Booster框架快速入门指引

1. 什么是Booster框架

Booster是整合Spring Boot和IView形成的一种快速应用开发框架，它集成了常见JavaEE应用所需要的通用功能，使开发人员能够快速搭建工程，专注于业务逻辑开发。

2. 面向的读者

我们假定读者已经具备Java、HTML和Javascript语言的开发经验，还需要了解:

1. Spring Framework
2. Spring Boot
3. Mybatis
4. VUE.js
5. IView

另外，为了编译整个工程，你需要了解:

1. Maven
2. NPM

3. 关于案例

案例中，我们将构建一个会议室管理预定系统，典型业务场景为：用户登录系统，预定查询会议室可用时间段，然后预定某一个时间段的会议室。

4. 开始入门

4.1 整体步骤

我们将按照以下步骤完成基于常见开发任务：

1. 环境准备
2. 创建工程；
3. 建立数据模型；
4. 添加页面组件；
5. 添加路由；
6. 添加Mapper；
7. 添加Service；
8. 添加REST API；

对于日常开发工作而言，仅需要3-8步既可。

4.2 环境准备

4.2.1 前端环境准备

1.Node.js

下载最新版本Node.js并安装，安装后执行`node -v`、`npm -v`命令检查是否安装成功

2.NPM镜像设置

配置NPM的registry地址，为公司的依赖包库，执行

`npm config set registry http://spr.foresealife.com/repository/npmjs-all/`

配置后可通过下面方式来验证是否成功

```
npm config get registry
// 或
npm info express
```

3.开发工具

Eclipse不支持vue开发，我们推荐使用WebStorm或者Visual Studio Code

4.2.2 后端环境准备

1.JDK

JDK需要安装JDK8，安装后需要配置环境变量：

```
JAVA_HOME：JDK的安装路径，如C:\jdk1.8，在这路径下你应该能够找到bin、lib等目录。
PATH：原来的内容;%JAVA_HOME%\bin
CLASSPATH：.;%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar
```

安装后，执行`java`、`javac`命令验证

2.Maven

下载Maven，解压，然后配置环境变量

```
PATH：原来的内容;%MAVEN_HOME%\bin
MAVEN_HOME：maven的解压目录，如D:\apache-maven-3.5.0
```

修改`conf/settings.xml`,在`mirrors`节点中新增以下内容，配置为公司内网的Maven仓库

```
<mirror>
      <id>central</id>
      <mirrorOf>*</mirrorOf>
      <name>central</name>
      <url>http://spr.foresealife.com/repository/maven-public/</url>
</mirror>
```

执行`mvn -v`命令验证

注意，后续创建工程的booster-archetype位于内网镜像仓库，如果找不到这个booster-archetype，请检查是否按照上面描述配置maven。

3.开发工具

开发工具推荐使用IntelliJ IDEA，或者Eclipse

4.3 创建工程

1.在任意空目录，执行以下命令创建工程：

```
mvn archetype:generate -B -DarchetypeGroupId=com.foresealife.booster  -
DarchetypeArtifactId=booster-archetype -DarchetypeVersion=1.4.0 -DgroupId=com.foresealife -
DartifactId=imeeting -Dversion=1.0.0 -Dpackage=com.foresealife.imeeting
```

参数说明:

- -DgroupId=com.foresealife   工程所属的groupId，使用所属组织的反序域名,这里使用的前海人寿的域名
- -DartifactId=imeeting   工程所属的artifact，使用系统名称，这里我们将系统命名为imeeting
- -Dversion=1.0.0   版本号
- -Dpackage=com.foresealife.imeeting   Java代码中的包名，如省略则使用groupId

执行完成后，生产的代码位于与artifact同名的路径下,这里为imeeting,结构如下：

```
├── imeeting-frontend            // 前端代码
│    ├── src                     // 前端源代码
├── backend                      // 后端源代码
│    ├── src                     // 后端源代码
│    ├── ...                     // 前端打包时用到的配置文件
├── installer                    // 打包逻辑
├── pom.xml                      // 项目的Maven POM
```

- imeeting-frontend为前端代码，这是一个标准的Javascript工程，使用npm进行管理，可以独立编译运行；
- imeeting-backend为后端代码，这个是一个标准的Java工程，使用maven进行管理，可以独立编译运行；
- imeeting-installer为打包逻辑，利用maven中提供的各种插件将前端和后端编译后的代码整合为一个可运行的jar包；

2. 创建Git工程

我们使用Git作为代码管理的工具，可以有效、高速地处理从很小到非常大的项目版本管理。

在imeeting根目录下创建git工程，前后端作为同一个项目管理

```
git init
git add .
git commit -m "create project"
```

3.下载前端依赖

```
cd imeeting-frontend
npm install
```

4.开发工具支持

Eclipse

执行`mvn eclipse:eclipse`命令，可以将项目创建为Eclipse工程, 完成后可以在eclipse中导入工程。

IntelliJ IDEA

在IDEA中打开工程时选中根路径下的pom.xml, 以工程方式打开则IDEA会自动导入整个工程。

4.4 编译和打包

在工程的根路径下执行如下命令可以完成编译打包：

```
mvn clean install
```

当然可以分别进行编译打包。 前端：

```
cd imeeting-frontend
npm run build
```

后端：

```
cd imeeting-backend
mvn clean install
```

打包:

```
cd imeeting-installer
mvn clean install
```

最终生成安装包位于

```
imeeting-installer/target/imeeting.zip
```

4.5 开发模式

在开发阶段，我们期望编码过程能实现所见即所得，避免复杂的打包部署过程，因此我们可以在前端根路径下：

```
npm run dev
```

在后端利用IDE支持，直接运行com.foresealife.imeeting.Application类启动应用。

或者在后端的target目录下执行以下命令启动后端：

```
java -jar  imeeting-backend-1.0.0.jar
```

在浏览器中打开http://localhost:8082访问，默认登录用用户为:chenh009,密码：Fsl@2017

4.6 建立数据模型

首先，我们需要建立会议室预定系统的数据模型。 在后端的建立实体类com.foresealife.imeeting.entity.MeetingRoom代表会议室

```
public class MeetingRoom {
    private int id;
    private String name;
    private int capacity;
    private boolean hasProjector;
    private boolean hasRemoteConferenceDevice;
    private boolean hasNetworkCable;
    private List<MeetingRoomReservation> reservationList;
    //省略了getter和setter
}
```

com.foresealife.imeeting.entity.MeetingRoomReservation代表预定

```
public class MeetingRoomReservation {
    private int id;
    private int roomId;
    private String user;
    private Date startTime;
    private Date endTime;
    private String description;
   //省略了getter和setter
}
```

示例中我们使用h2内存数据库，在后端的src/resource/schema-h2.sql中增加建表语句

```
-- 会议室预定 数据表
CREATE TABLE IF NOT EXISTS meeting_room (
  id integer PRIMARY KEY,
  name varchar(256),
  capacity integer,
  has_projector boolean default false ,
  has_remote_conference_device boolean default false ,
  has_network_cable boolean default false
);

delete from meeting_room ;

insert into meeting_room (id,
name,capacity,has_projector,has_remote_conference_device,has_network_cable) values (1,'视频会议
室',30,true,true,true);
insert into meeting_room (id, name,capacity) values (2,'电视会议室',10);

insert into meeting_room (id, name,capacity) values (3,'小会议室1',5);
insert into meeting_room (id, name,capacity) values (4,'小会议室2',8);

CREATE TABLE IF NOT EXISTS meeting_room_reservation (
  id integer auto_increment PRIMARY KEY,
  room_id integer,
  user varchar(256),
  start_time TIMESTAMP ,
  end_time TIMESTAMP,
  description varchar(256)
);
```

4.7 前端实现

4.7.1 添加VUE组件

我们需要在前端添加会议室查询和会议室预定界面，他们是标准的VUE组件，使用IView提供的控件。 首先，在前端增加会议室查询界面，创建src/view/meeting/meeting-room-query.vue文件，内容如下：

```
<template>
  <Form ref="meetingroom" :model="meetingroom" :rules='ruleValidate' :label-width="120">
    <FormItem label="参会人数" prop="capacity">
      <InputNumber :max="100" :min="0" v-model="meetingroom.capacity"></InputNumber>
    </FormItem>
    <FormItem label="需要投影仪" prop="hasProjector">
      <Select v-model="meetingroom.hasProjector">
        <Option value="false">否</Option>
        <Option value="true">是</Option>
      </Select>
    </FormItem>
    <FormItem label="需要视频会议设备" prop="hasRemoteConferenceDevice">
      <Select v-model="meetingroom.hasRemoteConferenceDevice">
        <Option value="false">否</Option>
        <Option value="true">是</Option>
      </Select>
    </FormItem>
    <FormItem label="需要网络接口" prop="hasNetworkCable">
      <Select v-model="meetingroom.hasNetworkCable">
        <Option value="false">否</Option>
        <Option value="true">是</Option>
      </Select>
    </FormItem>
    <FormItem>
      <Button type="primary" @click="handleSubmit('meetingroom')">提交</Button>
      <Button @click="handleReset('meetingroom')" style="margin-left: 8px">重置</Button>
    </FormItem>
    <iv-table border :data="meetingroomlist">
      <iv-table-column label="名称" prop="name"></iv-table-column>
      <iv-table-column label="容量" prop="capacity"></iv-table-column>
      <iv-table-column label="投影仪">
        <span slot-scope="scope">{{scope.row.hasProjector ? '有' : '无'}}</span>

      </iv-table-column>
```

```
        <iv-table-column label="视频会议设备">
          <span slot-scope="scope">{{scope.row.hasRemoteConferenceDevice ? '有' : '无'}}</span>
        </iv-table-column>
        <iv-table-column label="网络接口">
          <span slot-scope="scope">{{scope.row.hasNetworkCable ? '有' : '无'}}</span>
        </iv-table-column>
        <iv-table-column label="预定情况" :width="600">
          <ol slot-scope="scope">
            <li v-for="item in scope.row.reservationList" :key="item.id">
              {{getFormattedContent(item)}}
            </li>
          </ol>
        </iv-table-column>
      </iv-table>
    </Form>
</template>
<script>
import { listMeetingRooms } from '@/api/meetingroom'
import { format } from 'date-fns'

export default {
  data () {
    return {
      meetingroom: {
        capacity: 0,
        hasProjector: 'false',
        hasRemoteConferenceDevice: 'false',
        hasNetworkCable: 'false'
      },
      ruleValidate: {
        capacity: [
          { required: true, message: '请输入参会人数', trigger: 'blur' }
        ]
      },
      meetingroomlist: []
    }
  },
  methods: {
    handleSubmit (name) {
      listMeetingRooms(this.meetingroom).then(data => {
        this.meetingroomlist = data.data
        console.log(data)
      })
    },
    handleReset (name) {
      this.$refs[name].resetFields()
    },
    getFormattedContent (item) {
      return `${format(item.startTime, 'YYYY-MM-DD HH:mm')} - ${format(item.endTime, 'YYYY-MM-DD
HH:mm')}  ${item.user}  ${item.description}`
    }
  },

  mounted () {
    this.handleSubmit('meetingroom')
  }
}
</script>
```

然后增加会议室预定界面，创建src/view/meeting/meeting-room-reserve.vue文件,内容如下:

```
<template>
  <Form ref="reservation" :model="reservation" :rules="ruleValidate" :label-width="120">
    <FormItem prop="roomId" label="会议室">
      <Select v-model="reservation.roomId">
        <Option v-for="room in roomList" :value="room.id" :key="room.id">{{ room.name }}
</Option>
      </Select>
    </FormItem>
    <FormItem label="会议主题" prop="description">
      <Input type="text" v-model="reservation.description"></Input>
    </FormItem>
    <FormItem label="开始日期" prop="startTime">
      <DatePicker type="datetime" format="yyyy-MM-dd HH:mm" v-model="reservation.startTime">
</DatePicker>
    </FormItem>
    <FormItem label="结束日期" prop="endTime">
      <DatePicker type="datetime" format="yyyy-MM-dd HH:mm" v-model="reservation.endTime">
</DatePicker>
    </FormItem>
    <FormItem>
      <Button type="primary" @click="handleSubmit('reservation')">提交</Button>
      <Button @click="handleReset('reservation')" style="margin-left: 8px">重置</Button>
    </FormItem>
  </Form>
</template>
<script>
import {reserve, listMeetingRooms} from '@/api/meetingroom'
import {getUser} from '../../libs/util'

export default {
  data () {
    return {
      reservation: {
        roomId: '',
        user: getUser().userName,
        description: '',
        startTime: new Date(),
        endTime: new Date()
      },
      ruleValidate: {
        roomId: [{required: true, message: '请选择会议室'}],
        description: [
          {required: true, message: '请输入会议内容', trigger: 'blur'},

          {type: 'string', min: 3, message: '至少3个字', trigger: 'blur'}
        ],
        startTime: [{required: true, message: '请选择会议开始时间'}],
        endTime: [{required: true, message: '请选择会议结束时间'}]
      },
      roomList: []
    }
  },
  mounted () {
    this.init()
  },
  methods: {
    init () {
      listMeetingRooms({capacity: -1, hasProjector: false, hasRemoteConferenceDevice: false,
hasNetworkCable: false}).then((response) => {
        if (response.status === 200) {
          this.roomList = response.data
        }
      })
    },
    handleSubmit (name) {
      this.$refs[name].validate((valid) => {
        if (valid) {
          reserve(this.reservation).then((response) => {
            if (response.status === 200) {
              this.$Message.success('会议室预定成功')
            } else {
              this.$Message.error('预定失败:' + response.status + ',' + response.message)
            }
          })
        } else {
          this.$Message.error('表单数据不合法!')
        }
      })
    },
    handleReset (name) {
      this.$refs[name].resetFields()
    }
  }
}
</script>
```

这里，我们将调用后端的代码都封装到了单独的javascript文件中,创建src/api/meetingroom.js文件,内容如下:

```
import httpRequest from '@/libs/httpRequest'

export const listMeetingRooms = (params) => {
  return httpRequest.request({
    url: '/api/meetingroom',
    method: 'get',
    params: params
  })

}
```

```
export const reserve = (reservation) => {
  return httpRequest.request({
    url: '/api/meetingroom/reservation',
    method: 'post',
    data: reservation
  })
}
```

4.7.2 添加菜单和路由

我们需要把“会议室查询”和“会议预定”两个菜单添加到主界面，修改src/router/routers.js,在json数组中插入一个对象:

```
{
    path: '/meeting_room',
    name: '会议室管理',
    meta: {
      icon: 'social-buffer',
      title: '会议室管理'
    },
    component: Main,
    children: [
      {
        path: 'meeting_room_query',
        name: '会议室查询',
        meta: {
          icon: 'arrow-graph-up-right',
          title: '会议室查询'
        },
        component: () => import('@/view/meeting/meeting-room-query.vue')
      },
      {
        path: 'meeting_room_reserve',
        name: '会议室预定',
        meta: {
          icon: 'arrow-graph-up-right',
          title: '会议室预定'
        },
        component: () => import('@/view/meeting/meeting-room-reserve.vue')
      }
    ]
  }
```

4.8 后端实现

4.8.1 添加DAO

在后端创建数据访问接口com.foresealife.imeeting.mapper.MeetingRoomMapper

```
package com.foresealife.imeeting.mapper;
```

```
import com.foresealife.imeeting.entity.MeetingRoom;
import com.foresealife.imeeting.entity.MeetingRoomReservation;
import org.apache.ibatis.annotations.Mapper;

import java.util.List;
import java.util.Map;

@Mapper
public interface MeetingRoomMapper {
    List<MeetingRoom> getMeetingRoomList(Map<String ,Object> params);
    List<MeetingRoomReservation> getReservationListByRoomId(int roomId);
    void createReservation(MeetingRoomReservation reservation);
}
```

创建Mapper文件src/main/resources/mapper/meeting_room_mapper.xml,其内容如下:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-
mapper.dtd">
<mapper namespace="com.foresealife.imeeting.dao.mapper.MeetingRoomMapper">
    <resultMap id="meetingRoomResultMap" type="com.foresealife.imeeting.entity.MeetingRoom">
        <id property="id" column="id"/>
        <result property="name" column="name"/>
        <result property="capacity" column="capacity"/>
        <result property="hasProjector" column="has_projector"/>
        <result property="hasRemoteConferenceDevice" column="has_remote_conference_device"/>
        <result property="hasNetworkCable" column="has_network_cable"/>
    </resultMap>
    <resultMap id="reservationResultMap"
type="com.foresealife.imeeting.entity.MeetingRoomReservation">
        <id property="id" column="id"/>
        <result property="roomId" column="room_id"/>
        <result property="user" column="user"/>
        <result property="startTime" column="start_time"/>
        <result property="endTime" column="end_time"/>
        <result property="description" column="description"/>
    </resultMap>
    <select id="getMeetingRoomList" resultMap="meetingRoomResultMap">
        select * from meeting_room where 1=1
        <if test="hasProjector == true">
            and has_projector = #{hasProjector}
        </if>
        <if test="hasRemoteConferenceDevice == true">
            and has_remote_conference_device = #{hasRemoteConferenceDevice}
        </if>
        <if test="hasNetworkCable == true">
            and has_network_cable = #{hasNetworkCable}
        </if>
        <if test="capacity > 0">

            and capacity>=#{capacity}
        </if>
    </select>
    <select id="getReservationListByRoomId" resultMap="reservationResultMap">
            select * from meeting_room_reservation where room_id=#{roomId} and start_time>
CURRENT_DATE()
    </select>
    <insert id="createReservation"
parameterType="com.foresealife.imeeting.entity.MeetingRoomReservation">
        insert into meeting_room_reservation (room_id,user,start_time,end_time,description)
        values (#{roomId},#{user},#{startTime},#{endTime},#{description})
    </insert>
</mapper>
```

这里需要注意，namespace应该等于接口类的全名，查询的名字等于对应的方法名。

本系统配置了mybatis的mapper自动扫描，位于`classpath:/mapper`下的xml文件会被自动注册为mapper，如果需要更改扫描路径，请修改`application.properties`：

`mybatis.mapper-locations=classpath:/mapper/*.xml`

4.8.2 添加Service

通常，我们将业务逻辑封装在service中，但在本例中没有复杂的业务逻辑，因此service只是简单的访问DAO。 创建接口com.foresealife.imeeting.service.MeetingRoomService, 代码如下:

```java
package com.foresealife.imeeting.service;

import com.foresealife.imeeting.entity.MeetingRoom;
import com.foresealife.imeeting.entity.MeetingRoomReservation;

import java.util.List;

public interface MeetingRoomService {

    List<MeetingRoom> getMeetingRoomList(int capacity,  boolean hasProjector, boolean
hasRemoteConferenceDevice,boolean hasNetworkCable);
    List<MeetingRoomReservation> getReservationList(int roomId);
    void createReservation(MeetingRoomReservation reservation);
}
```

创建接口实现com.foresealife.imeeting.service.impl.ForesealifeMeetingRoomService,代码如下:

```java
package com.foresealife.imeeting.service.impl;
import com.foresealife.imeeting.mapper.MeetingRoomMapper;
import com.foresealife.imeeting.entity.MeetingRoom;
import com.foresealife.imeeting.entity.MeetingRoomReservation;
import com.foresealife.imeeting.service.MeetingRoomService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Date;
import java.util.HashMap;

import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

@Service
public class ForesealifeMeetingRoomService implements MeetingRoomService {
    @Autowired
    private MeetingRoomMapper meetingRoomMapper;

    @Override
    public List<MeetingRoom> getMeetingRoomList(int capacity, boolean hasProjector, boolean
hasRemoteConferenceDevice, boolean hasNetworkCable) {
        Map<String, Object> params = new HashMap<>();
        params.put("capacity", capacity);
        params.put("hasProjector", hasProjector);
        params.put("hasRemoteConferenceDevice", hasRemoteConferenceDevice);
        params.put("hasNetworkCable", hasNetworkCable);
        List<MeetingRoom> meetingRoomList = meetingRoomMapper.getMeetingRoomList(params);
        meetingRoomList.forEach(meetingRoom ->
meetingRoom.setReservationList(meetingRoomMapper.getReservationListByRoomId(meetingRoom.getId())
));
        return meetingRoomList;
    }

    @Override
    public List<MeetingRoomReservation> getReservationList(int roomId) {
        return meetingRoomMapper.getReservationListByRoomId(roomId);
    }

    @Override
    public void createReservation(MeetingRoomReservation reservation) {
        List<MeetingRoomReservation> reservationList =
meetingRoomMapper.getReservationListByRoomId(reservation.getRoomId());
        List<MeetingRoomReservation> conflict = reservationList.stream().filter(r ->
isReservationTimeConflict(reservation, r)).collect(Collectors.toList());
        if (conflict.isEmpty()) {
            meetingRoomMapper.createReservation(reservation);
        } else {
            throw new IllegalStateException("会议室预定失败：预定时间冲突");
        }
    }

    private boolean isReservationTimeConflict(MeetingRoomReservation newReservation,
MeetingRoomReservation reservation) {
        return isStartTimeConflict(newReservation.getStartTime(), reservation.getStartTime(),
reservation.getEndTime()) ||
                isEndTimeConflict(newReservation.getEndTime(), reservation.getStartTime(),
reservation.getEndTime());
    }

    private boolean isStartTimeConflict(Date startTime, Date start, Date end) {
        return startTime.equals(start)||(startTime.after(start) && startTime.before(end));
    }
    private boolean isEndTimeConflict(Date endTime, Date start, Date end) {
        return endTime.after(start) && endTime.before(end);
    }
}
```

4.8.3 添加REST API

我们需要在后端的web层添加REST接口，以便前端调用。 添加类com.foresealife.imeeting.web.api.MeetingRoomEndpoint，代码如下：

```
package com.foresealife.imeeting.web.api;
import com.foresealife.imeeting.entity.MeetingRoom;
import com.foresealife.imeeting.entity.MeetingRoomReservation;
import com.foresealife.imeeting.service.MeetingRoomService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import javax.websocket.server.PathParam;
import java.util.List;

@RestController
public class MeetingRoomEndpoint {
    @Autowired
    private MeetingRoomService meetingRoomService;

    @GetMapping("/api/meetingroom")
    public List<MeetingRoom> listMeetingRooms(@RequestParam int capacity, @RequestParam boolean
hasProjector, @RequestParam boolean hasRemoteConferenceDevice, @RequestParam boolean
hasNetworkCable) {
        return meetingRoomService.getMeetingRoomList(capacity, hasProjector,
hasRemoteConferenceDevice,hasNetworkCable);
    }

    @GetMapping("/api/meetingroom/reservation/{roomId}")
    public List<MeetingRoomReservation> getMeetingRoomReservation(@PathParam("roomId") int
roomId) {
        return meetingRoomService.getReservationList(roomId);
    }

    @PostMapping("/api/meetingroom/reservation")
    public void createReservation(@RequestBody MeetingRoomReservation reservation) {
        meetingRoomService.createReservation(reservation);
    }
}
```

5. 源代码

可以通过git获取本例完整的源代码：

```
git clone http://git.foresealife.com/booster/booster-example.git
```

6. 总结

至此，我们已经完成了一个简单的会议室预定应用。关于Booster框架的更多详细内容，请参考《Booster框架开发指南》。