <value>.*ServiceImpl\.*save.*</value>
           <value>.*ServiceImpl\.*delete.*</value>
           <value>.*ServiceImpl\.*update.*</value>
       </list>
     </property>
   </bean>
```

缓存频道 `getChannelList` 和菜单 `getMenuList` 信息，缓存以 load 开头的 Service 方法

在 Service 里执行 add、save、delete、update 开头的方法时清除缓存

八、 基于Quartz的任务调度管理

1. Service Bean实现任务调度的声明

```
<!-- ************************************Quartz配置****************************************** -->
<!-- 要调用的工作类 -->
<bean id="quartzJob" class="com.ckm.common.job.KmJob"></bean>
<!-- 定义调用对象和调用对象的方法 -->
<bean id="jobtask" class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">
    <!-- 调用的类 -->
    <property name="targetObject">
        <ref bean="quartzJob" />
    </property>
    <!-- 调用类中的方法 -->
    <property name="targetMethod">
        <value>work</value>
    </property>
</bean>
```

2. 定义调度时间

```
<!-- 定义触发时间 -->
<bean id="doTime" class="org.springframework.scheduling.quartz.CronTriggerBean">
    <property name="jobDetail">
        <ref bean="jobtask" />
    </property>
    <property name="cronExpression">
        <value>0 48 13 * * ?</value>
    </property>
</bean>
```

3. 管理调度器的总设置

```
<!--管理触发器的总设置,管理我们的触发器列表,可以在bean的list中放置多个触发器。-->
<bean id="startQuertz" lazy-init="false" autowire="no" class="org.springframework.scheduling.quartz.SchedulerFactoryBean">
    <property name="triggers">
        <list>
            <ref bean="doTime"/>
        </list>
    </property>
</bean>
```

4. 符调度时间表达式定义的说明

"0 0 12 * * ?" 每天中午12点触发
"0 15 10 ? * *" 每天上午10:15触发
"0 15 10 * * ?" 每天上午10:15触发
"0 15 10 * * ? *" 每天上午10:15触发
"0 15 10 * * ? 2005" 2005年的每天上午10:15触发

"0 * 14 * * ?" 在每天下午2点到下午2:59期间的每1分钟触发
"0 0/5 14 * * ?" 在每天下午2点到下午2:55期间的每5分钟触发
"0 0/5 14,18 * * ?" 在每天下午2点到2:55期间和下午6点到6:55期间的每5分钟触发
"0 0-5 14 * * ?" 在每天下午2点到下午2:05期间的每1分钟触发
"0 10,44 14 ? 3 WED" 每年三月的星期三的下午2:10和2:44触发
"0 15 10 ? * MON-FRI" 周一至周五的上午10:15触发
"0 15 10 15 * ?" 每月15日上午10:15触发
"0 15 10 L * ?" 每月最后一日的上午10:15触发
"0 15 10 ? * 6L" 每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6L 2002-2005" 2002年至2005年的每月的最后一个星期五上午10:15触发
"0 15 10 ? * 6#3" 每月的第三个星期五上午 10:15 触发

秒（0~59）

分钟（0~59）

小时（0~23）

天（月）（0~31，但是你需要考虑你月的天数）

月（0~11）

天（星期）（1~7 1=SUN 或 SUN，MON，TUE，WED，THU，FRI，SAT）

一个完整的例子

“0 0 12 ? * WED”表示每个星期三中午 12 点

字段 允许值 允许的特殊字符
秒 0-59 , - * /
分 0-59 , - * /
小时 0-23 , - * /
日期 1-31 , - * ? / L W C
月份 1-12 或者 JAN-DEC , - * /
星期 1-7 或者 SUN-SAT , - * ? / L C #
年（可选） 留空, 1970-2099 , - * /

九、 基于Log4j日志管理

1. 日志管理配置

日志管理采用成熟的 Log4j，通过配置文件 log4j_config.xml

一般级别信息

```
<appender name="INFO_FILE" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/usr/ckm/log/ckm-i.log" />
    <param name="Append" value="true" />
    <param name="MaxFileSize" value="500KB" />
    <param name="MaxBackupIndex" value="2" />
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%r [%thread  %X{txSymbol} ] %p %c [%x] - %m%n" />
    </layout>
    <filter class="com.ckm.log.tx.log4j.TxTraceFilter" />
    <filter class="org.apache.log4j.varia.LevelRangeFilter">
        <param name="LevelMin" value="DEBUG" />
        <param name="LevelMax" value="FATAL" />
    </filter>
</appender>
```

错误信息

```
<appender name="ERROR_FILE" class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="/usr/ckm/log/ckm-e.log" />
    <param name="Append" value="true" />
    <param name="MaxFileSize" value="500KB" />
    <param name="MaxBackupIndex" value="2" />
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%r [%thread %t %X{txSymbol} ] %p %c [%x] - %m%n" />
    </layout>
    <filter class="com.ckm.log.tx.log4j.TxTraceFilter" />
    <filter class="org.apache.log4j.varia.LevelRangeFilter">
        <param name="LevelMin" value="ERROR" />
        <param name="LevelMax" value="FATAL" />
    </filter>
</appender>
```

2. 程序中调用

定义：

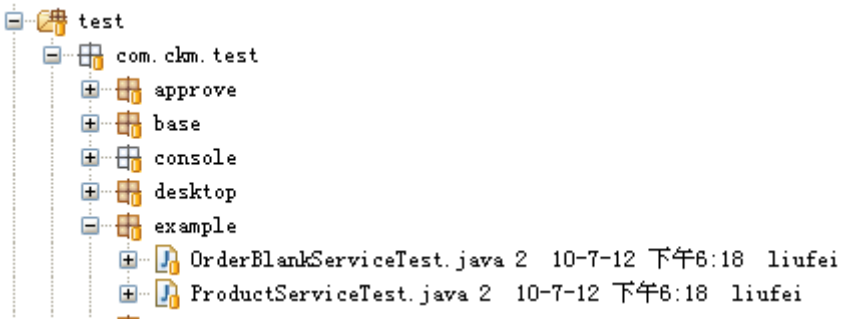
```
private static final Log logger = LogFactory.getLog(TaskAction.class);
```

使用：

```
try {
    taskService.deleteTaskModel(this.getIdArray(), this.getUserID());
    this.setReturnInfo(true);
} catch (Exception e) {
    this.setReturnInfo(false);
    this.setReturnError(e.getMessage());
    logger.error(e.toString());
}
return SUCCESS;
```

十、 基于Junit4.7 的测试平台

1. 平台提供独立的test包结构



2. 一个简单的测试类

```
package com.ckm.test.example;

import javax.annotation.Resource;

public class OrderBlankServiceTest extends BaseTest {
    @Resource
    private IOrderBlankService orderBlankService = null;

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void test01() {
        System.out.println(orderBlankService.getOrderBlankModelList().size());
    }
}
```

3. 重点说明

所有的测试类只需继承 BaseTest 基类

通过 annotation 方式注入 Service

测试方法的注解为@Test，例如 2)的 test01 方法

十一、 基于Ftp4j的ftp文件传输接口

对 ftp 的支持是基于开源的 Ftp4j 支持 socket4、socket5、http、ftp 代理
并对Ftp4j进行扩展支持异步传输模式

1. Ftp Server 配置信息

```
##########################################FTP服务器参数设置##########################################
#ftp服务器地
ftpServerIp = 192.168.0.1
#ftp服务器端口
ftpServerPort = 21
#ftp服务器用户登陆名
ftpClientUser = ftpins
#ftp服务器用户密码
ftpClientPass = ftpins
#是否使用代理连接 true为使用，false为不使用
isProxy = false
#ftp服务器代理服务器类型 SOCKS4 = "SOCKS4"; SOCKS5 = "SOCKS5"; HTTP = "HTTPT"; FTP = "FTPP";
proxyType = SOCKS4
#代理服务器地址
proxyHost = 127.0.0.1
#代理服务器端口
proxyPort = 1080
#代理服务器用户登陆名
proxyUser = anonymous
#代理服务器用户密码
proxyPass = ftp4j
```

2. 程序中的调用实例

同步传输

```
try {
    FtpClientApp ftpClientApp = new FtpClientApp();
    ftpClientApp.upload("d/log.jar", content, path);
    ftpClientApp.close();
} catch (FtpConnectException e) {
    e.printStackTrace();
}
```

异步传输

```
public static void main(String[] args) throws FtpConnectException, TransfersException {
    UpLoadThread opLoadThead = new UpLoadThread("d:/log.jar","/111/222");
    opLoadThead.start();
    opLoadThead.close();
}
```

十二、 基于Cxf的Web Service实现

1. 配置

2. 实例

3. 引用

十三、 应用服务的启动机制

1. 设计理念

基于可扩展性、方便、安全的理念
框加基础类提供接口 IappInit，在应用程序启动时会加载其接口方法 init()

```
public class AppInitImpl implements IAppInit {

    public void init(Properties props, ApplicationContext context) {
        // 装载配置文件信息
        PropertiesUtils.setProps(props);
        // 初始化日志数据
        initLog();
        // 初始化系统缓存资源
        initCacheData(context);
    }
```

在应程序启动时会默认加载配置文件信、日志数据、缓存资源
其中缓存资源包括：初始化角色资源、初始化数据字典

2. 扩展性

扩展的需要只需在 AppInitImpl 这个实现类里注入要加载的资源。

十四、 异常、错误信息处理机制

1. 所有的异常都在Acton里捕获并抛出提示信息如下

```
try {
    taskService.deleteTaskModel(this.getIdArray(), this.getUserID());
    this.setReturnInfo(true);
} catch (Exception e) {
    this.setReturnInfo(false);
    this.setReturnError(e.getMessage());
    logger.error(e.toString());
}
```

执行成功：`this.setReturnInfo(true);`

执行失败：`this.setReturnInfo(false);`

返回错误信息：`this.setReturnError(e.getMessage());`

对异常记录到日志中：`logger.error(e.toString());`

2. 统一的提示信息处理

在 Jsp 中的处理：

```
var ids = getCheckedId();
$.postJSON(delUrl,{ids:ids},function(json){
    if(json.success){//操作成功
        delSuccessCue();//删除成功
        loadList();//刷新数据列表
        refButton();//刷新按钮
    }else{
        errorCue(json.error);//操作失败，弹出错误信息
    }
});
```

errorCue(json.error);执行的效果

十五、 统一的Ctx上下文管理

1. 应用方式

```
@Service
public class OrderBlankServiceImpl extends BaseService implements IOrderBlankService {

@Resource
private IOrderBlankService orderBlankService;
```

默认的 Service Bean 管理方式只需要在 ServiceImpl 注解@Service 标签，在引用时用@Resource 标签。如上例；

如果你在 Service 标记时使用了名字如下：

```
@Service("orderBlankService")
public class OrderBlankServiceImpl extends BaseService implements IOrderBlankService {
```

框架中提供 Ctx 上下文管理可以在任何程序方法内，程序片断内得到其实例，如下：

```
IOrderBlankService orderBlankService = (IOrderBlankService)Ctx.getBean("orderBlankService");
orderBlankModelList = orderBlankService.getOrderBlankModelList();
```

2. 设计理念

一般情况下大部分的 Service Bean 按照默认方式
少数情况下比如 Service Bean 的作用域为 Session 级别的那么你要用@Resource 注入的话就只能设定当前 Action 的级别为 Session 级别，用 Ctx 方式就无限制，
还有在规则脚本中也可以通过 Ctx 方式得到 Service Bean，大大提高了规则脚本的功能。

十六、 基于Session级别的用户登陆信息管理

1. 用户信息的定义及存贮

用记登后系统会将该用户相关的信息存放在一个 Service Bean SessionUserInfo 中，如下：

```
/**
 * 用于整个系统中Session周期范围内的数据存放
 */
@Scope("session")
@Component("sessionUserInfo")
public class SessionUserInfo {
```

2. 用户信息的获取

它的级别就是 Session 级别的，在调用的时候就要用到 Ctx 上下文

```
SessionUserInfo sessionUserInfo = (SessionUserInfo)Ctx.getBean("sessionUserInfo");
System.out.println(sessionUserInfo.getUserID());
System.out.println(sessionUserInfo.getUserName());
```

如上 system 输出的即是登陆用户的用户 ID 和用户名称\

十七、 基于ServletContext的多系统Session共享方案

1. 应用 1：

```
//用以在同一应用服务器下不同应用之间的 Session 共享
ServletContext contextCninsure = request.getSession().getServletContext();
contextCninsure.setAttribute("CKM_SESSION", request.getSession());
```

2. 应用 2：

```
HttpSession session = (HttpSession)contextCninsure.getAttribute("CKM_SESSION ");
```

Session 信息即为应用 1 的 Session 信息

十八、 基于Struts2 的国际化管理

1. 采用标准的Struts2 国际化管理定义

为.properties 的资源文件

```
queryTitle=查询条件
queryTxt=查询
resetTxt=重置
addTxt=新增
modifyTxt=修改
copyTxt=复制
delTxt=删除
viewTxt=查看
submitTxt=提交
cancelTxt=取消
okTxt=确定
closeTxt=关闭
returnTxt=返回
openTxt=启用
disableTxt=停用
upstepTxt=上一步
nextstepTxt=下一步
saveTxt=保存
savereTxt=保存并返回
transactTxt=办理
```

2. 使用方式

在 Jsp 页面引用

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<s:i18n name="com.ckm.example.order.package">
```

使用方式：

```
<s:label key=""/>
<s:text name=""/>
<s:submit key=""/>
<s:property  value="%{getText('')}"/>
```

十九、 基于JbossRules 5 的规则引擎集成

1. Rules 脚本实例

//判断请假天数大于 10 天则需要总监级审批

```
global com.ckm.drools.context.SemanticContext ctx;

import java.util.HashMap;


rule "RULE_NAME1"

    when
        param:HashMap()
        eval (Integer.parseInt(param.get("sumday").toString()) < 10)
    then
       ctx.log(param.get("sumday"));
       ctx.addResult("processStepNum", "50");
end
```

2. 在Java程序中引用实例

```
/**
 * 处理流程分支,返回流程步骤顺序号(processStepNum)
 */
private Integer disposeFlowEmbranchment(String nodeID, Map _map) {
    BizFlowModel bizFlow = bizFlowService.getBizFlowModel(nodeID);
    if (bizFlow != null && bizFlow.getBizScript() != null) {
        ISemanticSessionFactory session = new SemanticSessionFactoryImpl(bizFlow.getBizScript());
        session.execute(_map);
        Object obj = session.getContext().getResult("processStepNum");
        if (obj != null) {
            return Integer.parseInt(obj.toString());
        }
    }
    return null;
}
```

3. 总结

引入 ckm-core.jar，调用提供的 Api 接口非常简单，以下三行代码搞定

ISemanticSessionFactory session = new SemanticSessionFactoryImpl(script);// **script:脚本**

session.execute(_map);//**执行**

Object obj = session.getContext().getResult("processStepNum");//**返回结果**

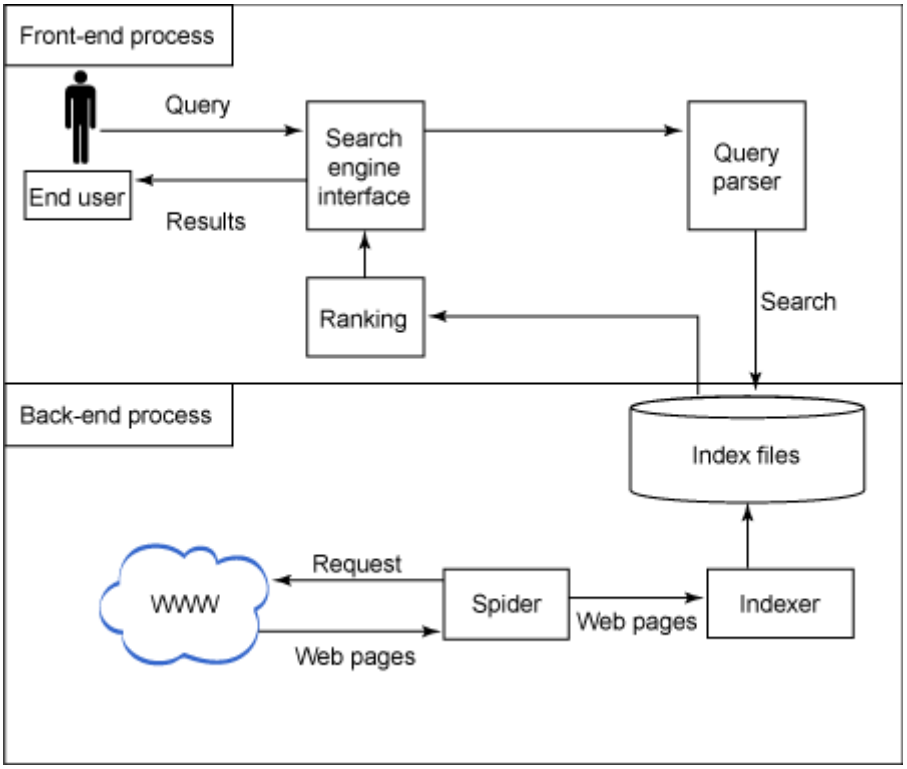
返回的结果对象类型可以是多种的，支持所有继承 Object 类型的对象

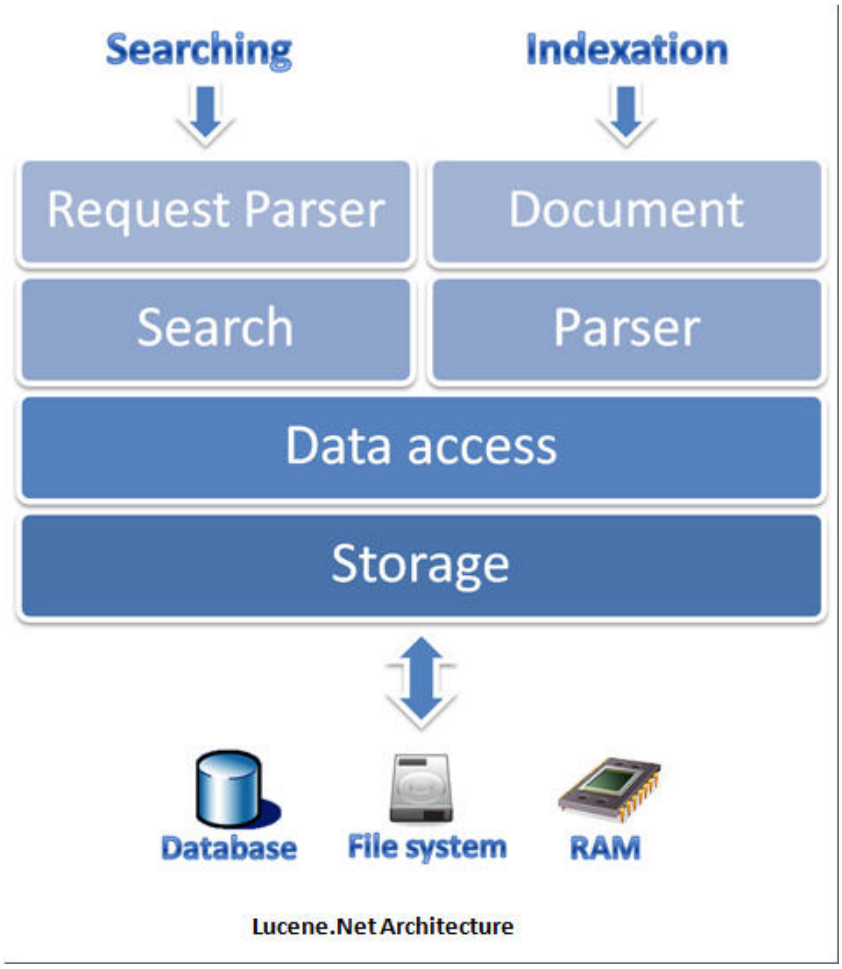
二十、 基于Apache Lucene全文检索引擎集成

Lucene 是 apache 软件基金会[4] jakarta 项目组的一个子项目，是一个开放源代码[5]的全文检索引擎工 Lucene

具包，即它不是一个完整的全文检索引擎，而是一个全文检索引擎的架构，提供了完整的查询引擎和索引引擎，部分文本分析引擎（英文与德文两种西方语言）。Lucene 的目的是为软件开发人员提供一个简单易用的工具包，以方便的在目标系统中实现全文检索的功能，或者是以此为基础建立起完整的全文检索引擎。

1. Lucene分析图


Front-end process
Query
Search engine interface
Query parser
End user
Results
Ranking
Search
Back-end process
Index files
Request
WWW
Spider
Web pages
Indexer
Web pages


Searching
Indexation
Request Parser
Document
Search
Parser
Data access
Storage
Database
File system
RAM
Lucene.Net Architecture

2. 实例

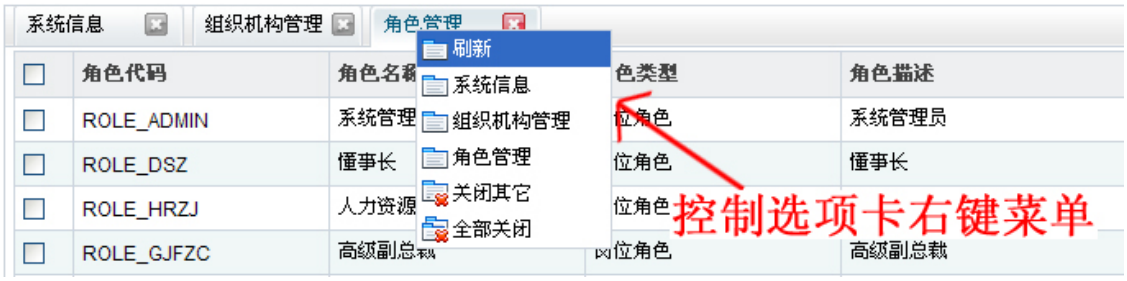
管理
知识
档案
搜 索
你输入的关键字：管理
找到相关内容约100,000,000篇，用时0.008秒
有效管理项目风险
风险管理被公认是项目管理的一个至关重要的组成部分。但是，令人惊讶的是，即使在这种重要性之下，实际运作中风险管理所得到的重视还是非常有限的。 今天的风险管理剩下什么？来自不同行业的100个项目经理在接受访问时表示，风险管理一般包括对照清单检查项目会出现哪些常见的风险，或者是项目开始之初进行风险初始评估
软件项目管理与计划
软件项目管理与计划 ========== 软件项目管理是对软件项目的开发管理。具体地说，就是对整个软件生存期的活动进行管理。本章内容主要包括：软件项目管理的内容，软件项目计划，软件配置管理，软件工程标准化与软件文档等。 16．1基础知识 16．1．1软件项目管理的内容软件
机房管理规定
机房管理一般规定总则为了安全有效地管理公司机房，保障公司计算机信息系统的安全应用和高效运行，特制定本规定。适用范围本规定适用于公司机房的管理。职责 IT中心负责本规定的制订和修订，并负责对本规定的执行情况开展定期或不定期的监督检查和指导工作。机房管理部门负责机房的日常管理工作。机房工作人员负责对机房设备进行操作、管理和日常维护保养。一般管理 公司机房的管理
绩效管理现状
首页 上一页 1 2 3 下一页 末页

二十一、 基于Jquery 的Web前端组件

1. 基于JS的Web界面框架

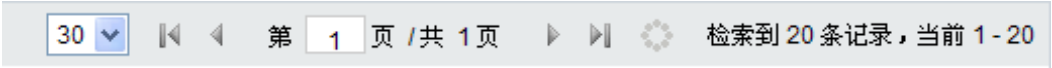
特点：以频道切换方式，显示多级菜单，按钮以资源方示展现，多选项卡操作方式

选项卡右键菜单：



2. 基于Jquery的Grid实现

分页控制



Grird 按钮控制

按钮样式：**name="btnMore"**（多选启用）**name="btnOnly"**（单选启用）

全选控制

	角色代码	角色名称	角色类型	角色描述
全选	ROLE_ZJ	总监	岗位角色	总监
	ROLE_CEO	总裁	岗位角色	总裁
	ROLE_XMSHY	项目审核员	岗位角色	项目审核员
	ROLE_ADMIN	系统管理员	岗位角色	系统管理员
	ROLE_HRZJ	人力资源总监	岗位角色	人力资源总监

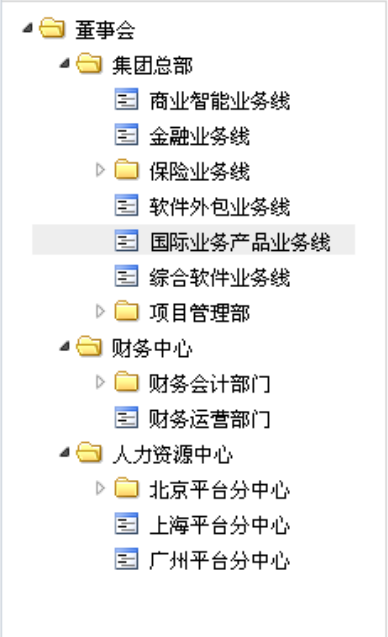
表格列排序

角色代码	角色名称	角色类型
ROLE_DOCADMIN	档案管理员	权限角色
ROLE_WORKGROUP	工作组组长权限	
ROLE_TASKADM	任务管理权限	

控制表格排序

3. 其它页面组件

1） 树形菜单演示



2） 树形表格演示

任务标题	所属机构	任务类型	执行对象	计划开始时间	计划完成时间	任务状态	进度	审核人员	指派人员
CKM项目开发	研发六部	项目		2010-11-07	2010-11-15	进行中	3%	部门经理	部门经理
我的桌面	研发六部	项目	组1		2010-11-15	未启动	0%	部门经理	部门经理
信息中心	研发六部	项目	组1		2010-11-15	未启动	0%	部门经理	部门经理
个人知识管理	研发六部	项目	组2		2010-11-15	未启动	0%	部门经理	部门经理
审批流程	研发六部	项目	组2		2010-11-15	进行中	14%	部门经理	部门经理
任务列表	研发六部	项目	宋彩金		2010-11-09	进行中	50%	李冰	李冰
员工请假流程	研发六部	项目	阮芳		2010-11-09	未启动	0%	李冰	李冰
员工借款流程	研发六部	项目	沈小勤		2010-11-10	未启动	0%	李冰	李冰
审批流程设置	研发六部	项目	罗涵予		2010-11-10	未启动	0%	李冰	李冰
流程节点设置	研发六部	项目	伍满姝		2010-11-10	进行中	20%	李冰	李冰
CKM项目帮助文档	研发六部	事宜	吴小微	2010-11-07	2010-11-09	未启动	0%	部门经理	部门经理
CKM项目管理文档	研发六部	事宜	关云长	2010-11-07	2010-11-09	未启动	0%	部门经理	部门经理
CKM项目开发文档	研发六部	事宜	李冰	2010-11-07	2010-11-09	已完成	100%	部门经理	部门经理

3） 弹出信息框组件

警告信息

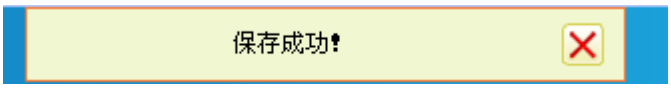
出错信息

确认信息

提示信息

告知型提示：

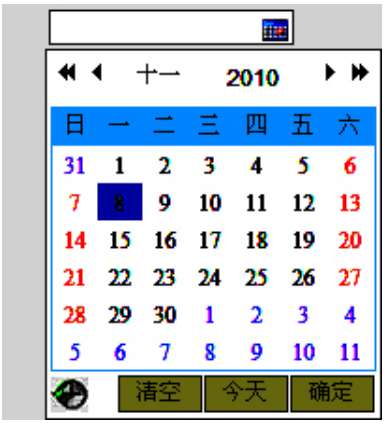
操作型提示：



屏幕右下角弹出式提示：

4） 弹出式操作框

5） 日期控件



6） 日历控件

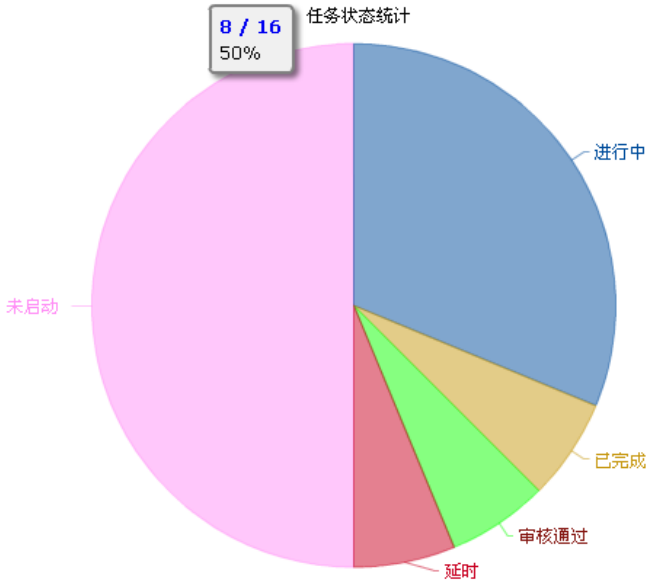
7） 自定义桌面组件

原始：

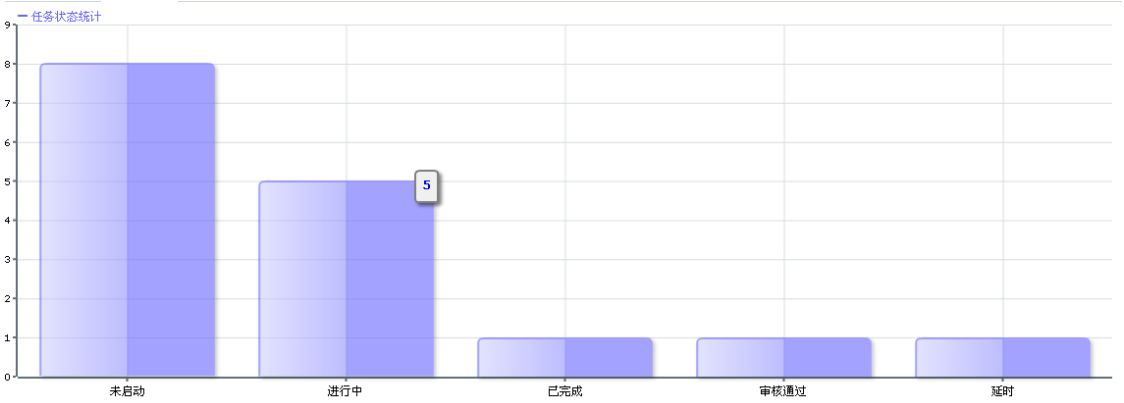
移动中：

8） 基于Open Flash Chart 的图表控件

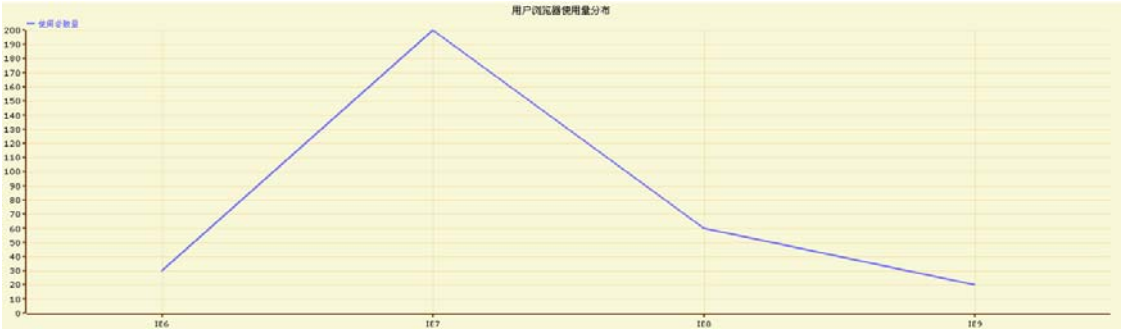
饼状图



柱状图



K 线图


用户浏览器使用量分布
使用企数量
200
190
180
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0
1E6
1E7
1E8
1E9

结束语

于 2010-11-08 日开始止于 2010-11-10 日，此为本开发平台的第一版说明文档，在长达 4 个月的 CKM 协同知识管理系统的开发和基础平台的完善中，当然还有很多细节和一些功能没有写进去，会在以后逐步完善。