简答题

1. 在 Spring 框架中，什么是控制反转？什么是依赖注入？使用控制反转和依赖注入有什么好处？

控制反转是指 IoC(Inversion of Control)容器，是面向对象编程中的一种设计原则，意为控制反转，它将程序中创建对象的控制权交给 Spring 框架来管理，以便降低计算机代码之间的耦合度，控制反转的实质是获得依赖对象的过程被反转了。这个过程由自身管理变为由 IoC 容器主动注入。这便是依赖注入，由 Ioc 容器在运行期间动态地将某种依赖关系注入对象之中。Ioc 容器通过类型或名称等信息将不同对象注入不同属性中。组件不做定位查询，只提供普通的 java 方法让容器去决定依赖关系。

使用控制反转和依赖注入使得对象与对象之间松散耦合，方便测试，利于功能复用，使得程序的整个体系结构变得非常灵活，同时单例模式防止内存溢出，避免频繁操作。

2. bean 的作用域分别设置为 singleton、prototype、request 时，bean 的创建、销毁有何区别？

- singleton

  当把一个 bean 定义设置为 singleton 作用域时，Spring IOC 容器只会创建该 bean 定义的唯一实例。这个单一实例会被存储到单例缓存（singleton cache）中，并且所有针对该 bean 的后续请求和引用都将返回被缓存的对象实例。创建实例时进行内存申请，销毁时完成垃圾回收。

- prototype

  当把一个 bean 定义设置为 prototype 作用域时,每一次请求都会产生一个新的 bean 实例，Spring 不能对一个 prototype bean 的整个生命周期负责，销毁 prototype 作用域的对象并释放所持有的资源，都不属于 Spring 所管，而是由客户端代码执行。创建和销毁代价比较大。

- request

  当把一个 bean 定义设置为 prototype 作用域时，针对每一次 HTTP 请求都会产生一个新的 bean，同时该 bean 仅在当前 HTTP request 内有效。

3. 简述基于注解的装配方式的基本用法。

声明 Bean 的注解：主要有以下四种，表示注解的是一个组件对象（Bean），四种注解功能相同，仅有语义区别。

- @Component，通用注解
- @Repository，注解数据访问层 Bean
- @Service，注解业务逻辑层 Bean
- @Controller，注解控制器层 Bean

辅助用的注解: @Scope 用于指定作用域 @Value 设置值

用注解声明的 Bean，可以通过组件扫描让 Spring Ioc 容器发现

注入 Bean 用的注解：主要有@Autowired， @Resource，@Qualifier

- @Autowired：可以对类成员变量、方法及构造函数进行注解，完成自动装配的工作，可以消除 setter 和 getter 方法，默认按照 Bean 的类型进行装配
- @Resource: 功能和@Autowired 一样，区别在于默认按照名称来装配
- @Qualifier：与@Autowired 配合使用，当@Autowired 注解需要按照名称来装配时需要和该注解一起使用，Bean 实例名称由@Qualifier 注解参数指定

4. 假如使用前后端分离架构设计个人通讯录管理系统，请设计一下 RESTFUL 风格的对通讯录进行 CRUD 操作的接口 API

SELECT * FROM ITEM;

ID	ADDRESS	EMAIL	NAME	PHONE	QQ
2	安徽省淮南市	1418226938@qq.com	李志毅	13053179645	1418226938

(1 row, 6 ms)

- **查询全部通讯录列表**

URL:'/listall'
METHOD: GET
REQUEST_ARGS: null
RESPONSE: JSON、{id, address, email, name, phone, qq}

- **查询特定通讯录用户项**

URL: '/listbyid'
METHOD: GET
REQUEST_ARGS: id -> 需要查询的元组 id
RESPONSE: JSON、{id, address, email, name, phone, qq}

- **查询特定通讯录用户项**

URL: '/listbyname'
METHOD: GET
REQUEST_ARGS: name -> 需要查询的用户名称
RESPONSE: JSON、{id, address, email, name, phone, qq}

- **修改特定通讯录用户项**

URL:'/updateone'
METHOD: POST
REQUEST_ARGS:
    id -> 需要修改的元组 id
    name -> 修改的用户名称
    address -> 修改后的地址
    email -> 修改后的邮箱
    phone -> 修改后的手机
    qq -> 修改后的 qq
RESPONSE: statusCode-200

- **删除特定通讯录用户项**

URL:'/delone'
METHOD: POST
REQUEST_ARGS:
    id -> 需要删除的元组 id
RESPONSE: statusCode-200

• **增加特定通讯录用户项**

URL:'/addone'

METHOD: POST

REQUEST_ARGS:

id -> 增加的元组 id

name -> 增加的用户名称

address -> 增加的地址

email -> 增加的邮箱

phone -> 增加的手机

qq -> 增加的 qq

RESPONSE: statusCode-200

个人通讯录管理系统 part3

1. 在添加联系人页面增加一处 AJAX 操作：用户输完电话号码后使用 AJAX 发起请求判断该号码是否已存在并予以提示

增加页面，当电话输入框变化时，即向”/ajax/checkphone”发送 POST 请求，携带参数为当前 phone 输入框值，后台在所有列表元素中查询，若存在电话相同，则返回”success”代表有号码已存在，此时将 id=”phoneAlert”的标签文本值设为“号码已存在”，否则为空不显示。

修改页面同理，只是电话输入框 id 变为”_phone”，存在标志标签 id 为”_phoneAlert”

Ajax 代码：

```
$(document).ready(
    //当增加页面 电话输入框变化时 发送ajax请求看是否电话已存在
    $("#phone").bind("input propertychange",function() {
        var phone = $("#phone").val();
        $.ajax({
            url: "/ajax/checkphone",
            type: "POST",
            data: {
                "phone":phone
            },
            success: function (result) {
                console.log(result);
                if(result === "success"){
                    $("#phoneAlert").text("号码已存在")
                }
                else{
                    $("#phoneAlert").text("")
                }

            },
            error: function (jqXHR, textStatus, errorThrown) {
                $("#phoneAlert").text("")
                alert('错误！');
            }
        })
    }),
```

index.html

```
<div class="input_div">
    <p>电话：</p>
    <input type="number" placeholder="11位电话号码" id="phone" name="phone" autocomplete="off"/>
    <text id="phoneAlert" style="color: #e20000"></text>
</div>

<div class="input_div">
    <p>电话：</p>
    <input type="number" placeholder="11位电话号码" id="_phone" name="phone" autocomplete="off"/>
    <text id="_phoneAlert" style="color: #e20000"></text>
```

IndexController

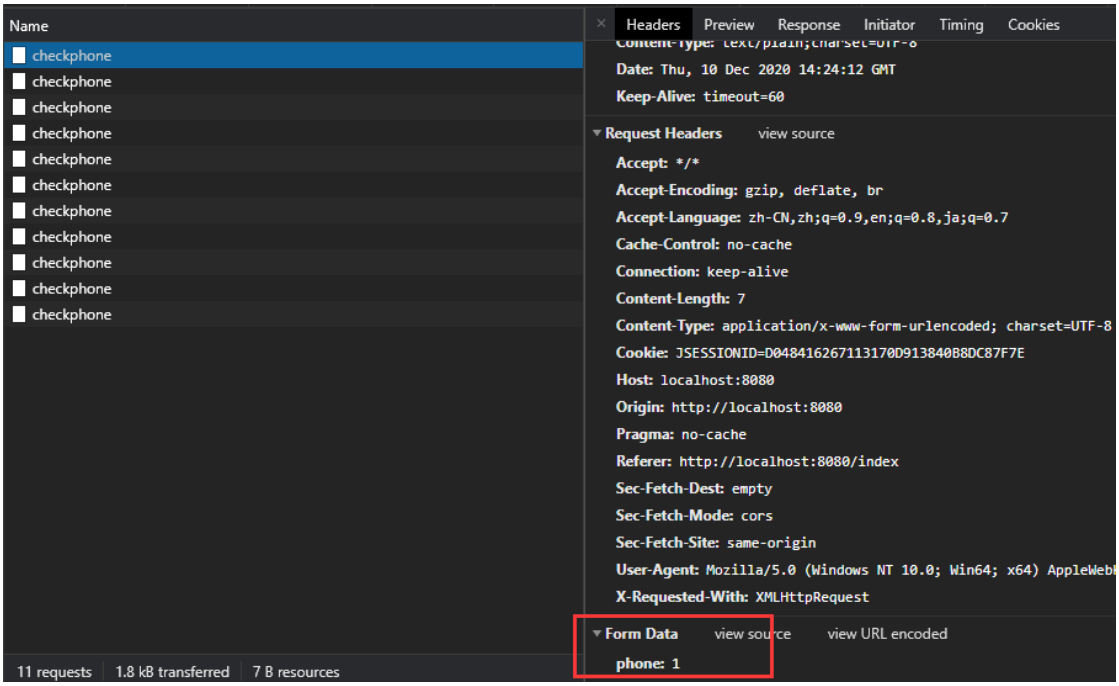
```
@ResponseBody
@PostMapping("/ajax/checkphone")//通过ajax检查当前电话是否存在
public String checkphone(@RequestParam("phone") String phone){
    System.out.println(phone);
    String result="";
    for (item each:items
         ) {
        System.out.println(each.getPhone());
        if(each.getPhone().equals(phone)){
            System.out.println("yes");//若电话已存在 则返回success
            result="success";
        }
    }
    return result;//否则返回""
}
```

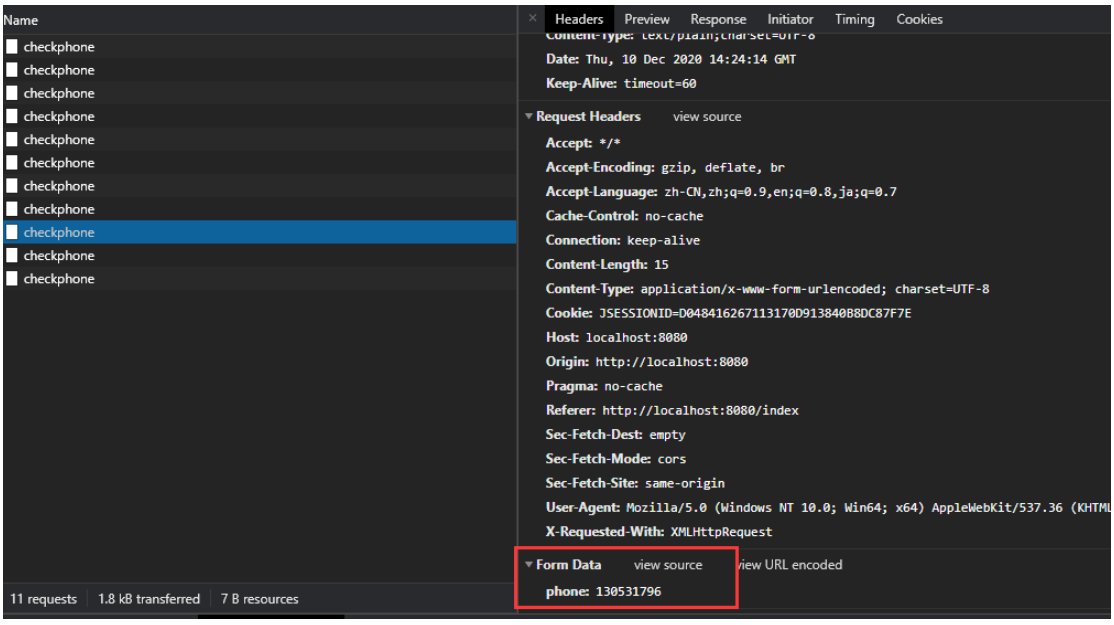
效果展示：

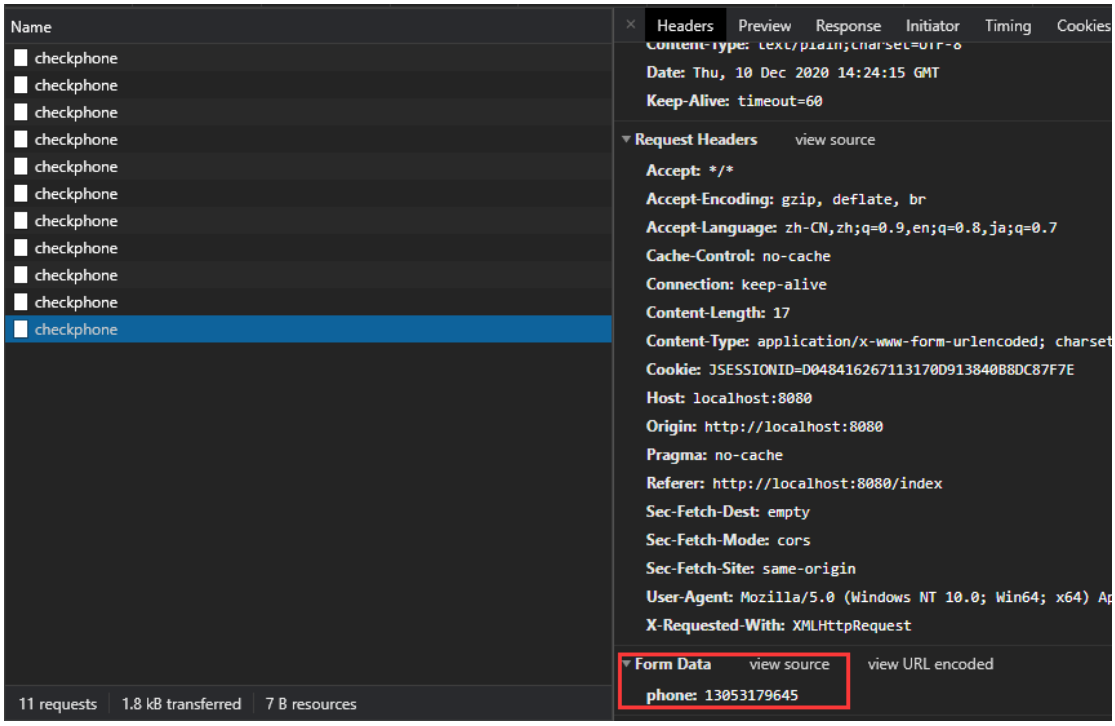
首先添加一个电话为 13053179645 的人员信息：

接着再添加一个 13053179645：

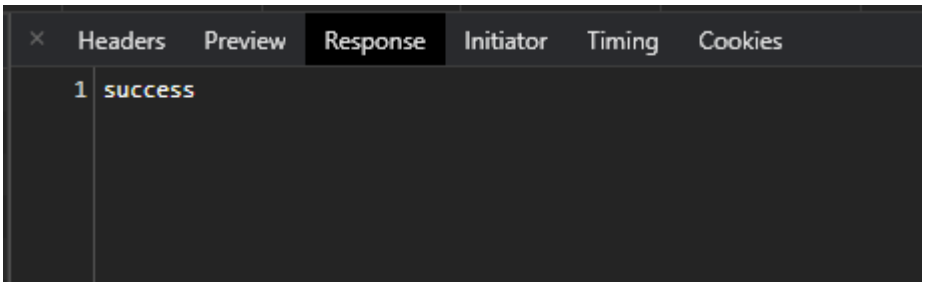
观察到在正好输入到 13053179645 时出现：







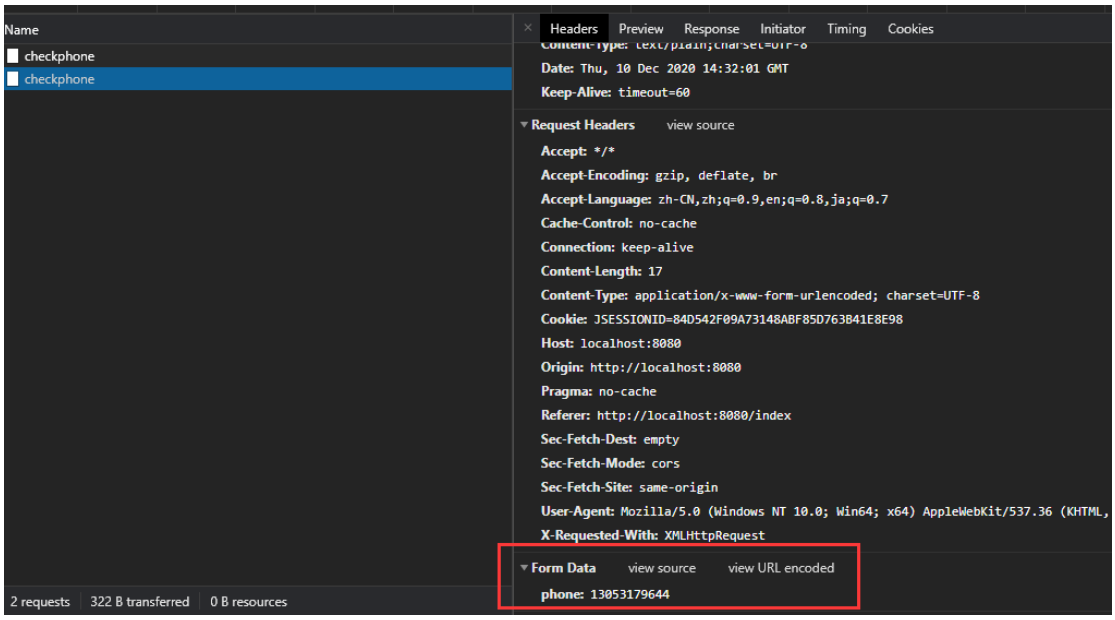
此时返回

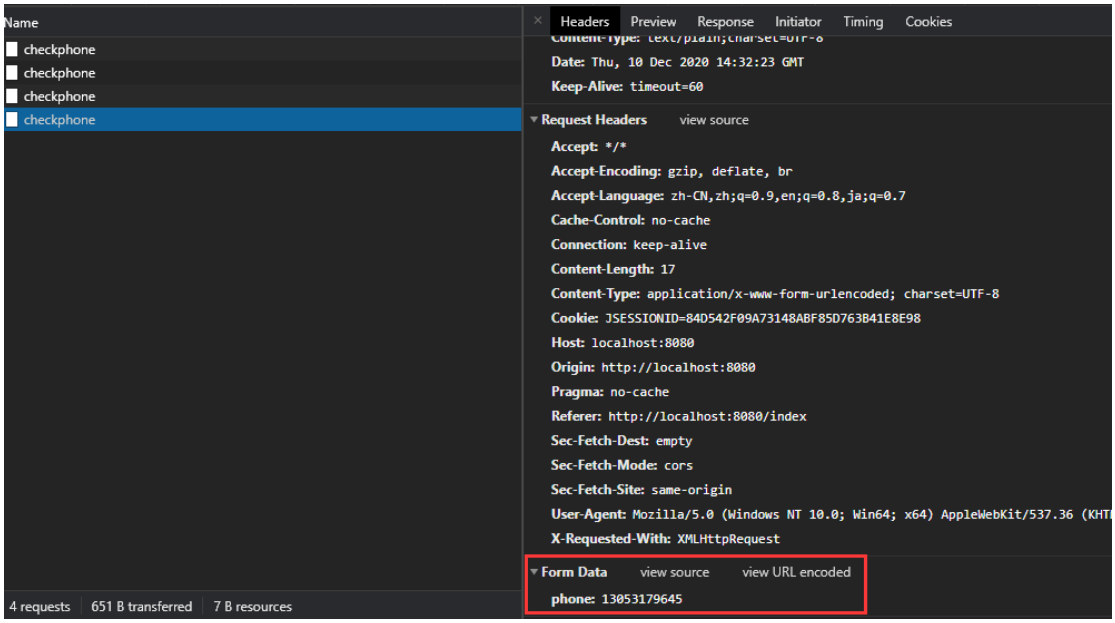


此时提示号码已存在：

修改页面同理：


住址
QQ号
修改
姓名：
李志毅
电话：
13053179644
邮箱：
1418226938@qq.com
住址：
北京市海淀区
QQ：
12345
保存
关闭


Name
checkphone
checkphone
2 requests
322 B transferred
0 B resources
Headers
Preview
Response
Initiator
Timing
Cookies
Date: Thu, 10 Dec 2020 14:32:01 GMT
Keep-Alive: timeout=60
Request Headers
view source
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,ja;q=0.7
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 17
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Cookie: JSESSIONID=84D542F09A73148ABF85D763B41E8E98
Host: localhost:8080
Origin: http://localhost:8080
Pragma: no-cache
Referer: http://localhost:8080/index
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
X-Requested-With: XMLHttpRequest
Form Data
view source
view URL encoded
phone: 13053179644


Name
checkphone
checkphone
checkphone
checkphone
4 requests
651 B transferred
7 B resources
Headers
Preview
Response
Initiator
Timing
Cookies
Date: Thu, 10 Dec 2020 14:32:23 GMT
Keep-Alive: timeout=60
Request Headers
view source
Accept: */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,ja;q=0.7
Cache-Control: no-cache
Connection: keep-alive
Content-Length: 17
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Cookie: JSESSIONID=84D542F09A73148ABF85D763B41E8E98
Host: localhost:8080
Origin: http://localhost:8080
Pragma: no-cache
Referer: http://localhost:8080/index
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
X-Requested-With: XMLHttpRequest
Form Data
view source
view URL encoded
phone: 13053179645


住址
QQ号
姓名:
李志毅
电话:
13053179645
号码已存在
邮箱:
1418226938@qq.com
住址:
北京市海淀区
QQ:
12345
保存
关闭

2.将通讯录的增删改查操作使用 JPA 技术予以实现。

设置数据库账户 sa，密码 password，同时设置在每次启动时清除数据库

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:~/test
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
```

实体 item 类代表一个用户，主键为 id

```
@Entity
public class item {
    private Long id;

    private String name;//名称
    private String phone;
    private String email;//邮箱
    private String address;
    private String qq;

    @Id
    @GeneratedValue//主键
    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }
```

增加一个 item 时，向数据库写入

```
//向index post数据，在通讯录list中添加post来的数据
@PostMapping("/index")
public String list(@RequestParam("name") String name,
                   @RequestParam("phone") String phone,
                   @RequestParam("email") String eamil,
                   @RequestParam("address") String address,
                   @RequestParam("qq") String qq,
                    HttpServletResponse response
                    ,Model model){
   System.out.println("成功接收POST"+name+phone+eamil);

    item it=new item();//创建item
    it.setId((long)count);
    count++;//id自增
    it.setName(name);
    it.setPhone(phone);
    it.setEmail(eamil);
    it.setAddress(address);
    it.setQq(qq);
    ItemRepository.save(it);//放入数据库
    items.add(it);
    model.addAttribute( s: "items",items);
    return "index";
}
```

修改时，先由主键获得元素，再根据参数更新数据库

```
//用于修改当前以index为索引的list中的元素值(修改个人信息)
@PostMapping("/changeItem")
public String changeItem(@RequestParam("id") int index,
                         @RequestParam("name") String name,
                         @RequestParam("phone") String phone,
                         @RequestParam("email") String eamil,
                         @RequestParam("address") String address,
                         @RequestParam("qq") String qq,Model model){
    System.out.println(index);
    long id = items.get(index).getId();//将List中的索引对应到数据库中元素的索引(index相当于下标，id才是真正的标识)
    Optional<item> itemOptional= ItemRepository.findById(id);
    if(itemOptional.isPresent()){
        item it=itemOptional.get();//修改主键为id的元组
        it.setName(name);
        it.setPhone(phone);
        it.setEmail(eamil);
        it.setAddress(address);
        it.setQq(qq);
        items.set(index,it);//在items中修改
        ItemRepository.save(it);//在数据库中保存
    }


    return "redirect:index";
}
```

在数据库中删除元素

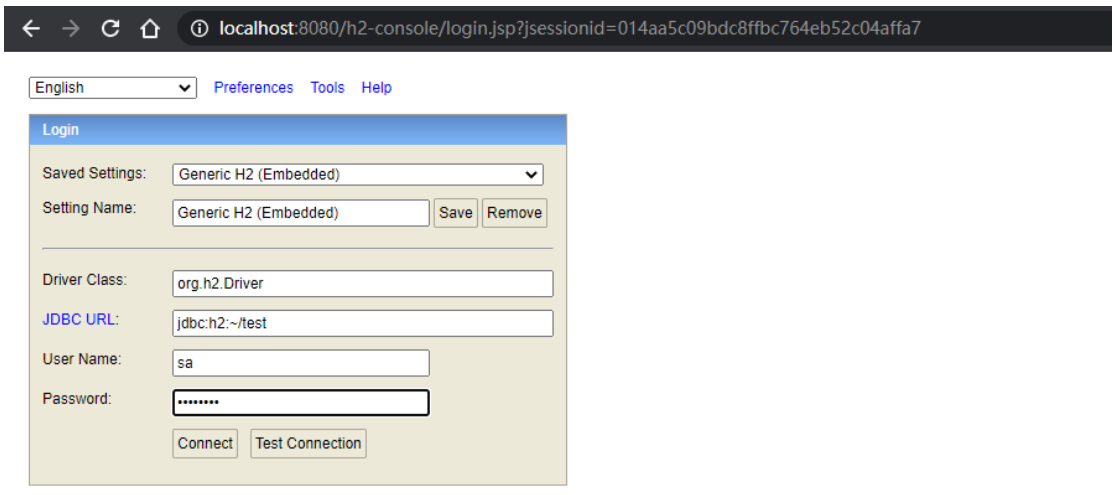
```
@GetMapping("/deleteItem")
public String DeleteItem(@RequestParam("itemId") int index){
    System.out.println(index);
    long id=items.get(index).getId();
    items.remove(index);

    ItemRepository.deleteById(id);

    return "redirect:index";
}
```

效果展示：

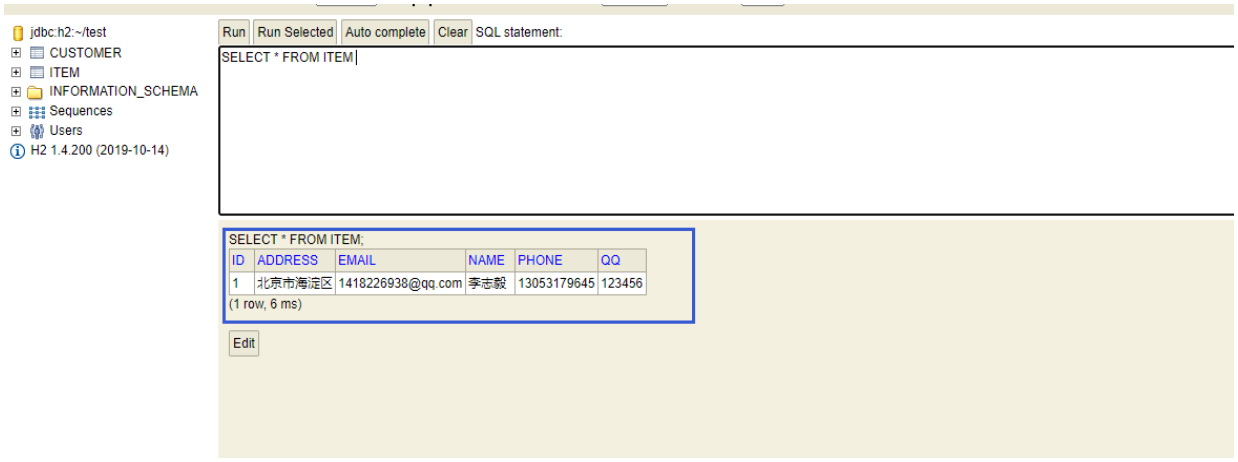
数据库登录：



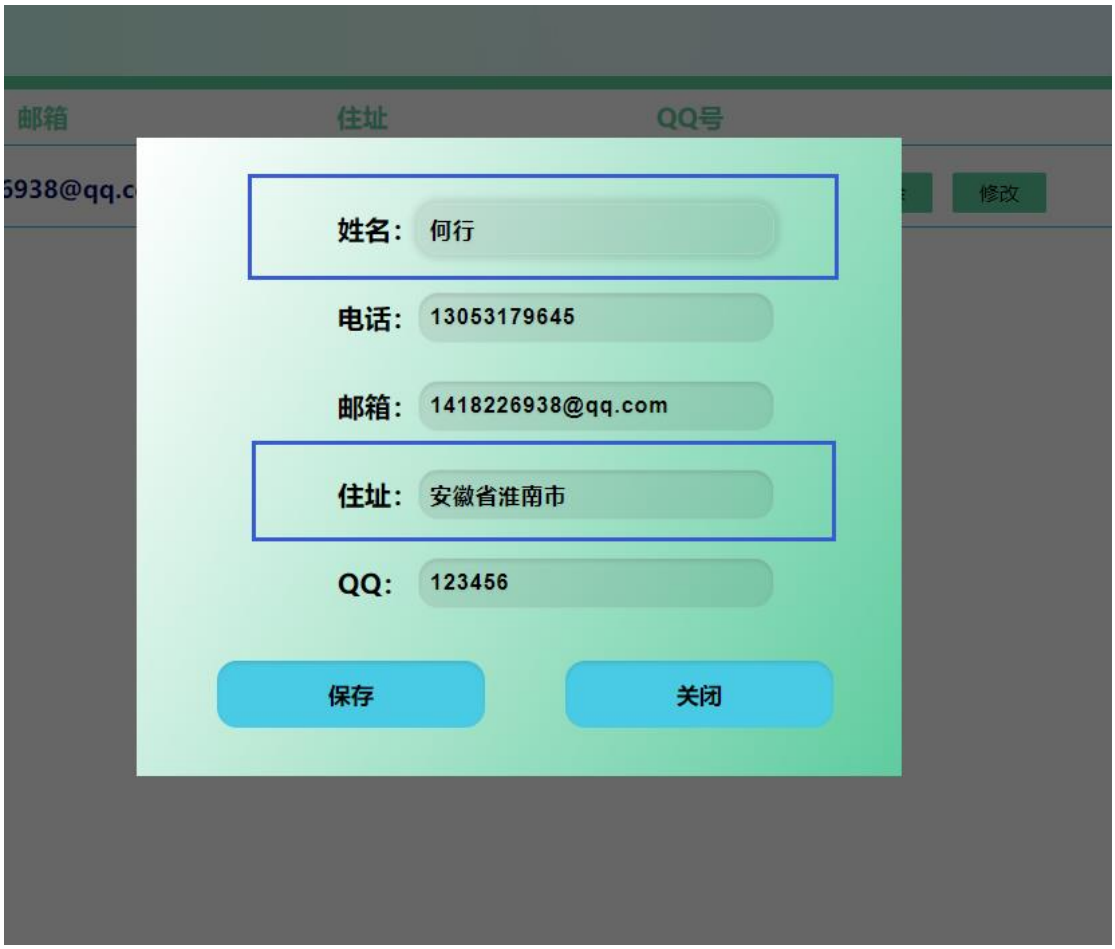
增加：

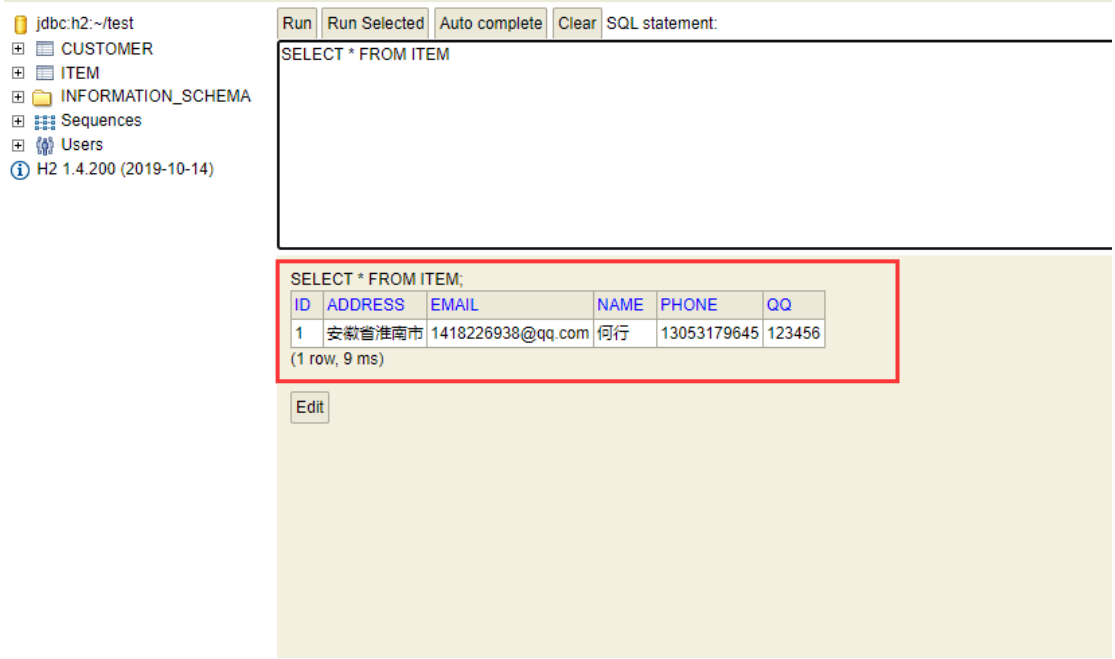

邮箱
住址
QQ号
姓名：
李志毅
电话：
13053179645
邮箱：
1418226938@qq.com
住址：
北京市海淀区
QQ：
123456
添加
关闭


在线通讯录
姓名
电话
邮箱
住址
QQ号
李志毅
13053179645
1418226938@qq.com
北京市海淀区
123456
删除
修改


jdbc:h2:~/test
CUSTOMER
ITEM
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM ITEM
SELECT * FROM ITEM;
ID
ADDRESS
EMAIL
NAME
PHONE
QQ
1
北京市海淀区
1418226938@qq.com
李志毅
13053179645
123456
(1 row, 6 ms)
Edit

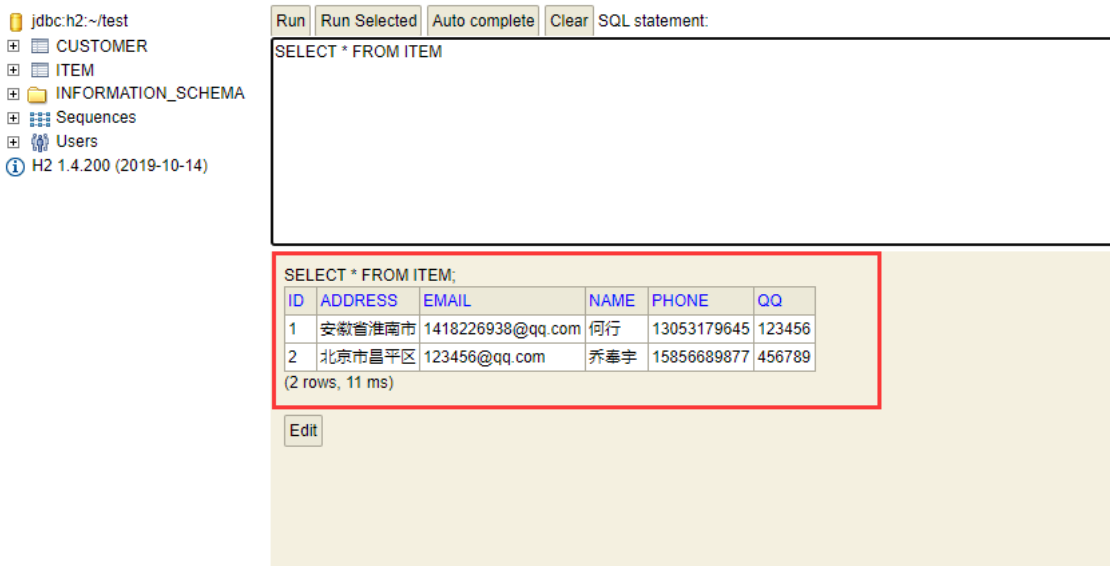
修改：


邮箱
住址
QQ号
修改
姓名：何行
电话：13053179645
邮箱：1418226938@qq.com
住址：安徽省淮南市
QQ：123456
保存
关闭


jdbc:h2:~/test
CUSTOMER
ITEM
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM ITEM
SELECT * FROM ITEM;
ID ADDRESS EMAIL NAME PHONE QQ
1 安徽省淮南市 1418226938@qq.com 何行 13053179645 123456
(1 row, 9 ms)
Edit

再增加一个后再删除第一个：

姓名	电话	邮箱	住址	QQ号	
何行	13053179645	1418226938@qq.com	安徽省淮南市	123456	删除 修改
乔奉宇	15856689877	123456@qq.com	北京市昌平区	456789	删除 修改



删除

删除后：

附录 github 地址：

https://github.com/hexing2333/AddressBook/tree/SpringData_version