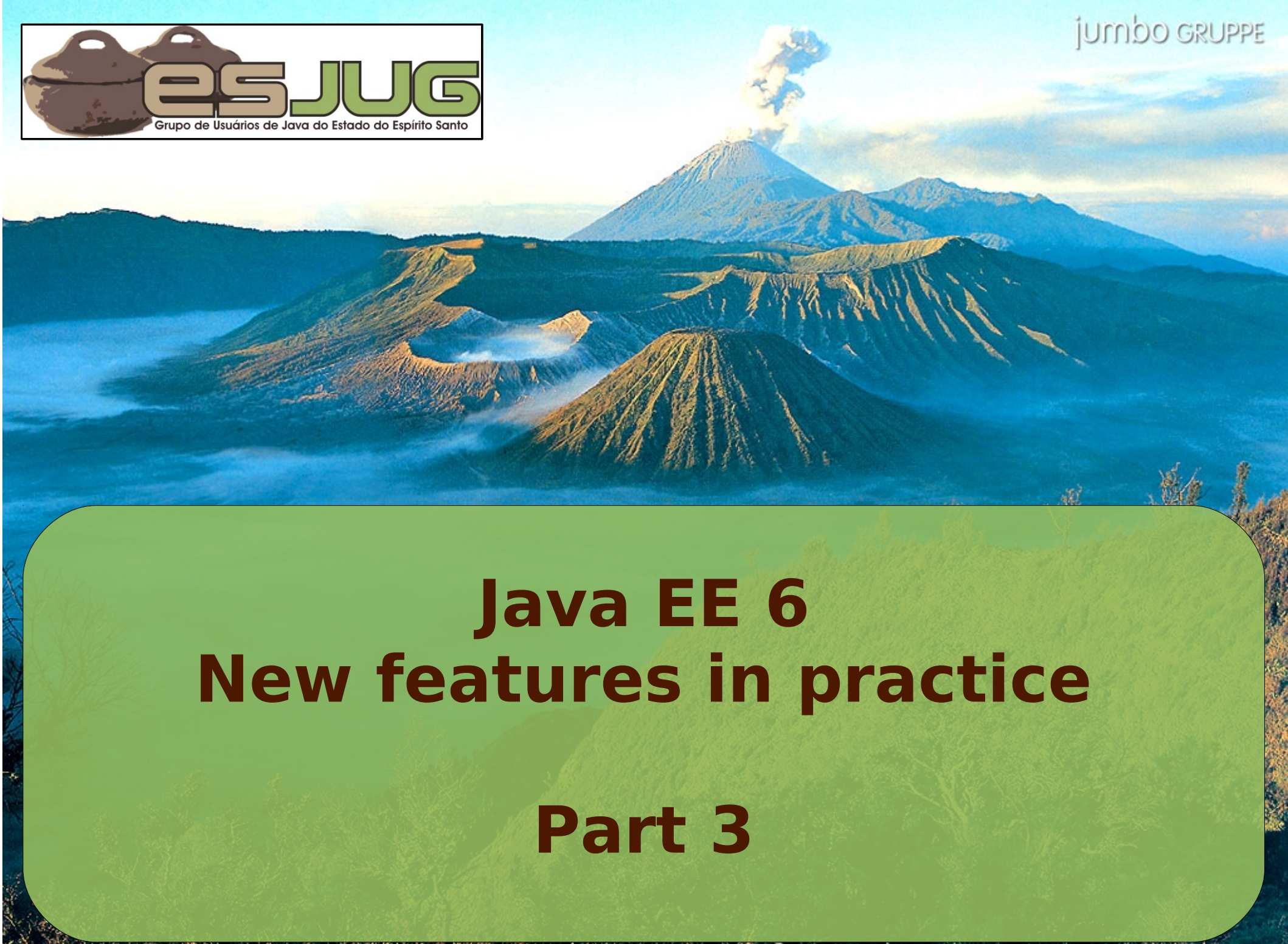


Java EE 6
New features in practice

Part 3

License for use and distribution

This material is available for non-commercial use and can be derived and/or redistributed, as long as it uses an equivalent license.

Attribution-Noncommercial-Share Alike 3.0 Unported

http://creativecommons.org/licenses/by-nc-sa/3.0/

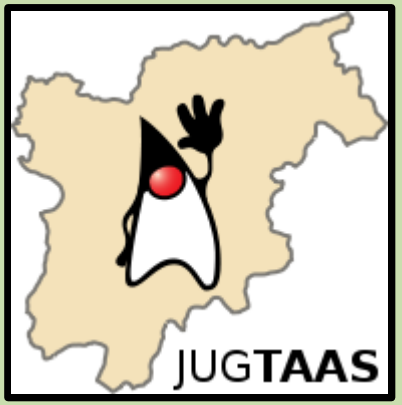
You are free to share and to adapt this work under the following conditions: (a) You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work); (b) You may not use this work for commercial purposes. (c) If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

About the author – Vítor Souza

- Education:
  - Computer Science graduate, masters in Software Engineering – (UFES, Brazil), taking PhD at U. Trento.
- Java:
  - Developer since 1999;
  - Focus on Web Development;
  - Co-founder and coordinator of ESJUG (Brazil).
- Professional:
  - Substitute teacher at Federal University of ES;
  - Engenho de Software Consulting & Development.
- Contact: vitorsouza@gmail.com

JUG TAAS = JUG Trento + JUG Bolzano

- Website:
  - http://www.jugtrento.org/
  - http://www.jugbz.org/
- Mailing list (in Italian, mostly): http://groups.google.com/group/jugtaa
- If you're interested in Java, join and participate!



JUGTRENTO.ORG
Java User Group

Agenda

- Auth^2 with JAAS;
- Servlets 3.0;
- More on JPA 2.0:
  - New commands in JPQL;
  - Support for pessimistic locking;
- Enhancements for EJBs.

Auth² with JAAS

Image source: http://www.freedigitalphotos.net/images/Security_g189-Keys_p17918.html

Auth² in Java = JAAS

- Authentication = guaranteeing the user is who she says she is;
- Authorization = guaranteeing the user can access resources she is authorized to;
- For Java applications, we can use JAAS: Java Authentication and Authorization Services;
  - Data integrity;
  - Confidentiality;
  - Non-repudiation;
  - Auditing.

Basic concepts of Auth²

- **Security realm**: set of security configurations registered under a name;
- **User**: an individual or software identified by an username and a password (credentials);
- **Group**: group of user;
- **Role**: a name associated with a set of access rights. Can be associated to users or groups.

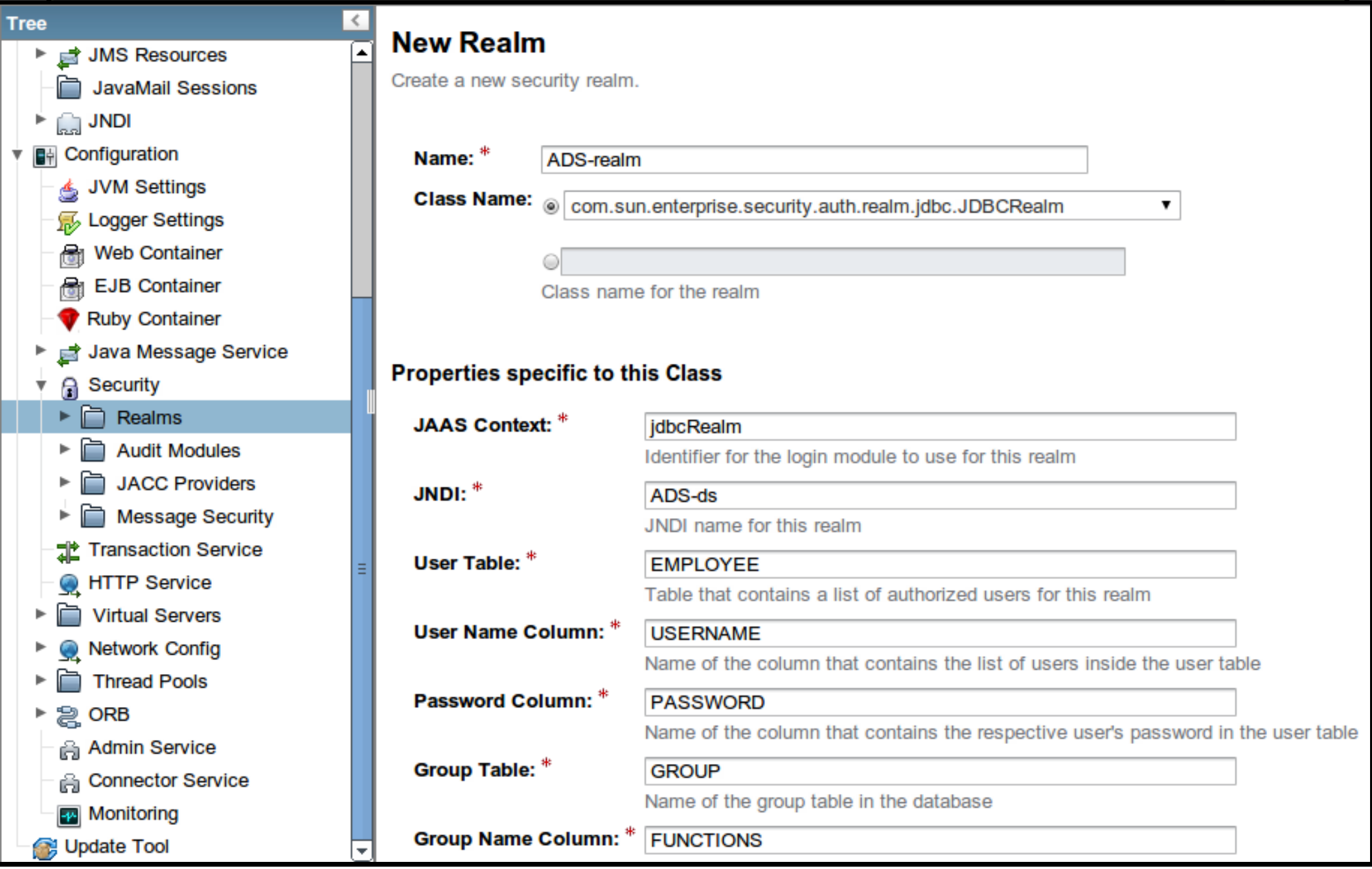
Authentication = what users exist and what are their passwords?

Authorization = which roles can access what?

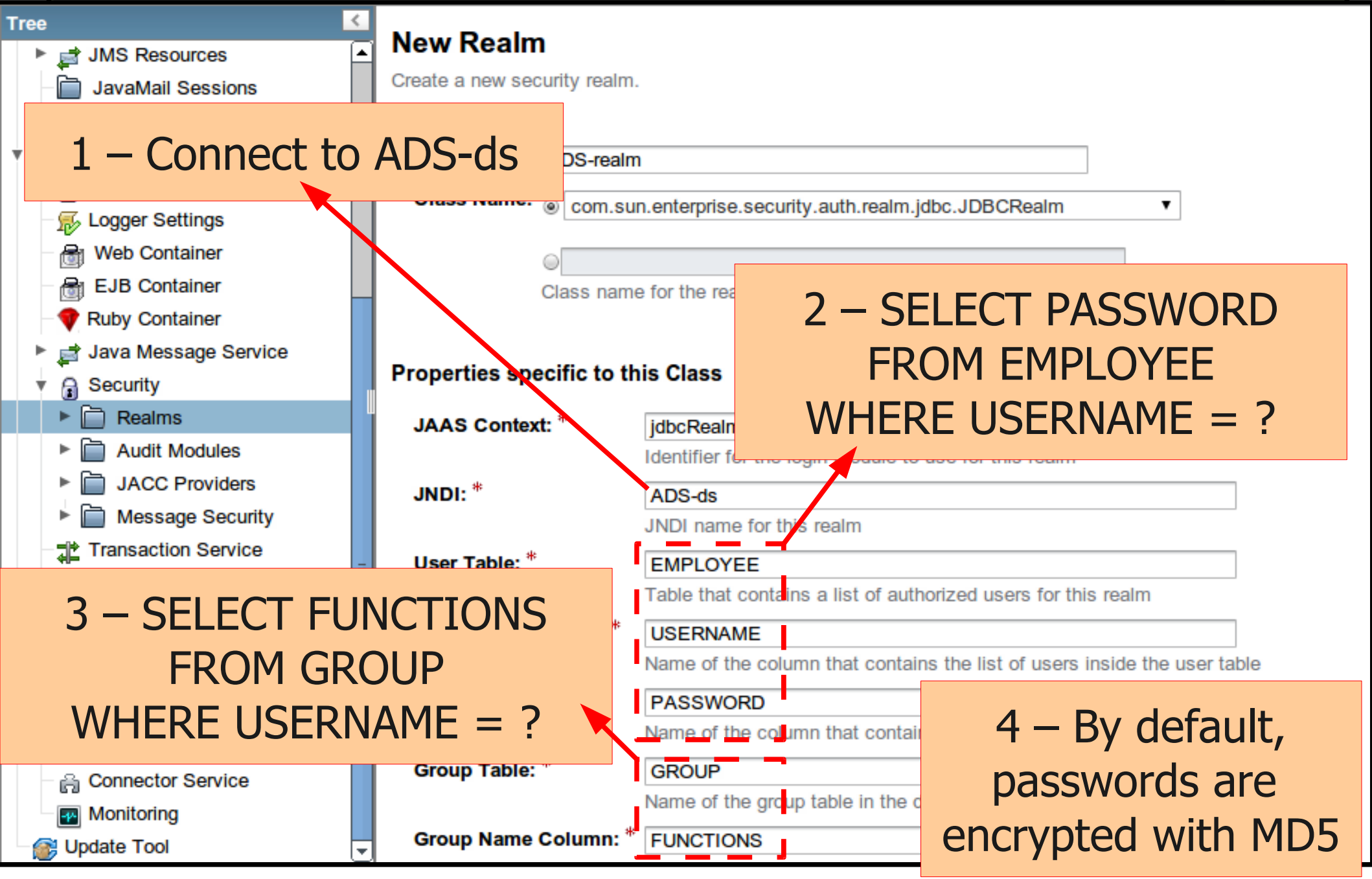
Realms in GlassFish

- Realm types:
  - Flat files;
  - JDBC;
  - Certificate;
  - Solaris;
  - LDAP / Microsoft Active Directory;
  - Any class implementing the Realm interface (proprietary).

Setting up a realm in Web Console



How JDBC Realm works

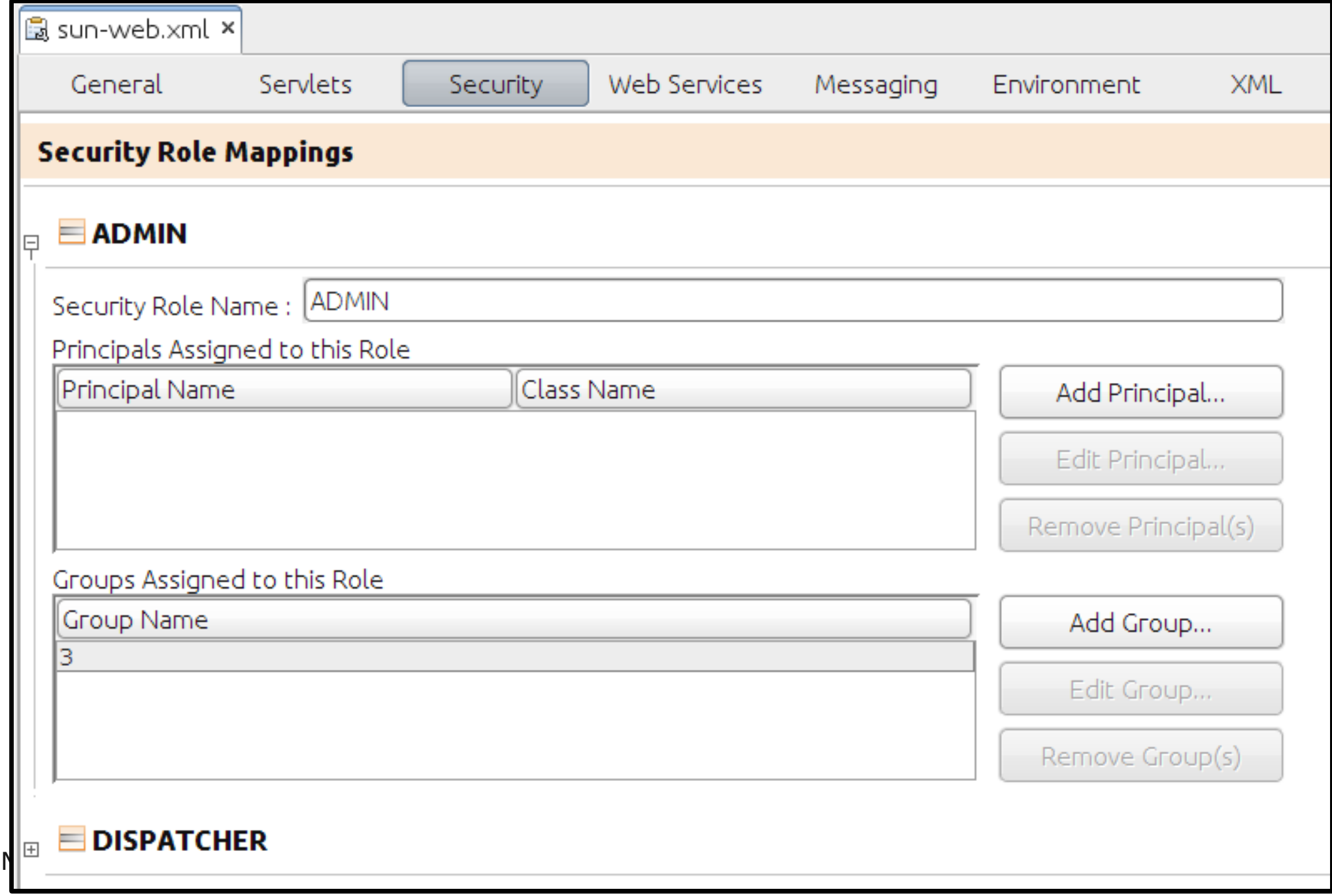


Configure GlassFish's sun-web.xml

```
<sun-web-app error-url="">
  <context-root>/ADS-war</context-root>
  <security-role-mapping>
    <role-name>OPERATOR</role-name>
    <group-name>0</group-name>
  </security-role-mapping>
  <security-role-mapping>
    <role-name>DISPATCHER</role-name>
    <group-name>1</group-name>
  </security-role-mapping>

  ...

  <class-loader delegate="true"/>
  <jsp-config>
    <property name="keepgenerated" value="true" />
  </jsp-config>
</sun-web-app>
```

Numerical values because FUNCTIONS is an enumeration!

Configure GlassFish's sun-web.xml

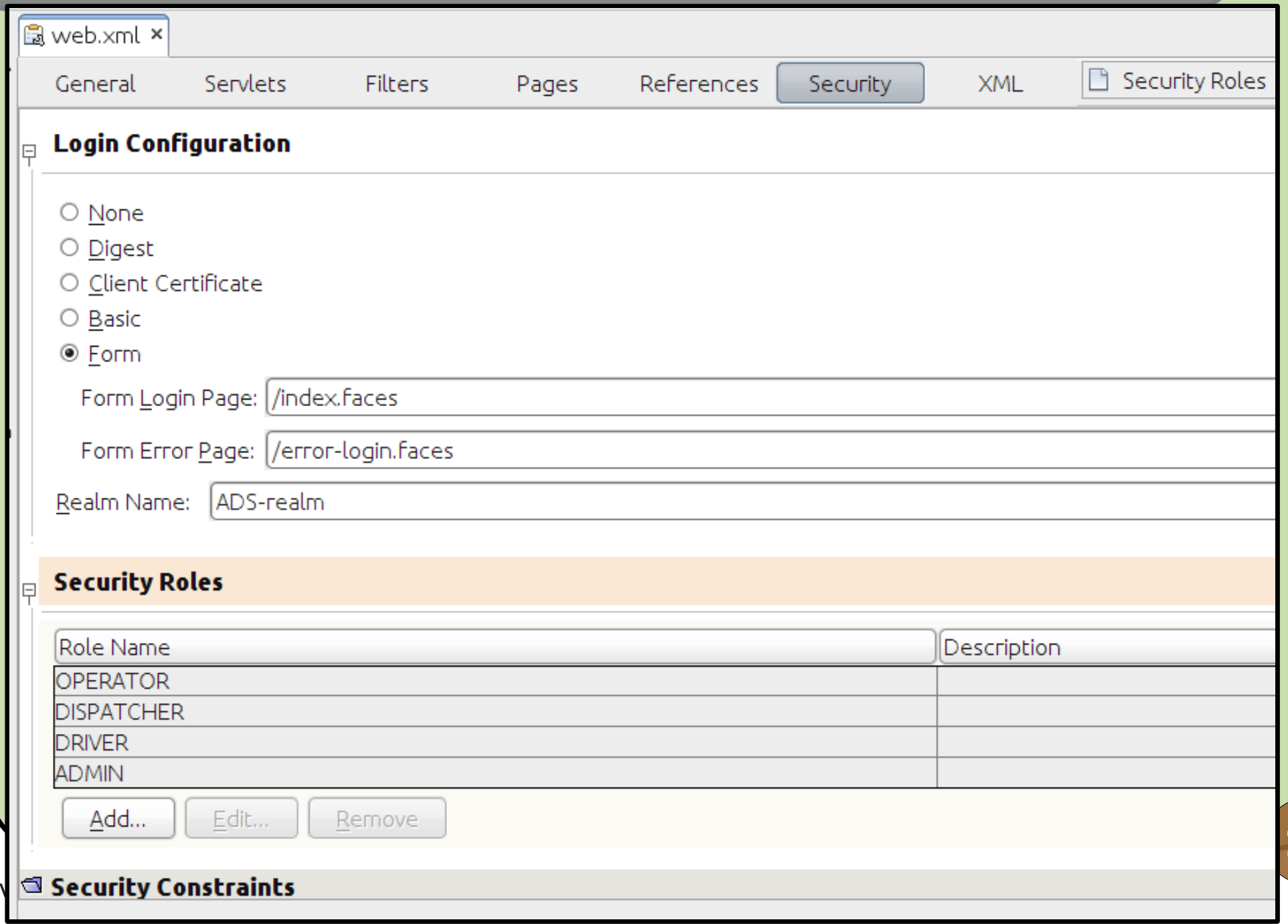


web.xml for FORM authentication

```
<web-app ...>
  <login-config>
    <auth-method>FORM</auth-method>
    <realm-name>ADS-realm</realm-name>
    <form-login-config>
      <form-login-page>/index.faces</form-login-page>
      <form-error-page>/error-login.faces</form-error-page>
    </form-login-config>
  </login-config>
  <security-role>
    <description />
    <role-name>OPERATOR</role-name>
  </security-role>
  <security-role>
    <description />
    <role-name>DISPATCHER</role-name>
  </security-role>

   ...

</web-app>
```

Should match the roles in sun-web.xml

web.xml for FORM authentication



The login form

```
<ui:decorate template="/templates/form.xhtml">
  <form id="form" method="POST" action="j_security_check">
    <ui:decorate template="/templates/field.xhtml">
      <ui:param name="id" value="form:username" />
      <ui:define name="nome">Username</ui:define>
      <input type="text" id="username" name="j_username" />
    </ui:decorate>
    <ui:decorate template="/templates/field.xhtml">
      <ui:param name="id" value="form:pwd" />
      <ui:define name="nome">Password</ui:define>
      <input type="password" id="pwd" name="j_password" />
    </ui:decorate>
    <ui:decorate template="/templates/buttons.xhtml">
      <input type="submit" value="Log in" />
    </ui:decorate>
  </form>
</ui:decorate>
```

Checking if user is authenticated

```
public class LoginManagerBean implements LoginManager,
Serializable {
  @Resource
  private SessionContext sessionCtx;

  @EJB
  private EmployeeDAO employeeDAO;

  public Employee checkJaasLogin() {
    Employee emp = null;
    Principal principal = sessionCtx.getCallerPrincipal();
    if (principal != null) {
      String username = principal.getName();
      if (! "ANONYMOUS".equals(username)) {
        emp = employeeDAO.retrieveByUsername(username);
      }
    }
    return emp;
  }
}
```

Form x programmatic authentication

- Form login:
  - Container is called directly;
  - Our application constantly checks for the principal.
- Programmatic login:
  - Our application is called;
  - The container is programmatically called from our application's code.

```
<form id="form" method="POST" action="j_security_check">
<input type="text" id="username" name="j_username" />
```

```
<h:form id="form">
<h:inputText id="username" value="#{loginBean.username}" />
```

Login method

```
public class LoginManagerBean ... {
  public void login(String username, String password) {
    Employee emp = employeeDAO.retrieveByUsername(username);
    String md5pwd = TextUtils.produceMd5Hash(password);
    String pwd = emp.getPassword();

    if ((pwd != null) && (pwd.equals(md5pwd))) {
      HttpServletRequest request =
(HttpServletRequest)FacesContext.getCurrentInstance().getExt
ernalContext().getRequest();
      request.login(username, password);

      currentUser = emp;
      pwd = password = null;
    }
    else {
      throw new LoginFailedException();
    }
  }
}
```

logout() also exists!

GlassFish also provides a proprietary solution:
com.sun.appserv.security.ProgrammaticLogin

Authorization for classes/methods

- Use of annotation `@RolesAllowed`:

```
@RolesAllowed("ADMIN")
public class AmbulanceCrudServiceBean implements
                          AmbulanceCrudService, Serializable {
   ...
}
```

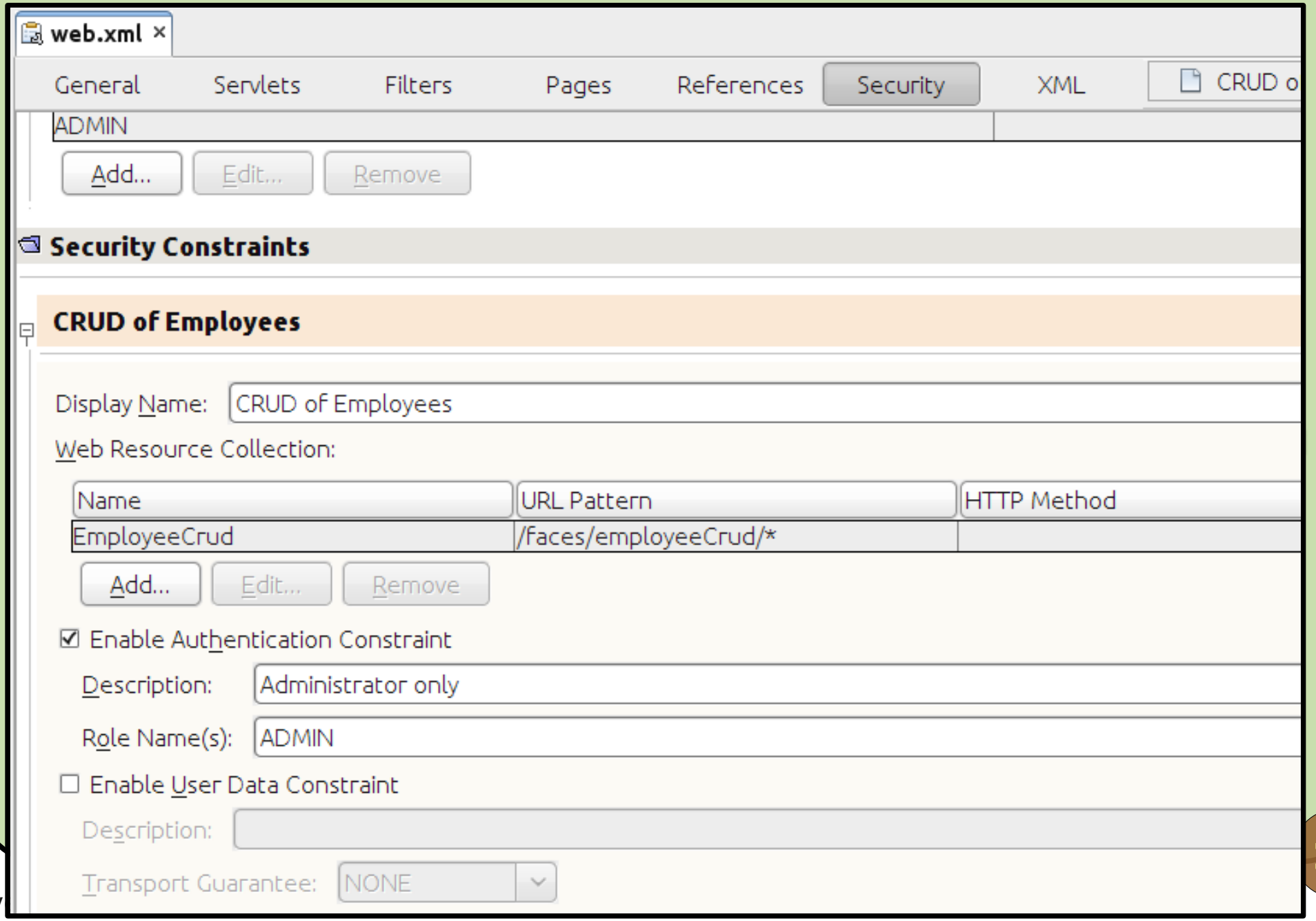
- Applies to the whole class or single methods;
- Limitation: does not extend to inherited methods;
- If a method is called and the user doesn't have the role, `javax.ejb.EJBAccessException` is thrown;
- Less useful: `@PermitAll` and `@DenyAll`.

Authorization for pages

```
<web-app ...>
  <security-constraint>
    <display-name>CRUD of Employees</display-name>
    <web-resource-collection>
      <web-resource-name>EmployeeCrud</web-resource-name>
      <description />
      <url-pattern>/faces/employeeCrud/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <description>Administrator only</description>
      <role-name>ADMIN</role-name>
    </auth-constraint>
  </security-constraint>

    ...

</web-app>
```

- Error 403 in case of violation.

Authorization for pages



Servlets 3.0

Image source: http://www.freedigitalphotos.net/images/Internet_g170-Global_Network_p21925.html

Servlet mapping with annotations

```
@WebServlet(name = "LogoutSrvlt", urlPatterns = {"/logout"})
public class LogoutServlet extends HttpServlet {
  protected void service(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
    // Destroys the session for this user.
    request.getSession(false).invalidate();

    // Redirects back to the initial page.
    response.sendRedirect(request.getContextPath());
  }
}
```

Filter mapping with annotations

```
@WebFilter(filterName = "CounterFltr", urlPatterns = {"/*"})
public class CounterFilter implements Filter {
  public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain) throws
IOException, ServletException {
    if (request instanceof HttpServletRequest) {
      HttpSession session =
                 ((HttpServletRequest)request).getSession();
      Object count = session.getAttribute("count");
      int c = (count == null) ? 0 :
                      Integer.parseInt(count.toString()) + 1;
      session.setAttribute("count", c);
    }
    chain.doFilter(request, response);
   }

  public void init(FilterConfig filterConfig) throws
ServletException { }
  public void destroy() { }
}
```

Extensibility of the Web layer

- With annotations, servlets, filters and listeners can be provided in JARs, no need for configuration;
- `ServletContext` provides methods for dynamic loading: `addServlet()`, `addFilter()`, ...;
- Also, a `web-fragment.xml` provided in the `META-INF` of the JAR is automatically loaded.

More on JPA 2.0

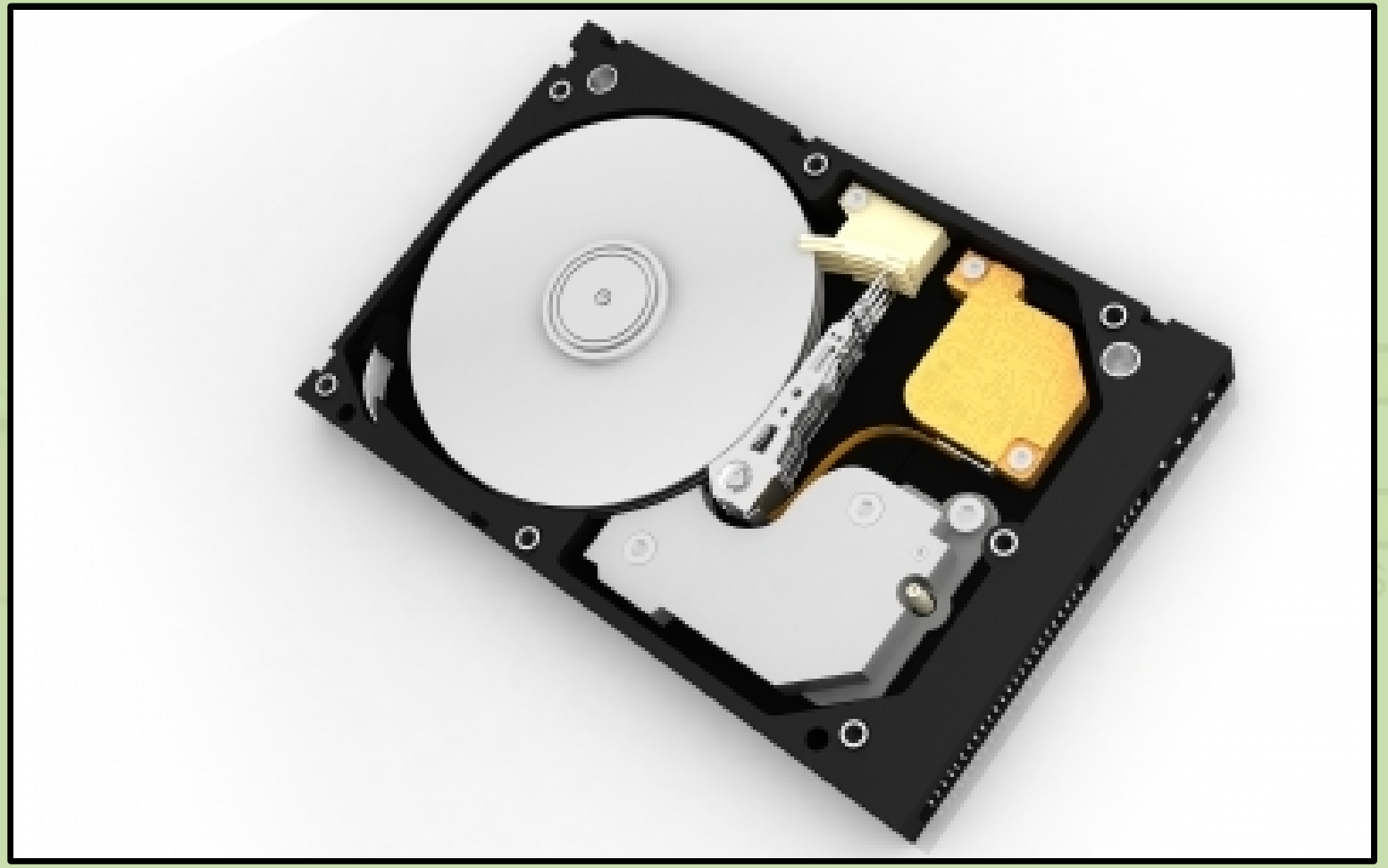


Image source: http://www.freedigitalphotos.net/images/Computers_g62-Hard_Disk_p13255.html

New operators

- Case expressions:

```
update Employee e set e.salary = case e.position
    when 'Director' then e.salary * 1.15
    when 'Manager' then e.salary * 1.10
    else e.salary * 1.05
end
```

- NULLIF:

```
select nullif(e.salary, -1) from Employee e
```

- COALESCE:

```
select coalesce(e.name, e.username) from Employee e
```

New operators

- INDEX:

```
-- Assuming a.drivers is a list instead of a set.
select d from Ambulance a join a.drivers d
where a.id = :id and index(d) between 0 and 4
```

- TYPE:

```
-- Assuming hierarchy of Employee instead of enum.
select e from Employee e
where type(e) in (Operator, Dispatcher)
```

- KEY, VALUE, ENTRY:

```
-- Assuming a.drivers is a map instead of a set.
select key(d), value(d) from Ambulance a join a.drivers d
where a.id = :id
```

Pessimistic Locking

- Optimistic = version column (few conflics);
- Pessimistic = locks (many conflics);
- Method lock() in EntityManager:

```
// cq is some CriteriaQuery that returns a single employee...
Employee emp = em.createQuery(cq).getSingleResult();
em.lock(emp, LockModeType.PESSIMISTIC_READ);
```

Lock modes

- **None**: no locking;
- **Optimistic**: new name for “read”, which already existed, = optimistic lock;
- **Optimistic, with force increment**: new name for “write”, which also existed;
- **Pessimistic read**: locks for writing (repeatable read);
- **Pessimistic write**: locks for everything (serialization);
- **Pessimistic, with force increment**: same as before, but forcing the increment of the version column.

Enhancements for EJBs

Image source: http://www.freedigitalphotos.net/images/Other_Business_g200-Desired_Outcome_p8711.html

No-interface EJB

- Before, EJBs had to be `@Local` or `@Remote`;
- Now, they can have no interface (`@LocalBean`). Public methods are made available:

```
@Stateless @LocalBean @Named
public class SomeStatelessBean {
    public void aMethod() { /* ... */ }
    public String anotherMethod() { /* ... */ }

    @PostConstruct
    public void init() {
        /* Initialization code... */
    }
}
```

Singleton

- We can also have Singleton EJBs:

```
@Stateless
@Singleton
@Named
public class HighlanderBean {
    /* There can be only one... */
}
```

- Note: singleton EJBs are thread-safe, serializing method calls...

Asynchronous calls

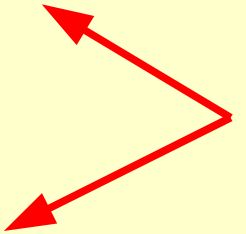
- Execute long methods in background:

```
public class RegisterCallServiceBean ... {
  @Asynchronous
  public Future<List<Call>> searchForSimilar(Call call) {
    List<Call> xList = callDAO.searchByX(call.getX());
    List<Call> yList = callDAO.searchByY(call.getY());
    // ...

    List<Call> similars = new ArrayList<Call>();
    similars.addAll(xList); // ...
    similars.remove(call);

    return new AsyncResult<List<Call>>(similars);
  }
}
```

Asynchronous calls

- Check if done:

```
public class RegisterCallAction ... {
  private Future<List<Call>> result;
  public List<Call> getSimilars() {
    if ((result != null) && (result.isDone())) return result.get();
     return null;
  }
  public boolean isDone() {
    return ((result != null) && (result.isDone()));
  }
  public void searchForSimilar() {
    result = registerCallService.searchForSimilar(call);
  }
}
```



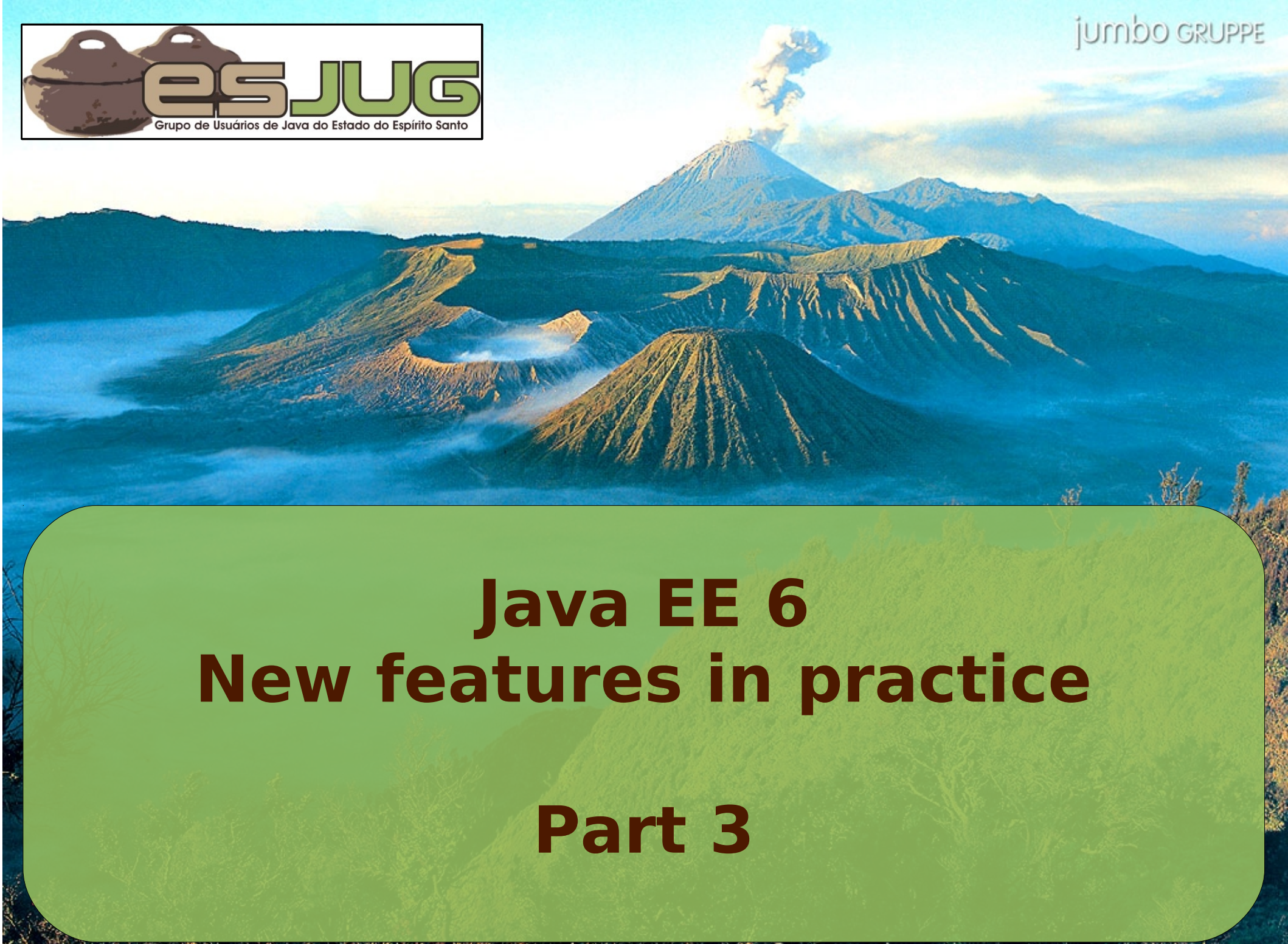
Call these using AJAX!

That's all folks...

Image source: http://www.freedigitalphotos.net/images/Coastal_And_Oceans_g117-In_Late_Summer_p20367.html

Conclusions

- Java EE 6 brings many new things;
- These things bring flexibility, extensibility and ease of development to the platform;
- In three presentations, we only introduced them:
  - Each topic can be explored in depth;
  - We leave this to you...
- Happy coding with Java EE 6!



Java EE 6
New features in practice

Part 3