



EBook Gratis

APRENDIZAJE mybatis

Free unaffiliated eBook created from
Stack Overflow contributors.

#mybatis

Tabla de contenido

Acerca de.....	1
Capítulo 1: Empezando con mybatis.....	2
Observaciones.....	2
Examples.....	2
Hola mundo usando Spring Boot y Maven.....	2
Spring Boot- Jersey-Mybatis-MySql REST Proyecto Maven paso a paso.....	4
Creditos.....	13

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [mybatis](#)

It is an unofficial and free mybatis ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mybatis.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con mybatis

Observaciones

Esta sección proporciona una descripción general de qué es mybatis y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de mybatis, y vincular a los temas relacionados. Dado que la Documentación para mybatis es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Hola mundo usando Spring Boot y Maven

1. Agregue las dependencias necesarias al proyecto POM (`mybatis` y `mybatis-spring`):

```
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>x.x.x</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.3.1-SNAPSHOT</version>
</dependency>
```

Se necesita una `SqlSessionFactory` para crear sesiones de mybatis. Spring permite que uno se configure rápidamente utilizando `SqlSessionFactoryBean` . Simplemente defina un bean `SqlSessionFactoryBean` y proporcione la referencia al bean `DataSource` :

```
@Bean
public SqlSessionFactoryBean sqlSessionFactoryBean(DataSource dataSource) {
    SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();
    sqlSessionFactoryBean.setDataSource(dataSource);
    return sqlSessionFactoryBean;
}
```

2. Definir un *mapeador* de mybatis. El asignador es una interfaz java que contendrá consultas SQL y traducirá las llamadas de métodos a consultas `JDBC` . Si se está utilizando una base de datos `HSQLDB` inicio de Spring, se puede crear la siguiente consulta. (Hay tablas involucradas; simplemente devuelve una cadena construida usando un parámetro proporcionado por el usuario).

```
public interface HelloWorldMapper {

    @Select("VALUES ('Hello ' || #{origin})")
```

```
String getString(@Param("origin") String origin);  
}
```

3. Registre los mapeadores existentes para que Mybatis y Spring puedan conocerlos. Agregue `@MapperScan` a la configuración de spring java con el nombre del paquete "raíz" que contiene las interfaces del asignador. Esta anotación registrará automáticamente las interfaces como resortes, que se pueden inyectar fácilmente en cualquier parte de la aplicación.

```
@Autowired  
private HelloWorldMapper helloWorldMapper;  
  
// invoking the mapper method  
helloWorldMapper.getString("World");
```

Aquí está la clase de configuración completa de Spring, con una invocación de prueba de la consulta:

```
package com.example;  
  
import org.apache.ibatis.session.SqlSessionFactory;  
import org.mybatis.spring.SqlSessionFactoryBean;  
import org.mybatis.spring.SqlSessionTemplate;  
import org.mybatis.spring.annotation.MapperScan;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.boot.context.TypeExcludeFilter;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.ComponentScan;  
import org.springframework.context.annotation.FilterType;  
  
import javax.annotation.PostConstruct;  
import javax.sql.DataSource;  
import java.lang.annotation.*;  
  
@SpringBootApplication  
@EnableAutoConfiguration  
@ComponentScan  
@MapperScan("com.example")  
public class DemoApplication {  
  
    @Autowired  
    private HelloWorldMapper helloWorldMapper;  
  
    @Bean  
    public SqlSessionFactory sqlSessionFactory(DataSource dataSource) throws Exception {  
        SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();  
        sqlSessionFactoryBean.setDataSource(dataSource);  
        return (SqlSessionFactory) sqlSessionFactoryBean.getObject();  
    }  
  
    @PostConstruct  
    public void init() {  
        System.out.println(helloWorldMapper.getString("World"));  
    }  
}
```

```

}

public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
}

}

```

Spring Boot- Jersey-Mybatis-MySql REST Proyecto Maven paso a paso

Requirement for Local development

- 1) MySQL server(I am Using XAMPP for MySQL server)
- 2) For Test API You can use Postman(optional)
- 3) Before run application,make sure MySQL server is running, properly prepare your application.properties file(DB_Name, Username,Password).

1.Agregue la siguiente dependencia, principal en su archivo POM.xml

```

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.0.RELEASE</version>
</parent>
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jersey</artifactId>
  </dependency>
  <dependency>
    <groupId>org.glassfish.jersey.media</groupId>
    <artifactId>jersey-media-multipart</artifactId>
    <version>2.18</version>
  </dependency>
  <dependency>

```

```

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
    </dependency>

    <dependency>
        <groupId>commons-validator</groupId>
        <artifactId>commons-validator</artifactId>
        <version>1.5.0</version>
    </dependency>

    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-sample-annotation</artifactId>
        <version>1.1.1</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.4.1</version>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
</dependencies>

```

Mi estructura de proyecto debería ser esta

```

src-->
  main-->
    java
      |-->com-->controller
      |           |--UserController.java
      |           |--CORSResponseFilter.java
      |           entity
      |           |--User.java
      |           mapper
      |           |--UserMapper.java
      |           JerseyConfig.java
      |           SampleMapperApplication.java
      |
    resources
      |-->application.properties

```

2.Cree una clase de entidad llamada Usuario en src \ main \ java \ com \ entity \ User.java

```

package com.entity;

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Pattern;

```

```

import javax.validation.constraints.Size;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy= GenerationType.AUTO)
    private int id;

    @Size(min = 2, max =50)
    @Pattern(regexp = "[A-Za-z. ]*", message = "First name requires valid character")
    private String firstname;

    @Size(min = 2, max =50)
    @Pattern(regexp = "[A-Za-z. ]*", message = "First name requires valid character")
    private String lastname;

    //@Column(unique = true)
    @NotNull(message = "Email requires valid value")
    @NotEmpty(message = "Email requires non empty value")
    @Email(message = "Email requires valid format")
    private String email;

    @Size(min = 2, max =50)
    @Pattern(regexp = "[A-Za-z.0-9 ]*", message = "First name requires valid character")
    private String password;

    @Pattern(regexp = "[0-9.\\-+ ]*", message = "Phone requires valid alphanumeric
characters")
    private String phone;

    private int age;
    /*setter getter method here*/

}

```

3.Cree una interfaz llamada UserMapper.java en src \ main \ java \ com \ mapper \ UserMapper.java

```

package com.mapper;

import com.entity.User;
import org.apache.ibatis.annotations.*;

import java.util.List;

@Mapper
public interface UserMapper {
    @Insert("INSERT INTO USER ( firstname,lastname, email,password,age ,phone ) VALUES (
#{user.firstname}, #{user.lastname},#{user.email},#{user.password},
#{user.age},#{user.phone})")
    Integer insertUser(@Param("user") User user) throws Exception;

    @Select("select * from user where email = #{email}")
    User findByEmail(@Param("email") String email);

    @Select("select * from user where email = #{email} AND id!=#{id}")

```



```

    User findByEmailNotUser(@Param("email") String email,@Param("id") Integer id);

    @Select("select * from user where id = #{id}")
    User findById(@Param("id") Integer id);

    @Select("select * from user where email = #{email} AND password = #{password}")
    User login(@Param("email") String email,@Param("password") String password);

    @Select("select * from user")
    List<User> getUsers();

    @Update("UPDATE USER SET firstname = #{user.firstname},lastname = #{user.lastname},email =
    #{user.email},phone = #{user.phone},age = #{user.age} WHERE id = #{user.id}")
    Integer updateUser(@Param("user") User user) throws Exception;

    @Delete("DELETE from user where id = #{id}")
    Integer deleteById(@Param("id") Integer id);

}

```

4.Crear dos clases UserController.java en src \ main \ java \ com \ controller \ UserController.java

```

package com.controller;

import com.entity.User;
import com.mapper.UserMapper;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;

import javax.validation.ConstraintViolation;
import javax.validation.Validation;
import javax.validation.Validator;
import javax.ws.rs.*;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import java.util.Collections;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Set;

@Controller
@Path("/users")
public class UserController {
    static final Logger logger = LoggerFactory.getLogger(UserController.class);
    private static Validator validator =
    Validation.buildDefaultValidatorFactory().getValidator();

    @Autowired
    private UserMapper userMapper;

    @POST
    @Consumes("application/json")

```

```

@Produces("application/json")
public Response create(User user) {
    LinkedHashMap<Object, Object> serviceResponse = new LinkedHashMap<Object, Object>();
    logger.info("Starting to create a user");

    try {
        Set<ConstraintViolation<User>> validateErrors = validator.validate(user);
        if (validateErrors.isEmpty()) {
            if (userMapper.findByEmail(user.getEmail()) != null) {
                serviceResponse.put("duplicate_email", "user already exist");
            } else {
                Integer createPerson = userMapper.insertUser(user);

                if (createPerson != 1) {
                    serviceResponse.put("created", "unable to create user");
                } else {
                    logger.info("Successfully created User.");
                    serviceResponse.put("created_msg", "Successfully created User");
                }
            }
            return
Response.status(Response.Status.CREATED).entity(serviceResponse).build();
        } else {
            logger.info("Failed to create a user due to field validation errors.");
            logger.debug("Unable to create a user due to validation errors using {}",
user);

            //JSONObject jsonObj = new JSONObject(validateErrors.toString());
            serviceResponse.put("error", validateErrors.toString());
            return Response.status(400).entity(serviceResponse).build();

        }
    } catch (Exception e) {
        logger.debug("<< create()");
        return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(serviceResponse).build();
    }
}

@GET
@Produces("application/json")
public Response getUsers() {

    LinkedHashMap<String, Object> response = new LinkedHashMap<String, Object>();
    try {
        List<User> listUsers = userMapper.getUsers();
        if (listUsers == null) {
            response.put("users", Collections.emptyMap());
        } else {
            response.put("total", listUsers.size());
            response.put("users", listUsers);
        }
        return Response.status(Response.Status.OK).entity(listUsers).build();
    } catch (Exception ex) {

        response.put("user", "Not Found");
        return Response.status(Response.Status.BAD_REQUEST).entity(response).build();
    }
}

```

```

@PUT
@Path("/{id}")
@Produces("application/json")
public Response updateUser(@PathParam("id") Integer id, User user) {
    //String personJson = gson.toJson(user);
    //logger.debug(">> create({})", personJson);
    //LinkedHashMap<Object, Object> apiResponse = new LinkedHashMap<>();
    LinkedHashMap<Object, Object> serviceResponse = new LinkedHashMap<Object, Object>();
    logger.info("Starting to create a person");

    try {
        Set<ConstraintViolation<User>> validateErrors = validator.validate(user);
        if (validateErrors.isEmpty()) {
            if (userMapper.findById(id) == null) {
                serviceResponse.put("msg", "User Not Found");
            } else {
                if (userMapper.findByEmailNotUser(user.getEmail(), user.getId()) != null)
                {
                    serviceResponse.put("duplicate_Email", "user already exist");
                } else {
                    int updateUser = userMapper.updateUser(user);
                    if (updateUser == 0) {
                        serviceResponse.put("created", "unable to update User");
                    } else {
                        logger.info("Successfully update user.");
                        serviceResponse.put("update", user);
                    }
                }
            }
            return Response.status(Response.Status.OK).entity(user).build();
        } else {
            logger.info("Failed to update a user due to field validation errors.");
            // logger.debug("Unable to update a user due to validation errors using {}",
personJson);
            serviceResponse.put("error", validateErrors.toString());

            return Response.status(400).entity(serviceResponse).build();
        }
    } catch (Exception e) {
        logger.debug("<< create()");
        return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(serviceResponse).build();
    }
}

@GET
@Path("/{id}")
@Produces("application/json")
public Response getPerson(@PathParam("id") int userId) {

    LinkedHashMap<String, Object> response = new LinkedHashMap<String, Object>();

    try {
        User user = userMapper.findById(userId);

        if (user == null) {
            response.put("User", Collections.emptyMap());
        } else {
            response.put("user", user);
        }
        return Response.status(Response.Status.OK).entity(user).build();
    }
}

```

```

    } catch (Exception ex) {

        response.put("user", "Not Found");
        return Response.status(Response.Status.BAD_REQUEST).entity(response).build();
    }
}

@DELETE
@Path("/{id}")
@Consumes("application/json")
@Produces("application/json")
public Response deleteUser(@PathParam("id") int userId) {

    //LinkedHashMap<Object, Object> apiResponse = new LinkedHashMap<>();
    LinkedHashMap<Object, Object> serviceResponse = new LinkedHashMap<Object, Object>();
    logger.info("Starting to delete a user");

    try {
        int deletePerson = userMapper.deleteById(userId);
        if (deletePerson == 0) {
            serviceResponse.put("delete", "unable delete user");
        } else {
            logger.info("Successfully delete user.");
            serviceResponse.put("delete", "Successfully delete user.");
        }
        return Response.status(Response.Status.OK).entity(serviceResponse).build();
    } catch (Exception e) {

        logger.debug("<< create()");
        return
Response.status(Response.Status.INTERNAL_SERVER_ERROR).entity(serviceResponse).build();
    }
}
}

```

5.Create la clase CORSResponseFilter.java en src \ main \ java \ com \ controller \ CORSResponseFilter.java

```

package com.controller;

import javax.ws.rs.container.ContainerRequestContext;
import javax.ws.rs.container.ContainerResponseContext;
import javax.ws.rs.container.ContainerResponseFilter;
import javax.ws.rs.core.MultivaluedMap;
import java.io.IOException;

public class CORSResponseFilter implements ContainerResponseFilter {
    public void filter(ContainerRequestContext requestContext, ContainerResponseContext
responseContext)
        throws IOException {

        MultivaluedMap<String, Object> headers = responseContext.getHeaders();

        headers.add("Access-Control-Allow-Origin", "*");
        //headers.add("Access-Control-Allow-Origin", "http://abcd.org"); //allows CORS
requests only coming from abcd.org
        headers.add("Access-Control-Allow-Methods", "GET, POST, DELETE, PUT");
    }
}

```

```
        headers.add("Access-Control-Allow-Headers", "X-Requested-With, Content-Type");
    }
}
```

6. Registre su Clase de configuración en la clase JerseyConfig src \ main \ java \ com \ JerseyConfig.java

```
package com;

import com.controller.CORSResponseFilter;
import com.controller.UserController;
import org.glassfish.jersey.server.ResourceConfig;
import org.springframework.context.annotation.Configuration;

import javax.ws.rs.ApplicationPath;

@Configuration
@ApplicationPath("/api")
public class JerseyConfig extends ResourceConfig {
    public JerseyConfig() {
        register(CORSResponseFilter.class);
        register(UserController.class);
    }
}
```

7. Cree su clase de aplicación llamada SampleMapperApplication en src \ main \ java \ com \ SampleMapperApplication.java

```
package com;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SampleMapperApplication implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(SampleMapperApplication.class, args);
    }
}
```

8. Prepare su archivo application.properties src \ main \ resources \ application.properties

```
# localhost:3306 will be replace your DB server url if you want. I am using MySQL local DB
server
spring.datasource.url = jdbc:mysql://localhost:3306/YOUR_DB_NAME
```

```
db.driver=com.mysql.jdbc.Driver
#if you want update, just replace create with update
spring.jpa.hibernate.ddl-auto=create
#If you want to change default port(8080) number, Comment out below line code
#server.port = 8090

# Username and password
spring.datasource.username = YOUR_DB_USER_NAME
spring.datasource.password =YOUR_DB_USER_PASSWORD
```

Para probar la API REST del usuario

```
List of User(GET) ---->http://localhost:8080/api/users
Create User(POST) ---->http://localhost:8080/api/users
Get User(GET) ----->http://localhost:8080/api/users/1
Update User(PUT) ----->http://localhost:8080/api/users/1
Delete User(Delete)--->http://localhost:8080/api/users/1
```

Lea Empezando con mybatis en línea: <https://riptutorial.com/es/mybatis/topic/6692/empezando-con-mybatis>

Creditos

S. No	Capítulos	Contributors
1	Empezando con mybatis	4444 , Abdullah Al Hafiz , Community , Fabien Benoit-Koch , Roman Konoval , Skyler Tao