pmlopes Restore reverted commit


```
/*******************************************************************************
 * Copyright (c) 2019 Stephane Bastian
 *
 * This program and the accompanying materials are made available under the 2
 * terms of the Eclipse Public License 2.0 which is available at
 * http://www.eclipse.org/legal/epl-2.0.
 *
 * SPDX-License-Identifier: EPL-2.0 3
 *
 * Contributors: 4
 *   Stephane Bastian - initial API and implementation
 *******************************************************************************/
package io.vertx.ext.auth.properties.impl;

import io.vertx.core.Future;
import io.vertx.core.Vertx;
import io.vertx.ext.auth.authentication.CredentialValidationException;
import io.vertx.ext.auth.authentication.Credentials;
import io.vertx.ext.auth.authentication.UsernamePasswordCredentials;
import io.vertx.ext.auth.authorization.Authorization;
import io.vertx.ext.auth.authorization.RoleBasedAuthorization;
import io.vertx.ext.auth.authorization.WildcardPermissionBasedAuthorization;
import io.vertx.ext.auth.properties.PropertyFileAuthentication;
import io.vertx.ext.auth.properties.PropertyFileAuthorization;

import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * @author <a href="mail://stephane.bastian.dev@gmail.com">Stephane Bastian</a>
 */
public class PropertyFileAuthenticationImpl implements PropertyFileAuthentication, PropertyFileAuthorization {
  private final static Logger logger = Logger.getLogger(PropertyFileAuthentication.class.getName());

  private static class User {
    final String name;
    String password;
    final Map<String, Role> roles;

    private User(String name) {
      this.name = Objects.requireNonNull(name);
      this.roles = new HashMap<>();
    }

    private void addRole(Role role) {
      Objects.requireNonNull(role);
      roles.put(role.name, role);
    }
  }

  private static class Role {
    final String name;
    final Set<String> permissions;

    private Role(String name) {
      this.name = Objects.requireNonNull(name);
      this.permissions = new HashSet<>();
    }

    private void addPermission(String permission) {
      Objects.requireNonNull(permission);
      permissions.add(permission);
    }
  }

  private final String path;

  private final Map<String, User> users = new HashMap<>();

  public PropertyFileAuthenticationImpl(Vertx vertx, String path) {
    Objects.requireNonNull(vertx);
    this.path = Objects.requireNonNull(path);
    final Map<String, Role> roles = new HashMap<>();

    String fileContent = vertx.fileSystem().readFileBlocking(path).toString(StandardCharsets.UTF_8);
    String[] lines = fileContent.split("\n");
    for (String line : lines) {
      if (line.length() == 0 || line.startsWith("#")) {
        // skip empty lines or comments
        continue;
      }

      if (line.startsWith("user.")) {
        logger.log(Level.FINE, () -> "read user line: " + line);
        String usernameAndRoles = line.substring(5);
        int index = usernameAndRoles.indexOf('=');
        String tmpName = index > 0 ? usernameAndRoles.substring(0, index).trim() : "";
        String tmpRoles = index > 0 ? usernameAndRoles.substring(index + 1).trim() : "";
        if (tmpName.length() > 0) {
          User user = new User(tmpName);
          users.put(tmpName, user);
          int roleIndex = 0;
```

```
            for (String tmpRole : tmpRoles.split(",")) {
              tmpRole = tmpRole.trim();
              if (roleIndex == 0) {
                user.password = tmpRole;
              } else if (tmpRole.length() > 0) {
                Role role = roles.get(tmpRole);
                if (role == null) {
                  role = new Role(tmpRole);
                  roles.put(tmpRole, role);
                }
                user.addRole(role);
              }
              roleIndex++;
            }
          } else {
            logger.log(Level.WARNING, () -> "read blank username - " + line);
          }
        } else if (line.startsWith("role.")) {
          logger.log(Level.FINE, () -> "read role line - " + line);
          String roleAndProperties = line.substring(5);
          int index = roleAndProperties.indexOf('=');
          String tmpName = index > 0 ? roleAndProperties.substring(0, index).trim() : "";
          String tmpProperties = index > 0 ? roleAndProperties.substring(index + 1).trim() : "";
          if (tmpName.length() > 0) {
            Role role = roles.get(tmpName);
            if (role == null) {
              role = new Role(tmpName);
              roles.put(tmpName, role);
            }
            for (String tmpProperty : tmpProperties.split(",")) {
              tmpProperty = tmpProperty.trim();
              if (tmpProperty.length() > 0) {
                role.addPermission(tmpProperty);
              }
            }
          } else {
            logger.log(Level.WARNING, () -> "read blank role - " + line);
          }
        } else {
          logger.log(Level.WARNING, () -> "read unknow line - " + line);
        }
      }
    }

    private Future<User> getUser(String username) {
      if (!users.containsKey(username)) {
        return Future.failedFuture("unknown user");
      }

      return Future.succeededFuture(users.get(username));
    }

    @Override
    public Future<io.vertx.ext.auth.User> authenticate(Credentials credentials) {
      final UsernamePasswordCredentials authInfo;
      try {
        try {
          authInfo = (UsernamePasswordCredentials) credentials;
        } catch (ClassCastException e) {
          throw new CredentialValidationException("Invalid credentials type", e);
        }
        authInfo.checkValid(null);
      } catch (RuntimeException e) {
        return Future.failedFuture(e);
      }

      return getUser(authInfo.getUsername())
        .compose(propertyUser -> {
          if (Objects.equals(propertyUser.password, authInfo.getPassword())) {
            io.vertx.ext.auth.User user = io.vertx.ext.auth.User.fromName(propertyUser.name);
            // metadata "amr"
            user.principal().put("amr", Collections.singletonList("pwd"));
            return Future.succeededFuture(user);
          } else {
            return Future.failedFuture("invalid username/password");
          }
        });
    }

    @Override
    public String getId() {
      // use the path as the id
      return path;
    }

    @Override
    public Future<Void> getAuthorizations(io.vertx.ext.auth.User user) {
      String username = user.principal().getString("username");
      return getUser(username)
        .onSuccess(record -> {
          Set<Authorization> result = new HashSet<>();
          for (Role role : record.roles.values()) {
            result.add(RoleBasedAuthorization.create(role.name));
            for (String permission : role.permissions) {
              result.add(WildcardPermissionBasedAuthorization.create(permission));
            }
          }
          user.authorizations().put(getId(), result);
        })
        .mapEmpty();
    }
  }
```