Disco 入门手册

简介

Disco 是基于 java 技术，能实现企业级 Java Web 应用程序快速开发的 MVC 框架。框架设计构思来源于国内众多项目实践，框架的设计及实现借鉴当前主要流行的开源 Web 框架（Rails、Struts、JSF、Tapestry），吸取其优点及精华，是一个完全由来自中国的开源爱好者开发、文档及注释全中文的开源框架。

Disco 由四个部分组成：

1、核心 MVC。Disco 的核心是一个基于模板技术实现的 MVC 框架；他能让你用非常简洁的代码写基于 Java 的 Web 应用。

2、容器及通用业务逻辑封装。作为一个旨在让基于 Java 的 Web 应用程序开发变得直接、快速的框架，Disco 提供了一个超级 IoC 容器，并对一些企业级应用中通用的业务逻辑如分页、查询、DAO 等进行了封装，提供了一套可以直接操作、应用企业资源的组件及 API。

3、代码生成引擎及工具。仅仅依靠一个灵活、简易的 MVC 核心引擎还不能最大、限度的提高开发速度，因此 Disco 提供了一个非常灵活、易用的代码生成引擎及工具，通过使用代码生成引擎，可以快速完成基于 JavaEE 平台的企业级应用程序生成。如数据库添删改查（CRUD）代码生成、自动页面模版生成、配置文件管理等。

4、Disco 插件体系，Disco 各种实用功能的扩展，可以灵活地通过基于插件的形式安装到 Disco 中，提供各种针对性的功能。如 ajax 实用插件、代码生成插件等。

Disco 的特点：

1、快速开发支持

Disco 是首要目标即实现基于 JavaEE 的 Web 应用程序快速开发。通过 Disco 的核心 MVC、通用业务逻辑抽象、代码自动生成、插件体系等几个部分有机组合，能实现企业级的 Java Web 应用程序开发。

2、零配置及约定配置

通过配置可以让程序变得更加的灵活、易维护及扩展，配置的滥用会造成维护配置麻烦。因些，Disco 基于尽可能简化配置的原则，实现了零配置支持，同时为了保证系统的灵活性及可扩展性，还提供了很多的约定配置支持。

3、优雅的视图支持，页面及程序完全分离。Disco 提供了非常优雅的视图支持能力，

不但实现了视图页面模板与程序逻辑的完全分离，克服了传统 jsp 页面难于维护的问题，而且还实现了对页面纯天然的支持能力，使得非常适用于企业级应用中的页面制作人员与程序的分工合作。

4、超级 IoC 容器

作为一个主要用于 Java 企业级应用程序开发的框架，Disco 实现了 IoC 容器，提供非常灵活的注入方式，并能支持 Spring、Guice 等异构容器实现。

5、Ajax 支持

Disco 内置了对远程 javascript 脚本调用功能，可以使用 javascript 直接访问服务端的业务组件。另外 Disco 通过使用 prototype.js 及其它一些来自开源社区 ajax 特效工具，提供了丰富的 Ajax 支持。

环境需求

安装配置 Disco

快速安装

进入快速体验，步骤如下：

1、 下载源码并解压;

2、 在命令行执行 bin\disco crud d:\myppp

3、 切换到 d:\mypp\bin 目录，执行 sample

4、 通过 http://localost:82/java/account/list 查看运行效果。

快速入门与示例

Disco 版的 Hello World！

下面，我们以一个老掉牙的示例"Hello World!"来开始 Disco 的应用程序，我们这里把"Hello World!"改成"喂，您好，Disco1.0 发布了，请支持国产开源项目！"，另外还将显示一个系统当前的时间。

第一步，建立项目：

打开 eclipse，新建一个 tomcat 项目（我使用的 tomcat 插件，如果是其它插件，通常是建立一个 web 项目），这里我们将 context 名称设为“ / ”，字符集设置为 utf-8。项目目录结构如下：

src 目录为源码目录，存放源码文件，lib 目录为 jar 包目录，存放需要用到的 jar 包。接下来新建一个包 com.disco.action，然后将 disco-1.0.jar 以及 Disco 的依赖包放到 lib 目录，并将它添加到 build path 中。

第二步，配置 web.xml 文件：

在 web-inf 目录下新建 web.xml 文件，输入一下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <servlet>
        <servlet-name>disco</servlet-name>

<servlet-class>cn.disco.web.ActionServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>disco</servlet-name>
        <url-pattern>*.java</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>disco</servlet-name>
        <url-pattern>/java/*</url-pattern>
    </servlet-mapping>

    <filter>
        <filter-name>CharsetFilter</filter-name>
        <filter-class>cn.disco.web.CharsetFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
```

```
            </init-param>
            <init-param>
                <param-name>ignore</param-name>
                <param-value>true</param-value>
            </init-param>
        </filter>

        <filter-mapping>
            <filter-name>CharsetFilter</filter-name>
            <url-pattern>/*</url-pattern>
        </filter-mapping>

    </web-app>
```

第三步，写 Disco Action HelloAction.java：

在包 com.disco.action 下建一个名为 HelloAction.java 的文件。内容如下：

```
    package com.disco.action;

    import java.util.Date;
    import cn.disco.web.IWebAction;
    import cn.disco.web.Module;
    import cn.disco.web.Page;
    import cn.disco.web.WebForm;

    public class HelloAction implements IWebAction {
        public Page execute(WebForm form, Module module) throws
Exception {
            form.addResult("msg","喂,您好,亲爱的Disco用户");//设置VO对象
msg的值。
            form.addResult("time",new Date());//设置VO对象time的值为当
前时间
            return new Page("hello", "/hello.html");
        }
    }
```

第三步，建立 Disco 显示页面模板文件

在/web-inf 目录下新建目录 views，并在这个目录下新建一个名为 hello.html 的文件，注意保存的时候请选择 utf-8 编码，helllo.html 文件的全部内容如下：

```
    <html>
        <head>
            <title>我的第一个Disco程序界面</title>
            <meta    http-equiv="Content-Type"    content="text/html;
charset=utf-8">
```

```
    </head>
    <body>
        $!msg<br>
        当前时间:$!time
    </body>
</html>
```

至此，项目建立完毕，项目结构如下图：

最后一步：启动 Tomcat 并运行 Hello Disco 应用程序

启动 Tomcat，然后在地址栏中输入 http://localhost:8080/hello.java 即可看到如下图所示的运行结果：

我们来简单介绍一下这个简单的 Disco 应用。在上面这个应用中，我们可以看到，Disco 应用主要包括两个部分：用 Java 实现的 Action 以及业务逻辑和一个 Html 模板（这里并不是使用的单纯的 Html，还有 Velocity 脚本）。这里我们简单介绍一下 Action 部分，Velocity 部分请参考《Disco-Velocity 脚本简明教程》，你可以通过这个地址下载该教程：http://www.disco.com/disco/Disco-Velocity.pdf。

在这个 Action 实现 IWebAction 接口，`IWebAction` 接口只有一个方法 `execute`。这个方法有两个参数：`WebForm` 和 `Module`，返回一个 `Page` 对象。`WebForm` 负责封装用于

用户端显示的数据，程序对 WebForm 进行处理，并根据 Module 封装的该模块的配置信息返回一个 Page 对象（本例中使用的是手动创建一个 Page 对象，也可以通过使用 module.findPage("")方法来获取一个配置好的 Page 对象），告诉框架返回哪个页面。在程序中用到了 WebForm 的 addResult 方法，这个方法主要是用来添加要在客户端显示的数据。以这个程序为例，在执行了 form.addResult("time",new Date());这一句之后，就可以在模板中使用$!time 来调用这个 Date 对象。

一个简单的例子，相信大家对 Disco 有了大概了解。好的，我们再稍微深入一点，将这个例子改造一下，实现稍微复杂一些的功能。我们在客户端增加一个文本框，输入用户名，提交到后台处理。我们先看看模板页：

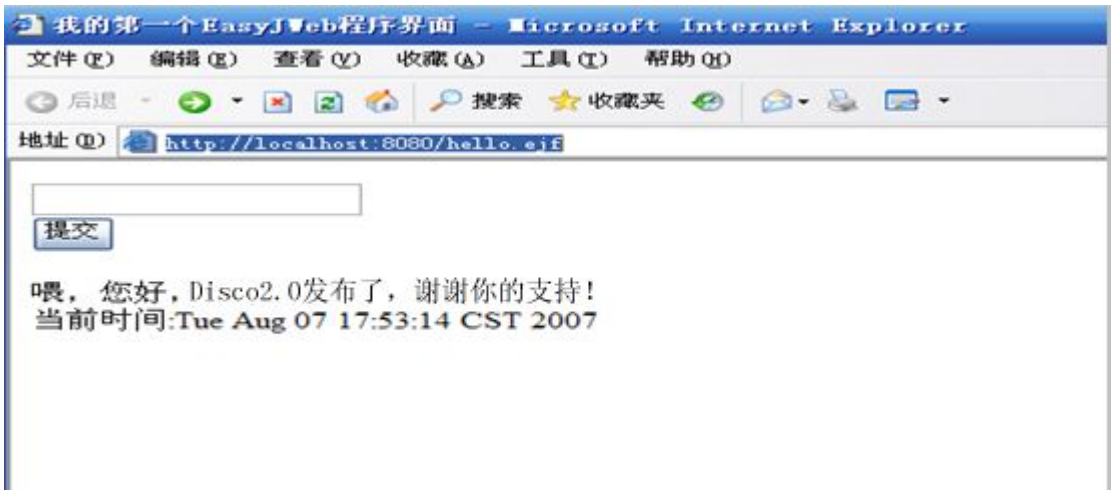
```
<html>
    <head>
        <title>我的第一个Disco程序界面</title>
        <meta      http-equiv="Content-Type"      content="text/html;
charset=utf-8">
    </head>
    <body>
        <form action="/hello.java">
            <input  name="userName"  type="text"  value="$!userName"
/><br>
            <input type="submit" value="提交" />
        </form>
        $!msg<br>
        当前时间:$!time
    </body>
    </html>
```

现在我们看看修改后的 Action：

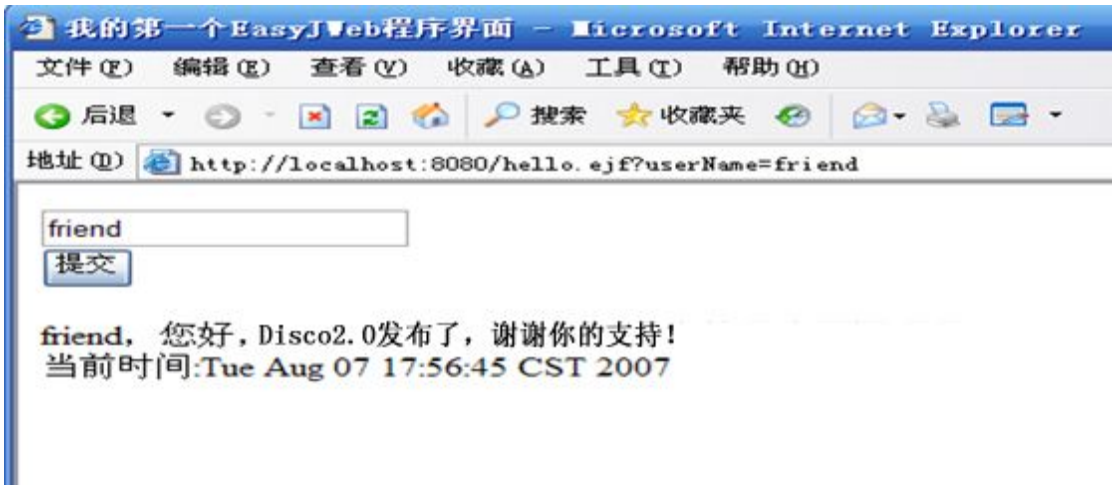
```
public class HelloAction implements IWebAction {

    public Page execute(WebForm form, Module module) throws Exception
{
        String userName = (String)form.get("userName");
        if(userName!=null&&!("".equals(userName))){
            form.addResult("msg", ""+userName+"，您好，Disco2.0发布了，
谢谢你的支持！");
        }else{
            form.addResult("msg", "喂，您好，Disco2.0发布了，谢谢你的支持！
");
        }
        form.addResult("time", new Date());
```

```
        return new Page("hello", "/hello.html");
    }
}
```

这里多了一句“String userName = (String)form.get("userName");”，这里的 form.get("userName")用来获取客户端表单域里的 userName 文本框的值。现在我们来运行一下这个例子。在客户端输入 http://localhost:8080/hello.java，界面如下：



我们输入“friend”，然后提交，会出现下面的界面：



到这里，相信大家应该了解了在 Disco 应用中如何获取客户端数据和向客户端传递数据。接下来我们再深入一点，将这个例子改为用户登录并显示用户的用户名和密码的例子，进一步的了解在 Disco 中是如何将对象数据传递给客户端并显示出来的。

在这个例子中我们首先创建一个 domain——User.java，代码如下：

```
public class User {

    private String name;
```

```
    private String password;

    public String getName() {
        return "tianyi";
    }

    public void setName(String name) {
        this.name = name;
    }
    public String getPassword() {
        return "123";
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

然后我们看看修改后的 Action，代码如下：

```
public class HelloAction implements IWebAction {

    public Page execute(WebForm form, Module module) throws
Exception {
        String userName = (String)form.get("userName");
        String password = (String)form.get("passsword");
        User user = new User();

    if(userName!=null&&!(user.getName().equals(userName))&&passwor
d!=null&&user.getPassword().equals(password)){
            form.addPo(user);
            form.addResult("msg", "登录成功！");
        }else{
            form.addResult("msg", "登录失败！");
        }
        form.addResult("time", new Date());
        return new Page("hello", "/hello.html");
    }

}
```

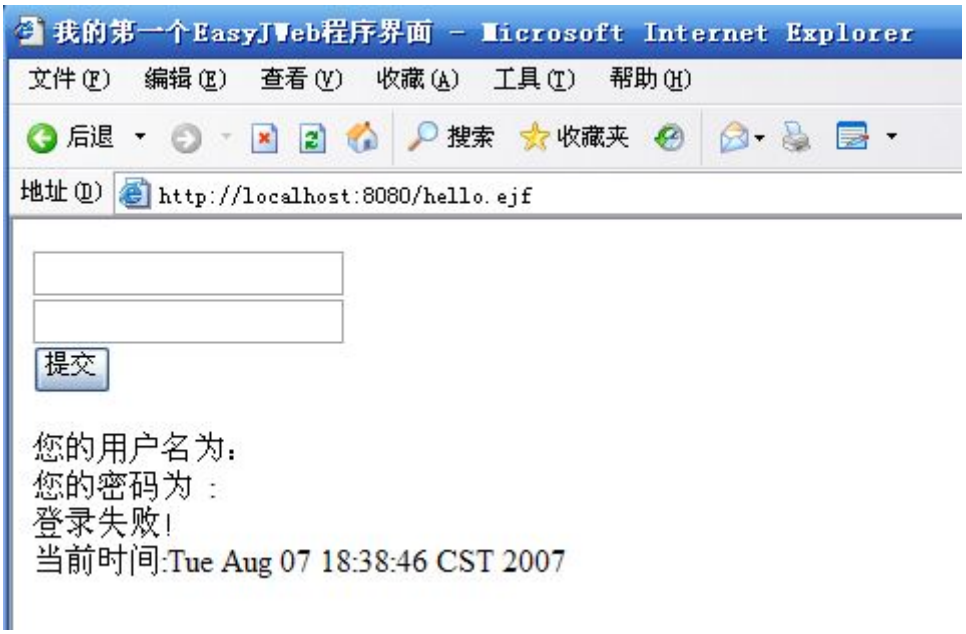
这里有一条语句 form.addPo(user)，这天语句用来将对象 user 的每一个属性值都传递到客户端。执行了这条语句之后在模板中可以直接使用$!name 来显示 user 的 name 属性值。这里还可以这样写 form.addResult("user", user)，然后在模板中这样调用：

$!user.name，$!user.password。具体的 Velocity 用法请参考 Disco 官方网站上的教程。

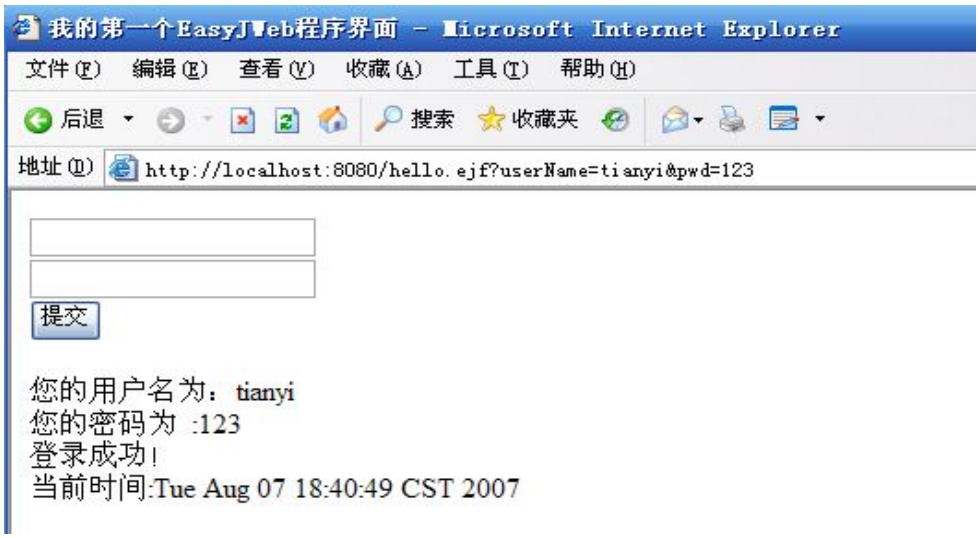
最后我们修改一下模板文件，代码如下：

```
<html>
    <head>
        <title>我的第一个Disco程序界面</title>
        <meta      http-equiv="Content-Type"      content="text/html;
charset=utf-8">
    </head>
    <body>
        <form action="/hello.java">
            <input name="userName" type="text" /><br>
            <input name="password" type="text" /><br>
            <input type="submit" value="提交" />
        </form>
        您的用户名为：$!name<br>
        您的密码为 :$!password<br>
        $!msg<br>
        当前时间:$!time
    </body>
    </html>
```

在浏览器中输入 http://localhost:8080/hello.java 访问，界面如下：



在输入框中分别输入 tianyi 和 123，提交，返回界面如下：



到这里相信大家对 Disco 的基本使用方法已经了解了，下面我们将介绍 Disco Tools 的用法。

Disco Crud

Disco Tools

前面我们创建了一个最简单的应用 Hello World，现在我们来创建一个完整的 Disco 应用。在这里我们将要用到 Disco Tools 这个包。Disco Tools 是 Disco 的一个重要部分。通常在 Action 层我们需要对客户端提交的数据做判断，进行流程控制，因此会有大量的 if else 语句。通过 Disco Tools 的业务引擎基本模型，只要编程者按照我们的模型规范进行编程，即可去除烦琐的 if else 语句。下面我们用一个例子来做简要说明。

AbstractCmdAction、AbstractPageCmdAction、AbstractCrudAction 这三个类是 Disco Tools 中最重要的三个抽象 Action 类，这三个类又以 AbstractCmdAction 为基础类。

AbstractCmdAction 类对 Action 命令进行封装，用于为提供命令式 WebAction 的写法，用户直接调用，可以减少书写繁锁的 if 语句。如果不使用 AbstractCmdAction，那么就像 struts 一样每一个操作都要写一个 Action，开发效率比较低。如果使用 AbstractCmdAction，就可以在一个 Action 中执行多个操作，只需要在客户端传递一个名为 cmd 的参数即可，AbstractCmdAction 会根据这个参数来选择执行哪个方法。假设现在客户端传递来的参数值是 save，那么将会执行 doSave()方法，其它的以此类推。

AbstractPageCmdAction 类继承自 AbstractCmdAction 类。AbstractPageCmdAction 对返回的 Page 对象进行了处理，用来支持更加灵活的参数调

用。使用这个类可以实现零配置，并且不用显示的返回 Page 对象，它会根据类名、命令参数值和配置好的 view 路径来自动创建一个 Page 对象。

AbstractCrudAction 继承自 AbstractPageCmdAction 类，AbstractCrudAction 类中对一些常用的、通用性比较强的业务逻辑方法如简单的添删改查操作进行了封装。如 doEdit()方法，这个方法通常用来指向一个编辑页面，通用性很高，因此做了封装，类似的还有很多，这里不一一列举。下面我们将分别举例对这三个类的用法进行简要讲解。

这是一个数据字典的添删改查应用，这里我们对只对字典目录进行添删改查操作。我们将分别使用这三个类来实现这个应用，比较这几个类的不同作用。现在我们来看看使用 AbstractCmdAction 类时的 Action，以下是 Action 的全部代码：

```
public class SystemDictionaryAction extends AbstractCmdAction {

    private ISystemDictionaryService service;

    public void setService(ISystemDictionaryService service) {
        this.service = service;
    }

    public Page doList(WebForm form, Module module){
        QueryObject query = new QueryObject();
        form.toPo(query);
        IPageList pageList = service.getSystemDictionaryBy(query);
        CommUtil.saveIPageList2WebForm(pageList, form);
        return new Page("list", "/news/dictionaryList.html");
    }

    public Page doAdd(WebForm form, Module module){
        return new Page("edit", "/news/dictionaryEdit.html");
    }

    public Page doEdit(WebForm form, Module module){
        Long                              id                              =
Long.parseLong(CommUtil.null2String(form.get("id")));
        form.addPo(this.service.getSystemDictionary(id));
        return new Page("edit", "/news/dictionaryEdit.html");
    }

    public Page doSave(WebForm form, Module module){
        SystemDictionary dic = form.toPo(SystemDictionary.class);
```

```
        this.service.addSystemDictionary(dic);
        return this.doList(form, module);
    }

    public Page doDel(WebForm form, Module module){
        Long                              id                              =
Long.parseLong(CommUtil.null2String(form.get("id")));
        this.service.delSystemDictionary(id);
        return this.doList(form, module);
    }

    public Page doUpdate(WebForm form, Module module){
        Long                              id                              =
Long.parseLong(CommUtil.null2String(form.get("id")));
        SystemDictionary dic = this.service.getSystemDictionary(id);
        form.toPo(dic);
        this.service.updateSystemDictionary(id, dic);
        return this.doList(form, module);
    }
}
```

AbstractCmdAction 实现了 IWebAction，在它的 execute 方法里边根据 cmd 的值来判断应该执行哪个方法。因此继承自这个类的 Action 就不需要再实现 execute 方法，只需要编写与操作相对应的 doXxxx() 方法。以 doSave() 方法为例，当 cmd 参数值为 save 时就会执行 doSave() 方法。在这个 Action 中，代码已经比较简洁了，也没有了 if else 语句，并且在一个 Action 里实现了关于 SystemDictionary 这个对象的添删改查操作。但是我们的目标是要越来越简单，越来越快捷，我们并不满足于此，接下来我们看看 AbstractPageCmdAction 类的使用。

下面同样是数据字典的添删改查操作，不同的是这个 Action 继承自 AbstractPageCmdAction 类，我们先看看这个 Action 的代码：

```
public class SystemDictionaryAction extends AbstractPageCmdAction {
    private ISystemDictionaryService service;

    public void setService(ISystemDictionaryService service) {
        this.service = service;
    }

    public void doList(WebForm form, Module module){
        QueryObject query = new QueryObject();
        form.toPo(query);
```

```
        IPageList pageList = service.getSystemDictionaryBy(query);
        CommUtil.saveIPageList2WebForm(pageList, form);
    }

    public void doEdit(WebForm form, Module module){
        Long id = Long.parseLong(CommUtil.null2String(form.get("id")));
        form.addPo(this.service.getSystemDictionary(id));
    }

    public void doSave(WebForm form, Module module){
        SystemDictionary dic = form.toPo(SystemDictionary.class);
        this.service.addSystemDictionary(dic);
    }

    public void doDel(WebForm form, Module module){
        Long id = Long.parseLong(CommUtil.null2String(form.get("id")));
        this.service.delSystemDictionary(id);
    }

    public void doUpdate(WebForm form, Module module){
        Long id = Long.parseLong(CommUtil.null2String(form.get("id")));
        SystemDictionary dic = this.service.getSystemDictionary(id);
        form.toPo(dic);
        this.service.updateSystemDictionary(id, dic);
    }
}
```

AbstractPageCmdAction 类继承自 AbstractCmdAction，这个类里边重写了 execute 方法，它先调用父类的 execute 方法，然后对父类的 execute 方法返回的值进行判断，如果为空并且 cmd 参数值不为空，就会根据调用的 Action 类的名字和 cmd 参数的值来创建一个 Page 对象返回给框架。

这个 Action 与前一个 Action 相比，可以看出来，所有的 doXxxx()方法都是 void 的了，不再 return 一个 Page 对象了。这就是 AbstractPageCmdAction 的作用，它会根据你调用的 Action 类的名字和客户端提交进来的命令参数的值自动创建一个 Page 对象，从而实现零配置，开发人员可以专注于业务逻辑，不用再关心流程控制。这样一来开发是不是又变得简单了一些、快捷了一些？但是我们的目标是不断前进的，我们不会满足现状，我们要越来越快。

好的，激动人心的时刻到来了，前面我们讲解的两个类虽然使开发已经很方便了，但是我们可以看到，Action 中还充斥着大量的 form.get()、form.addResult()等方法，并且很多方法中都有着这样一些相同的代码，那么我们是否能够再封装一下呢？现在我们来看看 AbstractCrudAction 类的使用。还是同样的应用：

```
public class SystemDictionaryAction extends AbstractCrudAction {
    private ISystemDictionaryService service;

    public void setService(ISystemDictionaryService service) {
        this.service = service;
    }

    @SuppressWarnings("unchecked")
    protected Class entityClass() {
        return SystemDictionary.class;
    }

    protected Object findEntityObject(Serializable id) {
        return service.getSystemDictionary((Long) id);
    }

    protected IPageList queryEntity(IQueryObject queryObject) {
        return service.getSystemDictionaryBy(queryObject);
    }

    protected void removeEntity(Serializable id) {
        service.delSystemDictionary((Long) id);
    }

    protected void saveEntity(Object object) {
        service.addSystemDictionary((SystemDictionary) object);
    }

    protected void updateEntity(Object object) {
        service.updateSystemDictionary(((SystemDictionary)
object).getId(),(SystemDictionary) object);
    }
}
```

我们可以看到 Action 的代码少了很多，业务逻辑的处理完全封装了，每个操作都只需要一条调用语句即可。现在我们来分析一下代码。首先，Action 里多出了一个方法 entityClass()，这个方法是 AbstractCrudAction 里申明的一个抽象方法，它在没

个 Action 里边的实现都不一样，分别返回各自要操作的类，提供给 AbstractCrudAction 里实现的一些业务逻辑方法使用。

接下来，我们会发现方法的命名也和以前不一样了，这里没有了 doXxxx()方法，取而代之的是 xxxxEntity()方法。这些方法都是在父类 AbstractCrudAction 里申明的抽象方法，在每个不同的 Action 里实现，AbstractCrudAction 里的业务逻辑方法调用这些不同的实现分别执行不同的操作，巧妙的对通用性比较强的代码进行了封装，使开发变得更加简单快捷。假设命令参数值为 save，在这里将不再执行 doSave()方法，而是执行 saveEntity()方法。这里并不是去掉了 doXxxx()方法，而是将这些方法放到了 AbstractCrudAction 类里边，这里的 saveEntyti()方法会被父类的 doSave()方法调用，因此实际上还是调用了 doSave()方法，只不过这个方法不需要开发人员自己来申明和实现了。