Table of Contents

ServletServer 1
REST Dienste 2
Life-cycle Objekte 4
Objektspezifische Aufrufe 4
Environment abhängiger Zugriff 4
Checkliste 4

Standardmässig wird bereits darauf geachtet das Object Zugriffe nur mit Authentifizierung möglich sind. Wenn man das nicht ausreicht, können in JVx weitere Sicherheitsvorkehrungen aktiviert werden:

ServletServer

Der `ServletServer` wird für die Kommunikation zwischen einem Desktop Client und dem Backend benötigt. Sie können den `ServletServer` vollständig deaktivieren wenn Sie keinen Desktop Client verwenden oder sie nicht direkt `HttpConnection` verwenden.

Um die möglichen Funktionen einzuschränken, können Sie im config.xml der Server Zone folgendes verwenden:

```
<security>
  <serializedaccess>
    <rules>
      <allow>Session!.*</allow>

      <deny>Session!.createSubSession(*)</deny>
    </rules>
  </serializedaccess>
</security>
```

Mit obigen Regeln definieren Sie das eine Anmeldung am Server möglich ist, aber es ist nicht möglich eine Sub Connection zu erzeugen. Es ist nicht möglich weitere Aufrufe erfolgreich durchzuführen. Die deny Regeln werden vor den allow Regeln angewandt!

Regeldefinition

Eine Regel wird definiert durch: **Objektname.Methodenname(*[param1[, param2]]*)**

- Die eckigen Klammern definieren optionale Felder
- Mit der wildcard Notation können Sie die Zugriffsteuerung feingranular steuern

Mehr Kontrolle

Wenn Sie mehr Kontrolle benötigen z.B. weil Sie zustandsabhängige Zugriffskontrolle benötigen, dann erreichen Sie das durch Implementierung von `ISerializedCommunicationAccessChecker`:

```
public class CustomAccessChecker implements
ISerializedCommunicationAccessChecker
{
    public void init(IServer pServer)
    {
    }

    public boolean isAccessAllowed(IRequest pRequest, ISession pSession,
String pObjectName, String pMethod,
```

```
                                        Object[] pParams,
ChangedHashtable<String, Object> pProperties, boolean pIsCallBack)
    {
        if (IConnection.OBJ_SESSION.equals(pObjectName))
        {
            return true;
        }

        if (pSession != null)
        {
            if ("myapplication".equals(pSession.getApplicationName())
            {
                if ("users".equals(pObjectName))
                {
                    return true;
                }
            }
        }

        return false;
    }
}
```

Die Klasse definieren Sie dann im `config.xml` der Server Zone:

```
<security>
  <serializedaccess>
    <checker>j.v.x.CustomAccessChecker</checker>
  </serializedaccess>
</security>
```

Sie können beliebig viele `checker` definieren.

Eine weitere Möglichkeit ist das Laden Ihres Checkers über den Standard Java Service Loader.

REST Dienste

Wenn Sie REST Dienste verwenden um den Zugriff auf Applikationsdaten mittels standard HTTP zu ermöglichen, wie z.B:

```
<servlet>
  <servlet-name>RestletServlet</servlet-name>
  <servlet-class>com.sibvisions.rad.server.http.rest.RESTServlet</servlet-
class>

  <init-param>
    <param-name>org.restlet.application</param-name>
    <param-value>com.sibvisions.rad.server.http.rest.RESTAdapter</param-
value>
```

```
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>RestletServlet</servlet-name>
  <url-pattern>/services/rest/*</url-pattern>
</servlet-mapping>
```

dann können Sie einzelne Dienste deaktivieren.

Admin Dienste

Die Admin Dienste generell deaktivieren:

```
<init-param>
    <param-name>adminservices.available</param-name>
    <param-value>false</param-value>
</init-param>
```

Die Admin Dienste sind Standardmässig aktiviert.

Um einzelne Dienste zu deaktivieren:

```
<init-param>
    <param-name>adminservice.NAME.enabled</param-name>
    <param-value>false</param-value>
</init-param>
```

Vordefinierte Admin Dienste:

- checkDB
- changePassword
- testAuthentication
- memoryStatus

Um alle Admin Dienste zu deaktivieren:

```
<init-param>
    <param-name>adminservices.enabled</param-name>
    <param-value>false</param-value>
</init-param>
```

Benutzerdefinierte Admin Dienste

Es gibt in JVx die Möglichkeit, die Admin Dienste zu erweitern. Das sind dann sogenannte benutzerdefinierte Admin Dienste. Um diese zu deaktiveren:

```
<init-param>
```

```
    <param-name>adminservice.NAME.enabled</param-name>
    <param-value>false</param-value>
</init-param>
```

Life-cycle Objekte

In den Life-cycle Objekten wird die Business Logik definiert. Die Standard Sicherheit sieht vor das alle public Methoden auch aufrufbar sind. Wenn Sie eine Methode auf `protected` oder `private` ändern, dann ist diese auch nicht mehr aufrufbar, z.B.:

```
protected IDBAccess getDataSource() throws Exception
{
    ...
}
```

Wenn eine Methode jedoch `public` sein muss, dann können Sie diese auch mittels Annotation vor externen Aufrufen schützen:

```
@NotAccessible
public IDBAccess getDataSource() throws Exception
{
    ...
}
```

Details zu `@Accessible` und `@NotAccessible` finden sie hier.

Objektspezifische Aufrufe

Wenn Sie in ihrer Applikation spezifische Zugriffe steuern wollen, dann können Sie das mit der Implementierung von IObjectAccessController erreichen. Dieses Interface wird bei jedem Aufruf eines Life-cycle Objektes geprüft. Dadurch können Sie auf Instanz Ebene feingranular steuern.

Die Zugriffsteuerung für REST funktioniert auch nur bei REST calls und die Zugriffsteuerung für `ServletServer` ebenfalls nur wenn eine `HttpConnection` verwendet wird. Der Einsatz des IObjectAccessController funktioniert unabhängig davon.

Environment abhängiger Zugriff

Wenn Sie mit Accessible und NotAccessible arbeiten, dann können Sie den Zugriff auch per Environment steuer, wie hier beschrieben.

Checkliste

- `ServletServer` deaktivieren oder Aufrufe einschränken

- REST Dienste komplett deaktivieren oder lediglich einzelne Dienste aktivieren
- Life-cycle Objekte kontrollieren (public, protected, private) und Einsatz von `@NotAccessible` prüfen
- Environment abhängiger Einsatz von `@Accessible` und `@NotAccessible`

From:
https://doc.sibvisions.com/ - **Documentation**

Permanent link:
https://doc.sibvisions.com/de/jvx/security

Last update: **2026/11/11 12:26**