

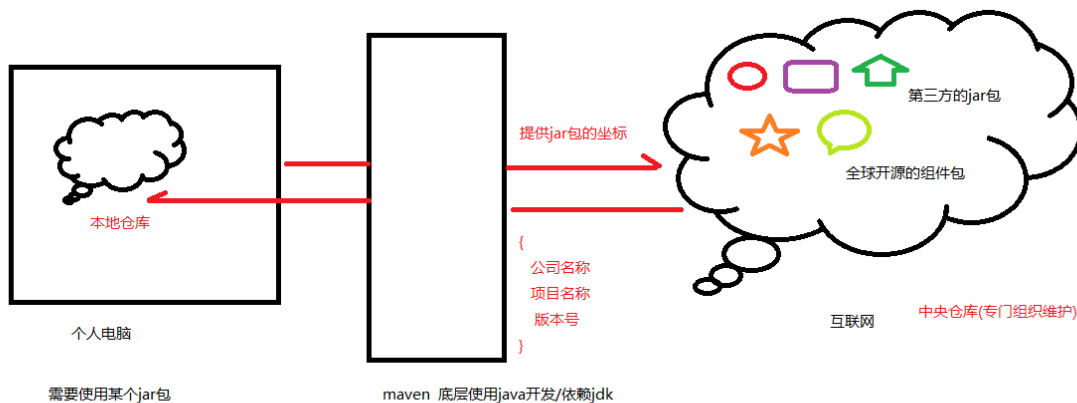
一、环境准备

JDK:建议至少是jdk17

IDEA:至少2021版 支持Spring Boot3.x

Visual Studio Code:傻瓜安装

二、Maven



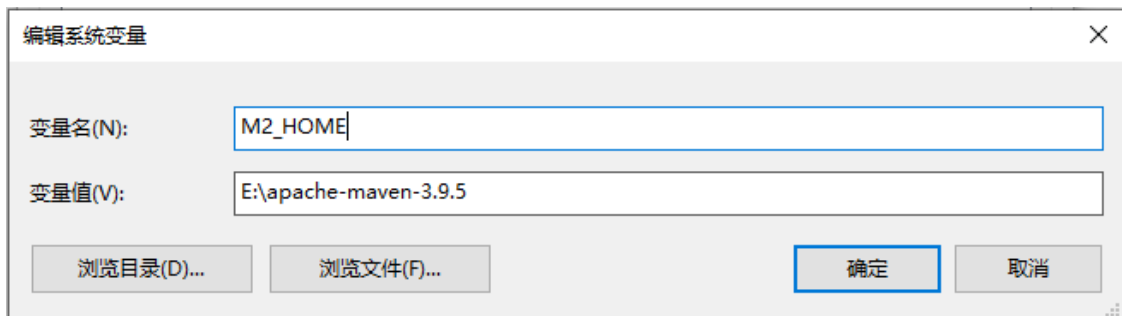
需要使用某个jar包

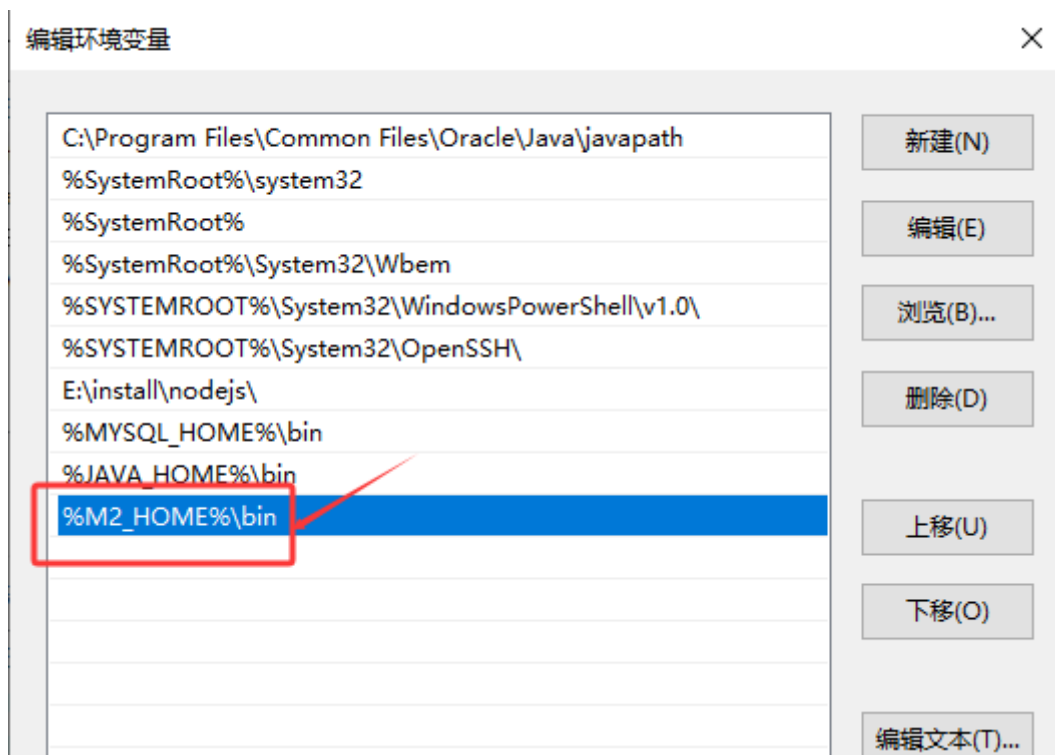
sqlserver.jar

- 1、如果本地没有，则通过maven从中央仓库中下载到本地仓库中，然后直接使用本地仓库
- 2、如果本地存在，则直接使用本地仓库中的jar包

安装maven:

1. 解压apache-maven-3.9.5-bin.zip
2. 配置环境变量





3. 验证maven

```
C:\Windows\system32\cmd.exe
Microsoft Windows [版本 10.0.19045.3570]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\Administrator>mvn -v
Apache Maven 3.9.5 (57804ffe001d7215b5e7bcb531cf83df38f93546)
Maven home: E:\apache-maven-3.9.5
Java version: 17.0.11, vendor: Oracle Corporation, runtime: E:\install\jdk17
Default locale: zh_CN, platform encoding: GBK
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Administrator>
```

4. 修改配置apache-maven-3.9.5\conf\settings.xml配置文件

本地仓库

```
<localRepository>E:\mvn_repo</localRepository>
```

根据实际情况，自己设计目录

```
<!-- localRepository
| The path to the local repository maven will use to store artifacts.
|
| Default: ${user.home}/.m2/repository
<localRepository>/path/to/local/repo</localRepository>
-->
<localRepository>E:\mvn_repo</localRepository>
<!-- interactiveMode
| This will determine whether maven prompts you when it needs input. If set to false,
| maven will use a sensible default value, perhaps based on some other setting, for
| the parameter in question.
```

中央仓库：使用阿里的镜像，提高下载速度。

```
<mirror>
  <id>alimaven</id>
  <mirrorOf>central</mirrorOf>
  <name>aliyun maven</name>
  <url>https://maven.aliyun.com/nexus/content/groups/public/</url>
</mirror>
```

```
<mirrors>
  <!-- mirror
  | Specifies a repository mirror site to use instead of a given repository. The repository that
  | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used
  | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
  |
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
-->
  <mirror>
    <id>alimaven</id>
    <mirrorOf>central</mirrorOf>
    <name>aliyun maven</name>
    <url>https://maven.aliyun.com/nexus/content/groups/public/</url>
  </mirror>
</mirrors>
```

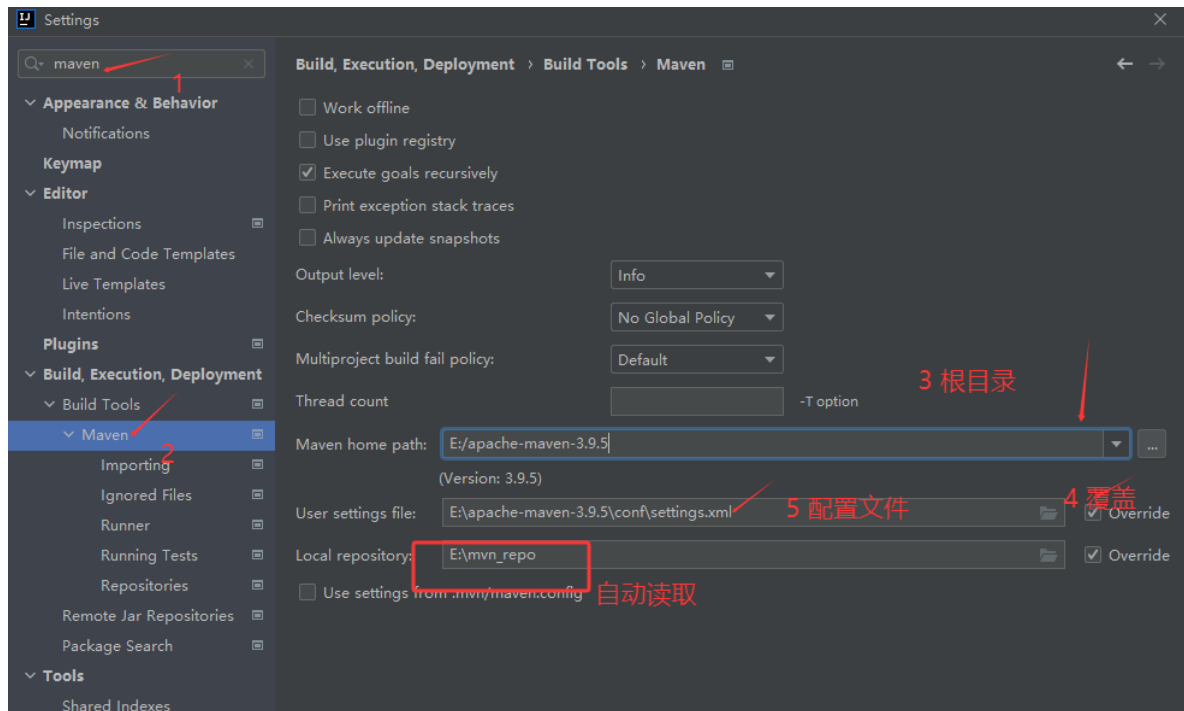
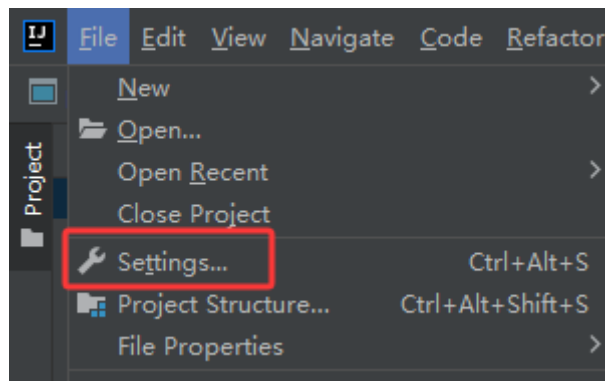
配置jdk的编译版本号:

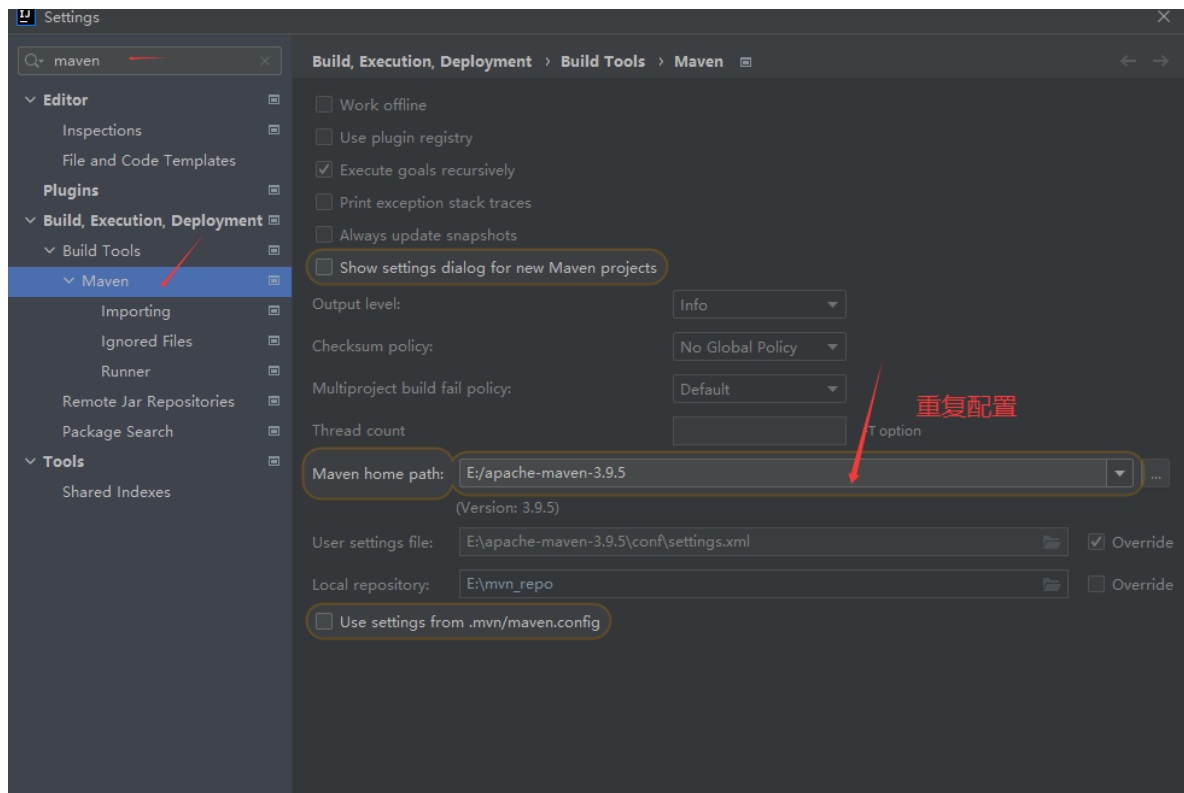
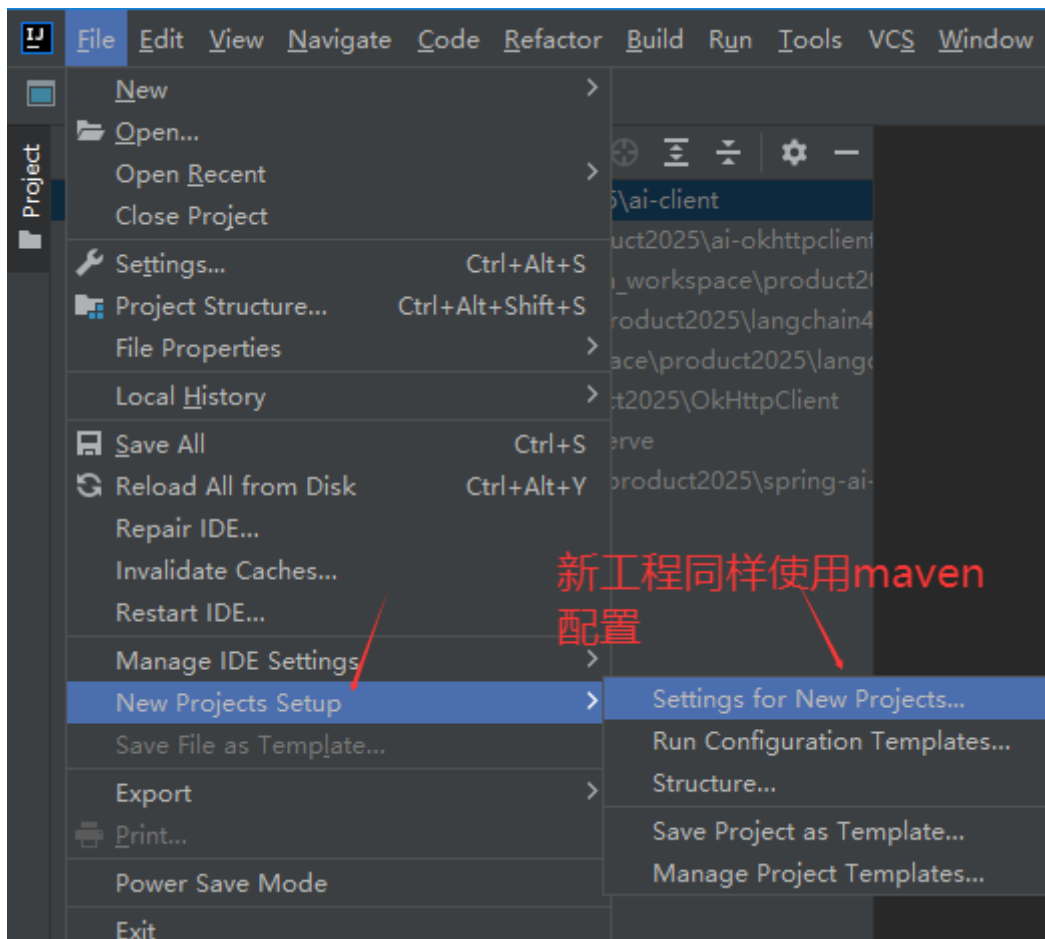
```
<profile>
  <id>jdk-17</id>
  <activation>
    <activeByDefault>true</activeByDefault>
    <jdk>17</jdk>
  </activation>
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <maven.compiler.compilerVersion>17</maven.compiler.compilerVersion>
  </properties>
</profile>
```

配置在profiles的子节点中

三、IDEA集成Maven

集成方案:



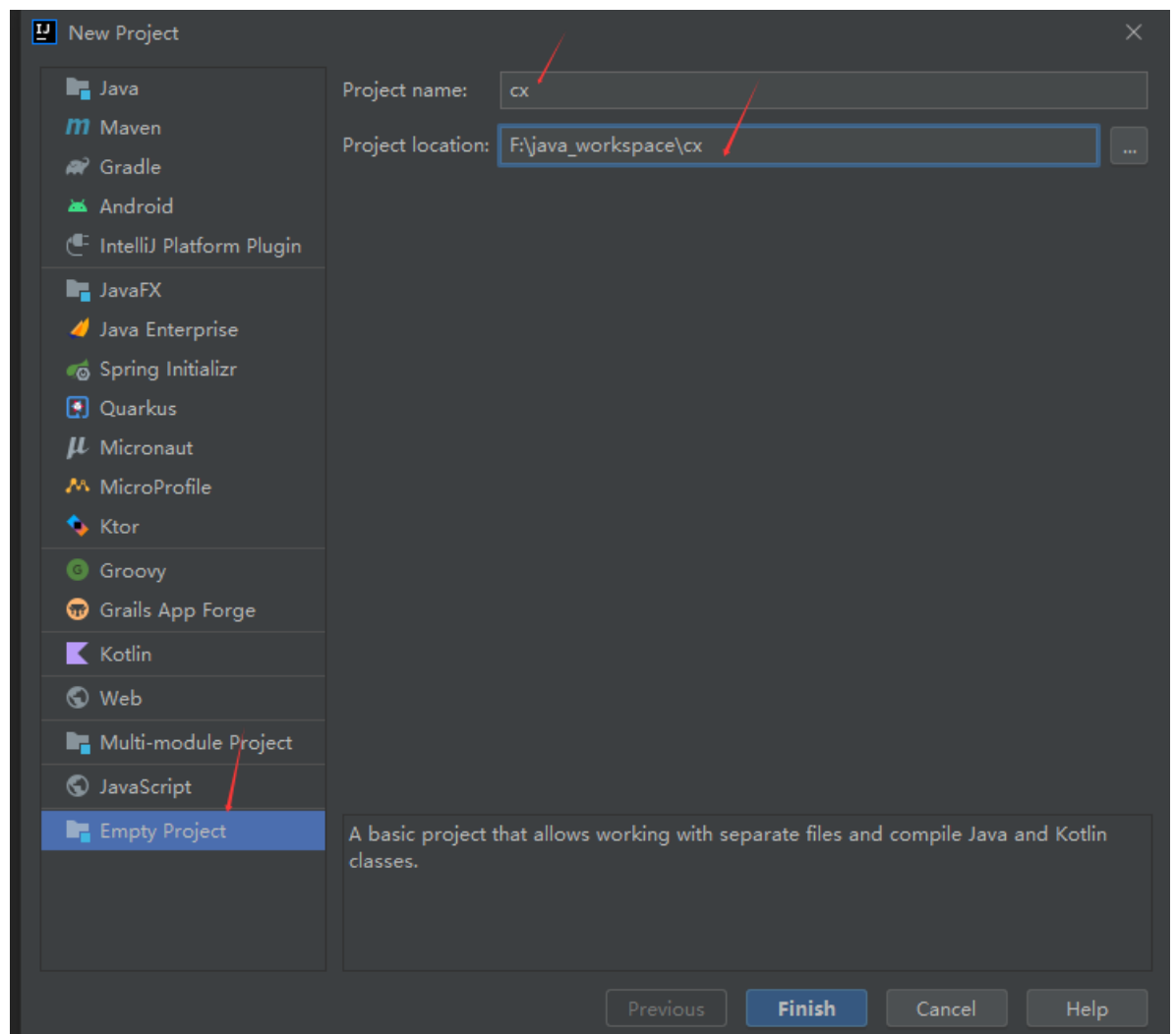
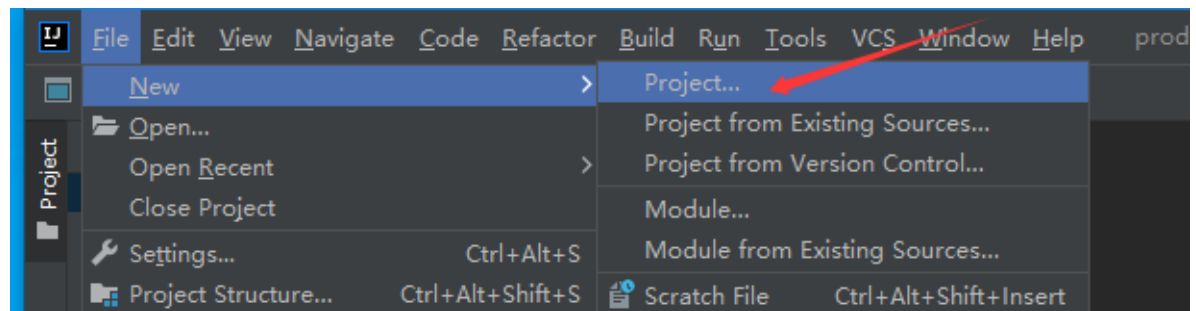


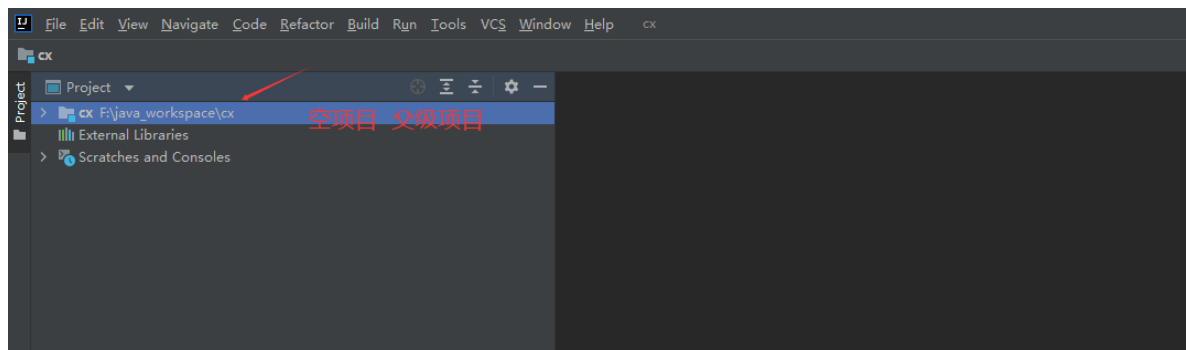
验证关注点：感受maven管理jar的强大之处

案例：创建一个maven工程，导入sqlserver的驱动包

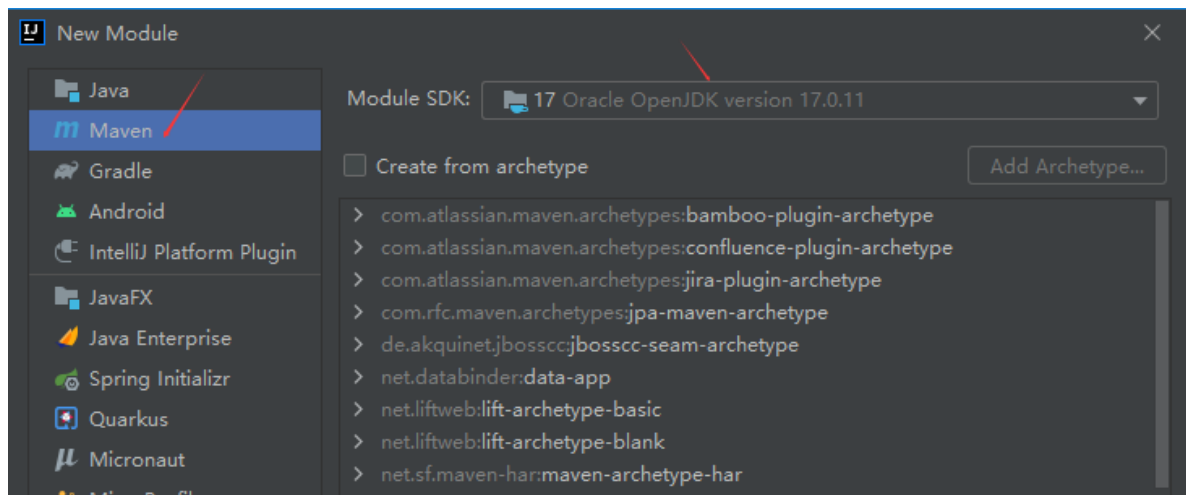
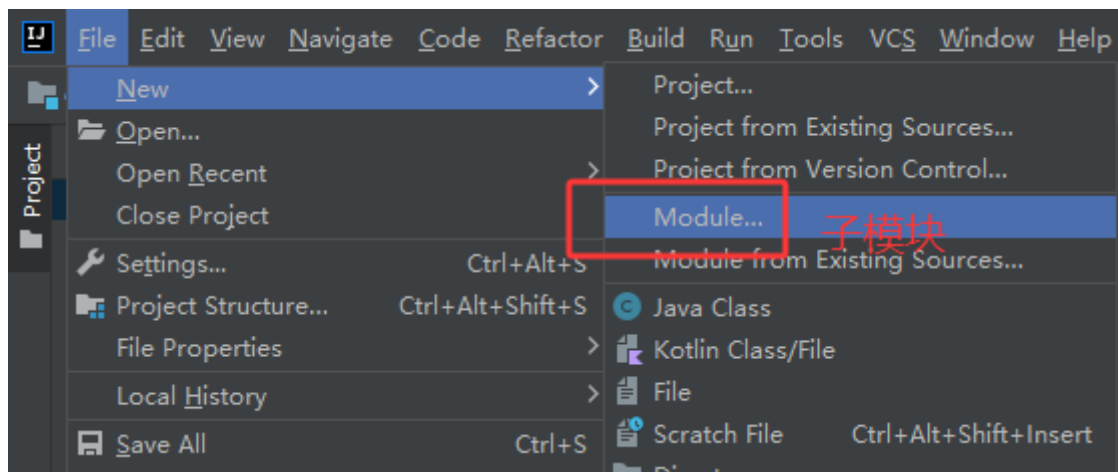
操作步骤

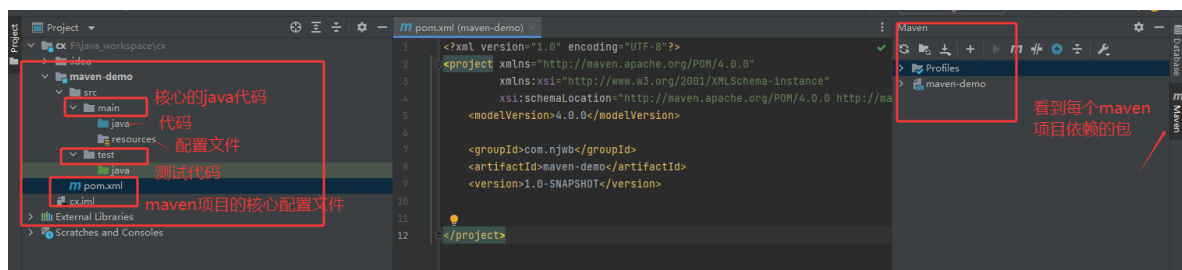
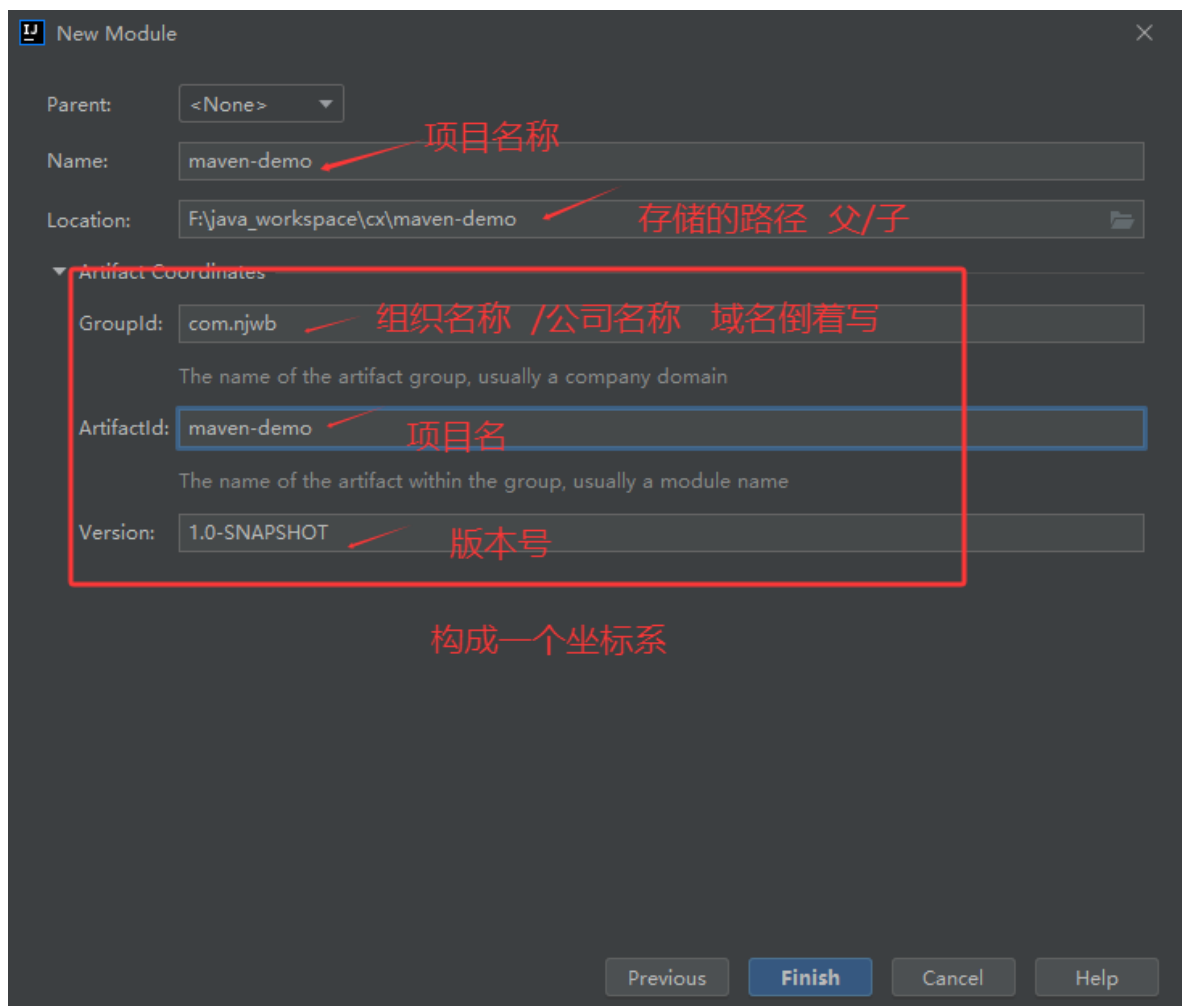
1、创建一个空项目：cx





2、创建子项目，是一个maven项目：maven-demo





3、导入sqlserver的驱动包

导入就是配置的过程-----》pom.xml文件中配置jar包的坐标信息


```
m pom.xml (maven-demo) x
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6
7   <groupId>com.njwb</groupId>
8   <artifactId>maven-demo</artifactId>
9   <version>1.0-SNAPSHOT</version>
10
11
12 </project>
```

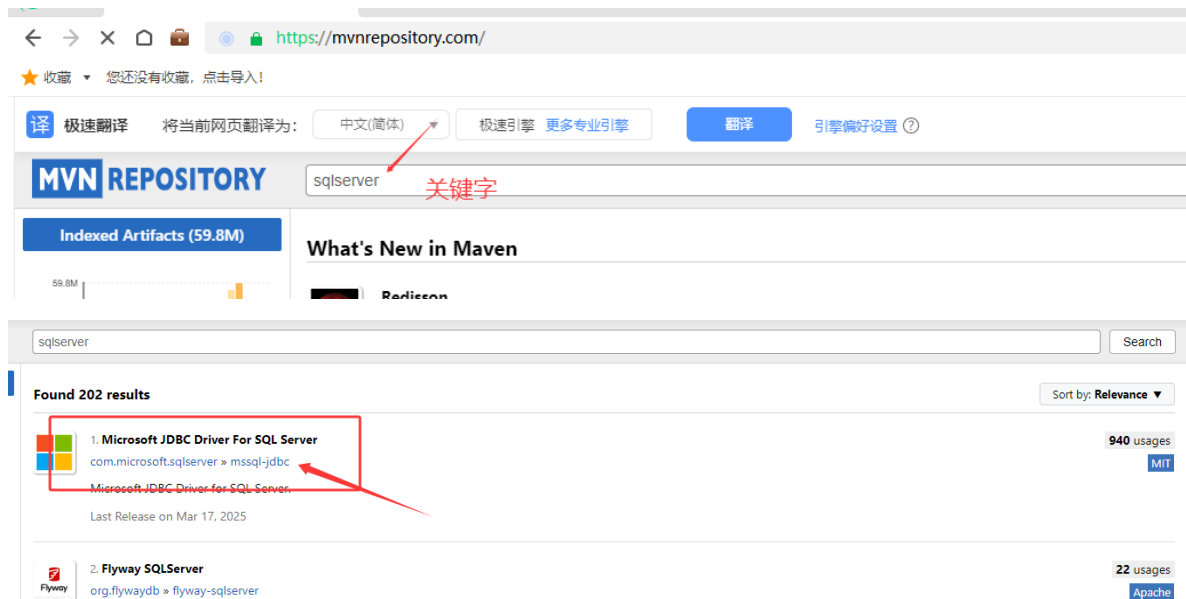
编写配置文件，导入相关的jar包

?? 怎么配置sqlserver

?? 得知道sqlserver包的坐标信息

查

<https://mvnrepository.com/>



Central (235)		Redhat GA (4)	ICM (3)		
Version ▼		Vulnerabilities	Repository	Usages	Date
12.10.x	12.10.0.jre11		Central	53	Mar 17, 2025
	12.10.0.jre8		Central	13	Mar 17, 2025
12.9.x	12.9.0.jre11-preview		Central	8	Dec 03, 2024
	12.9.0.jre8-preview		Central	11	Dec 03, 2024
12.8.x	12.8.1.jre11		Central	83	Aug 23, 2024
	12.8.1.jre8		Central	32	Aug 23, 2024
	12.8.0.jre11		Central	8	Jul 31, 2024
	12.8.0.jre8		Central	14	Jul 31, 2024
	12.7.1.jre11-preview		Central	2	Jul 09, 2024
12.7.x	12.7.1.jre8-preview		Central	6	Jul 09, 2024
	12.7.0		Central	0	Apr 08, 2024
	12.7.0.jre11-preview		Central	8	Apr 06, 2024
	12.7.0.jre8-preview		Central	6	Apr 05, 2024
12.6.x	12.6.4.jre11		Central	19	Aug 29, 2024
	12.6.4.jre8		Central	1	Aug 29, 2024
	12.6.3.jre11		Central	38	Jun 20, 2024
	12.6.3.jre8		Central	9	Jun 20, 2024
	12.6.2.jre11		Central	14	May 24, 2024
	12.6.2.jre8		Central	7	May 24, 2024
	12.6.1.jre11		Central	48	Feb 21, 2024

Maven
Gradle
SBT
Mill
Ivy
Grape
Leiningen
Buildr

Scope: Compile ▼

```

<!-- https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc -->
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>12.7.0</version>
  <scope>compile</scope>
</dependency>

```

坐标 ctr+c /ctr+v

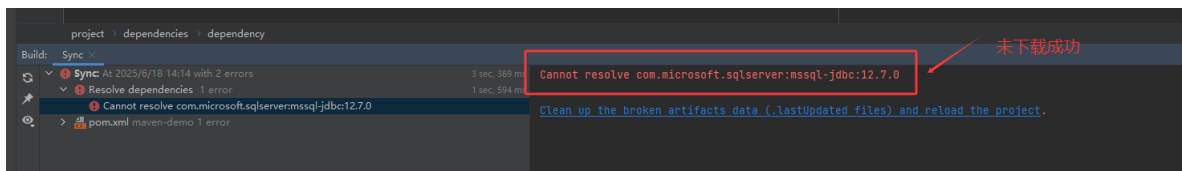
pom.xml (maven-demo)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4
5     <modelVersion>4.0.0</modelVersion>
6
7     <groupId>com.njwb</groupId>
8     <artifactId>maven-demo</artifactId>
9     <version>1.0-SNAPSHOT</version>
10
11     <dependencies>
12       <dependency>
13         <groupId>com.microsoft.sqlserver</groupId>
14         <artifactId>mssql-jdbc</artifactId>
15         <version>12.7.0</version>
16         <scope>compile</scope>
17       </dependency>
18     </dependencies>
19
20 </project>

```

更新配置 自动下载
C + V

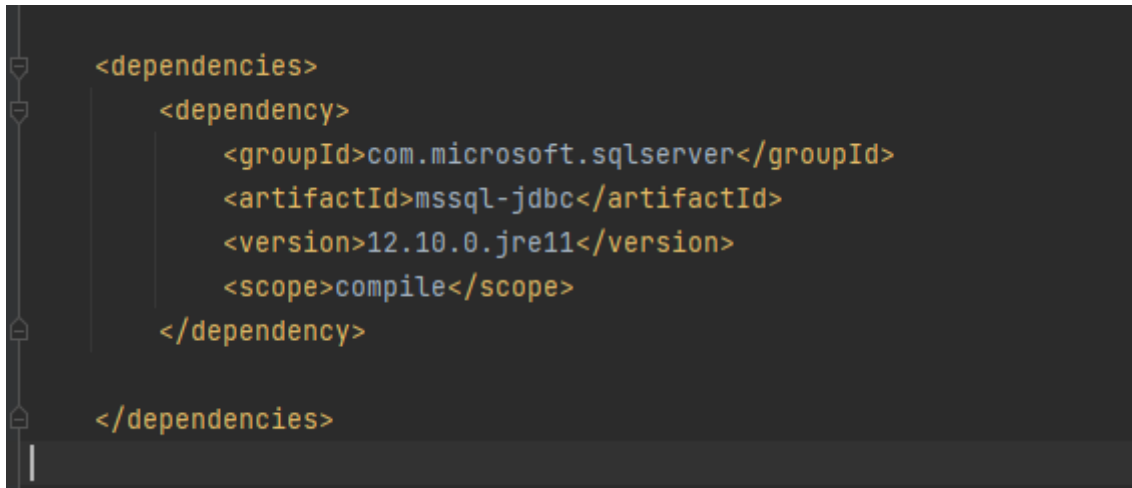


怎么办？

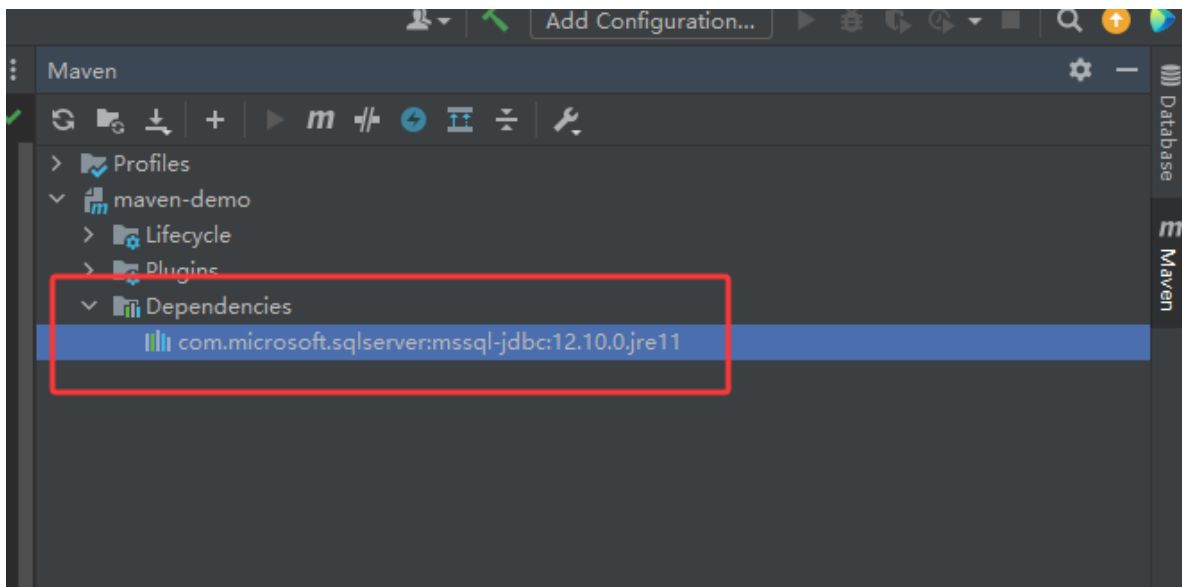
一般解决的办法有两种

a. 虽然有这样的坐标，但是中央仓库中的相关包可能被移除，所以下载不了

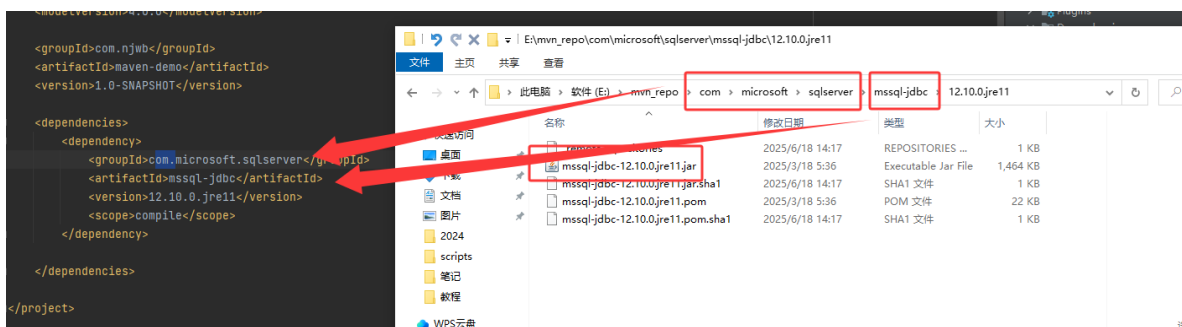
换版本号：12.10.0.jre11 【点击右上角的maven更新】



或者



或者去本地仓库检查是否存在该包



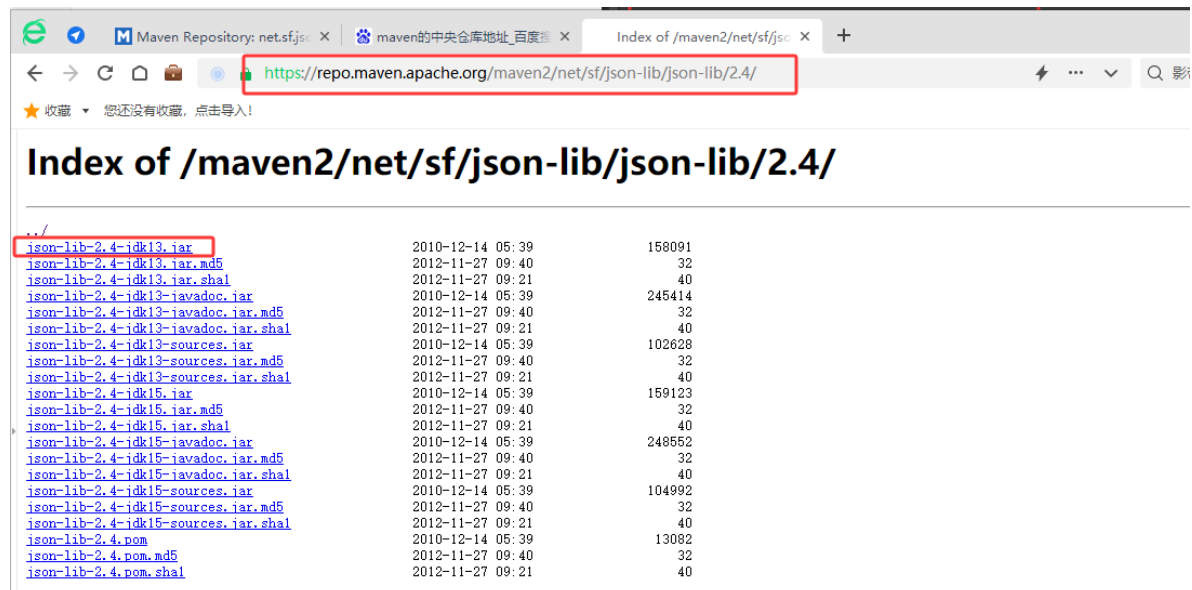
b.有坐标, 仓库中也有jar包, 但是仍然下载不下来

比如: jsonlib

```
<dependency>
  <groupId>net.sf.json-lib</groupId>
  <artifactId>json-lib</artifactId>
  <version>2.4</version>
</dependency>
```

到中央仓库中检查【小概率事件:需要携带jdk版本号】

<https://repo.maven.apache.org/maven2/>



Index of /maven2/net/sf/json-lib/json-lib/2.4/		
../		
json-lib-2.4-jdk13.jar	2010-12-14 05:39	158091
json-lib-2.4-jdk13.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk13.jar.sha1	2012-11-27 09:21	40
json-lib-2.4-jdk13-javadoc.jar	2010-12-14 05:39	245414
json-lib-2.4-jdk13-javadoc.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk13-javadoc.jar.sha1	2012-11-27 09:21	40
json-lib-2.4-jdk13-sources.jar	2010-12-14 05:39	102628
json-lib-2.4-jdk13-sources.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk13-sources.jar.sha1	2012-11-27 09:21	40
json-lib-2.4-jdk15.jar	2010-12-14 05:39	159123
json-lib-2.4-jdk15.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk15.jar.sha1	2012-11-27 09:21	40
json-lib-2.4-jdk15-javadoc.jar	2010-12-14 05:39	248552
json-lib-2.4-jdk15-javadoc.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk15-javadoc.jar.sha1	2012-11-27 09:21	40
json-lib-2.4-jdk15-sources.jar	2010-12-14 05:39	104992
json-lib-2.4-jdk15-sources.jar.md5	2012-11-27 09:40	32
json-lib-2.4-jdk15-sources.jar.sha1	2012-11-27 09:21	40
json-lib-2.4.pom	2010-12-14 05:39	13082
json-lib-2.4.pom.md5	2012-11-27 09:40	32
json-lib-2.4.pom.sha1	2012-11-27 09:21	40

```
<dependency>
  <groupId>net.sf.json-lib</groupId>
  <artifactId>json-lib</artifactId>
  <version>2.4</version>
  <classifier>jdk13</classifier>
</dependency>
</dependencies>
```

四、JUnit测试框架

1. 导入JUnit测试包

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.13.2</version>
  <scope>test</scope>
</dependency>
```

2. 定义测试类，就是一个普通类

```
package com.njwb.test;

public class TestUser {
}
```

3. 定义测试方法

要求：公共的 没有返回值的，且无参的方法

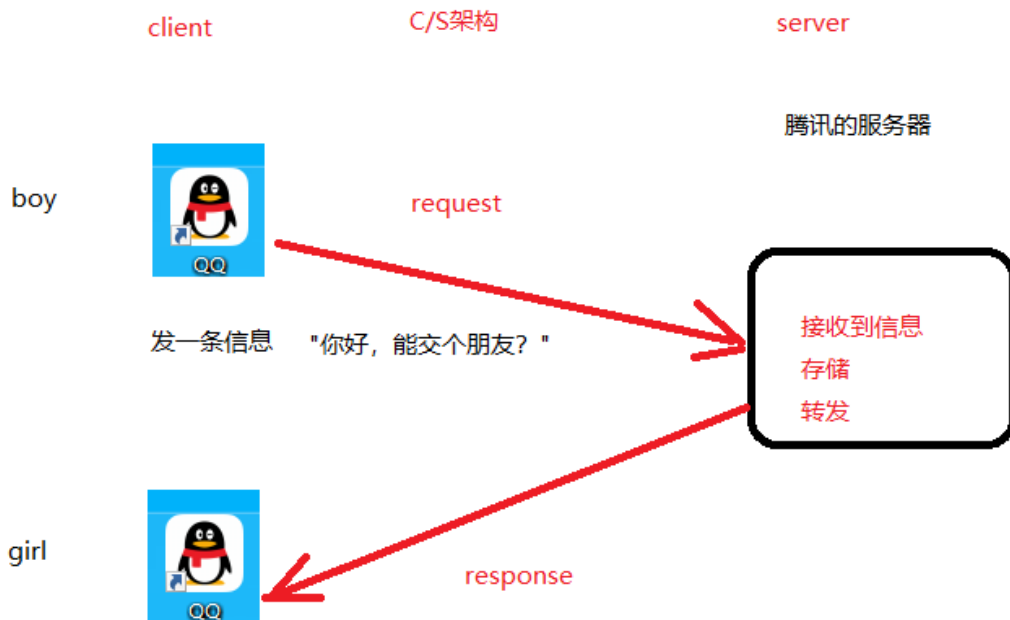
一个重要的注解：@Test

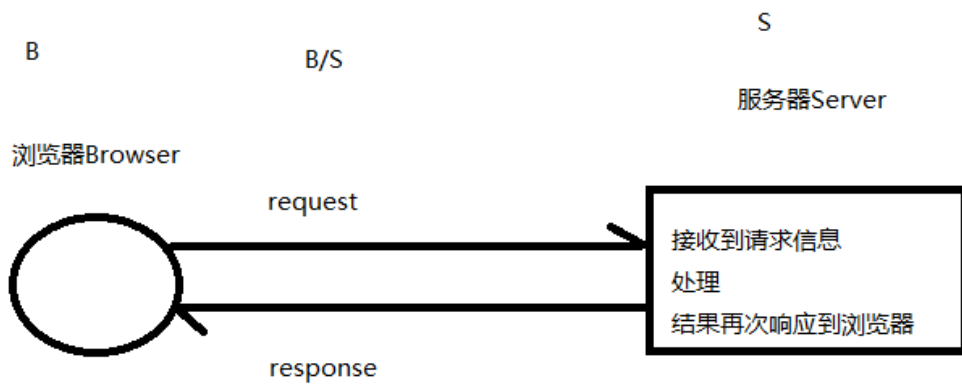
```
public class TestUser {
    @Test
    public void add(){
        System.out.println("add user success!");
    }

    @Test
    public void del(){
        System.out.println("del user success!");
    }
}
```

五、服务端基础

5.1 B/S架构





5.2 Tomcat服务器

1. 解压就是安装

脑 > 软件 (E:) > apache-tomcat-10.1.24 >

名称	^	修改日期	类型	大小
bin		2024/5/9 17:41	文件夹	
conf		2024/5/9 17:41	文件夹	
lib		2024/5/9 17:41	文件夹	
logs		2024/5/9 17:41	文件夹	
temp		2024/5/9 17:41	文件夹	
webapps		2024/5/9 17:41	文件夹	
work		2024/5/9 17:41	文件夹	
BUILDING.txt		2024/5/9 17:41	文本文档	22 KB
CONTRIBUTING.md		2024/5/9 17:41	Markdown File	7 KB
LICENSE		2024/5/9 17:41	文件	61 KB
NOTICE		2024/5/9 17:41	文件	3 KB
README.md		2024/5/9 17:41	Markdown File	4 KB
RELEASE-NOTES		2024/5/9 17:41	文件	7 KB
RUNNING.txt		2024/5/9 17:41	文本文档	17 KB

认识目录结构

bin: startup.bat(启动服务器) shutdown.bat (停止服务器)

conf:配置 server.xml(服务器的配置) web.xml (应用配置)

lib: tomcat自身依赖的jar包 (tomcat底层也是基于java实现)

logs:服务器启动过程中的日志信息

webapps:存放所有的web应用。

此电脑 > 软件 (E:) > apache-tomcat-10.1.24 > webapps

名称	修改日期	类型	大小
docs	2024/5/9 17:41	文件夹	
examples	2024/5/9 17:41	文件夹	
host-manager	2024/5/9 17:41	文件夹	
manager	2024/5/9 17:41	文件夹	
ROOT	2024/5/9 17:41	文件夹	

一个最基础的web应用需要哪些组件？

应用的名称

WEB-INF

web.xml

服务端的资源

图片、网页、音频....

2.手动发布一个应用，对外提供一个hello.html的资源信息

创建应用名称：cx

在cx下创建目录WEB-INF

在WEB-INF下定义web.xml

编辑web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
    https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
  version="6.0"
  metadata-complete="true">

  <welcome-file-list>
    <welcome-file>hello.html</welcome-file>
  </welcome-file-list>

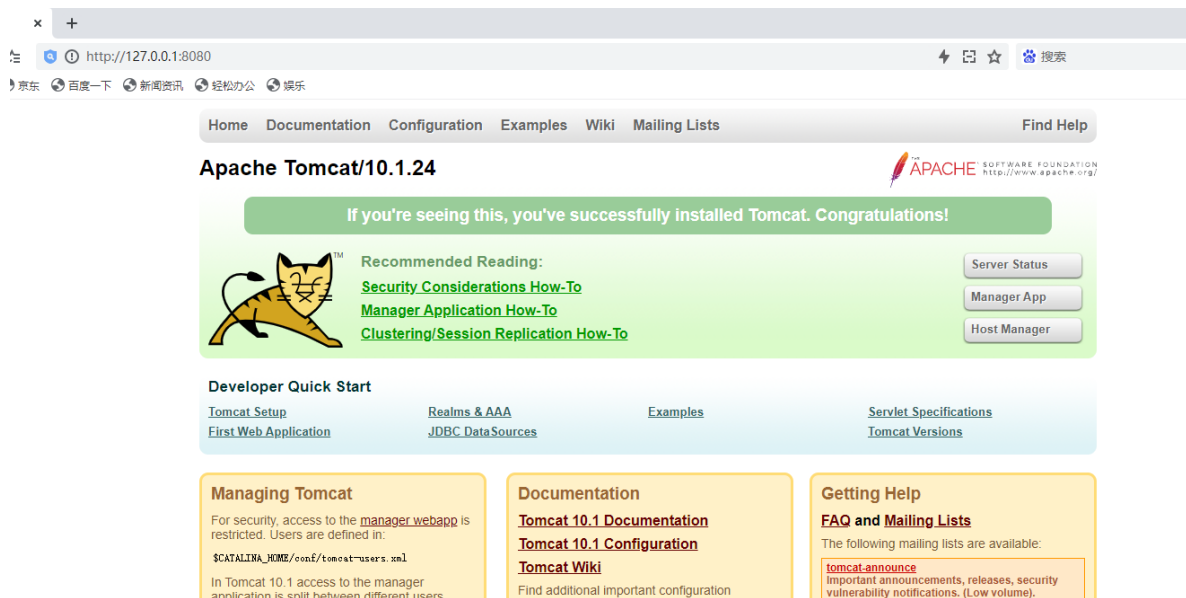
</web-app>
```


找不到则报404错误

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  你好，B/S！
</body>
</html>
```

执行startup.bat





目标资源是tomcat中的cx应用中的hello.html

<http://localhost:8080/cx/hello.html>

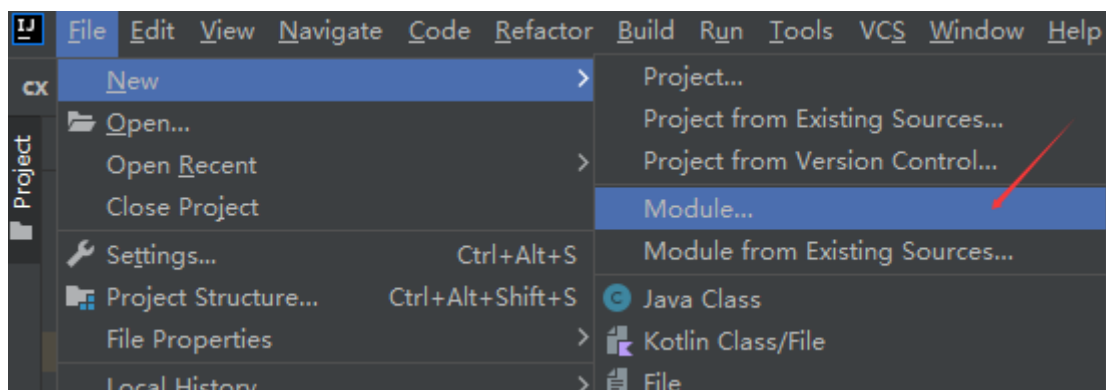
访问规则：

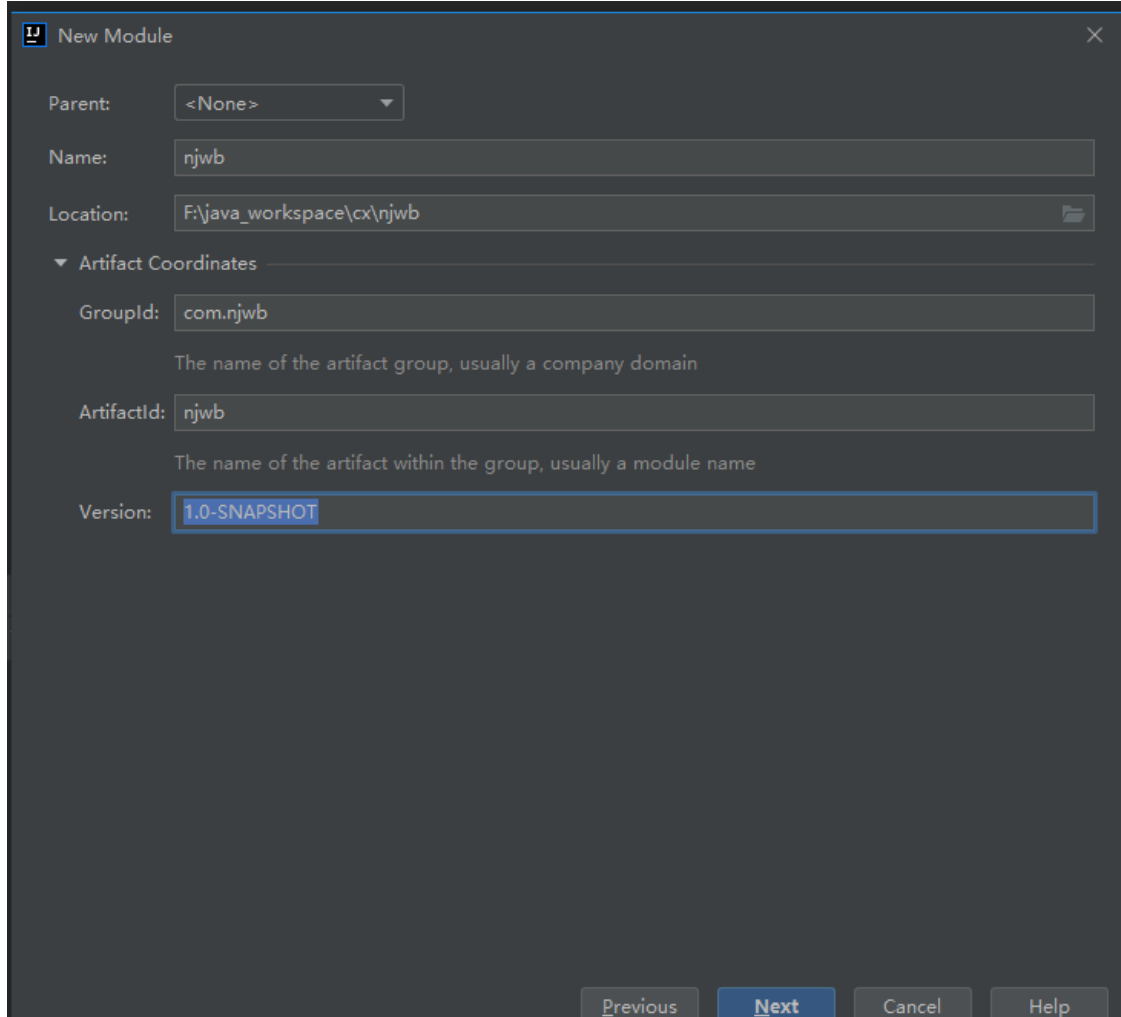
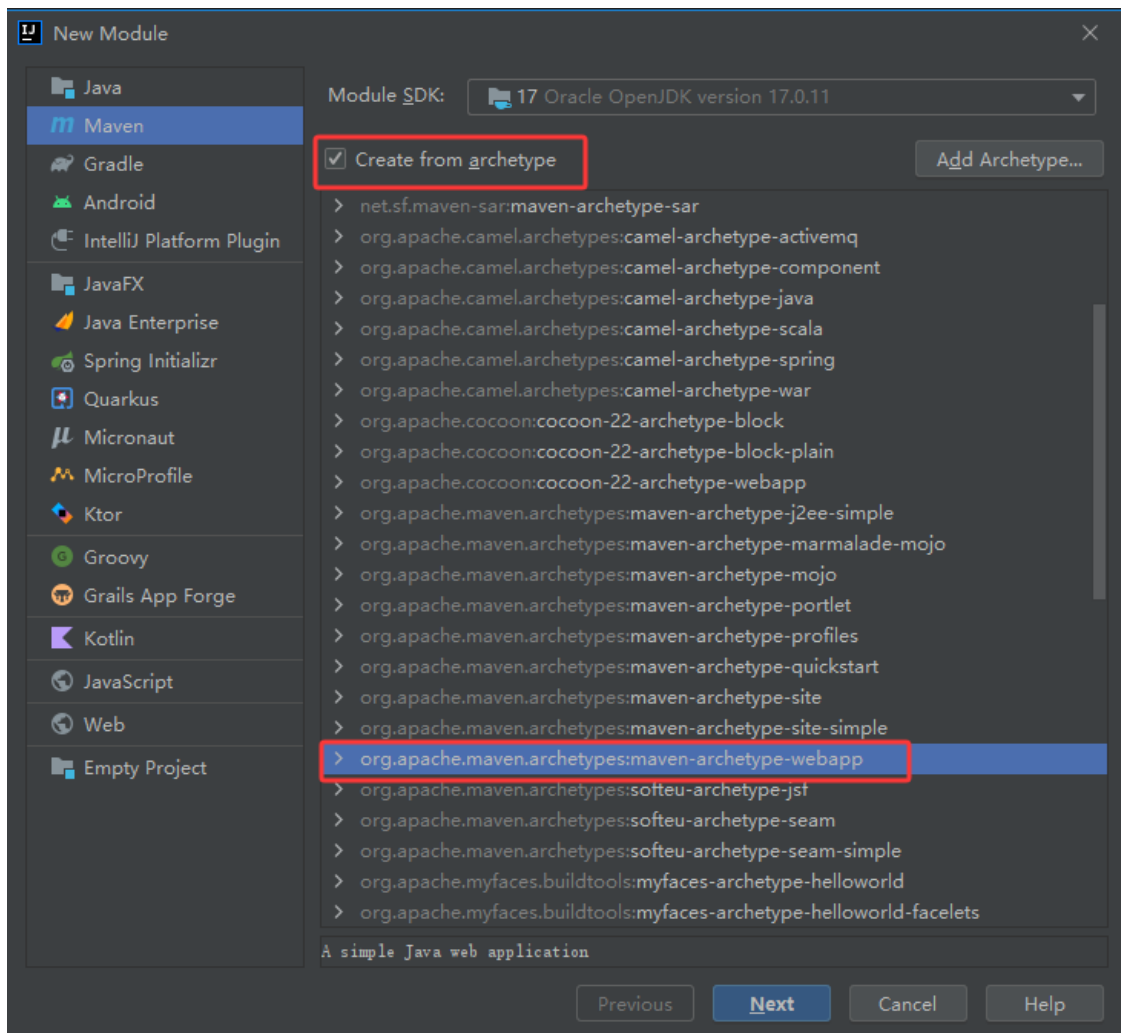
http://服务器的ip地址:端口号/应用的名称/应用下的资源的绝对路径

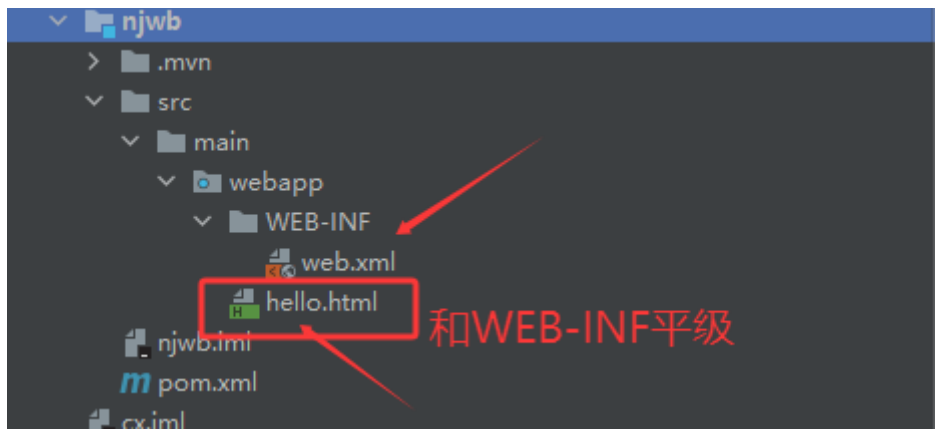
5.停止服务器 多按几次ctrl+c

5.3 IDEA创建应用与发布

1. 创建web应用



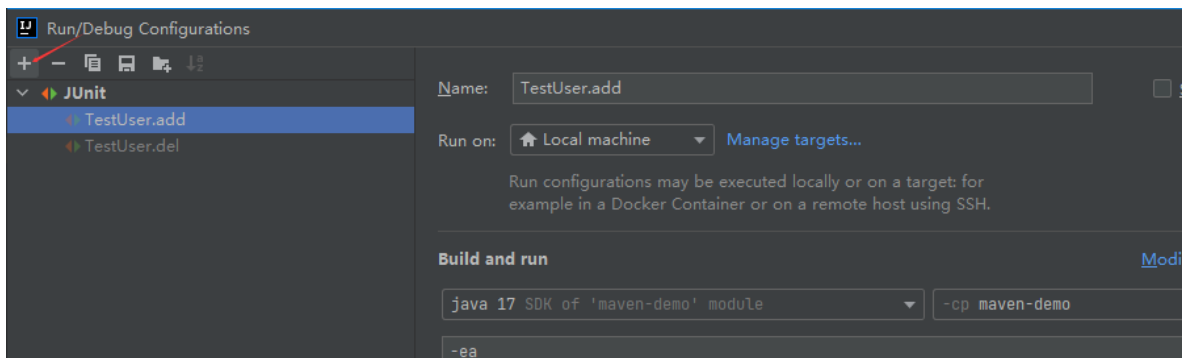
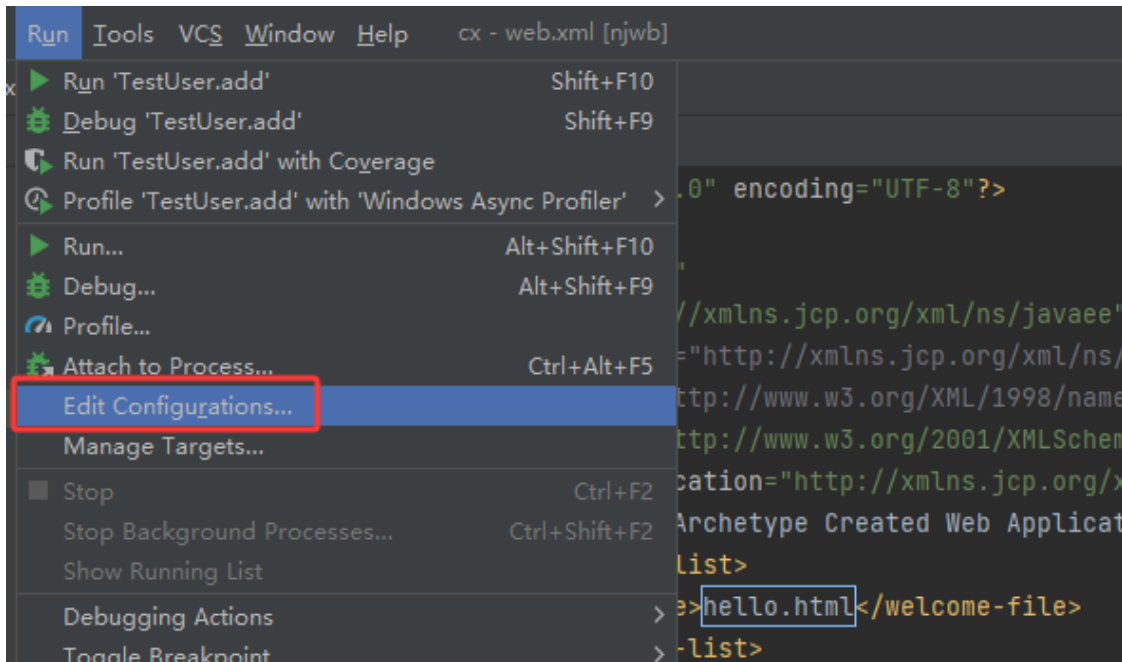


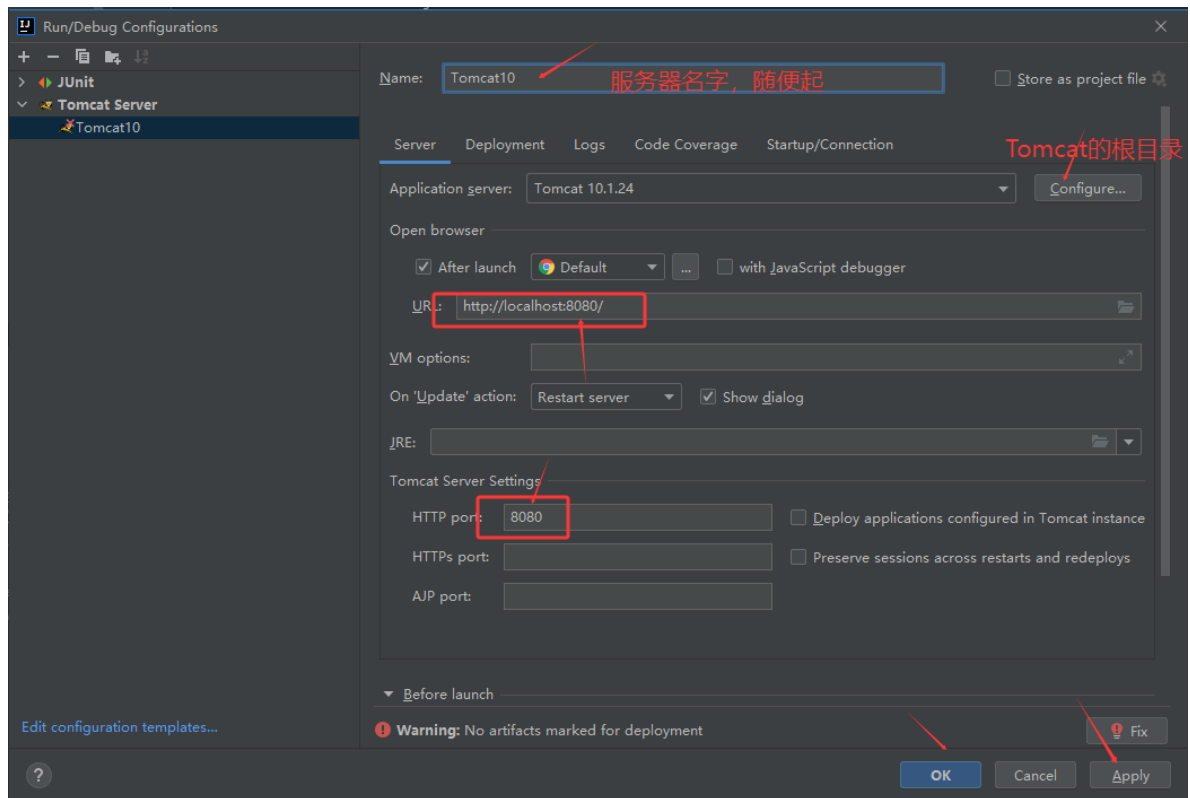
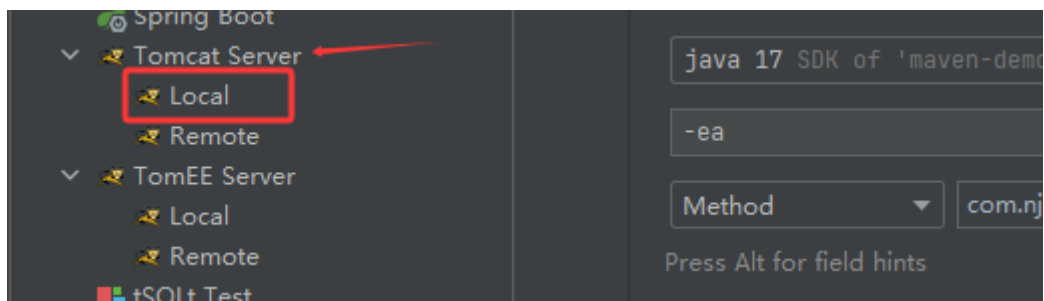


编辑web.xml

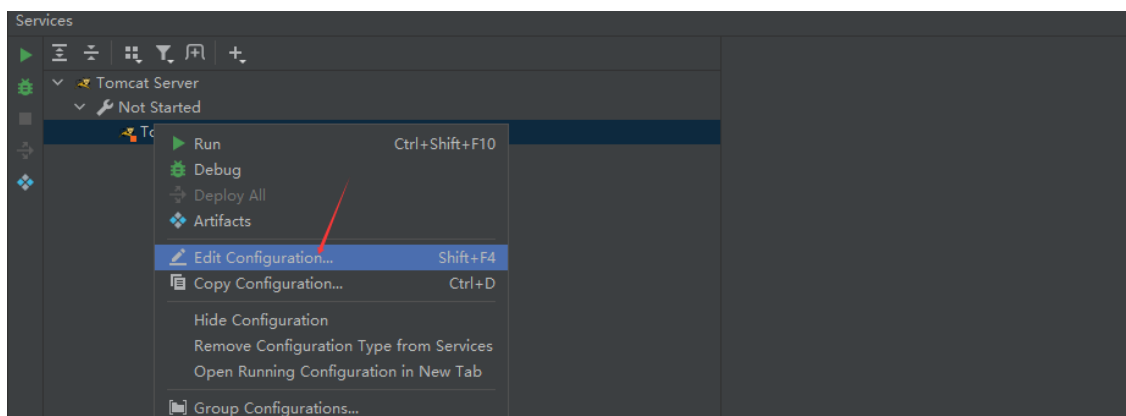
```
<welcome-file-list>
  <welcome-file>hello.html</welcome-file>
</welcome-file-list>
```

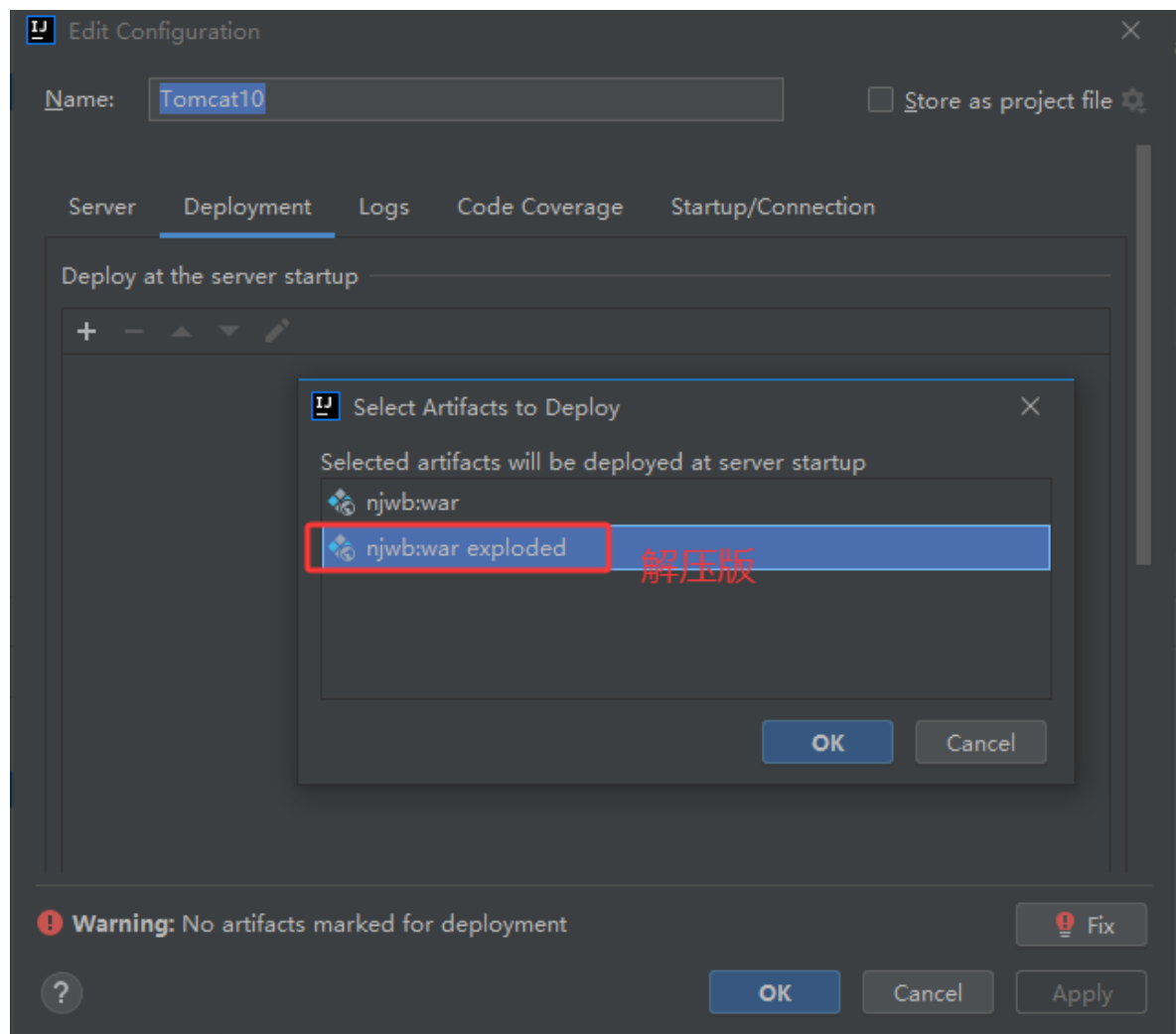
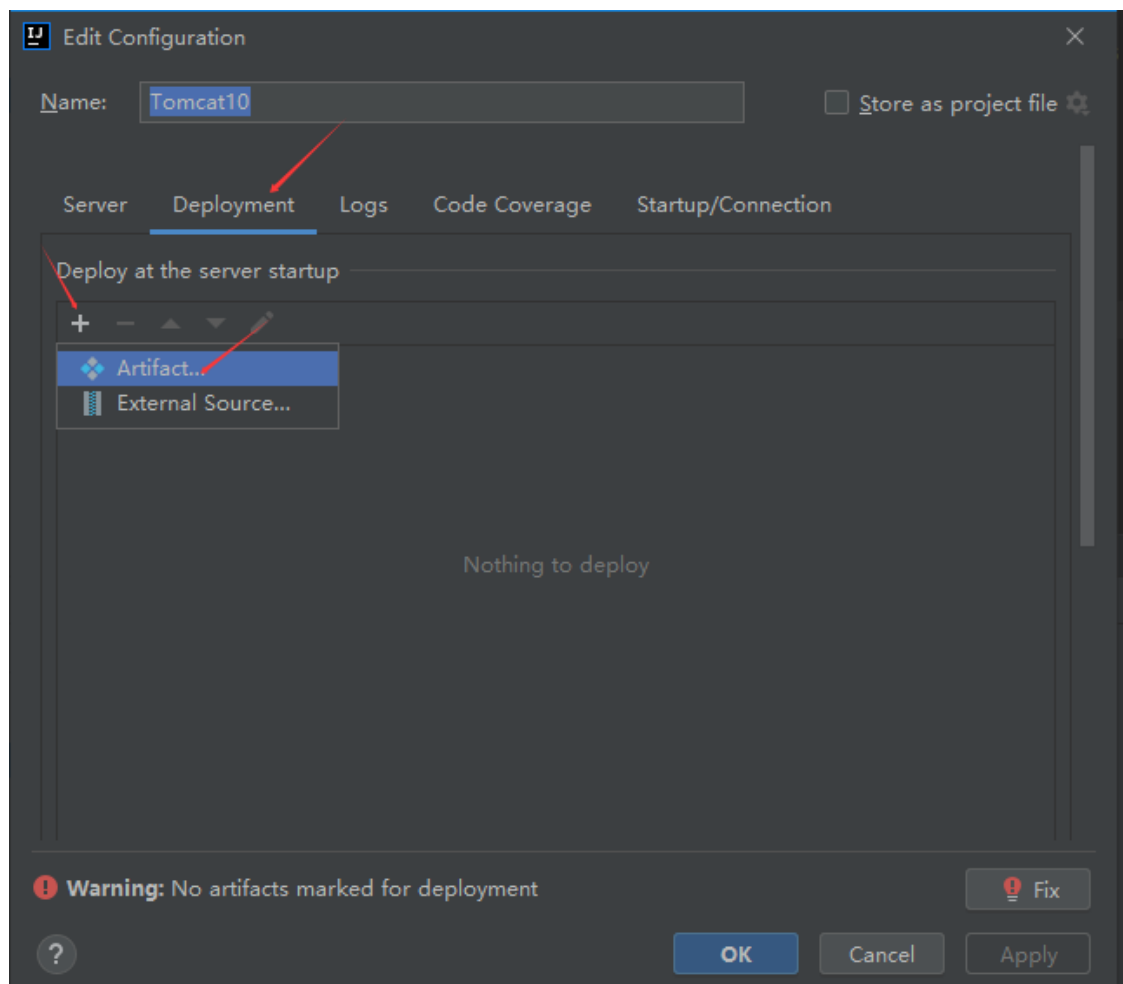
2. IDEA关联Tomcat

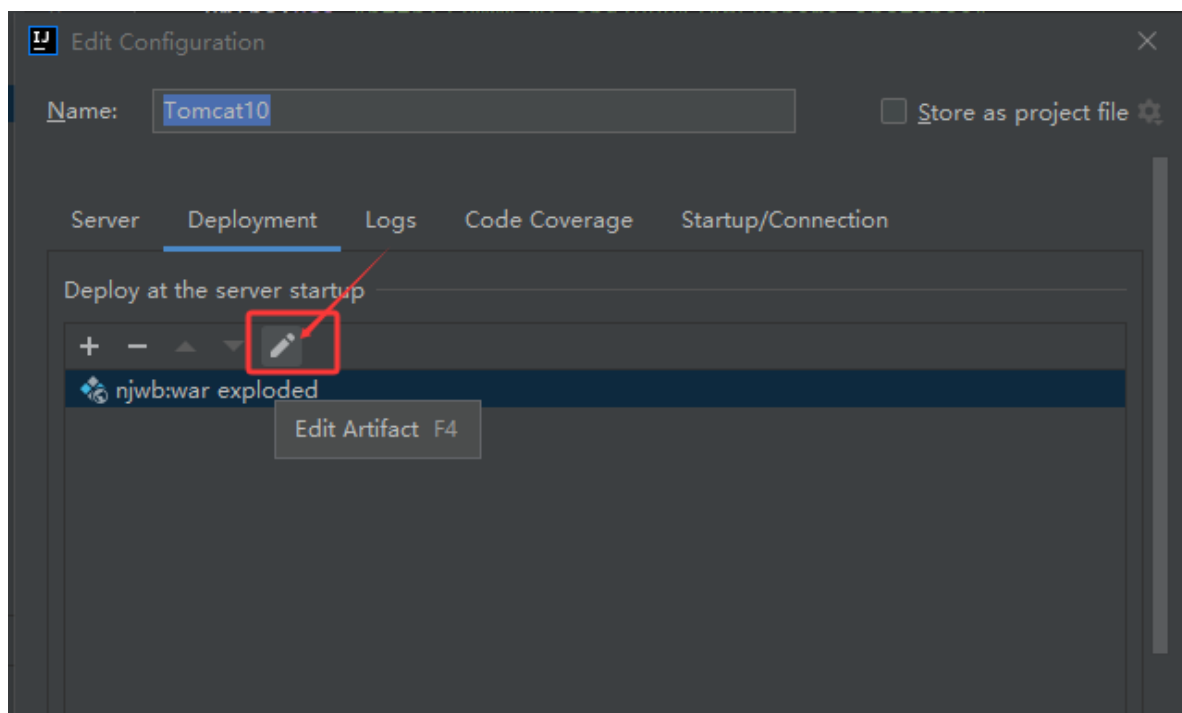
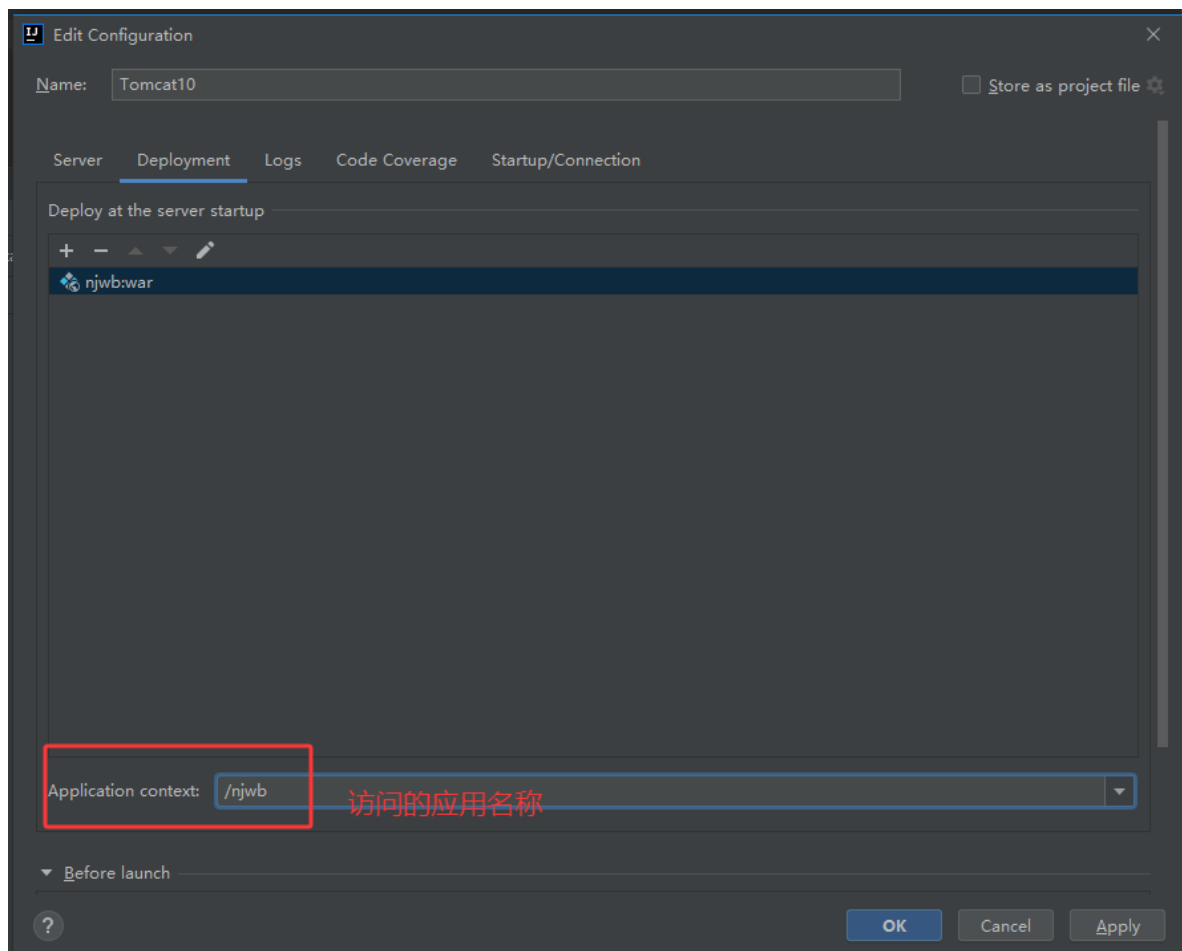


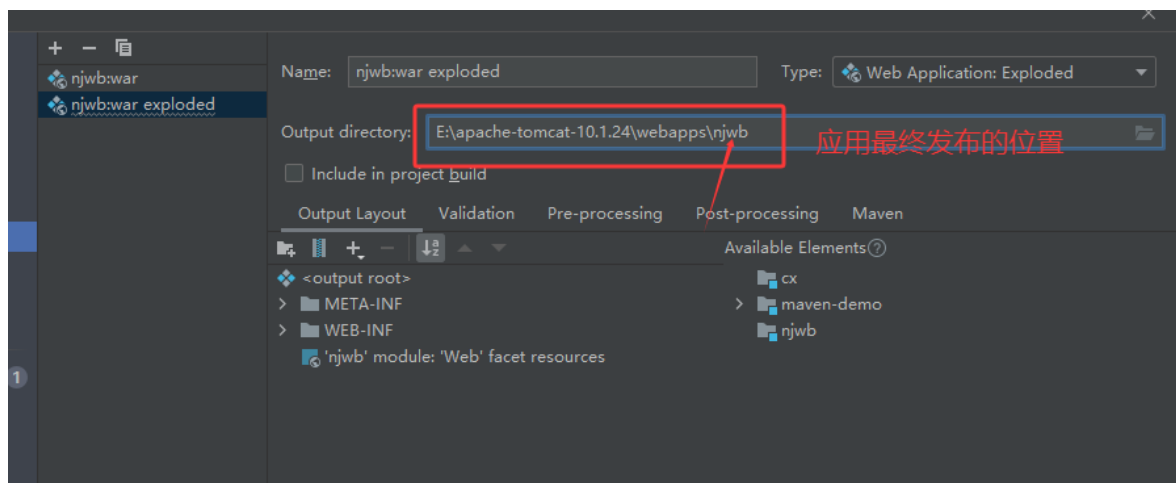


3. 部署应用至Tomcat



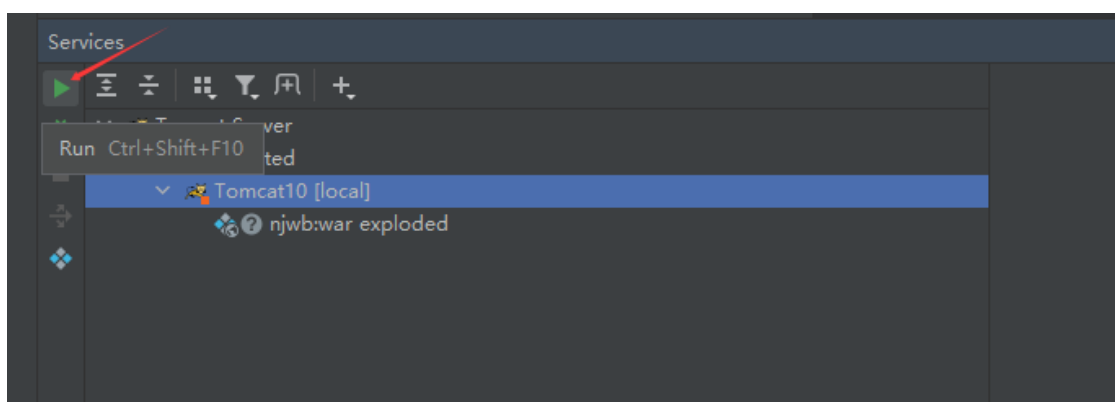






4. 启动Tomcat

如果其他位置上已经启动过Tomcat，请先停止（防止端口号冲突）



0619作业：使用JDBC连接SQLServer实现部门的CURD。

表名：dept

列名：

deptno

dname

loc

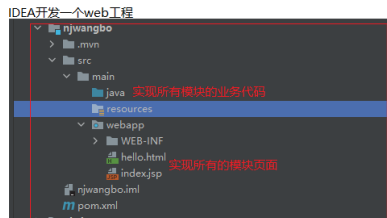
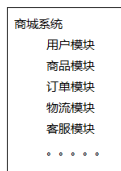
使用junit测试，添加一个部门、根据部门编号查询部门信息、修改部门、根据部门编号删除部门。

5.4 单体应用架构

什么是单体应用？

优点？

缺点？

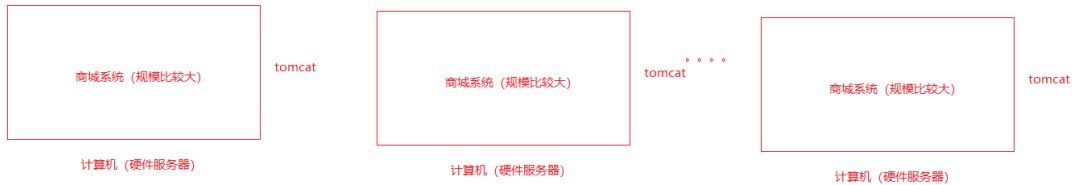


存放在一个应用中：单体应用

过去可以，但是现在不行
(功能太过复杂)

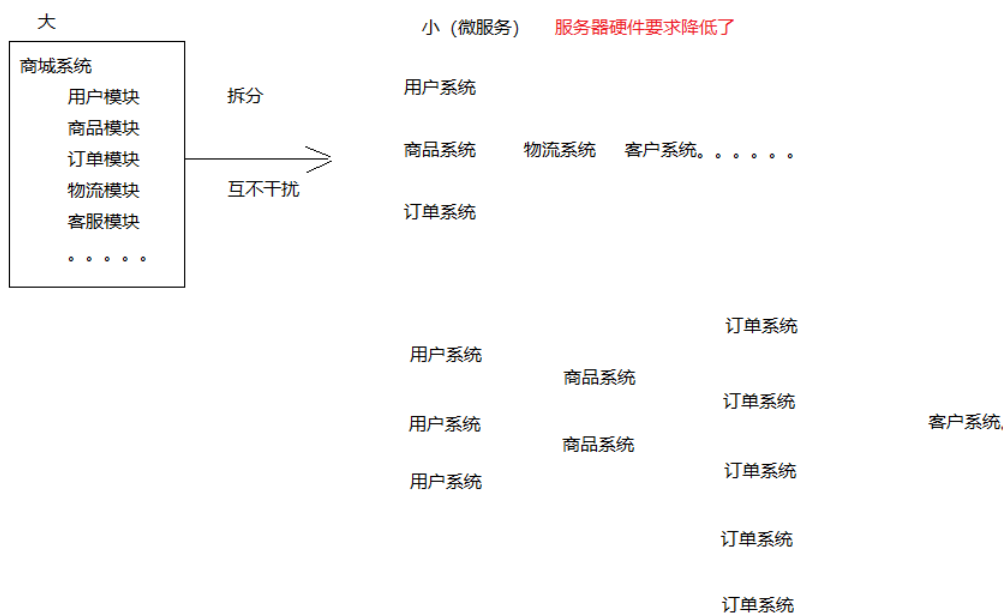
好处：方便代码的维护
方便代码的测试
方便代码的部署--只要丢到一个tomcat中

缺点：服务器的成本过高
系统升级时，所有的服务器都需要停（牵一发动全身）
服务器扩容：对标之前的服务器



服务器集群，统一对外提供服务

5.5 微服务应用架构



好处：

- 1.成本降低了
- 2.扩容时，按需扩容
- 3.更新时，独立更新

.....

缺点：应用之间的调用链路复杂，难以管理 ----管理工具 Spring Cloud Alibaba
如何快速开发微服务？ -----Spring Boot微服务开发技术

前端问题：界面问题

PC端
手机端：安卓 iOS 鸿蒙 ----->
平板
。。。。其他终端设备

专门的服务器

后端服务器



请求数据

响应结果数据

只提供数据


Java实现
一个个的微服务：Spring Boot应用

前端
独立的项目

HTML+CSS+JS
或
Vue3.....

前后端分离技术

六、Spring Boot

 **Spring Boot** 3.5.2

[OVERVIEW](#) [LEARN](#) [SUPPORT](#) [SAMPLES](#)

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

We take an opinionated view of the Spring platform and third-party libraries so you can get started with minimum fuss. Most Spring Boot applications need minimal Spring configuration.

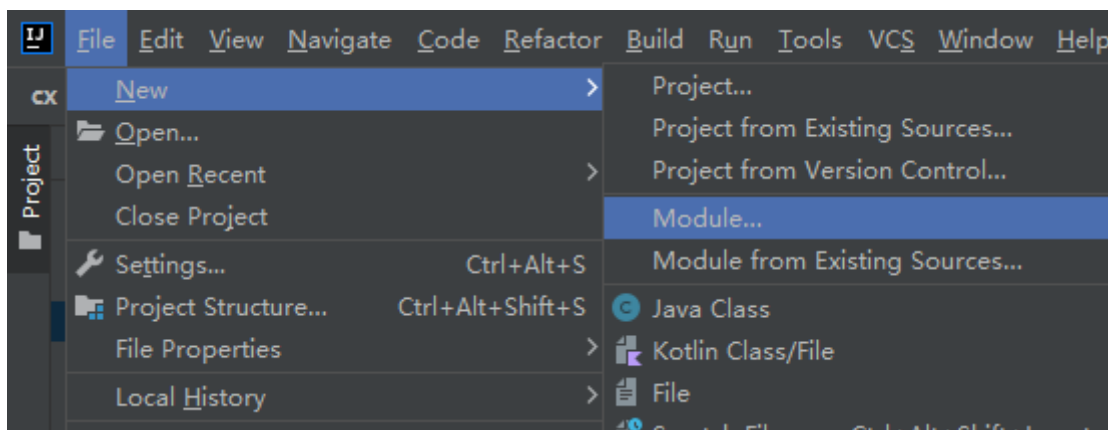
If you're looking for information about a specific version, or instructions about how to upgrade from an earlier release, check out [the project release notes section](#) on our wiki.

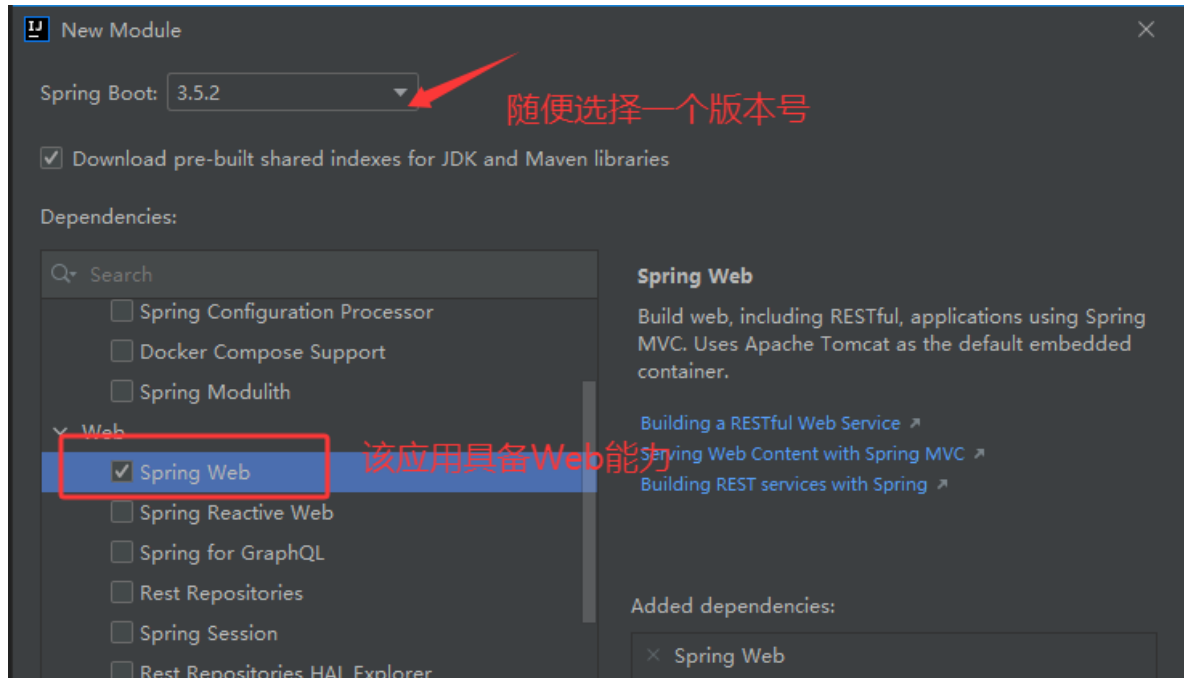
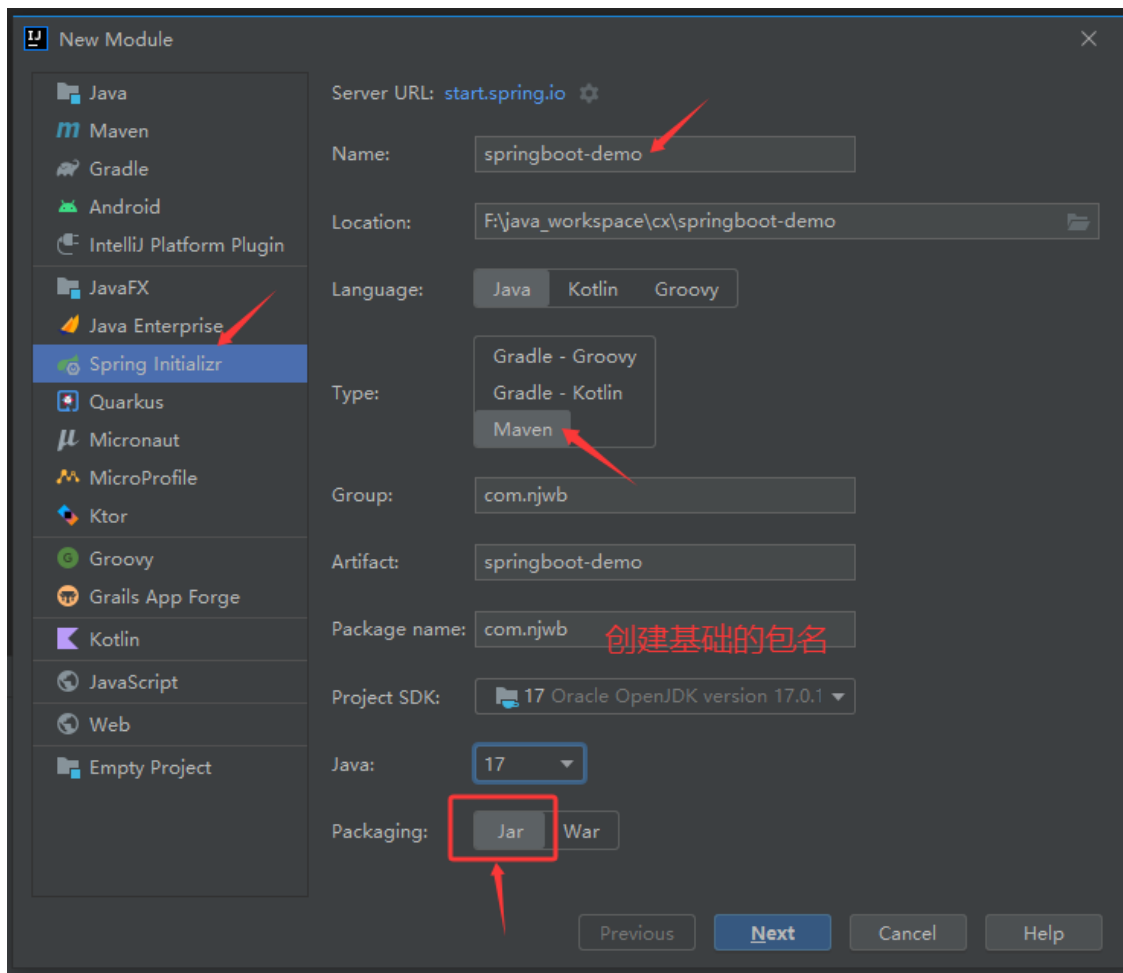
6.1 体验Spring Boot

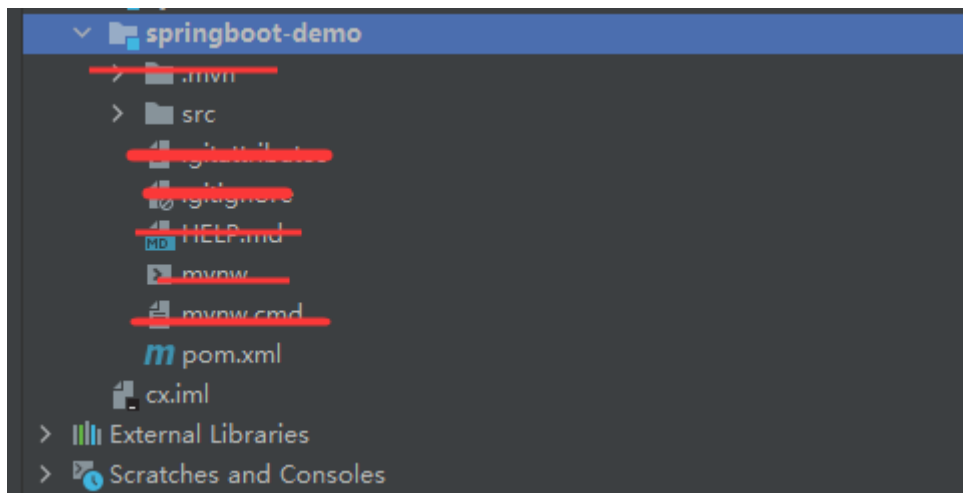
任务：浏览器发送服务端请求 <http://127.0.0.1:8080/hello>

服务端返回：“It's Cool!”

1. 创建一个Spring Boot的应用





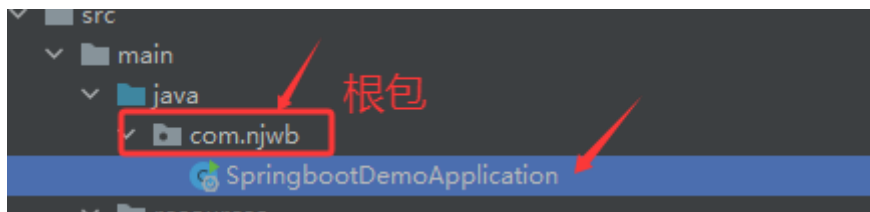


删掉

2.修改pom.xml,统一Spring Boot的版本为 3.4.6

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.4.6</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>
```

3.开发服务端



在根包下,创建一个子包 controller[接收所有的外部请求]

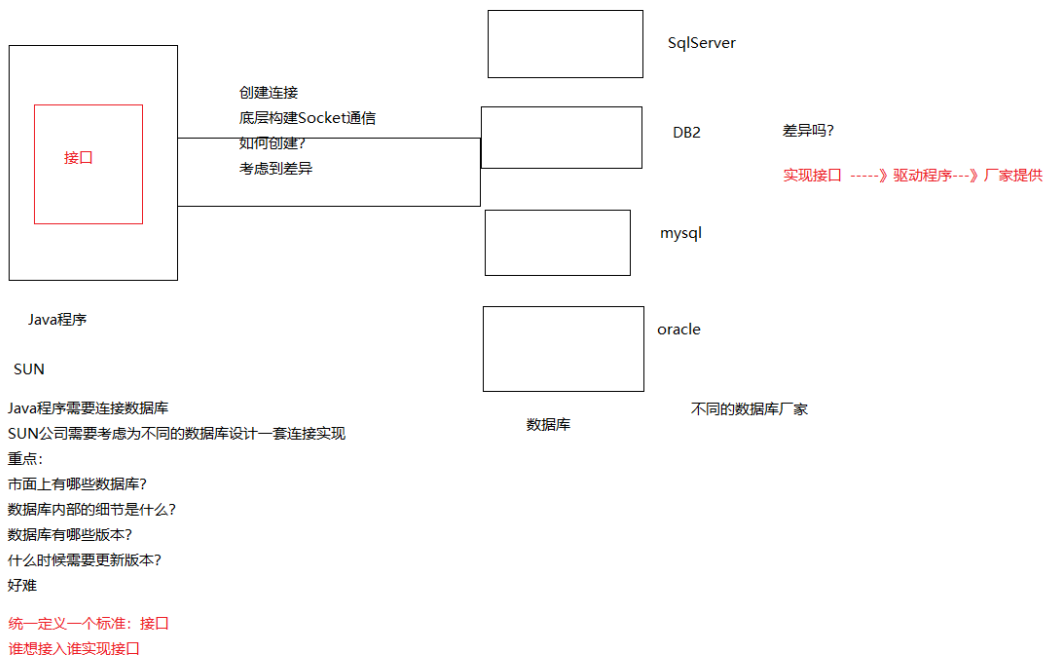
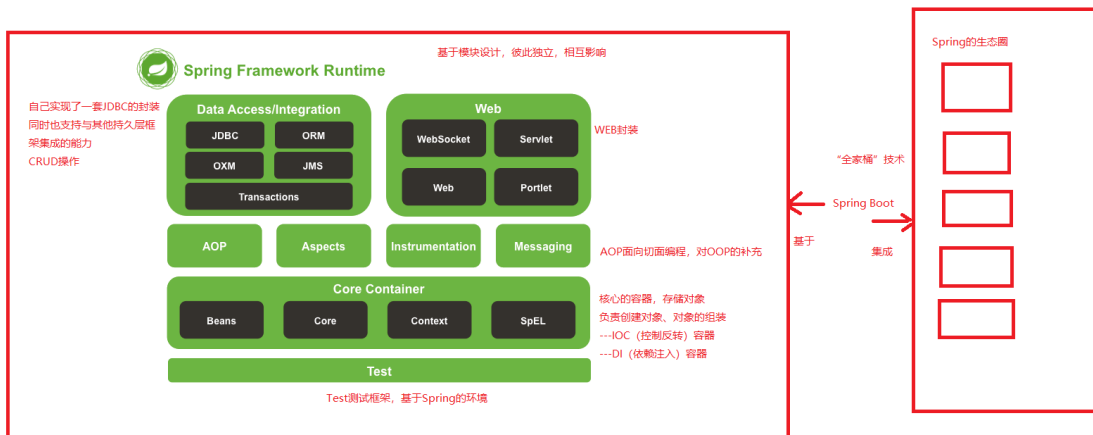
定义服务类

```
//@Controller
@RestController
public class HelloController {

    //专门接收"/hello"请求
    @RequestMapping("/hello")
    public String hello(){
        return "It's Cool!";
    }
}
```

4.main方法启动

6.2 Spring 简单介绍



程序设计优先考虑面向接口的编程.

主要谈核心容器

如何创建对象?

如果组装对象?

如果获取对象?

dao

impl

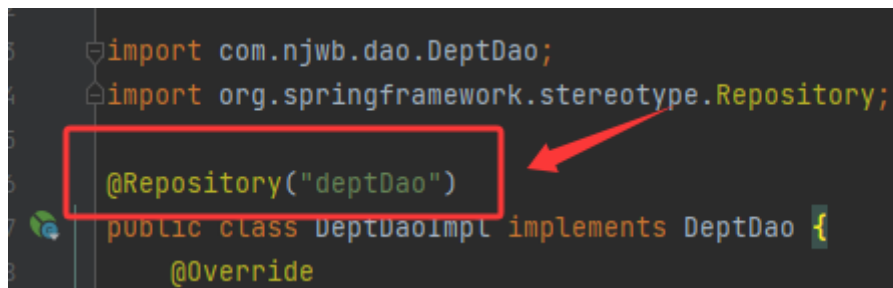
DeptDaoImpl实现类

DeptDao接口

```
public interface DeptDao {  
    void add();  
    void del();  
}
```

```
public class DeptDaoImpl implements DeptDao {  
    @Override  
    public void add() {  
  
    }  
  
    @Override  
    public void del() {  
  
    }  
}
```

让Spring的核心容器帮助我们实例化DeptDao对象



```
import com.njwb.dao.DeptDao;  
import org.springframework.stereotype.Repository;  
  
@Repository("deptDao")  
public class DeptDaoImpl implements DeptDao {  
    @Override
```

```
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class SpringbootDemoApplicationTests {
    //希望Spring 的容器将相关的对象注入进来
    @Autowired
    private DeptDao deptDao;
    @Test
    void contextLoads() {
        System.out.println(deptDao);
    }
}
```

创建对象的常用注解有哪些?

@Repository

@Service

@Controller

@RestController

@Component

区别是语义:

数据库持久层一般使用@Repository

service层使用@Service

控制器层使用@Controller或者@RestController

其他层使用@Component

```
public interface LogDao {
    void add();
}
@Repository
public class LogDaoImpl implements LogDao {
    @Override
    public void add() {
        System.out.println("add a log.....");
    }
}
```

```
//@Repository("deptDao")
//@Service("deptDao")
//@Controller
```



```
//@RestController
@Repository
public class DeptDaoImpl implements DeptDao {
    @Override
    public void add() {
        System.out.println("add a dept");
    }

    @Override
    public void del() {
        System.out.println("del a dept");
    }
}
```

```
package com.njwb.service;

public interface DeptService {
    void add();
}

@Service
public class DeptServiceImpl implements DeptService {
    @Autowired
    private DeptDao deptDao;
    @Autowired
    private LogDao logDao;
    @Override
    public void add() {
        //添加一个部门对象
        deptDao.add();

        //添加一条日志
        logDao.add();
    }
}
```

测试

```
@Autowired
private DeptService deptService;

@Test
void add() {
    deptService.add();
}
```

6.3 配置类

场景：

自定义的类，可以使用相应的注解实例化对象，如DeptDao、DeptService

非自定义的类，如何通过Spring实例化对象？

如：实例化一个Date对象

```
@Configuration//配置类
public class MyConfig {
    //实例化一个Date对象
    //配置一个Date对象
    //通过一个方法
    @Bean
    public Date date(){
        System.out.println("----date()-----");
        return new Date();//Spring框架调用后生成的对象存储到容器中，自然能够注入到程序中
    }
}
```

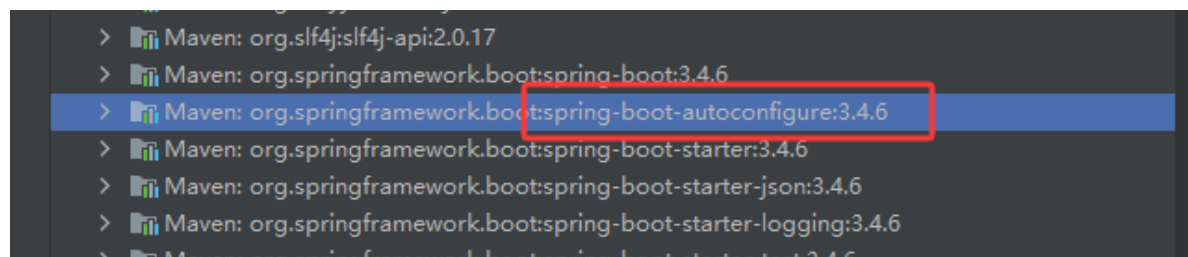
```
@Autowired
private Date date;
@Test
void date(){
    System.out.println(date);
}
```

6.4 配置项

Spring Boot:

Most Spring Boot applications need **minimal** Spring configuration.

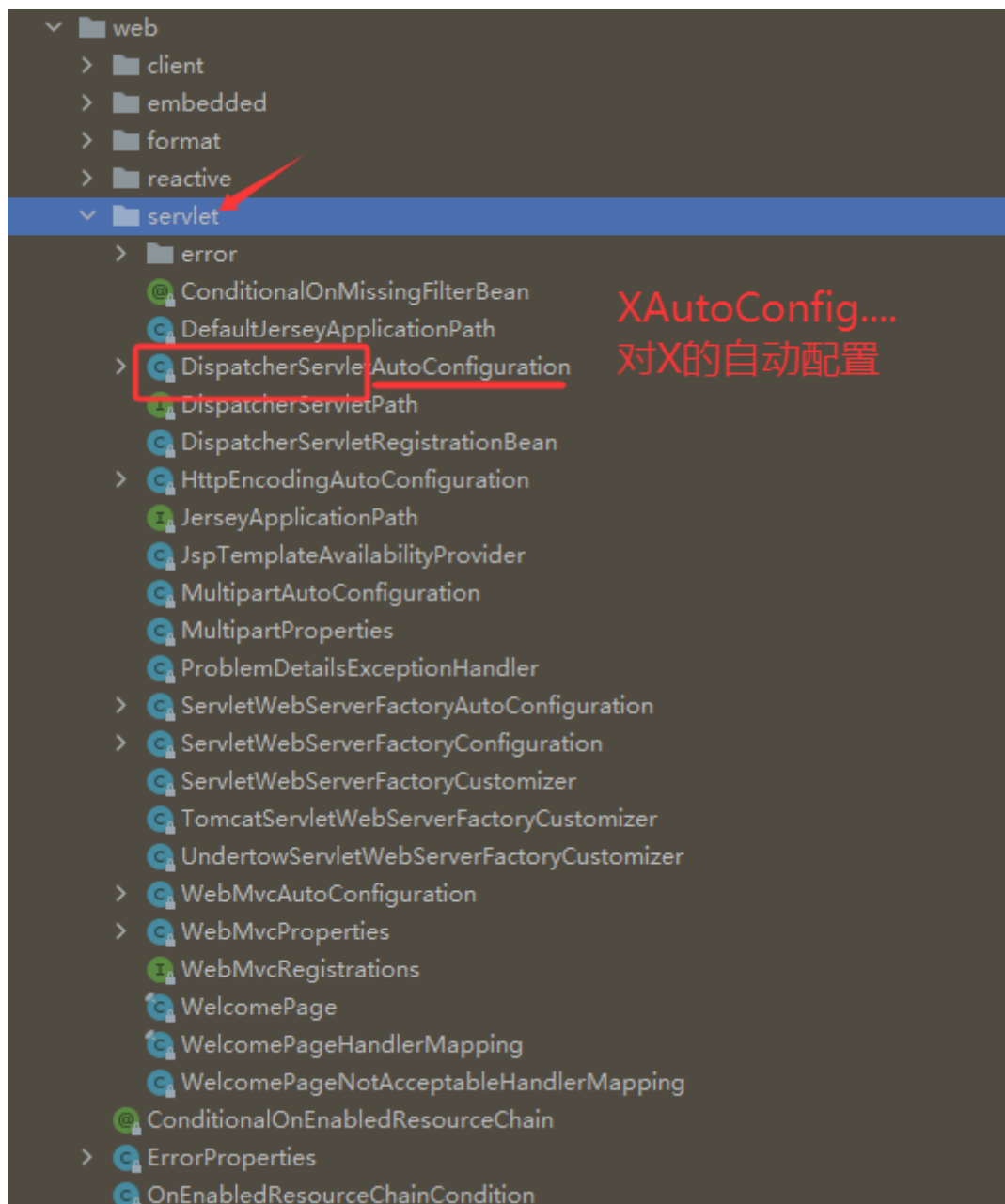
为什么使用Spring Boot构建应用后，可以使用最少配置？因为Spring Boot内部已经做了大量的自动配置。



7 META-INF

- ▼ org.springframework.boot.autoconfigure
 - > admin
 - > amqp
 - > aop
 - > availability
 - > batch
 - > cache
 - > cassandra
 - > codec
 - > condition
 - > container
 - > context
 - > couchbase
 - > dao
 - > data
 - > diagnostics.analyzer
 - > domain
 - > elasticsearch
 - > flyway
 - > freemarker
 - > graphql
 - > groovy.template

按照不同模块做自动配置



```
import ...

@AutoConfigureOrder(-2147483648)
@AutoConfiguration(
    after = {ServletWebServerFactoryAutoConfiguration.class}
)
@ConditionalOnWebApplication(
    type = Type.SERVLET
)
@ConditionalOnClass({DispatcherServlet.class})
public class DispatcherServletAutoConfiguration { 配置类
    public static final String DEFAULT_DISPATCHER_SERVLET_BEAN_NAME = "dispatcherServlet";
    public static final String DEFAULT_DISPATCHER_SERVLET_REGISTRATION_BEAN_NAME = "dispatcherServletRegistration";

    public DispatcherServletAutoConfiguration() {
    }
}
```

```

)
@Conditional({DispatcherServletAutoConfiguration.DispatcherServletRegistrationCondition.class})
@ConditionalOnClass({ServletRegistration.class})
@EnableConfigurationProperties({WebMvcProperties.class})
@Import({DispatcherServletAutoConfiguration.DispatcherServletConfiguration.class})
protected static class DispatcherServletRegistrationConfiguration {
    protected DispatcherServletRegistrationConfiguration() {
    }
}
@Bean

```

在配置类中，搜带有Properties的类

```

import ...

@ConfigurationProperties(
    prefix = "spring.mvc"
)
public class WebMvcProperties {
    private org.springframework.validation.DefaultMessageCodesResolver.Format messageCodesResolverFormat;
    private final WebMvcProperties.Format format = new WebMvcProperties.Format();
    private boolean dispatchTraceRequest = false;
    private boolean dispatchOptionsRequest = true;
    private boolean publishRequestHandledEvents = true;
    private boolean logRequestData;
    private boolean logResolvedException = false;
    private String staticPathPattern = "/*";
    private String webjarsPathPattern = "/webjars/**";
    private final WebMvcProperties.Async async = new WebMvcProperties.Async();
    private final WebMvcProperties.Servlet servlet = new WebMvcProperties.Servlet();
    private final WebMvcProperties.View view = new WebMvcProperties.View();
    private final WebMvcProperties.Contentnegotiation contentnegotiation = new WebMvcProperties.Contentnegotiation();
    private final WebMvcProperties.Pathmatch pathmatch = new WebMvcProperties.Pathmatch();
    private final WebMvcProperties.Problemdetails problemdetails = new WebMvcProperties.Problemdetails();
}

```

前缀 spring.mvc

属性值

```

application.properties
1 server.port=8888
2 server.servlet.context-path=/boot
3 spring.mvc.dispatch-options-request=false

```

任务：jdbc模块的配置，通过上述的自动配置原理的思路，查看源码，找到数据库的配置项是什么？

6.5 自定义配置

在哪里配置

resources/application.properties或者resources/application.yml

二者选一，先采用application.properties

设计配置

需求：Dept类{deptNo、dname、loc}，通过配置的形式，给属性注入数据。

```

public class Dept {
    private Integer deptNo;
    private String dname;
    private String loc;

    public Integer getDeptNo() {
        return deptNo;
    }

    public void setDeptNo(Integer deptNo) {
        this.deptNo = deptNo;
    }

    public String getDname() {
        return dname;
    }

    public void setDname(String dname) {
        this.dname = dname;
    }

    public String getLoc() {
        return loc;
    }

    public void setLoc(String loc) {
        this.loc = loc;
    }

    @Override
    public String toString() {
        return "Dept{" +
            "deptNo=" + deptNo +
            ", dname='" + dname + '\'' +
            ", loc='" + loc + '\'' +
            '}';
    }
}

```

设计的规则：

前缀 + 属性名

dept.deptNo

dept.dname

dept.loc

在application.properties编写

```

dept.deptNo=10
dept.dname=coding
dept.loc=js06

```

程序如何读取配置

方案一、

```
@Component//实例化Dept对象
public class Dept {
    //注入application.properties中配置的数据
    @Value("${dept.deptNo}")
    private Integer deptNo;
    @Value("${dept.dname}")
    private String dname;
    @Value("${dept.loc}")
    private String loc;
```

测试

```
@Autowired//注入一个对象    @Value注入一个基本的数据值
private Dept dept;
@Test
void dept(){
    System.out.println(dept);
}
```

唯一的不足：属性多了，需要一个个绑定 @Value("\${dept.deptNo}")

方案二、一次性批量绑定

```
@Component//实例化Dept对象
@ConfigurationProperties(prefix = "dept")
public class Dept {
    private Integer deptNo;
    private String dname;
    private String loc;
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-configuration-processor</artifactId>
</dependency>
```

采用yml文件

语法格式：

1. key: 空格+值
2. 层级关系 ： 使用缩进 【使用空格缩进或者回车键缩进】

```
server:
  port: 8888
  servlet:
    context-path: /boot

dept:
  dept-no: 20
  dname: testing
  loc: js05
```

建议使用yaml结构配置。

作业：

采用面向对象的设计方式，设计人有一部手机

人：姓名 性别 年龄

手机：品牌 价格

其中，数据采用yaml格式配置，最后程序读取配置信息，注入到对象中。

1. 设计配置

```
person:
  name: Lilei
  sex: M
  age: 20
  phone:
    brand: Huawei
    price: 6000
```

2. 定义Phone类

```
package com.njwb.entity;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "person.phone")
public class Phone {
    private String brand;
    private Double price;

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }
}
```



```

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Phone{" +
            "brand='" + brand + '\'' +
            ", price=" + price +
            '}';
    }
}

```

3. 定义Person类

```

package com.njwb.entity;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

@Component
@ConfigurationProperties(prefix = "person")
public class Person {
    private String name;
    private String sex;
    private Integer age;
    private Phone phone;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getSex() {
        return sex;
    }

    public void setSex(String sex) {
        this.sex = sex;
    }

    public Integer getAge() {
        return age;
    }

    public void setAge(Integer age) {
        this.age = age;
    }
}

```

```
public Phone getPhone() {
    return phone;
}

public void setPhone(Phone phone) {
    this.phone = phone;
}

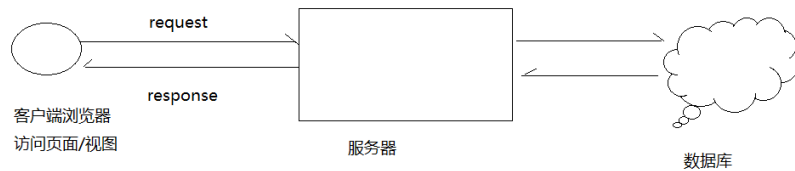
@Override
public String toString() {
    return "Person{" +
        "name='" + name + '\'' +
        ", sex='" + sex + '\'' +
        ", age=" + age +
        ", phone=" + phone +
        '}';
}
}
```

4. 测试

[illegible]

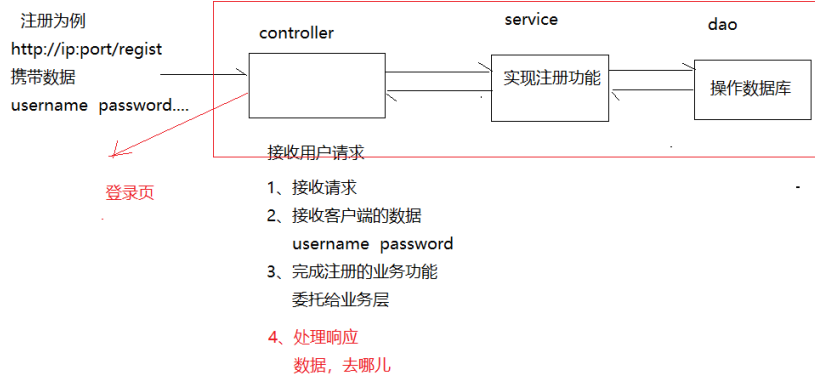
七、程序架构基础

7.1 后端分层



基于请求与响应的模式

服务器的三层模型



库

7.2 Restful架构

Restful: Representational State Transfer 表述层状态转移

2000年Roy Thomas Fielding 博士论文中提出

API设计规范:

1. 将一切数据视为资源
2. 利用Http请求方式，描述对资源的操作（增/删/改/查）
3. 通过Http响应状态码，描述对资源的操作结构

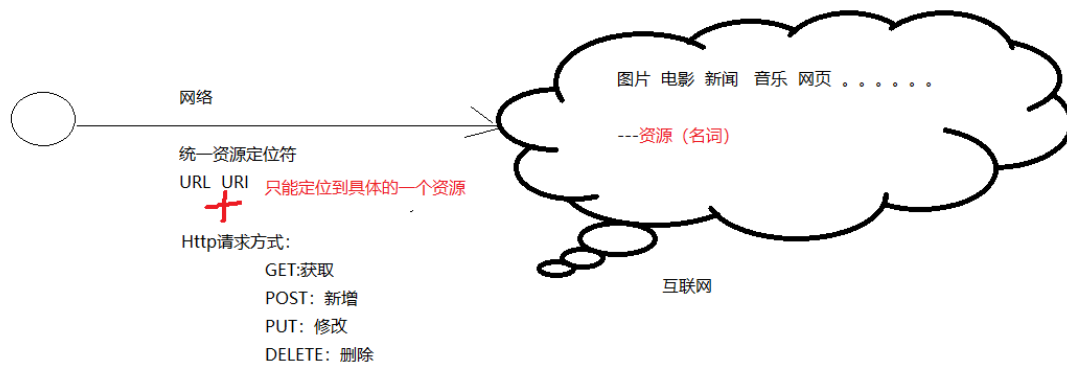
期望效果:

1. 看URL知道是什么资源。
2. 看Method知道要对资源进行什么操作
3. 看Response Code响应知道操作是否成功

Method规范: 用于描述操作（动词）

1. GET: 获取
2. POST: 新增
3. PUT: 修改
4. DELETE: 删除

URL规范: 只使用名词，不使用动词。



案例：

功能	传统设计	Restful设计风格
根据部门编号查询部门数据	/app/getByNo?no=10	/app/depts/10 + GET
查询所有的部门信息	/app/findAll	/app/depts + GET
修改部门信息	/app/update	/app/depts + PUT
新增	/app/add	/app/depts + POST
删除	/app/delByNo?no=10	/app/depts/10 + DELETE

响应的数据标准结构包含三部分：

响应码：如：200

响应的描述 如： success

响应数据： data

7.3 前后端交互数据

采用json格式

json：js的对象表示法，js能快速解析

使用json表示一个部门对象

```
{
    deptNo:10,
    dname:"研发部",
    loc:"101室"
```

```
}
```

使用json表示部门对象集合

```
[
```

```
{
```

```
    deptNo:10,
```

```
    dname:"研发部",
```

```
    loc:"101室"
```

```
},
```

```
{
```

```
    deptNo:20,
```

```
    dname:"测试部",
```

```
    loc:"102室"
```

```
}
```

```
.. . . . . . . . . .
```

```
]
```

Restful响应:

```
{
```

```
    code : 200,
```

```
    desc: "success",
```

```
    data: {    deptNo:10,dname:"研发部",loc:"101室"}
```

```
}
```

有的时候简化响应，只留数据部分

7.4 Restful案例

模拟实现部门的CRUD。

1. 定义实体类

```
//@Component//实例化Dept对象
//@ConfigurationProperties(prefix = "dept")
public class Dept {
    private Integer deptNo;
    private String dname;
    private String loc;
```

2. 定义控制器,实现查询所有部门

```

@RestController//支持Restful风格
public class DeptController {
    //使用内存模拟数据库
    private static Map<Integer, Dept> depts = new HashMap<Integer, Dept>();

    static {
        //初始化数据
        Dept dept1 = new Dept();
        dept1.setDeptNo(10);
        dept1.setDname("研发中心");
        dept1.setLoc("L101室");
        Dept dept2 = new Dept();
        dept2.setDeptNo(20);
        dept2.setDname("测试中心");
        dept2.setLoc("L102室");
        depts.put(dept1.getDeptNo(), dept1);
        depts.put(dept2.getDeptNo(), dept2);
    }

    //查询所有的部门信息
    // @RequestMapping(value = "/depts",method = RequestMethod.GET)
    @GetMapping("/depts")
    public List<Dept> findAll(){
        return new ArrayList(depts.values());
    }

    //根据部门编号查询部门信息

    //新增部门

    //修改部门

    //删除部门
}

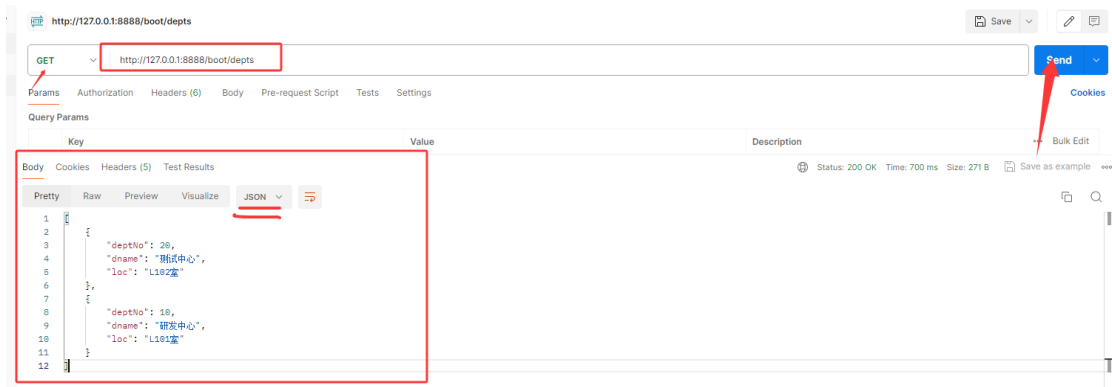
```

3. 启动，测试

java服务测试，常用的测试方案：

a.浏览器直接访问【get请求】，简单，但是对于其他的请求方式测试比较困难

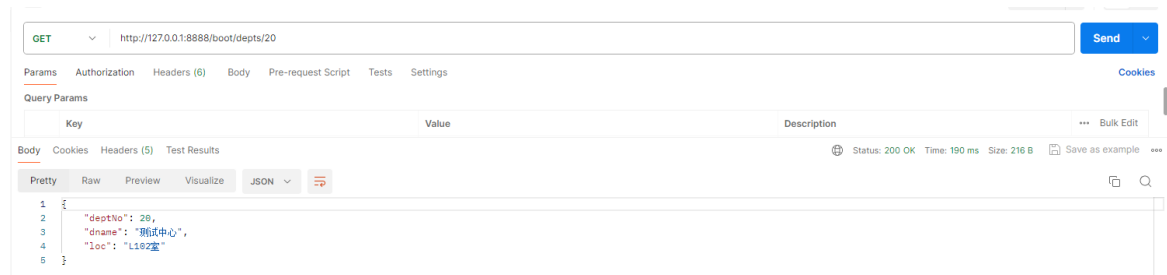
b.专门的测试工具 PostMan



根据部门编号查询部门信息

//根据部门编号查询部门信息

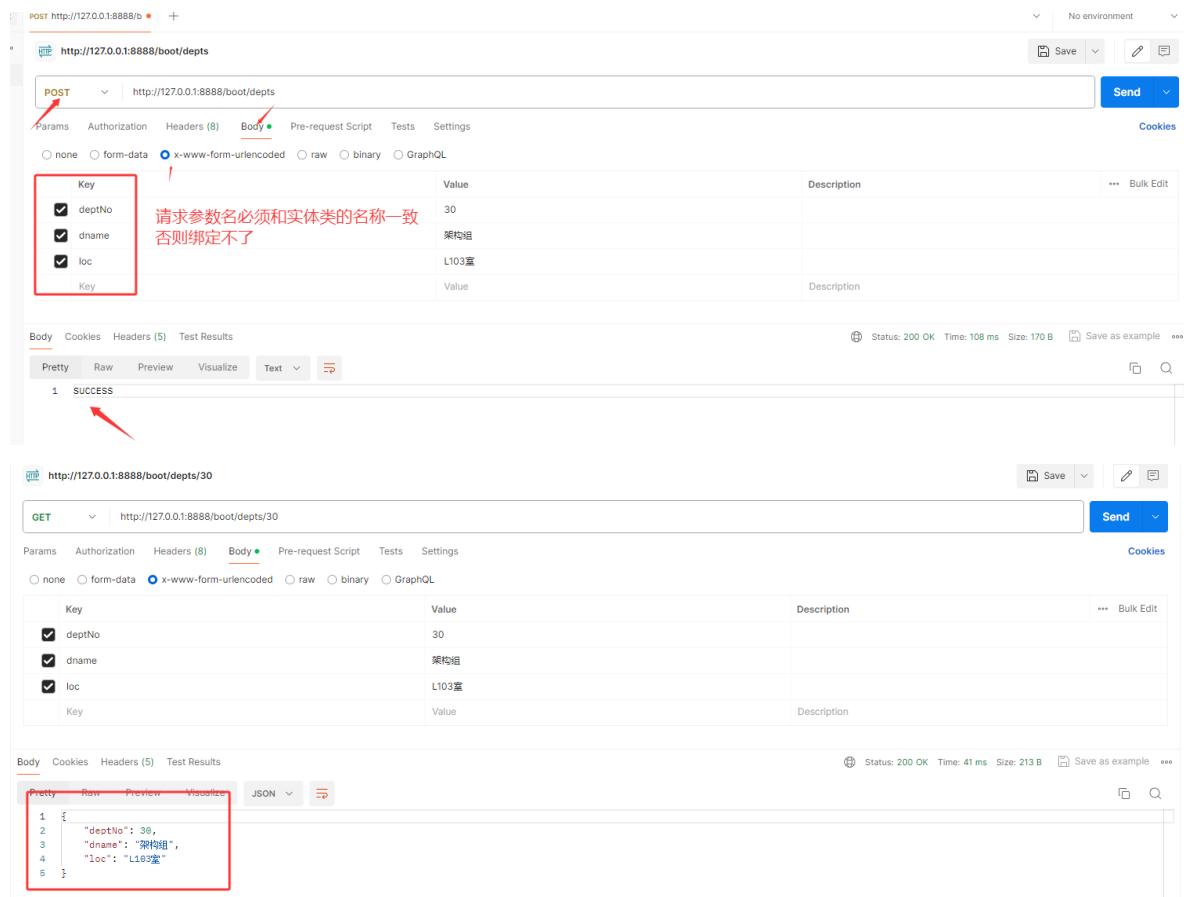
```
@GetMapping("/depts/{deptNo}")// {占位符} 表示动态参数，需要客户端传递参数过来
public Dept findByNo(@PathVariable("deptNo") Integer no){//接收参数，来自于占位符中的
    return depts.get(no);
}
```



新增部门:

//新增部门

```
@PostMapping("/depts")
public String add(Dept dept){//Dept接收客户端的表单数据
    depts.put(dept.getDeptNo(),dept);
    return "SUCCESS";
}
```



修改部门

```
//修改部门
@PutMapping("/depts")
public String modify(Department dept){//Dept接收客户端的表单数据
    //先查询旧的数据
    Department oldDept = depts.get(dept.getDeptNo());
    oldDept.setDname(dept.getDname());
    oldDept.setLoc(dept.getLoc());
    return "SUCCESS";
}
```

GET http://127.0.0.1:8888/boot/depts/10

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description
deptNo	30	
dname	架构组	
loc	L103室	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 126 ms Size: 216 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "deptNo": 10,
3   "dname": "研发中心",
4   "loc": "L101室"
5 }
```

PUT http://127.0.0.1:8888/boot/depts

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description
deptNo	10	
dname	特别研发组-国之重器	
loc	地下	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 31 ms Size: 170 B Save as example

Pretty Raw Preview Visualize Text

```
1 SUCCESS
```

GET http://127.0.0.1:8888/boot/depts/10

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Key	Value	Description
deptNo	10	
dname	特别研发组-国之重器	
loc	地下	

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 6 ms Size: 231 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "deptNo": 10,
3   "dname": "特别研发组-国之重器",
4   "loc": "地下"
5 }
```


删除

```
//删除部门
@DeleteMapping("/depts/{deptNo}")
public String del(@PathVariable("deptNo") Integer no){
    depts.remove(no);
    return "SUCCESS";
}
```

作业：仿照部门的CRUD，实现毕设课题的CRUD

毕设课题包含信息{

课题id

课题名称

指导老师名称

指导老师的邮箱

}

实现：查询所有课题、根据id查询课题、发布课题、修改课题、根据id删除课题功能，采用Postman测试。

作业完成后提交到群内文件夹。

7.5 前后端交互

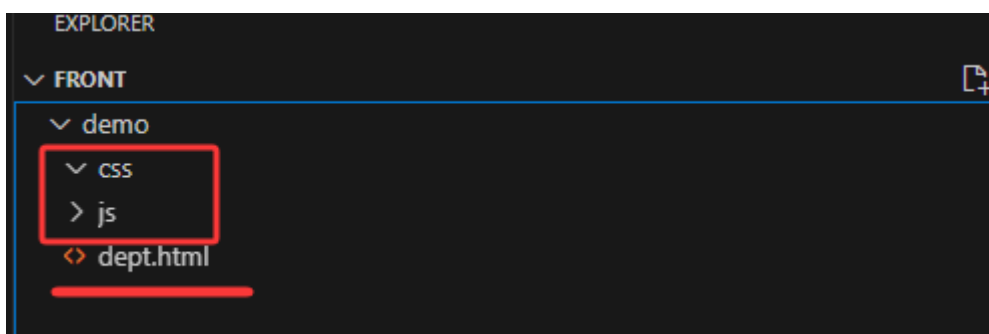
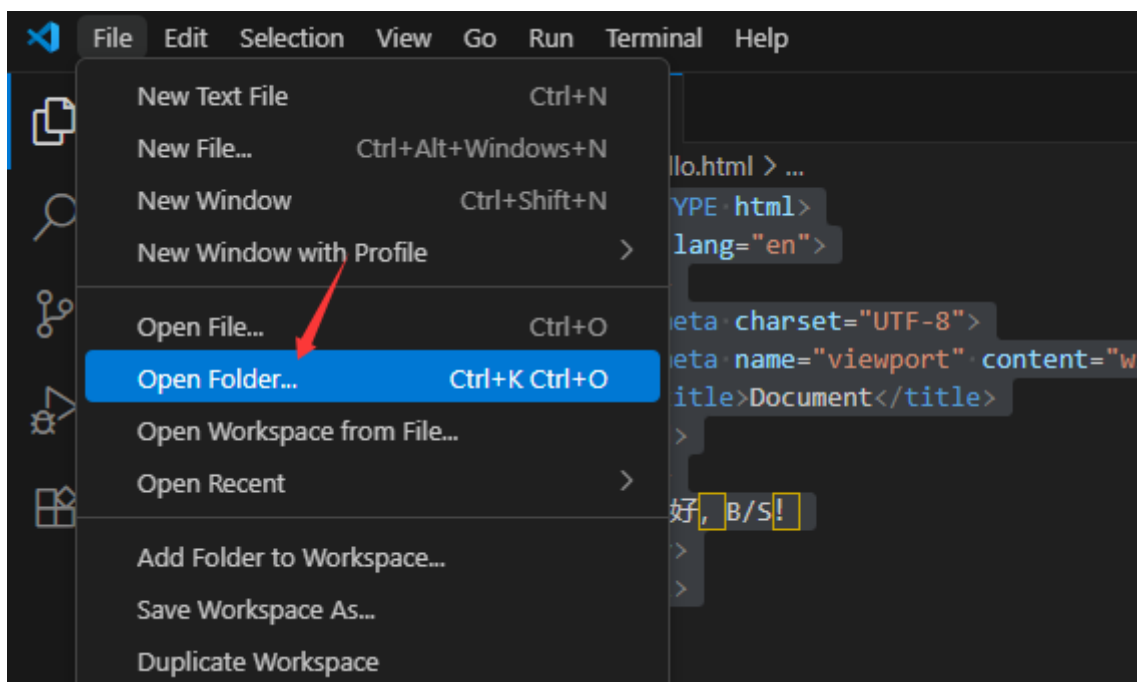
前端：HTML+CSS+JS ：界面的显示，使用vscode开发

1、采用VSCode创建项目





打开vscode



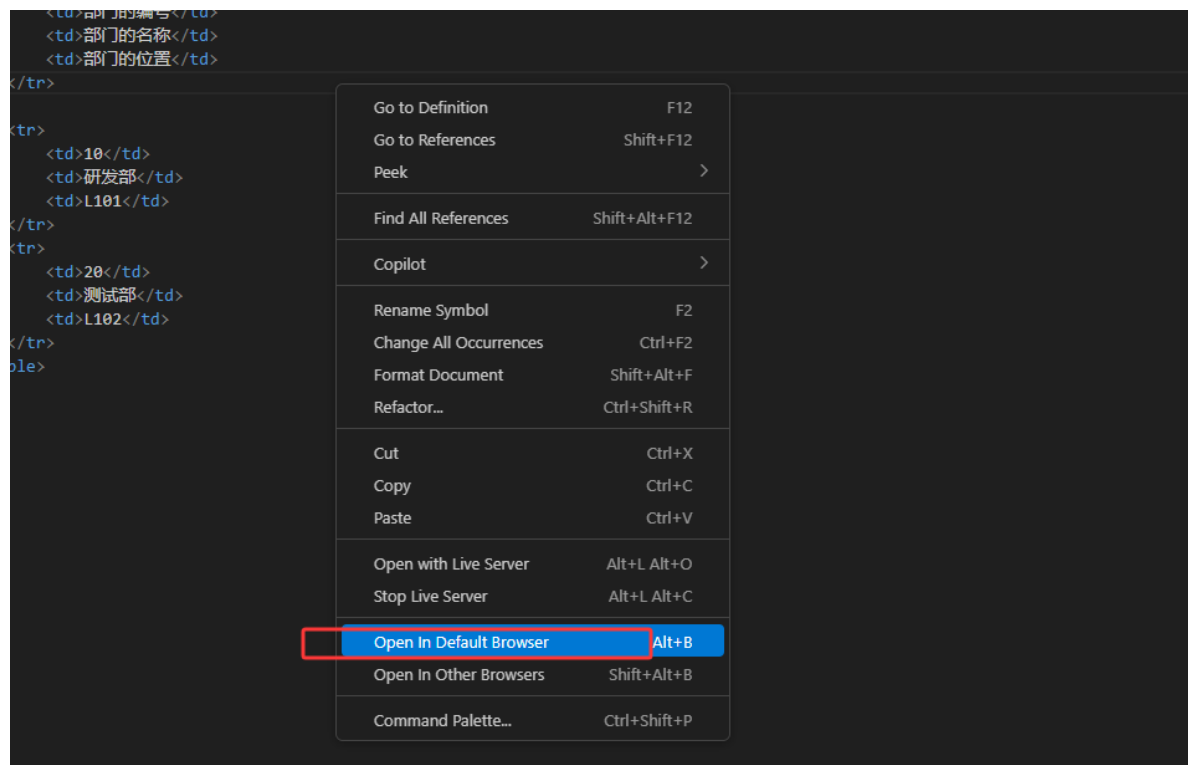
2、dept.html文件

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- ! + 回车 生成html代码的骨架 -->
  <table>
```

```
<tr>
  <td>部门的编号</td>
  <td>部门的名称</td>
  <td>部门的位置</td>
</tr>

<tr>
  <td>10</td>
  <td>研发部</td>
  <td>L101</td>
</tr>
<tr>
  <td>20</td>
  <td>测试部</td>
  <td>L102</td>
</tr>
</table>
</body>
</html>
```

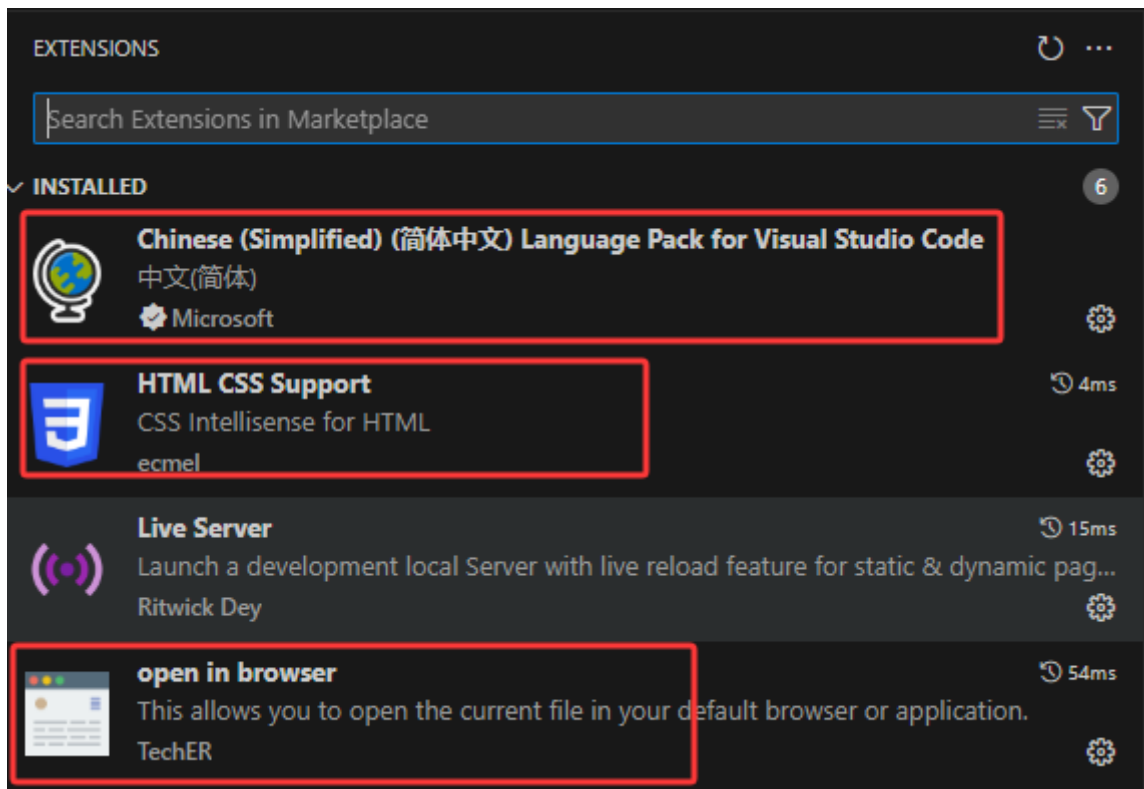
3、访问html文件



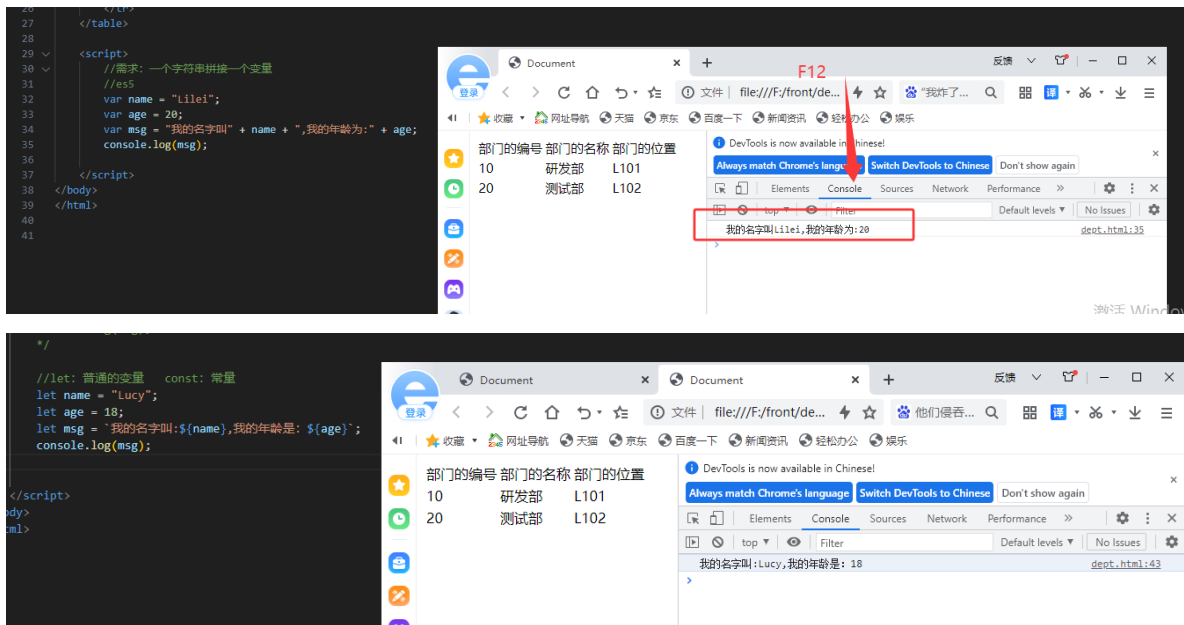
为了后面方便测试：安装几个插件



```
demo > <> dept.html > html > body > table > c
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="wi
6     <title>Document</title>
7 </head>
8 <body>
9     <!-- ! + 回车 生成html代码的骨架 --
10    <table>
11        <tr>
12            <td>部门的编号</td>
13            <td>部门的名称</td>
14            <td>部门的位置</td>
15        </tr>
16
17        <tr>
18            <td>10</td>
19            <td>研发部</td>
20            <td>L101</td>
21        </tr>
22        <tr>
23            <td>20</td>
24            <td>测试部</td>
25            <td>L102</td>
26        </tr>
27    </table>
28 </body>
29 </html>
30
31
```

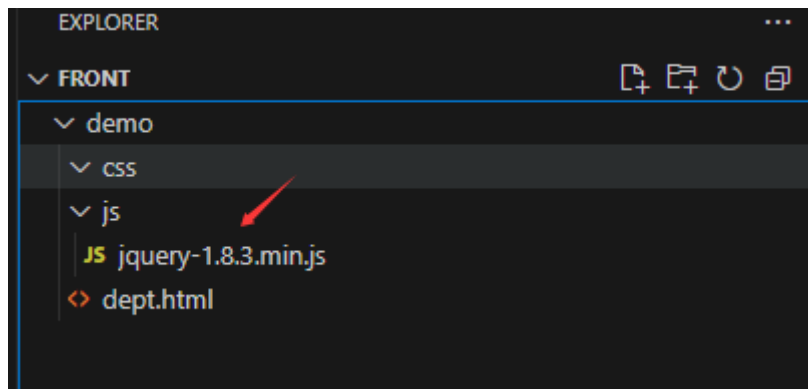


4、ES6模板字符串替换



5、ajax交互

导入jquery库



编写js代码

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- ! + 回车 生成html代码的骨架 -->
  <table>
    <tr>
      <td>部门的编号</td>
      <td>部门的名称</td>
      <td>部门的位置</td>
    </tr>

    <tr>
      <td>10</td>
      <td>研发部</td>
      <td>L101</td>
    </tr>
    <tr>
      <td>20</td>
      <td>测试部</td>
      <td>L102</td>
    </tr>
  </table>
  <!-- 导入jquery库 -->
  <script src="js/jquery-1.8.3.min.js"></script>
  <script>
    //加载所有的部门数据
    function loadDepts(){
      //远程调用服务器
      //$.ajax(); 调用
      //$.ajax({传参, 参数也是json格式});
      $.ajax({
        url : "http://127.0.0.1:8888/boot/depts",
        type : "GET",
        dataType:"json",
        success:function(depts){
          console.log(depts);
        }
      });
    }
  </script>
</body>
</html>
```

```

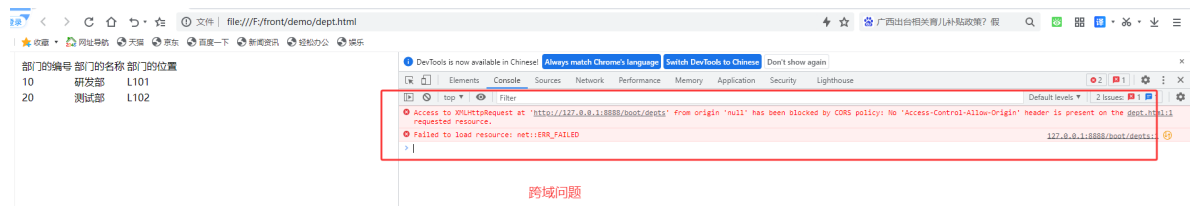
    }
    loadDepts();

</script>
</body>
</html>

```

启动Spring Boot应用

页面访问



在Spring Boot应用端配置跨域

新建配置类CorsConfig:

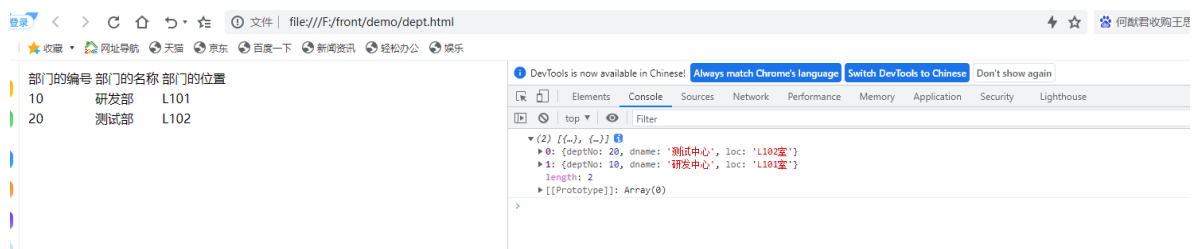
```

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**") // 匹配所有路径
            .allowedOriginPatterns("*") // 允许所有域名（或指定如
"http://localhost:3000"）
            .allowedMethods("GET", "POST", "PUT", "DELETE") // 允许的请求方法
            .allowedHeaders("*") // 允许所有请求头
            .allowCredentials(true) // 允许携带 Cookie
            .maxAge(3600); // 预检请求缓存时间（秒）
    }
}

```

重新启动Spring Boot应用



解析json数据, 动态生成DOM元素

```
<!DOCTYPE html>
```

```

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <!-- ! + 回车 生成html代码的骨架 -->
  <table id = "deptTable">
    <tr>
      <td>部门的编号</td>
      <td>部门的名称</td>
      <td>部门的位置</td>
    </tr>

    <!-- <tr>
      <td>10</td>
      <td>研发部</td>
      <td>L101</td>
    </tr>
    <tr>
      <td>20</td>
      <td>测试部</td>
      <td>L102</td>
    </tr> -->
  </table>
  <!-- 导入jquery库 -->
  <script src="js/jquery-1.8.3.min.js"></script>
  <script>
    //加载所有的部门数据
    function loadDepts(){
      //远程调用服务器
      //$.ajax(); 调用
      //$.ajax({传参, 参数也是json格式});
      $.ajax({
        url : "http://127.0.0.1:8888/boot/depts",
        type : "GET",
        dataType:"json",
        success:function(depts){
          console.log(depts);
          for(let i = 0; i < depts.length; i++){
            let dept = depts[i];
            console.log(`部门编号${dept.deptNo}, 部门名称:${dept.dname},
            部门的位置:${dept.loc}`);
            let trHtml = `<tr><td>${dept.deptNo}</td>
            <td>${dept.dname}</td><td>${dept.loc}</td></tr>`;
            //$("#deptTable")获取id为deptTable的元素对象
            $("#deptTable").append(trHtml);

          }
        }
      });
    }
    loadDepts();
  </script>

```



```
</script>
</body>
</html>
```

八、MySQL

8.1 安装

MySQL5.X

MySQL8.x 【主流版本】

视频

8.2 使用

```
C:\Users\Administrator>mysql -uroot -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.27 MySQL Community Server - GPL

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _
```

查看数据库

```
show databases;
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| jd_db      |
| mysql      |
| performance_schema |
| sys        |
+-----+
5 rows in set (0.02 sec)

mysql>
```

系统库，不要动

创建数据库

```
create database 数据库名称 default charset utf8;
```

```
mysql> create database db_ai default charset utf8;
Query OK, 1 row affected, 1 warning (0.17 sec)

mysql> show databases;
+-----+
| Database |
+-----+
| db_ai    |
| information_schema |
| jd_db    |
| mysql    |
| performance_schema |
| sys      |
+-----+
6 rows in set (0.00 sec)

mysql>
```

使用数据库

```
use 数据库名称;
```

```
mysql> use db_ai
Database changed
mysql>
```

创建表

```
create table 表名(
    字段名1  字段类型1,
    字段名2  字段类型2,
    .....
    字段名N  字段类型N
)engine=InnoDB;
```

其中常用的字段类型：

int：整数

double：小数

char:字符串

varchar:不定长字符串

date:日期 年月日

datetime:日期 年月日时分秒

主键自增长

```
如: id int primary key auto_increment
```

```
create table dept(
    deptno int primary key,
    dname  varchar(20),
    loc    varchar(20)
)engine=InnoDB;
```

```
mysql> use db_ai
Database changed
mysql> create table dept(
  -> deptno int primary key,
  ->   dname varchar(20),
  ->   loc   varchar(20)
  -> )engine=InnoDB;
Query OK, 0 rows affected (0.26 sec)

mysql> _
```

查看表结构

`desc` 表名;

```
mysql> desc dept;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| deptno | int           | NO   | PRI | NULL    |       |
| dname  | varchar(20)   | YES  |     | NULL    |       |
| loc    | varchar(20)   | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> _
```

8.3 Spring Boot整合MyBatis

MyBatis是什么？

MyBatis 是一款优秀的持久层框架，它支持自定义 SQL、存储过程以及高级映射。MyBatis 免除了几乎所有的 JDBC 代码以及设置参数和获取结果集的工作。MyBatis 可以通过简单的 XML 或注解来配置和映射原始类型、接口和 Java POJO（Plain Old Java Objects，普通老式 Java 对象）为数据库中的记录。

整合步骤:

1. 导入组件包

MyBatis和Spring Boot整合的包

MySQL驱动包

```
<dependency>
  <groupId>org.mybatis.spring.boot</groupId>
  <artifactId>mybatis-spring-boot-starter</artifactId>
  <version>3.0.4</version>
</dependency>
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.1.0</version>
</dependency>
```

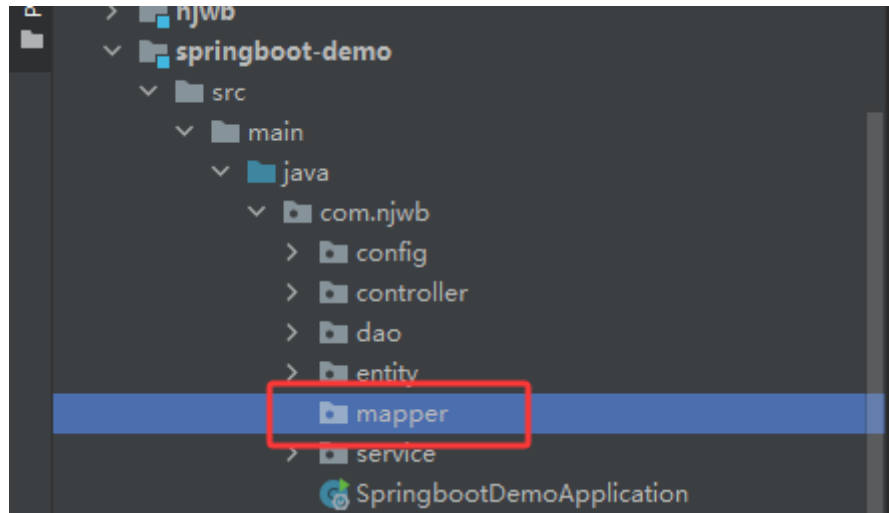
2. 配置

application.yml

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://127.0.0.1:3306/db_ai?
    useUnicode=true&characterEncoding=utf8&useSSL=false&serverTimezone=Asia/Shanghai
    username: root
    password: root
  mybatis:
    type-aliases-package: com.njwb.entity
```

3. 代码

新建mapper包【和dao包作用一模一样，只是在MyBatis框架中，习惯使用mapper，更能体现映射】



新建一个接口，DeptMapper 【相当于DeptDao】，专门处理数据的CRUD

```
import org.apache.ibatis.annotations.Mapper;

@Mapper//Spring 框架会帮助创建一个DeptMapper的代理对象
//代理对象会自动调用MySQL数据库，完成CRUD操作
public interface DeptMapper {
}
```

做什么操作，就在接口中定义相应的方法

这里是添加部门

```

@Mapper//Spring 框架会帮助创建一个DeptMapper的代理对象
    //代理对象会自动调用MySQL数据库，完成CRUD操作
public interface DeptMapper {

    //参数的占位符  #{ }
    //如果参数是实体对象，则必须写成#{实体的属性名}
    @Insert("insert into dept(deptno,dname,loc) values(#{deptNo},#{dname},#{loc})")
    void insert(Dept dept);
}

```

定义业务接口DeptServ

```

public interface DeptServ {
    void add(Dept dept); //添加部门
}

```

业务接口实现类DeptServImpl

```

@Service
@Transactional//事务 要么全部成功，要么全部失败【自动回滚】
public class DeptServImpl implements DeptServ {
    @Autowired
    private DeptMapper deptMapper;
    @Override
    public void add(Dept dept) {
        deptMapper.insert(dept);
    }
}

```

4. 测试

```

@SpringBootTest
class DeptServTests {
    //希望Spring 的容器将相关的对象注入进来
    @Autowired
    private DeptServ deptServ;
    @Test
    void add() {
        Dept dept = new Dept();
        dept.setDeptNo(99);
        dept.setDname("rocket");
        dept.setLoc("L99");
        deptServ.add(dept);
    }
}

```

5. 再实现查询所有部门的操作

DeptMapper接口中新增方法

```
//框架会自动将一行 deptno,dname,loc数据
//装载到Dept对象中  deptno ---deptNo属性上
//                      dname ---dname属性上
//要求：实体的属性名必须和表的列名一模一样【忽略大小写】
//暂时不考虑不一样的情况
@Select("select deptno,dname,loc from dept")
List<Dept> selectList();
```

DeptServ接口中新增方法

```
public interface DeptServ {
    void add(Dept dept);//添加部门

    List<Dept> selectList();
}
```

DeptServImpl实现新增方法

```
@Service
@Transactional//事务 要么全部成功，要么全部失败【自动回滚】
public class DeptServImpl implements DeptServ {
    @Autowired
    private DeptMapper deptMapper;
    @Override
    public void add(Dept dept) {
        deptMapper.insert(dept);
    }

    @Override
    public List<Dept> selectList() {
        return deptMapper.selectList();
    }
}
```

测试

```
@Test
void selectList() {
    deptServ.selectList().forEach(dept -> System.out.println(dept));
}
```

6. 重构DeptController

查询所有

```

@Autowired
private DeptServ deptServ;

@GetMapping("/depts")
public List<Dept> findAll(){
    return deptServ.selectList();
}

```

7. 测试前后端

先启动Spring Boot应用

再访问dept.html

作业：重构DeptController,实现剩余的接口功能，数据使用数据库中真实数据，客户端可以直接采用Postman测试。

DeptMapper

```

@Mapper//Spring 框架会帮助创建一个DeptMapper的代理对象
//代理对象会自动调用MySQL数据库，完成CRUD操作
public interface DeptMapper {

    //参数的占位符  #{ }
    //如果参数是实体对象，则必须写成#{实体的属性名}
    @Insert("insert into dept(deptno,dname,loc) values(#{deptNo},#{dname},#{loc})")
    void insert(Dept dept);

    //框架会自动将一行 deptno,dname,loc数据
    //装载到Dept对象中  deptno ---deptNo属性上
    // dname ---dname属性上
    //要求：实体的属性名必须和表的列名一模一样【忽略大小写】
    //暂时不考虑不一样的情况
    @Select("select deptno,dname,loc from dept")
    List<Dept> selectList();

    //当参数是简单类型时【单个值对象，如整数、字符串】，占位符可以随便写  #{a}  #{xyz}
    //建议写的有意义
    @Select("select deptno,dname,loc from dept where deptno = #{deptNo}")
    Dept getByNo(int deptNo);

    @Update("update dept set dname = #{dname} ,loc = #{loc} where deptno = #{deptNo}")
    void update(Dept dept);

    @Delete("delete from dept where deptno = #{deptNo}")
    void delByNo(int deptNo);
}

```

DeptServ

```
public interface DeptServ {

    //持久层的本质工作：CRUD，所以再Mapper层定义的方法更加取向于增删改查的字样
    // insert delete update select selectList selectOne
    //业务层本质工作：功能性 命名趋向于功能
    void add(Dept dept);//添加部门

    List<Dept> selectList();

    Dept findByNo(int deptNo);

    void modify(Dept dept);

    void delete(int deptNo);
}
```

DeptServImpl

```
@Service
@Transactional//事务 要么全部成功，要么全部失败【自动回滚】
public class DeptServImpl implements DeptServ {
    @Autowired
    private DeptMapper deptMapper;
    @Override
    public void add(Dept dept) {
        deptMapper.insert(dept);
    }

    @Override
    public List<Dept> selectList() {
        return deptMapper.selectList();
    }

    @Override
    public Dept findByNo(int deptNo) {
        return deptMapper.getByNo(deptNo);
    }

    @Override
    public void modify(Dept dept) {
        deptMapper.update(dept);
    }

    @Override
    public void delete(int deptNo) {
        deptMapper.delByNo(deptNo);
    }
}
```

DeptController


```

@RestController//支持Restful风格
public class DeptController {

    //查询所有的部门信息
    @Autowired
    private DeptServ deptServ;

    @GetMapping("/depts")
    public List<Dept> findAll(){
        return deptServ.selectList();
    }

    //根据部门编号查询部门信息
    @GetMapping("/depts/{deptNo}")// {占位符} 表示动态参数，需要客户端传递参数过来
    public Dept findByNo(@PathVariable("deptNo") Integer no){//接收参数，来自于占位
符中的
        return deptServ.findByNo(no);
    }
    //新增部门
    @PostMapping("/depts")
    public String add(Dept dept){//Dept接收客户端的表单数据
        deptServ.add(dept);
        return "SUCCESS";
    }
    //修改部门
    @PutMapping("/depts")
    public String modify(Dept dept){//Dept接收客户端的表单数据
        deptServ.modify(dept);
        return "SUCCESS";
    }

    //删除部门
    @DeleteMapping("/depts/{deptNo}")
    public String del(@PathVariable("deptNo") Integer no){
        deptServ.delete(no);
        return "SUCCESS";
    }

}

```

测试

九、Http客户端

Http客户端多种多样，各有使用场景。

1. Postman，纯面向数据，替代浏览器，简单易用，便于测试。

2. 浏览器，真实场景，界面显示友好，需要写HTML/CSS/JS，适合功能性测试。

3. 程序客户端，封装了底层通信的细节，提供易用的API接口

如：HttpClient、RestTemplate、OKHttpClient等

适合客户端程序中访问基于Http请求协议的其他服务端程序。

这里主要介绍OkHttpClient

9.1 OKHttpClient概述

OkHttpClient 一个广泛使用的**HTTP客户端库**，来进行网络请求操作。OKHttp是一个高效的HTTP客户端库，支持HTTP/2、连接池、透明压缩和其他高级功能，使得它在Android开发以及Java后端开发中被广泛采用。

OKHttp是Square公司开发的一个开源项目，主要用于处理HTTP请求和响应。它与标准的Java HttpURLConnection相比，提供了更高效、简单、灵活的API，支持同步和异步请求。

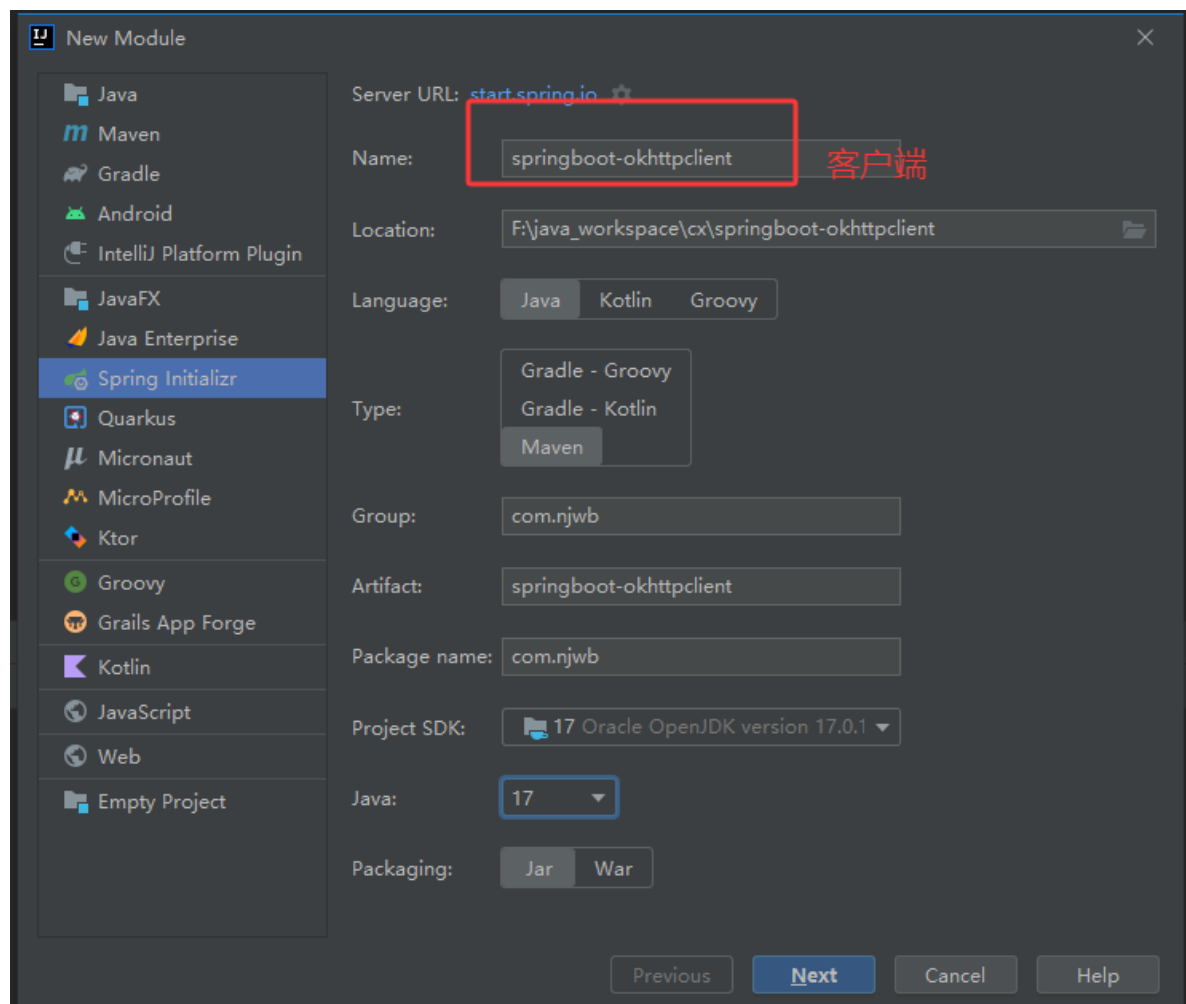
下面介绍OKHttpClient基本的使用

9.2 服务端程序构建

就使用DeptController

9.2 客户端构建

创建一个Spring Boot客户端应用



1、导入OKClient依赖

```
<dependency>
  <groupId>com.squareup.okhttp3</groupId>
  <artifactId>okhttp</artifactId>
  <version>4.11.0</version>
</dependency>
```

2、使用OkClient步骤

- 创建OkHttpClient对象

```
OkHttpClient client = new OkHttpClient()
```

- 构建请求，默认是GET请求

```
Request request = new Request.Builder().url(url).build();
```

或者

```
Request request = new Request.Builder().url(url).post(formBody).build();
```

或者

```
Request request = new Request.Builder().url(url).put(formBody).build();
```

或者

```
Request request = new Request.Builder().url(url).delete().build();
```

- 调用

同步调用

```
Response response = client.newCall(request).execute();
```

异步调用

```
client.newCall(request).enqueue(new Callback(){});
```

9.2.1 GET请求

```
@Test
void getOne() throws Exception { //测试查询所有
    //1. 创建OkHttpClient对象
    OkHttpClient client = new OkHttpClient();
    //2. 构建get请求
    String url = "http://127.0.0.1:8888/boot/depts/99";
    Request request = new Request.Builder().url(url).build();
    //3. 远程调用
    Response response = client.newCall(request).execute();
    //4. 输出响应结果
    //rest api响应 要么就纯字符串，要么就是json格式的字符串
    if(response.isSuccessful()){
        System.out.println(response.body().string());
    }
}
```

```
@Test
void getList() throws Exception { //测试查询所有
```

```

//1.创建OkHttpClient对象
OkHttpClient client = new OkHttpClient();
//2.构建get请求
String url = "http://127.0.0.1:8888/boot/depts";
Request request = new Request.Builder().url(url).build();
//3.远程调用
Response response = client.newCall(request).execute();
//4.输出响应结果
//rest api响应 要么就纯字符串，要么就是json格式的字符串
if(response.isSuccessful()){
    System.out.println(response.body().string());
}
}

```

9.2.2 POST请求

```

@Test
public void add()throws Exception{
    OkHttpClient client = new OkHttpClient();
    String url = "http://127.0.0.1:8080/dept";

    FormBody formBody = new FormBody.Builder()
        .add("deptNo", "30")
        .add("dname", "架构组")
        .build();

    Request request = new Request.Builder().url(url).post(formBody).build();
    Response response = client.newCall(request).execute();
    if(response.isSuccessful()){
        System.out.println(response.body().string());
    }
}

```

9.2.3 PUT请求

```

@Test
public void modify()throws Exception{
    OkHttpClient client = new OkHttpClient();
    String url = "http://127.0.0.1:8080/dept";

    FormBody formBody = new FormBody.Builder()
        .add("deptNo", "30")
        .add("dname", "架构中心")
        .build();

    Request request = new Request.Builder().url(url).put(formBody).build();
    Response response = client.newCall(request).execute();
    if(response.isSuccessful()){
        System.out.println(response.body().string());
    }
}

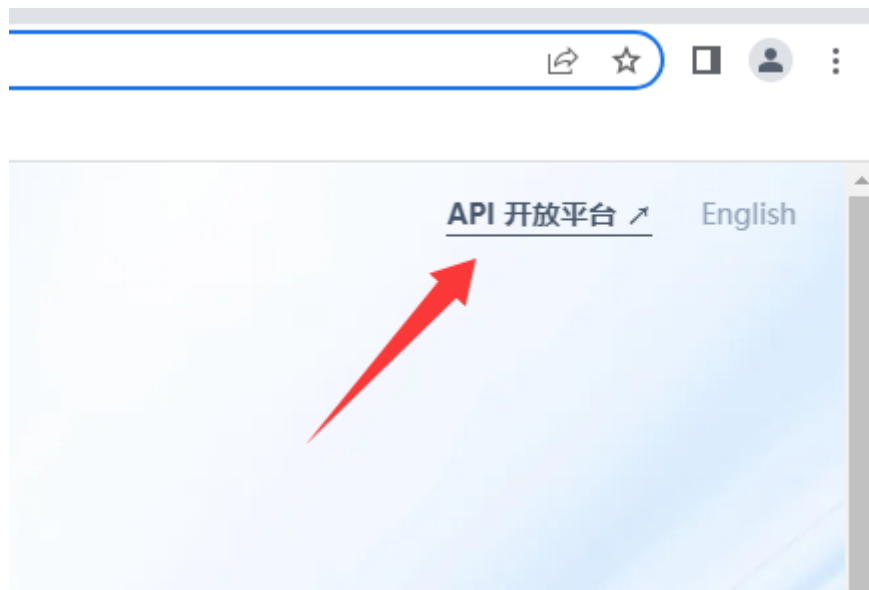
```

9.2.4 DELETE请求

```
@Test
public void deleteByNo() throws Exception{
    OkHttpClient client = new OkHttpClient();
    String url = "http://127.0.0.1:8080/dept/30";
    Request request = new Request.Builder()
        .url(url)
        .delete()
        .build();
    Response response = client.newCall(request).execute();
    System.out.println(response.body().string());
}
```

十、DeepSeek

DeepSeek官网: <https://www.deepseek.com/>



做两件事情:

1. 充值 就1元
2. 申请key, 保存

10.1 DeepSeek API

10.1.1 介绍

对话补全接口: 根据输入的上下文, 来让模型补全对话内容。

base_url : <https://api.deepseek.com>

api_key：使用者自行注册

出于与 OpenAI 兼容考虑，也可以将 `base_url` 设置为 `https://api.deepseek.com/v1` 来使用，但注意，此处 `v1` 与模型版本无关。

`deepseek-chat` 模型已全面升级为 DeepSeek-V3，接口不变。通过指定 `model='deepseek-chat'` 即可调用 DeepSeek-V3

`deepseek-reasoner` 是 DeepSeek 最新推出的推理模型 DeepSeek-R1。通过指定 `model='deepseek-reasoner'`，即可调用 DeepSeek-R1。

请求方式：POST


请求地址：<https://api.deepseek.com/v1/chat/completions>

MediaType: "application/json"

请求头：

- Content-Type：application/json
- Accept：application/json
- Authorization：Bearer
TOKEN为apiKey

请求体：

 > [API 文档](#) > [对话 \(Chat\)](#) > [对话补全](#)

对话补全

POST `https://api.deepseek.com/chat/completions`

根据输入的上下文，来让模型补全对话内容。

Request

APPLICATION/JSON

关注点：请求体的构造

Response响应文本：

Responses

200 (No streaming)

200 (Streaming)

OK, 返回一个 `chat completion` 对象。

APPLICATION/JSON

Schema

Example (from schema)

Example

SCHEMA

<code>id</code> string	REQUIRED
该对话的唯一标识符。	
<code>> choices</code> object[]	REQUIRED
<code>created</code> integer	REQUIRED
创建聊天完成时的 Unix 时间戳（以秒为单位）。	
<code>model</code> string	REQUIRED
生成该 completion 的模型名。	
<code>system_fingerprint</code> string	REQUIRED
This fingerprint represents the backend configuration that the model runs with.	
<code>object</code> string	REQUIRED
Possible values: [<code>chat.completion</code>]	
对象的类型, 其值为 <code>chat.completion</code> 。	
<code>> usage</code> object	

关注点：响应结果需要从json数据中将choice解析出来。

10.1.2 OKHttpClient接入

1、每次请求都需要使用OKHttpClient，采用单例模式封装OKHttpClient对象，便于复用。

```
public class OkHttpClientUtil {
    private static final OkHttpClient OK_HTTP_CLIENT;
    static {
        OK_HTTP_CLIENT = new OkHttpClient();
    }
    public static OkHttpClient create(){
        return OK_HTTP_CLIENT;
    }
}
```

2、准备解析工具，待请求结束后能正常解析出响应结果。

```
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10.1</version>
</dependency>
```

```

public class DeepSeekResponseParser {
    private static final Gson gson = new Gson();

    public static String parseResponse(String jsonResponse) {
        JsonObject jsonObject =
        JsonParser.parseString(jsonResponse).getAsJsonObject();
        // 获取choices数组的第一个元素
        JsonObject firstChoice =
        jsonObject.getAsJsonArray("choices").get(0).getAsJsonObject();
        JsonObject message = firstChoice.getAsJsonObject("message");

        return message.get("content").getString();
    }
}

```

测试数据:

```

{"id":
"2c87498d-8a2c-4b46-bf5b-e0f5c58d0a3a",
"object":"chat.completion",
"created":1748336370,
"model":"deepseek-chat",
"choices":[
    {"index":0,"message":{"role":"assistant",
        "content":"我是DeepSeek Chat，由深度求索公司创造的智能AI助手！👉 我的使命
是帮助你解答问题、提供信息、陪你聊天，甚至帮你处理各种文本和文件。无论是学习、工作，还是日常生活
中的疑问，都可以问我哦！😊 \n\n有什么我可以帮你的吗？"},
        "logprobs":null,"finish_reason":"stop"}],
"usage":{"prompt_tokens":4,"completion_tokens":62,"total_tokens":66,
"prompt_tokens_details":
{"cached_tokens":0,"prompt_cache_hit_tokens":0,"prompt_cache_miss_tokens":4},"s
ystem_fingerprint":"fp_8802369eaa_prod0425fp8"}

```

3、接入

```

public class DeepSeekService {
    private String baseUrl = "https://api.deepseek.com/v1";
    private String apiKey = "sk-efa43b3367964c6395ae15ec7d7414ee";
    public String chatWithDeepSeek(String userMessage){
        MediaType mediaType = MediaType.parse("application/json; charset=utf-
8");
        //构造请求体
        String requestBody = "{\n" +
            "  \"model\": \"deepseek-chat\",\n" +
            "  \"messages\": [\n" +
            "    {\n" +
            "      \"role\": \"user\", \"content\": \"" + userMessage +
            "\"\n" +
            "    },\n" +
            "    {\n" +
            "      \"role\": \"assistant\", \"content\": \"\n" +
            "        \"temperature\": 0.7,\n" +
            "        \"max_tokens\": 2048\n" +
            "      }\n" +
            "    ]\n" +
            "  }";
        RequestBody body = RequestBody.create(requestBody.getBytes(),mediaType);
    }
}

```



```

        Request request = new Request.Builder().url(baseUrl +
"/chat/completions")
        .post(body)
        .addHeader("Content-Type","application/json ")
        .addHeader("Accept","application/json")
        .addHeader("Authorization","Bearer " + apiKey).build();
        Response response = null;
        try {
            response = OkHttpClientUtil.create().newCall(request).execute();

            return
DeepSeekResponseParser.parseResponse(response.body().string());
        } catch (Exception e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

10.2 LangChain4j

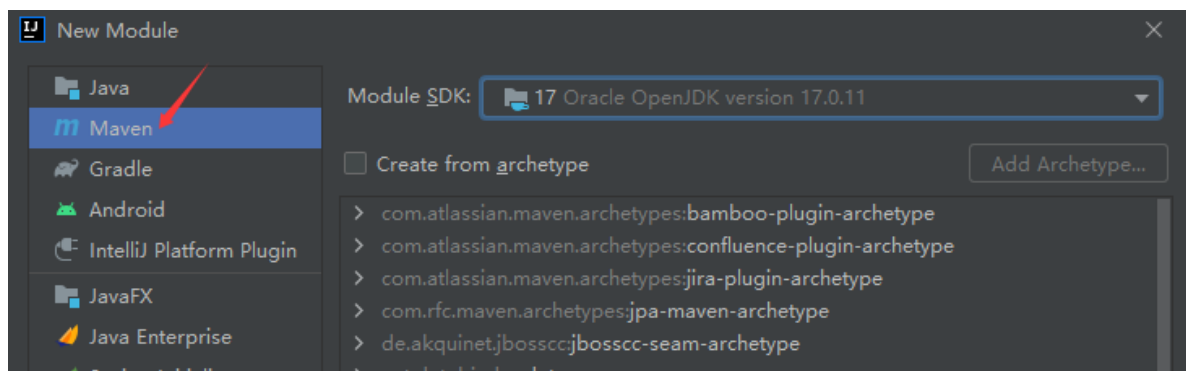
10.2.1 介绍


中文官网: <https://docs.langchain4j.info/>



10.2.2 初识LangChain4j

1. 创建普通的maven工程 langchain4j-demo



 New Module
 ✕

Parent: <None>

Name:

Location:

▼ Artifact Coordinates

GroupId:

The name of the artifact group, usually a company domain

ArtifactId:

The name of the artifact within the group, usually a module name

Version:

2. 导入依赖

```

<dependencies>
    <!--
        接入open-ai的包
        未来如果采用LangChain4j接入其他的大模型，可能会导入langchain4j-其他模型的
        jar包
    -->
    <dependency>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j-open-ai</artifactId>
        <version>1.0.0-beta3</version>
    </dependency>
    <!--    langchain4j的核心依赖包    -->
    <dependency>
        <groupId>dev.langchain4j</groupId>
        <artifactId>langchain4j</artifactId>
        <version>1.0.0-beta3</version>
    </dependency>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
    
```

3. 接入Open AI

```

public class TestOpenAI {
    @Test
    public void chatwithOpenAI(){
        //创建ChatModel
        String baseUrl = "http://langchain4j.dev/demo/openai/v1";
        ChatLanguageModel model = OpenAiChatModel.builder()
            .baseUrl(baseUrl)
    
```

```

        .apiKey("demo")
        .modelName("gpt-4o-mini").build();

    //开始聊天
    String answer = model.chat("你好，你是谁?");
    System.out.println(answer);
}
}

```

4. 运行

```

✓ Tests passed: 1 of 1 test - 8 sec 125 ms
E:\install\jdk17\bin\java.exe ...
SLF4J(W): No SLF4J providers were found.
SLF4J(W): Defaulting to no-operation (NOP) logger implementation
SLF4J(W): See https://www.slf4j.org/codes.html#noProviders for further details.
你好！我是一个人工智能助手，旨在回答问题和提供信息。有什么我可以帮助你的吗？
Process finished with exit code 0

```

10.2.3 接入DeepSeek

LangChain4j接入DeepSeek的思路：和接入Open AI是一样的【底层基于同样的标准】

只需要变：

baseUrl

apiKey

modelName

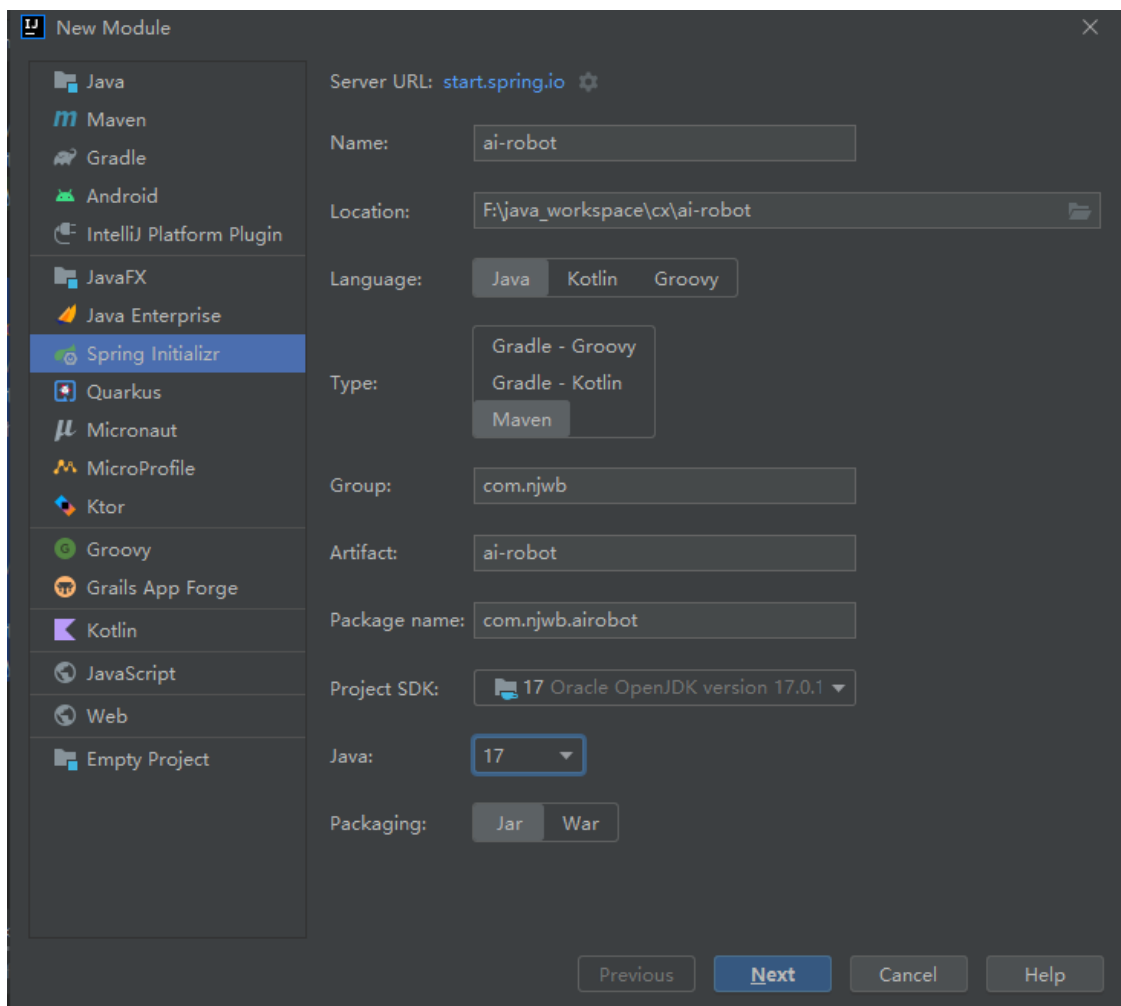
```

@Test
public void chatwithDeepSeek(){
    //创建ChatModel
    String baseUrl = "https://api.deepseek.com";
    ChatLanguageModel model = OpenAiChatModel.builder()
        .baseUrl(baseUrl)
        .apiKey("sk-710dc0de2d5044e9bc3b24863e428792")
        .modelName("deepseek-chat").build();
    //开始聊天
    // String answer = model.chat("你好，你是谁?");
    String answer = model.chat("中国四大淡水湖");
    System.out.println(answer);
}

```

10.2.4 整合Spring Boot

1. 创建Spring Boot应用



添加web服务

Spring Boot版本选择3.4.6

2. 编辑application.yml文件

```
server:
  port: 9001
  servlet:
    context-path: /ai
```

3. 配置跨域

```

@Configuration
public class CorsConfig implements WebMvcConfigurer {
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**") // 匹配所有路径
            .allowedOriginPatterns("*") // 允许所有域名（或指定如
"http://localhost:3000"）
            .allowedMethods("GET", "POST", "PUT", "DELETE") // 允许
的请求方法

            .allowedHeaders("*") // 允许所有请求头
            .allowCredentials(true) // 允许携带 Cookie
            .maxAge(3600); // 预检请求缓存时间（秒）
    }
}

```

3. Spring Boot整合LangChain4j

pom.xml 导入组件

```

<dependency>
    <groupId>dev.langchain4j</groupId>
    <artifactId>langchain4j-open-ai-spring-boot-starter</artifactId>
    <version>1.0.0-beta3</version>
</dependency>

```

定义控制器ChatController:

```

@RestController
public class ChatController {
    @Autowired
    private ChatLanguageModel model;
    @GetMapping("/chat/{message}")
    public String chat(@PathVariable("message") String message){
        return model.chat(message);
    }
}

```

测试:

```

robot.AiRobotApplication : Starting AiRobotApplication using Java 17.0.11 with PID 16908 (F:\java_workspac
robot.AiRobotApplication : No active profile set, falling back to 1 default profile: "default"
aded.tomcat.TomcatWebServer : Tomcat initialized with port 9001 (http)
alina.core.StandardService : Starting service [Tomcat]
alina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.41]
Tomcat].[localhost].[/ai] : Initializing Spring embedded WebApplicationContext
etWebServerApplicationContext : Root WebApplicationContext: initialization completed in 813 ms
aded.tomcat.TomcatWebServer : Tomcat started on port 9001 (http) with context path '/ai'
robot.AiRobotApplication : Started AiRobotApplication in 1.906 seconds (process running for 2.963)

```

使用Postman测试

http://127.0.0.1:9001/ai/chat/你是谁

GEThttp://127.0.0.1:9001/ai/chat/你是谁

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQL

Key	Value
<input checked="" type="checkbox"/> deptNo	300
<input checked="" type="checkbox"/> dname	chating组
<input checked="" type="checkbox"/> loc	L300室
Key	Value

BodyCookiesHeaders (8)Test Results

PrettyRawPreviewVisualizeText

1我是DeepSeek Chat，由深度求索公司（DeepSeek）研发的智能AI助手！🤖👋

2

3我可以帮你解答各种问题，比如学习、工作、编程、生活小技巧等等。无论是写作灵感、数据分析，还是日常闲聊，我都会尽力帮助你！😊

4

5有什么我可以帮你的吗？

http://127.0.0.1:9001/ai/chat/江苏十三太保

GEThttp://127.0.0.1:9001/ai/chat/江苏十三太保

ParamsAuthorizationHeaders (8)BodyPre-request ScriptTestsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQL

Key	Value	Description
<input checked="" type="checkbox"/> deptNo	300	
<input checked="" type="checkbox"/> dname	chating组	
<input checked="" type="checkbox"/> loc	L300室	
Key	Value	Description

BodyCookiesHeaders (8)Test Results

PrettyRawPreviewVisualizeText

1“江苏十三太保”是网络流行语，指江苏省下辖的13个地级市。由于这些城市经济实力较强、发展较为均衡，且在某些领域各具特色，因此被网友戏称为“十三太保”，带有调侃和自嘲的意味。以下是具体介绍：

2

3---

4

5### **由来与背景**

61. **名称来源**：

7- “太保”原为古代官职或武侠小说中的角色，网友借用此词形容江苏13个地级市“各自为战”、实力强劲的特点。

8- 2020年新冠疫情初期，江苏省医疗队伍驰援湖北时以“市”为单位独立行动，网友调侃“散装江苏”，进一步强化了“十三太保”的称呼。

9

102. **经济基础**：

11- 江苏所有地级市均位列全国GDP百强（截至2023年），且区域发展相对均衡，苏南、苏中、苏北均有亮点。

12

13---

14

15### **十三市概况（按2023年GDP排序）**

161. **苏州**：经济总量长期居江苏第一、全国前列，工业强市（电子、纺织、制造业），下辖昆山、张家港等全国百强县。

172. **南京**：省会，科教文化中心，高校云集（如南大、东南大学），主打软件、生物医药等产业。

183. **无锡**：人均GDP领先，物联网、集成电路产业突出，民营经济活跃。

194. **南通**：“长三角北翼经济中心”，建筑业、船舶制造业发达，毗邻上海。

205. **常州**：装备制造（如轨道交通）、新能源产业（光伏、动力电池）优势明显。

216. **徐州**：苏北中心城市，淮海经济区核心，工程机械（徐工集团）、交通枢纽。

227. **扬州**：历史文化名城，旅游、服务业、制造业发达。

238. **镇江**：长江重要港口，化工、医药产业突出。

249. **泰州**：医药、纺织、船舶产业发达。

2510. **盐城**：沿海经济中心城市，新能源（光伏、风电）产业崛起。

2611. **连云港**：新亚欧大陆桥东端起点，港口经济、海洋产业突出。

2712. **宿迁**：电商之都（京东物流总部），白酒、纺织服装产业。

2813. **淮安**：运河文化名城，食品、医药产业。

29

10.2.5 声明式AI

导入依赖

```
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j-spring-boot-starter</artifactId>
  <version>1.0.0-beta3</version>
</dependency>
```

定义ai接口

```
@AiService//两层含义 1: 是一个@Service,IOC会自动创建实现类对象
public interface Assistant {

    String chat(String message);//2: AI功能, 会自动调用大模型
}
```

重构ChatController

```
@RestController
public class ChatController {
    @Autowired
    private Assistant assistant;
    @GetMapping("/chat/{message}")
    public String chat(@PathVariable("message")String message){
        return assistant.chat(message);
    }
}
```

测试

http://127.0.0.1:9001/ai/chat/江苏足球

GET http://127.0.0.1:9001/ai/chat/江苏足球

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

☐ none ☐ form-data ☒ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Key	Value	Description
<input checked="" type="checkbox"/> deptNo	300	
<input checked="" type="checkbox"/> dname	chating组	
<input checked="" type="checkbox"/> loc	L300室	
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 1

Pretty Raw Preview Visualize Text

```
1 江苏足球在中国职业足球版图中曾有过辉煌，但也经历了挫折和重组。以下是关于江苏足球的主要信息梳理：
2
3 ---
4 |
5 ### **1. 江苏苏宁时代（2015-2021）**
6 - **俱乐部成立**：前身为1994年成立的江苏迈特，后历经舜天、苏宁等时期。2015年苏宁集团接手，更名为江苏苏宁足球俱乐部。
7 - **高光时刻**：
8   - **2020年中超冠军**：在特殊赛制下击败广州恒大，队史首夺中超冠军。
9   - **亚冠表现**：2016年亚冠小组赛不败出线，2017年晋级16强。
10 - **解散风波**：2021年2月，苏宁集团因财务危机宣布停止运营俱乐部，中超新科冠军被迫退出职业联赛，震惊中国足球。
11
12 ---
13
14 ### **2. 江苏足球重组（2021年后）**
15 - **新俱乐部成立**：
16   - **南京城市**：2019年成立，现征战中甲联赛（次级联赛）。
17   - **苏州东吴**：2015年成立，目前在中甲联赛。
18   - **南通支云**：2016年成立，2023年升入中超，成为江苏足球新代表。
```

10.2.6 流式响应

需要使用到StreamingChatLanguageModel对象，默认并未实例化，需要我们自行配置。

1、导入依赖

```
<dependency>
  <groupId>dev.langchain4j</groupId>
  <artifactId>langchain4j-reactor</artifactId>
  <version>1.0.0-beta1</version>
</dependency>
```

2、定义配置类

```
@Configuration
public class DeepSeekConfig {
    @Value("${langchain4j.open-ai.chat-model.base-url}")
    private String baseUrl;
    @Value("${langchain4j.open-ai.chat-model.api-key}")
    private String apiKey;
    @Value("${langchain4j.open-ai.chat-model.model-name}")
    private String modelName;
    @Bean
    public OpenAiStreamingChatModel openAiStreamingChatModel(){
        return OpenAiStreamingChatModel.builder()
            .baseUrl(baseUrl)
            .apiKey(apiKey)
            .modelName(modelName)
            .build();
    }
}
```


3、编写控制器

```
@RestController
public class FluxChatController {
    @Autowired
    private StreamingChatLanguageModel streamingChatLanguageModel;
    @RequestMapping(value = "/chat/{message}", produces =
"text/stream;charset=UTF-8")
    public Flux<String> chat(@PathVariable("message")String message){
        return Flux.create(emitter->{
            streamingChatLanguageModel.chat(message, new
StreamingChatResponseHandler() {
                @Override
                public void onPartialResponse(String partialResponse) {
                    emitter.next(partialResponse);
                }

                @Override
                public void onCompleteResponse(ChatResponse chatResponse) {
                    emitter.complete();
                }

                @Override
                public void onError(Throwable throwable) {
                    emitter.error(throwable);
                }
            });
        });
    }
}
```

10.2.7 前后端联调测试

前端项目代码见AI-Chat。

AI机器人对话页面：chat.html，核心业务逻辑封装在chat.js中。

chat.js通过ajax远程调用服务端，获取信息，动态写入到对话框中。

十一、实训项目

11.1 业务需求

用户模块：

1. 用户注册

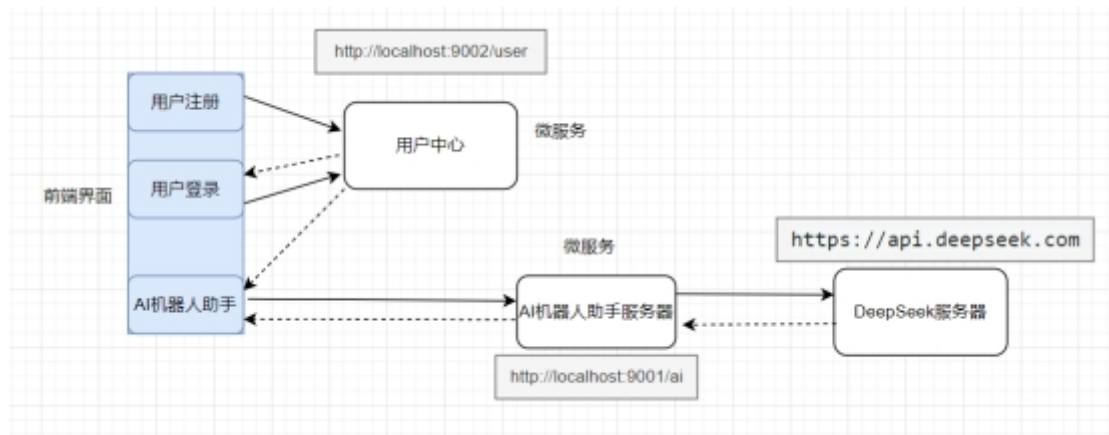
系统提供注册页面，输入手机号码、密码和确认密码，点击注册，请求用户中心，实现用户注册功能。页面需要验证手机号码格式是否正确、密码和确认密码是否一致，如果出错，给出相应提示。一个手机号码只能注册一次，注册成功后，跳转到登录页面。

2. 用户登录

系统提供登录页面，输入手机号码、密码，点击登录，请求用户中心，实现用户登录功能。登录成功后跳转到AI机器人助手界面，登录失败后提示“手机号码不正确”或者“密码不正确”。

AI机器人助手模块：系统提供AI机器人对话补全功能，用户输入问题，系统调用AI机器人助手服务器，采用流式响应补全内容。

11.2 系统流程



11.3 系统架构

1. 前后端分离
2. Spring Boot分布式微服务架构
3. Restful API接口
4. MySQL数据库存储数据
5. LangChain4J Http客户端

任务：

- 1、完成所有的功能 小组为单位
- 2、答辩材料

代码演示 功能演示 优秀代码的分享

PPT：简单 所学 心得 未来的期许 建议

- 3、实训的报告

