Hibernate-Vaadin-Utils

Autor Administrador
miércoles, 08 de febrero de 2012
Modificado el miércoles, 08 de febrero de 2012

Hibernate-Vaadin-Utils es una API para la rápida creación de CRUDs usando Hibernate y Vaadin. La API fue concebida para que la creación de un componente visual CRUD fuera tan fácil como: CrudComponent<Entidad> crud = new CrudComponent<Entidad>(Entidad.class);Mi experiencia en desarrollo de software me ha mostrado que la funcionalidad CRUD es muy frecuente en las aplicaciones empresariales. Tener una tabla con datos y un formulario para crearlos y modificarlos es una característica recurrente en todo tipo de aplicaciones empresariales. Es posible encontrar componentes que encapsulan esta funcionalidad para casi cualquier framework web. De hecho, para Vaadin existe el add-on "ReinCRUD" con una licencia de US$350 (a la hora de escribir este artículo).Como la funcionalidad CRUD es tan recurrente y común en las aplicaciones empresariales, comencé con el desarrollo de un conjunto de clases que sirvieran de base para la creación de CRUDs más avanzados y con funcionalidad particular a las necesidades de cada aplicación.Comencé su desarrollo en Julio de 2011 y a la fecha incluye no sólo utilidades para la creación de CRUDs sino para la generación de reportes, internacionalización y envío de correos electrónicos. Por ahora no quiero publicar el código fuente de la API, pero sí ofrecer un JAR para su uso libre sin royalties. Descarga: http://alejandroduarte.hollosite.com/hibernate-vaadin-utils-0.149.jar

Hibernate-Vaadin-Utils es una API para la rápida creación de CRUDs usando Hibernate y Vaadin. La API fue concebida para que la creación de un componente visual CRUD fuera tan fácil como: CrudComponent<Entidad> crud = new CrudComponent<Entidad>(Entidad.class); Mi experiencia en desarrollo de software me ha mostrado que la funcionalidad CRUD es muy frecuente en las aplicaciones empresariales. Tener una tabla con datos y un formulario para crearlos y modificarlos es una característica recurrente en todo tipo de aplicaciones empresariales. Es posible encontrar componentes que encapsulan esta funcionalidad para casi cualquier framework web. De hecho, para Vaadin existe el add-on "ReinCRUD" con una licencia de US$350 (a la hora de escribir este artículo). Como la funcionalidad CRUD es tan recurrente y común en las aplicaciones empresariales, comencé con el desarrollo de un conjunto de clases que sirvieran de base para la creación de CRUDs más avanzados y con funcionalidad particular a las necesidades de cada aplicación. Comencé su desarrollo en Julio de 2011 y a la fecha incluye no sólo utilidades para la creación de CRUDs sino para la generación de reportes, internacionalización y envío de correos electrónicos. Por ahora no quiero publicar el código fuente de la API, pero sí ofrecer un JAR para su uso libre sin royalties. El código no está documentado aún (javadoc) ni he escrito un tutorial apropiado. Sin embargo, a continuación describo un ejemplo de uso.

Supongamos que queremos implementar un CRUD de usuarios, para esto será necesario crear una clase Usuario y "mapearla" usando las anotaciones de Hibernate. Luego se marcará la clase con @CrudTable(stringRepresentationProperty="login") y la propiedad "password" con @CrudField(showInTable=true, isPassword=true), de la siguiente manera:

```
@Entity
@CrudTable(stringRepresentationProperty="login")
public class Usuario extends Dto implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;

    @Column(nullable = false, unique=true, length=255)
    private String login;

    @CrudField(showInTable=true, isPassword=true)
    @Column(nullable = false, length=255)
    private String password;

    public Usuario() {
        super();
    }

    @Override
    public String toString() {
        return login;
    }
    @Override
    public Long getId() {
        return id;
    }
    @Override
    public void setId(Object id) {
        this.id = (Long) id;
    }
```

```
    public String getLogin() {
        return login;
    }
    public void setLogin(String login) {
        this.login = login;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
```

} stringRepresentationProperty, permite especificar la propiedad que será usada para filtrar los datos en las tablas (sí, se requiere un refactor de nombre para reflejar el uso real). @CrudField(showInTable=true, isPassword=true) especifica que la propiedad anotada (password) será mostrada en la tabla y que será encriptada en la base de datos. La implementación del componente CRUD respectivo, sería de la siguiente forma: CrudComponent<Usuario> usuariosCrud = new CrudComponent<Usuario>(Usuario.class, new AppFieldFactory()); AppFieldFactory sería una subclase de DefaultCrudFieldFactory (proporcionada en la API): public class InfocontFieldFactory extends DefaultCrudFieldFactory {

```
    @Override
    public Field createCustomField(Object bean, Item item, String pid, Component uiContext, Class<?> propertyType) {
        Field field = null;

        if("login".equals(pid)) {
            TextField loginField = new TextField();
            loginField.setWidth("230px");
        }

        return field;
    }
    @Override
    public void addValidators(Field field, Object bean, Item item, String pid, Component uiContext, Class<?> propertyType,
CrudField crudFieldAnnotation) {

        if("login".equals(pid)) {
            field.addValidator(new LoginValidator());
        }

        super.addValidators(field, bean, item, pid, uiContext, propertyType, crudFieldAnnotation);
    }
```

} En esta clase se usa el método createCustomField para crear fields de forma personalizada y addValidators para agregar Validators a los fields creados. La API incluye clases de soporte para la creación de las sesiones de Hibernate. Se usa el excelente add-on "HbnContainer" (https://vaadin.com/directory#addon/hbncontainer) con una modificación de la clase HbnContainer (se incluye el código fuente de esta clase en el JAR). Cada clase del dominio (entidad) debería tener una clase Container correspondiente (derivada de DefaultHbnContainer). Se debe crear una clase Factory para estos Containers: public class AppContainerFactory extends ContainerFactory {

```
    @SuppressWarnings("rawtypes")
    @Override
    public DefaultHbnContainer getContainer(Class<?> clazz, Component component) {
        if(clazz.equals(Usuario.class)) {
            return getUsuarioContainer();
        }

        return getDefaultFactory().getContainer(clazz, component);
    }

    public static UsuarioContainer getUsuarioContainer() {
        return new UsuarioContainer();
    }
```

} Para luego crear la clase de la aplicación: public class Applicacion extends UtilApplication {

```
    @Override
    public void init() {
        super.init();
        ContainerFactory.init(new AppContainerFactory());
        MainWindow mainWindow;
```

```
            mainWindow = new MainWindow();
            setMainWindow(mainWindow);
        }
    }
```

Por último, se debe crear un archivo de configuración "configuration.properties":

```
db.driver=com.mysql.jdbc.Driver
db.url=jdbc:mysql://localhost:3306/cis
db.user=root
db.password=admin
db.persistenceUnit=cis
db.schemaGeneration=update
db.show_sql=false
ui.usuario.login=Login
ui.usuario.password=Contraseña
```

Las propiedades ui.usuario.login y ui.usuario.password, configuran las etiquetas de los TextFields en los formularios CRUD.