

学习路线:

- 全新路线

课程特色:

- ◆ 前后端分离开发，基于接口交互（接口文档）。
- ◆ 前端-基于Vue脚手架，构建工程化的前端项目。
- ◆ 后端-基于主流SpringBoot高效学习SSM。
- ◆ 通过案例贯穿整个课程体系，学以致用。
- ◆ 参照企业开发模式，需求分析 - 表结构设计 - 接口文档 - 功能实现 - 测试。

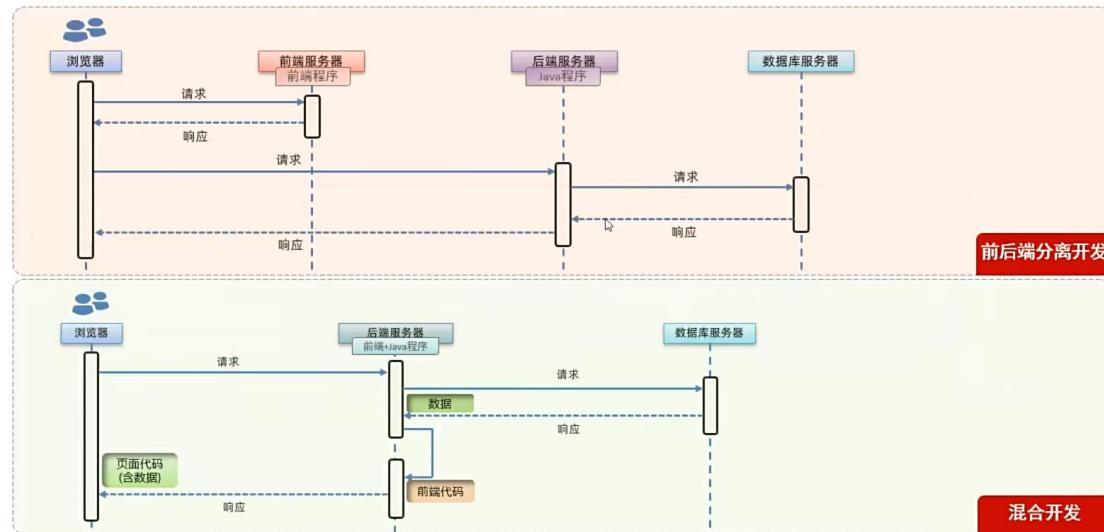


1. Web 开发

Web: 全球广域网，也称为万维网(www World Wide Web)，能够通过浏览器访问的网站。

Web 网站的开发模式：

前后端分离是主流。



2. Web 前端开发

2.1 Web 标准

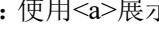
- Web 标准也称为网页标准，由一系列的标准组成，大部分由 W3C(World Wide Web Consortium, 万维网联盟)负责制定。
- 三个组成部分：

HTML: 负责网页的结构（页面元素和内容）。

HTML(HyperText Markup Language):超文本标记语言。

超文本: 超越了文本的限制，比普通文本更强大。除了文字信息，还可以定父图片、音频、视频等内容。

标记语言: 由标签构成的语言

➤ HTML 标签都是预定义好的。例如：使用[展示超链接](#)，使用展示图片，展示视频。

➤ HTML 代码直接在浏览器中运行，HTML 标签由浏览器解析。

CSS: 负责网页的表现（页面元素的外观、位置等页面样式，如：颜色、大小等）。

➤ CSS(Cascading Style Sheet): 层叠样式表，用于控制页面的样式（表现）。

JavaScript: 负责网页的行为（交互效果）。

2.2 HTML&CSS

2.2.1 入门

HTML 结构标签：

```
<html>
  <head>
    <title>标题</title>
  </head>
  <body>
    ...
  </body>
</html>
```

特点：

HTML 标签不区分大小写

HTML 标签属性值单双引号都可以

HTML 语法松散

2.2.2 基础标签&样式

标题

1. 标题标签

标签：`<h1>...</h1>`(h1→h6 重要程度依次降低)

注意：HTML 标签都是预定义好的，不能自己随意定义。

2. 水平线标签 `<hr>`

3. 图片标签 ``

绝对路径：绝对磁盘路径(D:/xxxx)、绝对网络路径(<https://xxxx>)

相对路径：从当前文件开始查找。(./: 当前目录， ../: 上级目录)

4. span:

``是一个在开发网页时大量会用到的没有语义的布局标签

特点：一行可以显示多个（组合行内元素），宽度和高度默认由内容撑开

5. 超链接:

`<a>`

属性：

href:指定资源访问的 url
target:指定在何处打开资源链接

- _self:默认值，在当前页面打开
- _blank:在空白页面打开

css 引入方式:

● CSS引入方式：

- 行内样式：写在标签的style属性中（不推荐）
- 内嵌样式：写在style标签中（可以写在页面任何位置，但通常约定写在head标签中）
- 外联样式：写在一个单独的.css文件中（需要通过 link 标签在网页中引入）

③ `h1 {
 xxx: xxx;
 xxx: xxx;
}`

`<link rel="stylesheet" href="css/news.css">`

② `<style>
h1 {
 xxx: xxx;
 xxx: xxx;
}</style>`

属性名: 属性值;

① `<h1 style="xxx: xxx; xxx: xxx;">中国新闻网</h1>`

颜色表示:

表示方式	表示含义	取值
关键字	预定义的颜色名	red、green、blue...
rgb表示法	红绿蓝三原色，每项取值范围：0-255	rgb(0,0,0)、rgb(255,255,255)、rgb(255,0,0)
十六进制表示法	#开头，将数字转换成十六进制表示	#000000、#ff0000、#cccccc，简写：#000、#ccc

颜色属性:

color: 设置文本内容的颜色

css 选择器：用来选取需要设置样式的元素（标签）

元素选择器	<code>元素名称 { color: red; }</code>	<code>h1 { color: red; }</code>	<code><h1> Hello CSS </h1></code>
id选择器	<code>#id属性值 { color: red; }</code>	<code>#hid { color: red; }</code>	<code><h1 id="hid"> CSS id Selector</h1></code>
类选择器	<code>.class属性值 { color: red; }</code>	<code>.cls { color: red; }</code>	<code><h1 class="cls">CSS class Selector</h1></code>

优先级：id>类>元素

正文

1. 音频、视频标签

`<audio> <video>`

2. 换行、段落标签

换行：`
`； 段落：`<p>`

3. 文本加粗标签

` `

4. CSS样式

`line-height: 设置行高`

`text-indent: 定义第一个行内容的缩进`

`text-align: 规定元素中的文本的水平对齐方式`

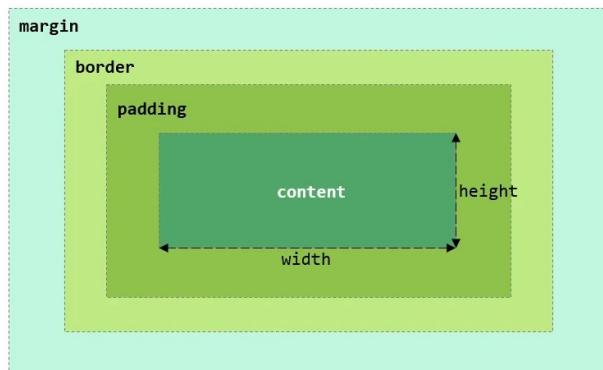
5. 注意

在HTML中无论输入多少个空格，只会显示一个。 可以使用空格占位符：`&nbsp`

页面布局：

盒子： 页面中所有的元素（标签），都可以看做是一个盒子，由盒子将页面中的元素包含在一个矩形区域内，通过盒子的视角更方便的进行页面布局

盒子模型组成： 内容区域(content)、内边距区域(padding)、边框区域(border)、外边距区域(margin)



布局标签：实际开发网页中，会大量频繁的使用 div 和 span 这两个没有语义的布局标签。

标签: <div>

特点:

- **div 标签:**

- 一行只显示一个（独占一行）
- 宽度默认是父元素的宽度，高度默认由内容撑开
- 可以设置宽高(width、height)

- **span 标签:**

- 一行可以显示多个
- 宽度和高度默认由内容撑开
- 不可以设置宽高(width、height)

CSS 属性

width: 设置宽度

height: 设置高度

border: 设置边框的属性，如: 1px solid #000;

padding: 内边距

margin: 外边距

注意：如果只需要设置某一个方位的边框、内边距、外边距，可以在属性名后加上-位置，

如：padding-top、padding-left、padding-right…

表格标签

场景：在网页中以表格(行、列)形式整齐展示数据，如：班级表。

标签	描述	属性/备注
<table>	定义表格整体，可以包裹多个 <tr>	border: 规定表格边框的宽度
		width: 规定表格的宽度
		cellspacing: 规定单元之间的空间。
<tr>	表格的行，可以包裹多个 <td>	
<td>	表格单元格(普通)，可以包裹内容	如果是表头单元格，可以替换为 <th>

表单标签:

<form>

负责数据采集功能：注册 登录等。

表单项标签:

表单项：不同类型的 input 元素、下拉列表、文本域等。

<input>: 定义表单项，通过 type 属性控制输入形式

<select>: 定义下拉列表

<textarea>: 定义文本域

| type取值 | 描述 | 形式 |
|--------------------------|-----------------------|---|
| text | 默认值，定义单行的输入字段 | <input type="text" value="张无忌"/> |
| password | 定义密码字段 | <input type="password"/> |
| radio | 定义单选按钮 | <input type="radio"/> 男 <input type="radio"/> 女 |
| checkbox | 定义复选框 | <input type="checkbox"/> Java <input type="checkbox"/> Game |
| file | 定义文件上传按钮 | <input type="file"/> 选择文件 未选择任何文件 |
| date/time/datetime-local | 定义日期/时间/日期时间 | <input type="date"/> 年 /月/日 <input type="time"/> 年 /月/日 --:-- |
| number | 定义数字输入框 | <input type="number" value="18"/> |
| email | 定义邮件输入框 | <input type="email" value="java@163.com"/> |
| hidden | 定义隐藏域 | |
| submit / reset / button | 定义提交按钮 / 重置按钮 / 可点击按钮 | <input type="submit"/> 提交 <input type="reset"/> 重置 <input type="button"/> |

action: 规定当提交表单时向何处发送表单数据， URL

method: 规定用于发送表单数据的方式。GET（默认）、POST

get: 表单数据拼接在 u 后面，?username=java,大小有限制

post: 表单数据在请求体中携带，大小没有限制

表单项必须要有 name 属性才可以提交。

2.3 JavaScript

JavaScript(简称：JS)是一门跨平台、面向对象的脚本语言。是用来控制网页行为的，它能使网页可交互。

JavaScript 和 Java 是完全不同的语言，不论是概念还是设计。但是基础语法类似。

JavaScript 在 1995 年由 Brendan Eich 发明，并于 1997 年成为 ECMA 标准。

ECMAScript6(ES6)是最新的 JavaScript 版本(发布于 2015 年)。

2.3.1 引入方式

- 内部脚本：将 JS 代码定义在 HTML 页面

JavaScript 代码必须位于<script></script>标签之间

在 HTML 文档中，可以在任意地方，放置任意数量的<script>

一般会把脚本置于<body>元素的底部，可改善显示速度

- 外部脚本：将 JS 代码定义在外部 S 文件中，然后引入到 HTML 页面中

外部 JS 文件中，只包含 JS 代码，不包含<script>标签

<script>标签不要自闭合

2.3.2 基础语法

分号可有可无。

输出语句:

使用 `window.alert()` 写入警告框

使用 `document.write()` 写入 HTML 输出

使用 `console.log()` 写入浏览器控制台

变量:

JavaScript 中用 **var** 关键字(variable 的缩写)来声明变量。

JavaScript 是一门弱类型语言，变量可以存放不同类型的值。

变量名需要遵循如下规则：

组成字符可以是任何字母、数字、下划线(_)或美元符号(\$)

数字不能开头

建议使用驼峰命名

`var` 作用域为全局

ECMAScript6 新增了 `let` 关键字来定义变量。它的用法类似于 `var`，但是所声明的变量，只在 `let` 关键字所在的代码块内有效，且不允许重复声明。

ECMAScript6 新增了 `const` 关键字，用来声明一个只读的常量。一旦声明，常量的值就不能改变。

数据类型:

原始类型和引用类型。`typeof x` 获取类型

① 原始类型

- **number**: 数字 (整数、小数、NaN(Not a Number))
- **string**: 字符串，单双引皆可
- **boolean**: 布尔。true, false
- **null**: 对象为空
- **undefined**: 当声明的变量未初始化时，该变量的默认值是 undefined

运算符:

② 运算符

- 算术运算符: +, -, *, /, %, ++, --
- 赋值运算符: =, +=, -=, *=, /=, %=
- 比较运算符: >, <, >=, <=, !=, ==, ===
- 逻辑运算符: &&, ||, !
- 三元运算符: 条件表达式 ? true_value: false_value

==会进行类型转换， ===不会进行类型转换

类型转换

- 字符串类型转为数字：
 - 将字符串字面值转为数字。如果字面值不是数字，则转为NaN。
- 其他类型转为boolean：
 - Number: 0 和 NaN 为 false，其他均转为 true。
 - String: 空字符串为 false，其他均转为 true。
 - Null 和 undefined：均转为 false。

流程控制语句：

流程控制

- if...else if ...else...
- switch
- for
- while
- do ... while ↗

2.3.3 函数

函数（方法）是被设计为执行特定任务的代码块。

定义：JavaScript 函数通过 function 关键字进行定义，语法为：

```
function functionName(参数 1, 参数 2){  
    //要执行的代码  
}
```

或者：

```
var functionName=function(参数 1, 参数 2){  
    //要执行的代码  
}
```

注意：

形式参数不需要类型。因为 JavaScript 是弱类型语言

返回值也不需要定义类型，可以在函数内部直接使用 return：返回即可

调用：函数名称（实际参数列表），可以传递多个参数，但是会使用前 n 个

2.3.4 对象

● Array

```
var 变量名 = new Array(元素列表); //方式一  
var 变量名 =[元素列表]; //方式二
```

长度可变，类型可变，中间可以跳过。

| 属性 | 描述 |
|---------------------|----------------|
| <code>length</code> | 设置或返回数组中元素的数量。 |

方法

| 方法 | 描述 |
|------------------------|--------------------------|
| <code>forEach()</code> | 遍历数组中的每个有值的元素，并调用一次传入的函数 |
| <code>push()</code> | 将新元素添加到数组的末尾，并返回新的长度。 |
| <code>splice()</code> | 从数组中删除元素。 |

● String

`var 变量名 = new String("...");`/方式一
`var 变量名 = "...";` /方式二

| 属性 | 描述 |
|---------------------|---------|
| <code>length</code> | 字符串的长度。 |

方法：

| 方法 | 描述 |
|--------------------------|----------------------|
| <code>charAt()</code> | 返回在指定位置的字符。 |
| <code>indexOf()</code> | 检索字符串。 |
| <code>trim()</code> | 去除字符串两边的空格 |
| <code>substring()</code> | 提取字符串中两个指定的索引号之间的字符。 |

● JSON

JS 自定义对象：

```
var 对象名={  
    属性名 1: 属性值 1,  
    属性名 2: 属性值 2,  
    属性名 3: 属性值 3,  
    函数名称: function(形参列表) {}  
    //函数名称(形参列表) {}  
};
```

JSON: JavaScript Object Notation, JavaScript 对象标记法。是通过 JavaScript 对象标记法书写的文本。由于其语法简单，层次结构鲜明，现多用于作为数据载体，在网络中进行数据传输。

定义：

- 数字（整数或浮点数）
- 字符串（在双引号中）
- 逻辑值(true 或 false)
- 数组（在方括号中）
- 对象（在花括号中）
- null

JSON 字符串转为 JS 对象

```
var jsobject = JSON.parse(userstr);
```

JS 对象转为 SON 字符串

```
var jsonstr = JSON.stringify(jsobject);
```

● BOM

Browser Object Model 浏览器对象模型，允许 JavaScript 与浏览器对话，JavaScript 将浏览器

的各个组成部分封装为对象。

组成:

- Window:浏览器窗口对象

Window

- 介绍: 浏览器窗口对象。
- 获取: 直接使用window, 其中 window. 可以省略。 `window.alert("Hello Window");` `alert("Hello Window");`
- 属性:
 - `history`: 对 History 对象的只读引用。请参阅 [History 对象](#)。
 - `location`: 用于窗口或框架的 Location 对象。请参阅 [Location 对象](#)。
 - `navigator`: 对 Navigator 对象的只读引用。请参阅 [Navigator 对象](#)。
- 方法
 - `alert()`: 显示带有一段消息和一个确认按钮的警告框。
 - `confirm()`: 显示带有一段消息以及确认按钮和取消按钮的对话框。
 - `setInterval()`: 按照指定的周期 (以毫秒计) 来调用函数或计算表达式。
 - `setTimeout()`: 在指定的毫秒数后调用函数或计算表达式。

- Navigator:浏览器对象

- Screen:屏幕对象

- History:历史记录对象

- Location:地址栏对象

Location

- 介绍: 地址栏对象。
- 获取: 使用 `window.location` 获取, 其中 window. 可以省略。
`window.location属性;` `location属性;`
- 属性:
 - `href`: 设置或返回完整的URL。 `location.href = "https://www.itcast.cn";`

● DOM

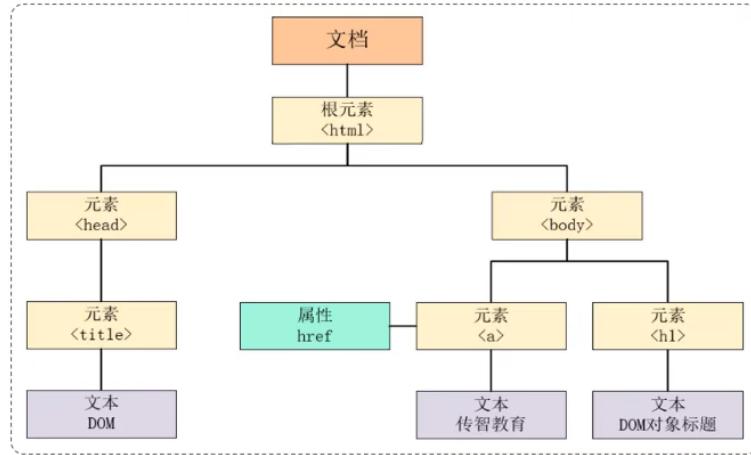
概念: Document Object Model, 文档对象模型。

将标记语言的各个组成部分封装为对应的对象:

- Document:整个文档对象
- Element:元素对象
- Attribute:属性对象
- Text:文本对象
- Comment:注释对象

JavaScript 通过 DOM,就能够对 HTML 进行操作:

- 改变 HTML 元素的内容
- 改变 HTML 元素的样式(CSS)
- 对 HTML DOM 事件作出反应
- 添加和删除 HTML 元素



DOM是W3C（万维网联盟）的标准，定义了访问HTML和XML文档的标准，分为3个不同的部分：

1. Core DOM - 所有文档类型的标准模型

- ◆ Document: 整个文档对象
- ◆ Element: 元素对象
- ◆ Attribute: 属性对象
- ◆ Text: 文本对象
- ◆ Comment: 注释对象

```
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <h1>DOM对象标题</h1>
    <a href="https://itcast.cn">传智教育</a>
  </body>
</html>
```

2. XML DOM - XML 文档的标准模型

3. HTML DOM - HTML 文档的标准模型

- ◆ Image:
- ◆ Button : <input type='button'>

● HTML中的Element对象可以通过Document对象获取，而Document对象是通过window对象获取的。

● Document对象中提供了以下获取Element元素对象的函数：

1. 根据id属性值获取，返回单个Element对象

```
var h1 = document.getElementById('h1');
```

2. 根据标签名称获取，返回Element对象数组

```
var divs = document.getElementsByTagName('div');
```

3. 根据name属性值获取，返回Element对象数组

```
var hobbies = document.getElementsByName('hobby');
```

4. 根据class属性值获取，返回Element对象数组

```
var cls = document.getElementsByClassName('cls');
```

2.3.5 事件监听

事件：发生在HTML元素上的“事情”。比如：

| 事件名 | 说明 |
|-------------|--------------|
| onclick | 鼠标单击事件 |
| onblur | 元素失去焦点 |
| onfocus | 元素获得焦点 |
| onload | 某个页面或图像被完成加载 |
| onsubmit | 当表单提交时触发该事件 |
| onkeydown | 某个键盘的键被按下 |
| onmouseover | 鼠标被移到某元素之上 |
| onmouseout | 鼠标从某元素移开 |

事件监听: JavaScript 可以在事件被侦测到时执行代码,

事件绑定:

方式一：通过 HTML 标签中的事件属性进行绑定

```
<input type="button" onclick="on()" value="按钮1">

<script>
    function on(){
        alert('我被点击了!');
    }
</script>
```

方式二：通过 DOM 元素属性绑定

```
<input type="button" id="btn" value="按钮2">

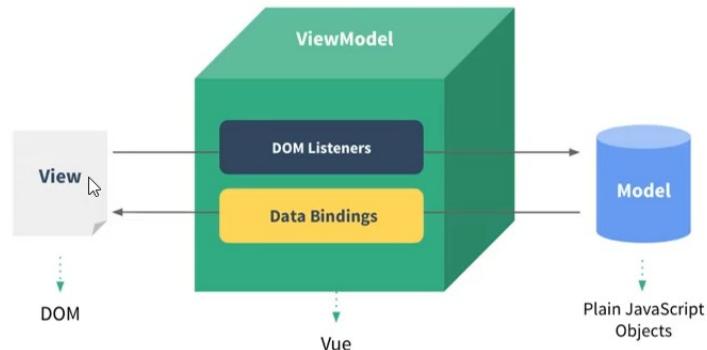
<script>
    document.getElementById('btn').onclick=function(){
        alert('我被点击了!');
    }
</script>
```

2.4 Vue

Vue 是一套**前端框架**, 免除原生 JavaScript 中的 DOM 操作, 简化书写。

基于 MVVM(Model-View-ViewModel)思想, 实现数据的双向绑定, 将编程的关注点放在数据上。

官网: <https://v2.cn.vuejs.org/>



插值表达式:

形式: {{表达式}}。

内容可以是:

- 变量
- 三元运算符
- 函数调用
- 算术运算

2.4.1 Vue 常用指令

指令: HTML 标签上带有 v-前缀的特殊属性, 不同指令具有不同含义。例如: v-if, v-for...

| 指令 | 作用 |
|-----------|---------------------------------|
| v-bind | 为HTML标签绑定属性值, 如设置 href , css样式等 |
| v-model | 在表单元素上创建双向数据绑定 |
| v-on | 为HTML标签绑定事件 |
| v-if | |
| v-else-if | 条件性的渲染某元素, 判定为true时渲染,否则不渲染 |
| v-else | |
| v-show | 根据条件展示某元素, 区别在于切换的是display属性的值 |
| v-for | 列表渲染, 遍历容器的元素或者对象的属性 |

2.4.2 Vue 生命周期

生命周期: 指一个对象从创建到销毁的整个过程。

生命周期的八个阶段: 每触发一个生命周期事件, 会自动执行一个生命周期方法 (钩子), 允许开发者在不同的阶段插入代码。

| 状态 | 阶段周期 |
|---------------|------|
| beforeCreate | 创建前 |
| created | 创建后 |
| beforeMount | 挂载前 |
| mounted | 挂载完成 |
| beforeUpdate | 更新前 |
| updated | 更新后 |
| beforeDestroy | 销毁前 |
| destroyed | 销毁后 |

创建阶段 (Creation)

在这个阶段, Vue 实例被创建, 但还没有被挂载到 DOM 上。此时, 可以在实例中设置初始数据、计算属性等。

挂载阶段 (Mounting)

Vue 实例被挂载到 DOM 上, 这意味着它的模板已经被解析并渲染。

更新阶段 (Updating)

当数据变化时, Vue 会重新渲染视图。这一阶段涉及响应式系统的工作, 触发视图的更新。

销毁阶段 (Unmounting)

Vue 实例被销毁，事件监听器和子实例会被清理。此时，开发者可以执行清理操作，例如取消订阅事件、清除定时器等。

2.5 Ajax

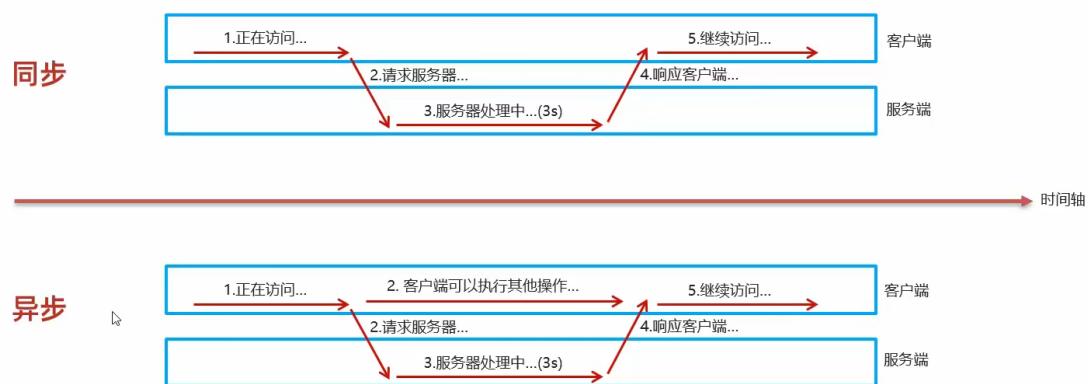
概念：Asynchronous JavaScript And XML, 异步的 JavaScript 和 XML.

作用：

●**数据交换**：通过 Ajax 可以给服务器发送请求，并获取服务器响应的数据。

●**异步交互**：可以在不重新加载整个页面的情况下，与服务器交换数据并更新部分网页的技术，如：搜索联想、用户名是否可用的校验等等。

同步与异步：



2.5.1 Axios

Axios 对原生的 Ajax 进行了封装，简化书写，快速开发。

步骤：

1. 引入 Axios 的 js 文件

```
<script src="js/axios-0.18.0.js"></script>
```

2. 使用 Axios 发送请求，并获取响应结果

```
axios({
  method: "get",
  url: "http://yapi.smart-xwork.cn/mock/169327/emp/list"
}).then((result) => {
  console.log(result.data);
});
```

成功回调函数

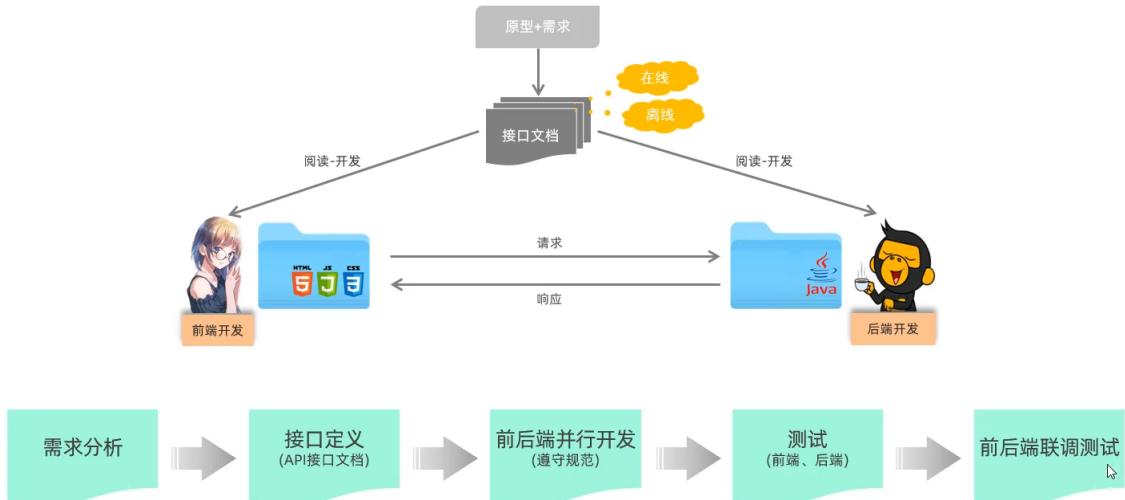
```
axios({
  method: "post",
  url: "http://yapi.smart-xwork.cn/mock/169327/emp/deleteById",
  data: "id=1"
}).then((result) => {
  console.log(result.data);
});
```

成功回调函数

请求方式别名:

```
axios.get(url[, config])  
axios.delete(url[, config])  
axios.post(url[, data[, config]])  
axios.put(url[, data[,config]])
```

2.6 前后端分离开发



YApi

介绍: YApi 是高效、易用、功能强大的 api 管理平台，旨在为开发、产品、测试人员提供更优雅的接口管理服务

地址: <http://yapi.smart-xwork.cn/>

功能:

API 接口管理，Mock 服务。

2.7 前端工程化

2.7.1 环境准备

Vue-cli: Vue-cli 是 Vue 官方提供的一个脚手架，用于快速生成一个 Vue 的项目模板。

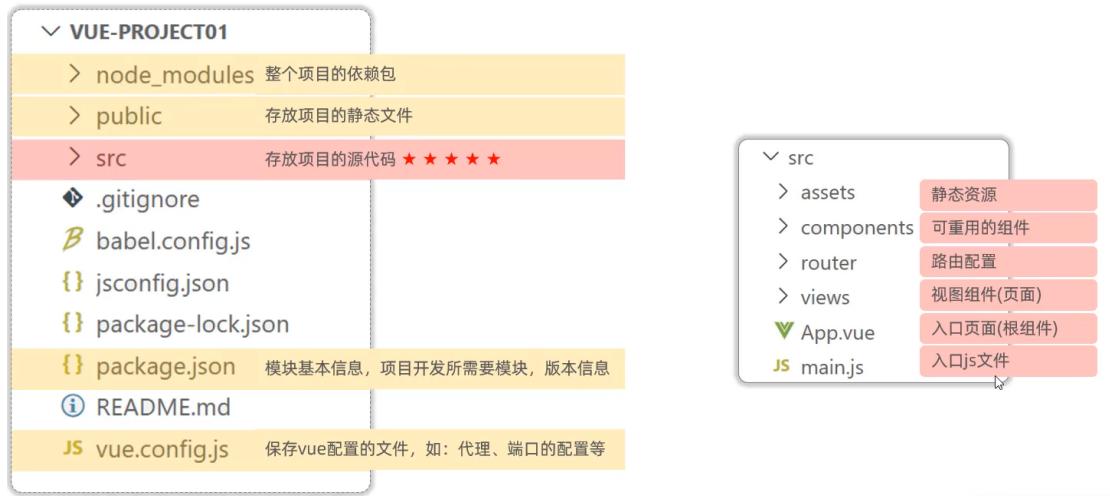
Vue-cli 提供了如下功能:

- 统一的目录结构
- 本地调试
- 热部署
- 单元测试
- 集成打包上线

依赖环境: NodeJS

2.7.2 Vue 项目

目录结构:



2.7.3 开发流程

默认首页: index.html

入口文件: main.js

Vue 的组件文件以.vue 结尾, 每个组件由三个部分组成:

<template>: 模板部分, 由它生成 HTML 代码;

<script>: 控制模板的数据来源和行为;

<style>: css 样式部分。

2.8 Element

Element: 是饿了么团队研发的, 一套为开发者、设计师和产品经理准备的基于 Vue2.0 的桌面端组件库。

组件: 组成网页的部件, 例如超链接、按钮、图片、表格、表单、分页条等等。

官网: <https://element.eleme.cn/#/zh-CNListener>

常见组件:

Table 表格

Pagination 分页

Dialog 对话框

Form 表单: 由输入框、选择器、单选框、多选框等控件组成, 用以收集、校验、提交数据。

3 Web 后端开发

Spring:

官网: spring.io

Spring 发展到今天已经形成了一种开发生态圈，Spring 提供了若干个子项目，每个项目用于完成特定的功能。

Spring Boot 可以帮助我们非常快速的构建应用程序、简化开发提高效率。本质还是 Spring 框架。

3.1 SpringBootWeb

1. 创建 springboot 工程，填写模块信息，并勾选 web 开发相关依赖。
2. 删除 demo。创建请求处理类 HelloController,添加请求处理方法 hello,并添加注解
3. 运行启动类，打开浏览器测试。、

3.2 HTTP 协议

3.2.1 概述

Hyper Text Transfer Protocol,超文本传输协议，规定了浏览器和服务器之间数据传输的规则。



特点:

1. 基于 TCP 协议：面向连接，安全
2. 基于请求-响应模型的：一次请求对应一次响应
3. HTTP 协议是无状态的协议：对于事务处理没有记忆能力。每次请求-响应都是独立的。
缺点：多次请求间不能共享数据。
优点：速度快

3.2.2 请求协议

请求行：请求数据第一行（请求方式、资源路径、协议）

请求头：第二行开始，格式 key: value

HTTP-请求数据格式

```
POST /brand HTTP/1.1
Accept: application/json, text/plain, */*
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Content-Length: 161
Content-Type: application/json; charset=UTF-8
Cookie: Idea-8296eb32=841b16f0-0cfe-495a-9cc9-d5aaa71501a6; JSESSIONID=0FDE4E430876BD9C5C955F061207386F
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/...
```

| | |
|-----------------|---|
| Host | 请求的主机名 |
| User-Agent | 浏览器版本, 例如Chrome浏览器的标识类似Mozilla/5.0 ... Chrome/79, IE浏览器的标识类似Mozilla/5.0 (Windows NT ...) like Gecko |
| Accept | 表示浏览器能接收的资源类型, 如text/*, image/*或者/*表示所有; |
| Accept-Language | 表示浏览器偏好的语言, 服务器可以据此返回不同语言的网页; |
| Accept-Encoding | 表示浏览器可以支持的压缩类型, 例如gzip, deflate等。 |
| Content-Type | 请求主体的数据类型。 |
| Content-Length | 请求主体的大小 (单位: 字节)。 |

请求体: POST 请求, 存放请求参数

请求方式-GET: 请求参数在请求行中, 没有请求体, 如: /brand/findAll?name=oppo&status=1。

GET 请求大小是有限制的。

请求方式-POST: 请求参数在请求体中, POST 请求大小是没有限制的。

3.2.3 响应协议

请求行: 响应数据第一行(协议、状态码、描述)

状态码大类:

| | |
|-----|--|
| 1xx | 响应中-临时状态码, 表示请求已经接收, 告诉客户端应该继续请求或者如果它已经完成则忽略它。 |
| 2xx | 成功-表示请求已经被成功接收, 处理已完成。 |
| 3xx | 重定向-重定向到其他地方; 让客户端再发起一次请求以完成整个处理。 |
| 4xx | 客户端错误-处理发生错误, 责任在客户端。如: 请求了不存在的资源、客户端未被授权、禁止访问等。 |
| 5xx | 服务器错误-处理发生错误, 责任在服务端。如: 程序抛出异常等。 |

| 状态码 | 英文描述 | 解释 |
|-----|--|--|
| 200 | OK | 客户端请求成功, 即 处理成功 , 这是我们最想看到的状态码 |
| 302 | Found | 指示所请求的资源已移动到由 Location 响应头给定的 URL, 浏览器会自动重新访问到这个页面 |
| 304 | Not Modified | 告诉客户端, 你请求的资源至上次取得后, 服务端并未更改, 你直接用你本地缓存吧。隐式重定向 |
| 400 | Bad Request | 客户端请求有 语法错误 , 不能被服务器所理解 |
| 403 | Forbidden | 服务器收到请求, 但是 拒绝提供服务 , 比如: 没有权限访问相关资源 |
| 404 | Not Found | 请求 资源不存在 , 一般是URL输入有误, 或者网站资源被删除了 |
| 405 | Method Not Allowed | 请求方式有误, 比如应该用GET请求方式的资源, 用了POST |
| 428 | Precondition Required | 服务器 要求有条件的请求 , 告诉客户端要想访问该资源, 必须携带特定的请求头 |
| 429 | Too Many Requests | 指示用户在给定时间内发送了 太多请求 (“限速”), 配合 Retry-After (多长时间后可以请求)响应头一起使用 |
| 431 | Request Header Fields Too Large | 请求 头太大 , 服务器不愿意处理请求, 因为它的头部字段太大。请求可以在减少请求头域的大小后重新提交。 |
| 500 | Internal Server Error | 服务器发生 不可预期的错误 。服务器出异常了, 赶紧看日志去吧 |
| 503 | Service Unavailable | 服务器尚未准备好处理请求, 服务器刚刚启动, 还未初始化好 |

响应头: 第二行开始, 格式 key: value

| | |
|------------------|---|
| Content-Type | 表示该响应内容的类型，例如text/html, application/json。 |
| Content-Length | 表示该响应内容的长度（字节数）。 |
| Content-Encoding | 表示该响应压缩算法，例如gzip。 |
| Cache-Control | 指示客户端应如何缓存，例如max-age=300表示可以最多缓存300秒。 |
| Set-Cookie | 告诉浏览器为当前页面所在的域设置cookie。 |

响应体：最后一部分，存放响应数据

3.2.4 协议解析



3.3 Tomcat

Web 服务器是一个软件程序，对 HTTP 协议的操作进行封装，使得程序员不必直接对协议进行操作，让 Web 开发更加便捷。主要功能是“提供网上信息浏览服务”。

概念：Tomcat:是 Apache 软件基金会一个核心项目，是一个开源免费的轻量级 Web 服务器，支持 Servlet/小 SP 少量 JavaEE 规范。

JavaEE: Java Enterprise Edition, Java 企业版。指 Java 企业级开发的技术规范总和。包含 13 项技术规范：JDBC、JNDI、EJB、RMI、JSP、Servlet、XML、JMS、Java IDL、JTS、JTA、JavaMail、JAF

Tomcat 也被称为 Web 容器、Servlet 容器，支持 servlet、jsp 等少量 JavaEE 规范。。Servlet 程序需要依赖于 Tomcat 才能运行。

官网：<https://tomcat.apache.org/>

3.4 请求响应

请求(HttpServletRequest): 获取请求数据

响应.HttpServletResponse): 设置响应数据

BS 架构：Browser/Server，浏览器/服务器架构模式。客户端只需要浏览器，应用程序的逻辑和数据都存储在服务端。

CS 架构：Client/Server，客户端/服务器架构模式。

3.4.1 请求

- **Postman**

网页调试，发送网页 HTTP 请求，用于接口测试。

- **简单参数**

参数名与形参变量名相同，定义形参即可接收参数。可以自动进行类型转换。

```
@RequestMapping("/simpleParam")
public String simpleParam(String name, Integer age){
    System.out.println(name + " : " + age);
    return "OK";
}
```

@RequestParam 注解

方法形参名称与请求参数名称不匹配，通过该注解完成映射

该注解的 required 属性默认是 true，代表请求参数必须传递

- **实体参数**

规则：请求参数名与形参对象属性名相同，即可直接通过 POJO 接收

- **数组集合参数**

数组参数：请求参数名与形参数组名称相同且请求参数为多个，定义数组类型形参即可接收参数

集合：请求参数名与形参中集合变量相同，通过@RequestParam 绑定参数关系

- **日期参数**

使用@DateTimeFormat 注解完成日期参数格式转换

- **Json 参数**

JSON 数据键名与形参对象属性名相同，定义 POJO 类型形参即可接收参数，需要使用 @RequestBody 标识

- **路径参数**

通过请求 URL 直接传递参数，使用 {...} 来标识该路径参数，需要使用@PathVariable 获取路径参数

3.4.2 响应

@ResponseBody

类型：方法注解、类注解

位置：Controller 方法上/类上

作用：将方法返回值直接响应，如果返回值类型是实体对象/集合，将会转换为 JSON 格式响应

说明：@RestController=@Controller+@ResponseBody；

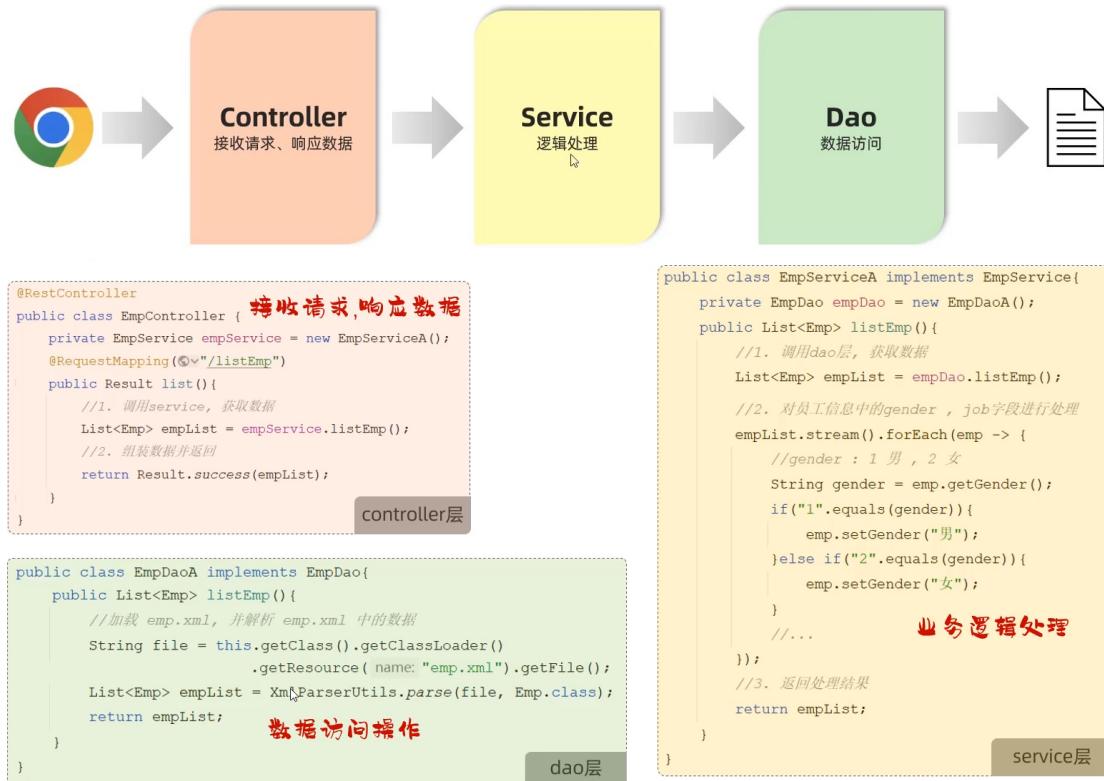
统一响应结果

Result(code、msg、data)

3.5 分层解耦

3.5.1 三层架构

- **controller:** 控制层，接收前端发送的请求，对请求进行处理，并响应数据。
- **service:** 业务逻辑层，处理具体的业务逻辑。
- **dao:** 数据访问层(Data Access Object)（持久层），负责数据访问操作，包括数据的增、删、改、查。



3.5.2 分层解耦

内聚: 软件中各个功能模块内部的功能联系。

耦合: 衡量软件中各个层/模块之间的依赖、关联的程度。

软件设计原则: 高内聚低耦合。

控制反转: Inversion Of Control, 简称 **IOC**。对象的创建控制权由程序自身转移到外部（容器），这种思想称为控制反转。

依赖注入: Dependency Injection,简称 **DI**。容器为应用程序提供运行时，所依赖的资源，称之为依赖注入。

Bean 对象: IOC 容器中创建、管理的对象，称之为 bean。

3.5.3 IOC&DI

- ①. Service 层及 Dao 层的实现类，交给 IOC 容器管理。 @Component
- ②. 为 Controller 及 Service 注入运行时，依赖的对象。 @Autowired
- ③. 运行测试。

3.5.4 IOC

将对象的控制权交给 IOC 容器

| 注解 | 说明 | 位置 |
|-------------|-----------------|-----------------------------|
| @Component | 声明bean的基础注解 | 不属于以下三类时，用此注解 |
| @Controller | @Component的衍生注解 | 标注在控制器类上 |
| @Service | @Component的衍生注解 | 标注在业务类上 |
| @Repository | @Component的衍生注解 | 标注在数据访问类上（由于与mybatis整合，用的少） |

注意事项

声明 bean 的时候，可以通过 value 属性指定 bean 的名字，如果没有指定，默认为类名首字母小写。

使用以上四个注解都可以声明 bean,但是在 springboot 集成 web 开发中，声明控制器 bean 只能用 @Controller。

Bean 组件扫描：

- 前面声明 bean 的四大注解，要想生效，还需要被组件扫描注解 @ComponentScan 扫描。
- @ComponentScan 注解虽然没有显式配置，但是实际上已经包含在了启动类声明注解 @SpringBootApplication 中， 默认扫描的范围是启动类所在包及其子包。

3.5.5 DI

@Autowired 注解， 默认是按照类型进行，如果存在多个相同类型的 bean,将会报出错误
解决：

@Primary
@Autowired + @Qualifier("name")
@Resource(name = "...")

@Autoeired 和@Resource 区别：

@Autowired 是 spring 框架提供的注解，而@Resource 是 JDK 提供的注解。
@Autowired 默认是按照类型注入，而@Resource 默认是按照名称注入。

3.6 案例

开发规范-Restful

- REST(Representational State Transfer)，表述性状态转换，它是一种软件架构风格。

传统风格

- http://localhost:8080/user/getById?id=1 GET: 查询id为1的用户
- http://localhost:8080/user/saveUser POST: 新增用户
- http://localhost:8080/user/updateUser POST: 修改用户
- http://localhost:8080/user/deleteUser?id=1 GET: 删除id为1的用户

REST风格

- http://localhost:8080/users/1 GET: 查询id为1的用户
- http://localhost:8080/users POST: 新增用户
- http://localhost:8080/users PUT: 修改用户
- http://localhost:8080/users/1 DELETE: 删除id为1的用户

URL定位资源
HTTP动词描述操作

REST 是风格，是约定方式，约定不是规定，可以打破。

描述模块的功能通常使用复数，也就是加 s 的格式来描述，表示此类资源，而非单个资源。
如：users、emps、books…

开发流程：



● 文件上传

服务端 MultipartFile

本地存储

在服务端，接收到上传上来的文件之后，将文件存储在本地服务器磁盘中。

阿里云 OSS

阿里云对象存储 OSS (Object Storage Service)，是一款海量、安全、低成本、高可靠的云存储服务。使用 OSS，您可以通过网络随时存储和调用包括文本、图片、音频和视频等在内的各种文件。

使用第三方服务：

准备→参照 SDK 编写入门程序→集成使用



AccessKey ID: LTAI5tQopad8sr5sjkAT3czo

AccessKey Secret: hSIXEo5B0hUKy8N96VnkW3BjRikyPt

SpringBoot 属性配置方式：

- application.properties
key=value
key1=value1
- application.yml
server:

- port: 8080
 - address: 127.0.0.1
 - application.yaml
- ```
server:
 port: 8080
 address: 127.0.0.1
```

- 常见配置文件格式对比

- XML

```
<server>
 <port>8080</port>
 <address>127.0.0.1</address>
</server>
```

臃肿

- properties

```
server.port=8080 层级结构不清晰
server.address=127.0.0.1
```

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/tlias
spring.datasource.username=root
spring.datasource.password=1234
```

- yaml/yml

```
server:
 port: 8080 简洁、数据为中心
 address: 127.0.0.1
```

```
spring:
 datasource:
 driver-class-name: com.mysql.cj.jdbc.Driver
 url: jdbc:mysql://localhost:3306/tlias
 username: root
 password: 1234
```

## yml

- 基本语法

大小写敏感

数值前边必须有空格，作为分隔符

使用缩进表示层级关系，缩进时，不允许使用 Tab 键，只能用空格(idea 中会自动将 Tab 转换为空格)

缩进的空格数目不重要，只要相同层级的元素左侧对齐即可

#表示注释，从这个字符一直到行尾，都会被解析器忽略

- 数据格式

- 对象/Map集合：

```
user:
 name: zhangsan
 age: 18
 password: 123456
```

- 数组/List/Set集合：

```
hobby:
 - java
 - game
 - sport
```

### @ConfigurationProperties(prefix = “\*\*\*.\*\*\*\*”)

配置文件的名称要和实体类对应，实体类交给 IOC 管理，自动将配置项注入到 bean 对象。

@Data

@Component

### @ConfigurationProperties(prefix = “\*\*\*.\*\*\*\*”)

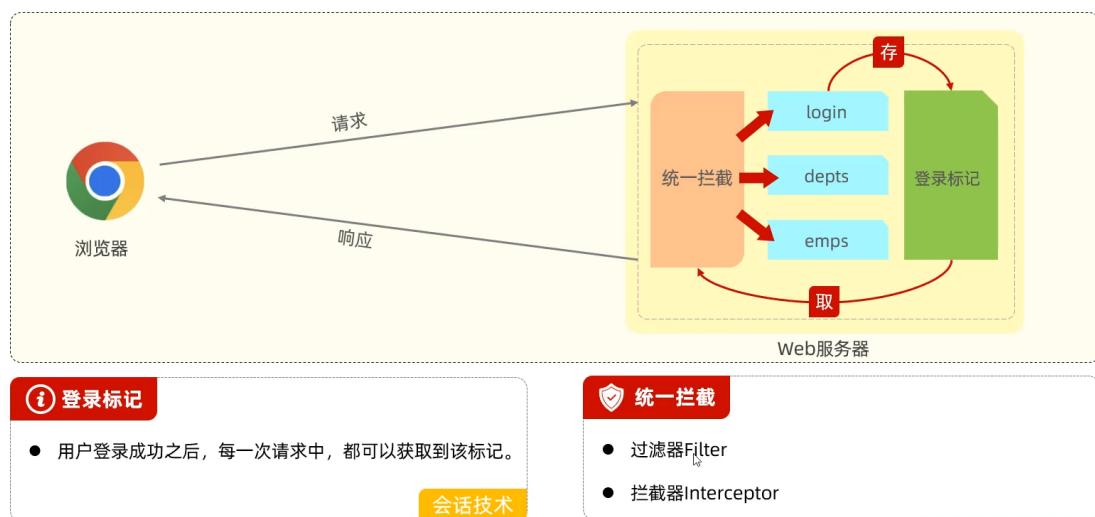
@Value 注解只能一个一个的进行外部属性的注入。

@ConfigurationProperties 可以批量的将外部的属性配置注入到 bean 对象的属性中。

## 3.7 登录认证

### 3.7.1 登录校验

问题：在未登录情况下，我们也可以直接访问部门管理、员工管理等功能。



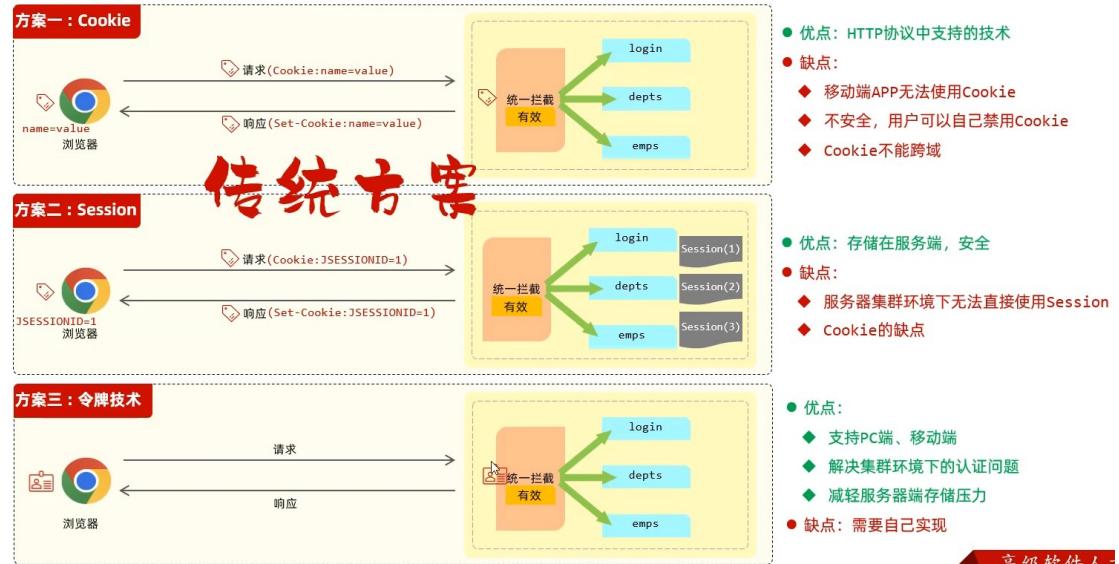
### 会话技术

**会话：**用户打开浏览器，访问 Web 服务器的资源，会话建立，直到有一方断开连接，会话结束。在一次会话中可以包含多次请求和响应。

**会话跟踪：**一种维护浏览器状态的方法，服务器需要识别多次请求是否来自于同一浏览器，以便在同一次会话的多次请求间共享数据。

http 是无状态的，效率高，但是彼此独立，无法共享数据。

跨域分为三个维度：协议、IP/域名、端口



JWT 令牌

JSON Web Token: 定义了一种简洁的、自包含的格式，用于在通信双方以 json 数据格式安全的传输信息。由于数字签名的存在，这些信息是可靠的。

组成：

第一部分：Header（头），记录令牌类型、签名算法等。例如：{"alg":"HS256","type":"JWT"}  
第二部分：Payload（有效载荷），携带一些自定义信息、默认信息等。例如：  
{"id": "1", "username": "Tom"}

第三部分：Signature（签名），防止 Token 被篡改、确保安全性。将 header、payload，并加入指定秘钥，通过指定签名算法计算而来。



**Base64**: 是一种基于64个

#### 4. 登录认证

登录成功生成令牌；  
后续每个请求，都要携带 JWT 令牌，系统在每次请求处理之前，先校验令牌，通过后，

再处理。

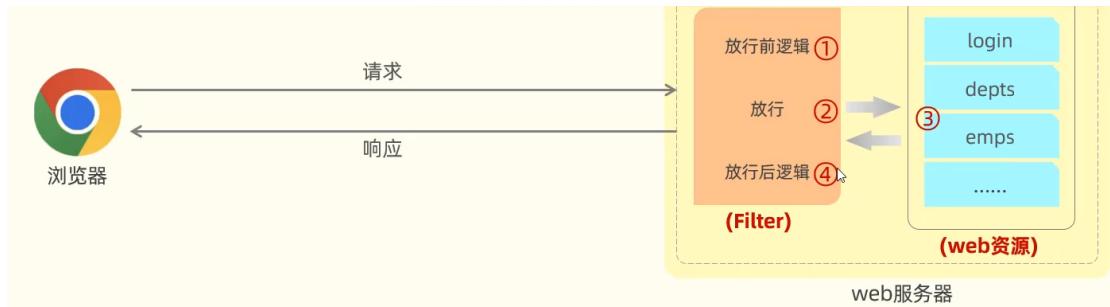
## 過濾器 Filter

概念：Filter 过滤器，是 JavaWeb 三大组件(Servlet、Filter、Listener)之一。

过滤器可以把对资源的请求拦截下来，从而实现一些特殊的功能。

过滤器一船

拦截流程:

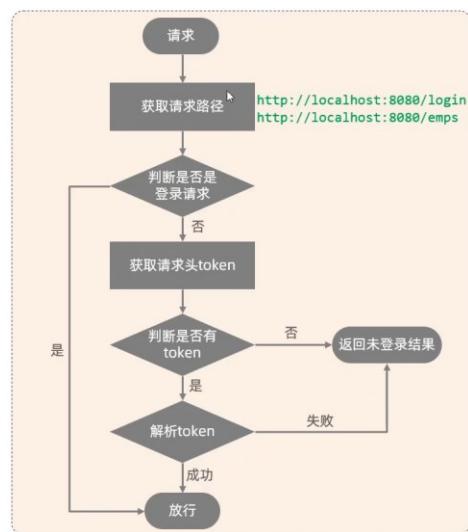


过滤器链：

一个 web 应用里配置多个过滤器，形成控制器链。

顺序：注解配置的 Filter,优先级是按照过滤器类名（字符串）的自然排序。

登录校验过滤器：



## 拦截器 Interceptor

概念：是一种动态拦截方法调用的机制，类似于过滤器。Spring 框架中提供的，用来动态拦截控制器方法的执行。

作用：拦截请求，在指定的方法调用前后，根据业务需要执行预先设定的代码。

拦截路径	含义	举例
/*	一级路径	能匹配/depts, /emps, /login, <b>不能匹配 /depts/1</b>
/**	任意级路径	能匹配/depts, /depts/1, /depts/1/2
/depts/*	/depts下的一级路径	能匹配/depts/1, <b>不能匹配/depts/1/2, /depts</b>
/depts/**	/depts下的任意级路径	能匹配/depts, /depts/1, /depts/1/2, <b>不能匹配/emps/1</b>

### Filter 与 Interceptor

- 接口规范不同：过滤器需要实现Filter接口，而拦截器需要实现HandlerInterceptor接口。
- 拦截范围不同：过滤器Filter会拦截所有的资源，而Interceptor只会拦截Spring环境中的资源。

### 3.7.2 异常处理

全局异常处理器:

@RestControllerAdvice= @ControllerAdvice + @ResponseBody

方法:

@ExceptionHandler

## 4 事务管理&AOP

### 4.1 事务管理

#### 4.1.1 事务

事务是一组操作的集合，它是一个不可分割的工作单位，这些操作要么同时成功，要么同时失败。

操作：

开启事务(一组操作开始前，开启事务): start transaction/begin;

提交事务(这组操作全部成功后，提交事务): commit;

回滚事务(中间任何一个操作出现异常，回滚事务): roll back;

#### 4.1.2 Spring 事务管理

注解：@Transactional

位置：业务(service)层的方法上、类上、接口上

作用：将当前方法交给 spring 进行事务管理，方法执行前，开启事务；成功执行完毕，提交事务；出现异常，回滚事务

#### 4.1.3 事务进阶

##### ● rollbackFor

默认情况下，只有出现 RuntimeException 才回滚异常。rollbackFor 属性用于控制出现何种异常类型，回滚事务。

##### ● propagation

事务传播行为：指的就是当一个事务方法被另一个事务方法调用时，这个事务方法应该如何进行事务控制。

属性值	含义
REQUIRED	【默认值】需要事务，有则加入，无则创建新事务
REQUIRES_NEW	需要新事务，无论有无，总是创建新事务
SUPPORTS	支持事务，有则加入，无则在无事务状态中运行
NOT_SUPPORTED	不支持事务，在无事务状态下运行，如果当前存在已有事务，则挂起当前事务
MANDATORY	必须有事务，否则抛异常
NEVER	必须没事务，否则抛异常
...	

场景

REQUIRED:大部分情况下都是用该传播行为即可。

REQUIRES NEW:当我们不希望事务之间相互影响时，可以使用该传播行为。比如：下订单前需要记录日志，不论订单保存成功与否，都需要保证日志记录能够记录成功。

## 4.2 AOP 基础

AOP: Aspect Oriented Programming(面向切面编程、面向方面编程), 其实就是面向特定方法编程。(动态代理)

### 4.2.1 SpringAOP

导入依赖 spring-boot-starter-app

编写 AOP 程序

#### 场景

记录操作日志

权限控制

事务管理

#### 优势

代码无侵入

减少重复代码

提高开发效率

维护方便

### 4.2.3 AOP 核心概念

连接点: JoinPoint, 可以被 AOP 控制的方法 (暗含方法执行时的相关信息)

通知: Advice, 指哪些重复的逻辑, 也就是共性功能 (最终体现为一个方法)

切入点: PointCut, 匹配连接点的条件, 通知仅会在切入点方法执行时被应用 (实际控制的方法)

切面: Aspect, 描述通知与切入点的对应关系(通知+切入点)

目标对象: Target, 通知所应用的对象

## 4.3 AOP 进阶

### 4.3.1 通知类型

@Around: 环绕通知, 此注解标注的通知方法在目标方法前、后都被执行

@Before: 前置通知, 此注解标注的通知方法在目标方法前被执行

@After: 后置通知, 此注解标注的通知方法在目标方法后被执行, 无论是否有异常都会执行

@AfterReturning: 返回后通知, 此注解标注的通知方法在目标方法后被执行, 有异常不会执行

@AfterThrowing: 异常后通知, 此注解标注的通知方法发生异常后执行

可以抽取切入点表达式@PointCut, 在其他地方复用

### 4.3.2 通知顺序

1. 不同切面类中，  
默认按照切面类的类名字母排序：  
目标方法前的通知方法：字母排名靠前的先执行  
目标方法后的通知方法：字母排名靠前的后执行
2. 用@Order（数字）加在切面类上来控制顺序  
目标方法前的通知方法：数字小的先执行  
目标方法后的通知方法：数字小的后执行

### 4.3.3 切入点表达式

切入点表达式：描述切入点方法的一种表达式

作用：主要用来决定项目中的哪些方法需要加入通知

常见形式：

1. **execution(...)**: 根据方法的签名来匹配

execution 主要根据方法的返回值、包名、类名、方法名、方法参数等信息来匹配，语法为：

execution(访问修饰符? 返回值 包名.类名.? 方法名(方法参数) throws 异常?)

其中带? 的表示可以省略的部分

- ◆ 访问修饰符：可省略(比如：public、protected)
- ◆ 包名.类名：可省略
- ◆ throws 异常：可省略(注意是方法上声明抛出的异常，不是实际抛出的异常)
- 可以使用通配符描述切入点

◆ \* : 单个独立的任意符号，可以通配任意返回值、包名、类名、方法名、任意类型的一个参数，也可以通配包、类、方法名的一部分

```
execution(* com.*.service.*.update*)
```

◆ ... : 多个连续的任意符号，可以通配任意层级的包，或任意类型、任意个数的参数

```
execution(* com.itheima..DeptService.(..))
```

根据业务需要，可以使用且(&&)、或(||)、非(!)来组合比较复杂的切入点表达式。

书写建议

- 所有业务**方法名**在**命名**时尽量**规范**，方便切入点表达式快速匹配。如：查询类方法都是 find 开头，更新类方法都是 update开头。
- 描述切入点方法通常基于**接口描述**，而不是直接描述实现类，**增强拓展性**。
- 在满足业务需要的前提下，**尽量缩小切入点的匹配范围**。如：包名匹配尽量不使用 ..，使用 \* 匹配单个包。

2. **@annotation(...)** :根据注解匹配

@annotation(com.itheima.anno.Log)

### 4.3.4 连接点

在 Spring 中用 JoinPoint 抽象了连接点，用它可以获取方法执行时的相关信息，如目标类名、方法名、方法参数等。

对于@Around 通知，获取连接点信息只能使用 ProceedingJoinPoint

对于其他四种通知，获取连接点信息只能使用 JoinPoint，它是 ProceedingJoinPoint 的父类型

## 4.4 AOP 案例

增、删、改相关接口的操作日志记录到数据库表中。

自定义注解@Log

定义切面类，完成记录操作日志的逻辑。

## 5 SpringBoot 原理

### 5.1 配置优先级

文件属性配置：

- 1 application.properties
- 2 **application.yml** 主流
- 3 application.yaml

Java 系统属性配置：

-Dserver.port=9000



命令行参数：

--server.port=10010

### 5.2 Bean 管理

#### 5.2.1 获取 bean

默认情况下，Spring 项目启动时，会把 bean 都创建好放在 IOC 容器中  
名称：

Object getBean(String name) name 是类型首字母小写

类型：

<T> T getBean(Class<T> requiredType)

类型和名称：

<T> T getBean(String name, Class<T> requiredType)

#### 5.2.2 bean 作用域

@Scope

Spring 支持五种作用域，后三种在 web 环境才生效：

singleton 容器内同名称的 bean 只有一个实例（单例）（默认）

prototype 每次使用该 bean 时会创建新的实例（非单例）

request 每个请求范围内会创建新的实例（web 环境中，了解）

session 每个会话范围内会创建新的实例（web 环境中，了解）

application 每个应用范围内会创建新的实例（web 环境中，了解）

默认 bean 在容器启动时初始化。

@Lazy 延迟到第一次使用时初始化

### 5.2.3 第三方 bean 管理

如果要管理的 bean 对象来自于第三方（不是自定义的），是无法用@Component 及衍生注解声明 bean 的，就需要用到@Bean 注解。

## 5.3 SpringBoot 优先级

### 5.3.1 起步依赖

maven 依赖传递

### 5.3.2 自动配置

SpringBoot 的自动配置就是当 spring 容器启动后，一些配置类、bean 对象就自动存入到了 IOC 容器中，不需要我们手动去声明，从而简化了开发，省去了繁琐的配置操作。

原理

方案一：@ComponentScan

方案二：@import

普通类

配置类

ImportSelector 接口实现类

@EnableXxxx 注解，封装@Import 注解

源码：

#### @SpringBootApplication

该注解标识在 SpringBoot 工程引导类上，是 SpringBoot 中最最重要的注解。该注解由三个部分组成：

- @SpringBootConfiguration：该注解与@Configuration 注解作用相同，用来声明当前也是一个配置类。
- @ComponentScan：组件扫描，默认扫描当前引导类所在包及其子包。
- @EnableAutoConfiguration：实现自动化配置的核心注解。

#### @Conditional

作用：按照一定的条件进行判断，在满足给定条件后才会注册对应的 bean 对象到 Spring IOC 容器中。

位置：方法、类

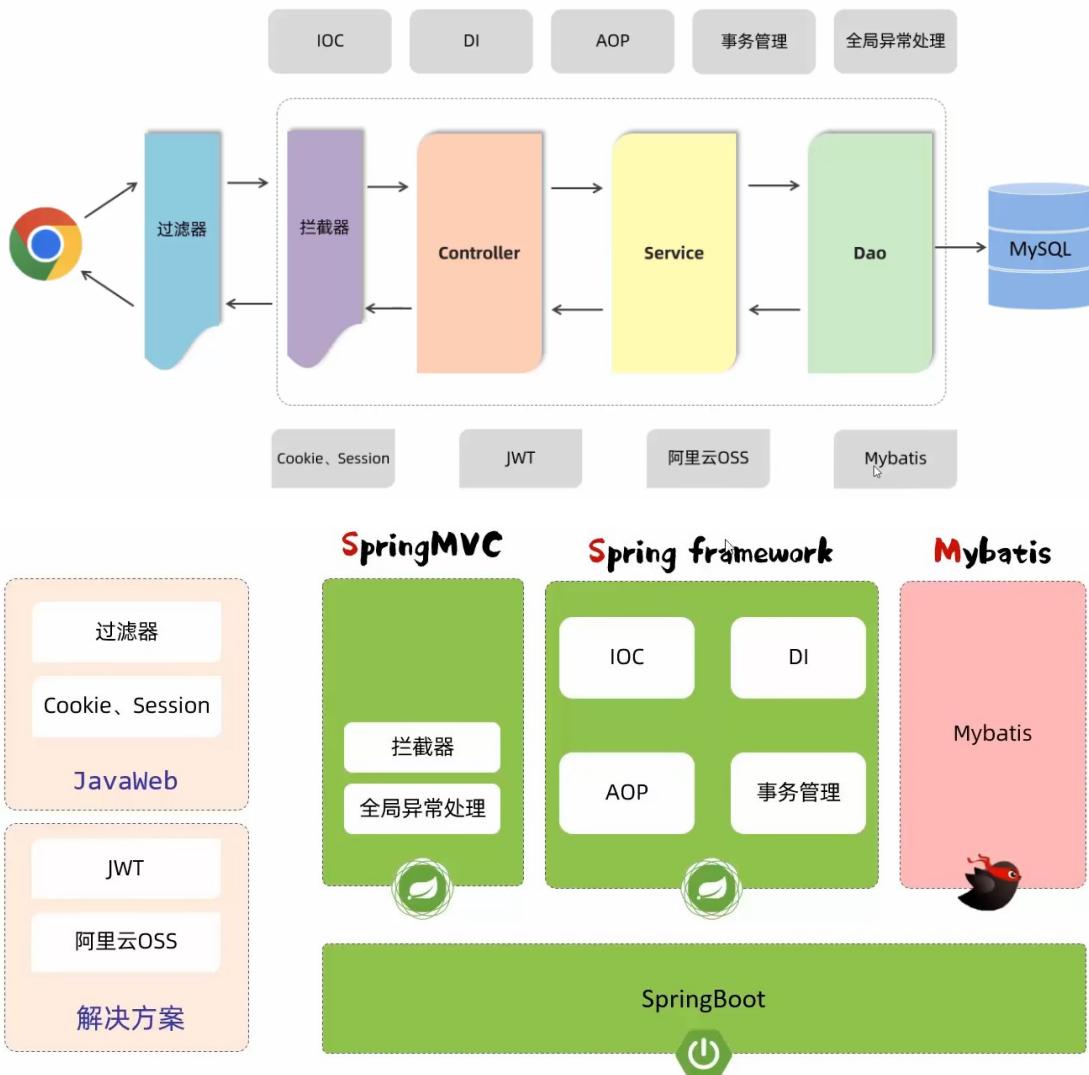
@Conditional 本身是一个父注解，派生出大量的子注解：

@ConditionalOnClass：判断环境中是否有对应字节码文件，才注册 bean 到 IOC 容器。

@ConditionalOnMissingBean：判断环境中没有对应的 bean（类型或名称），才注册 bean 到 IOC 容器。

@ConditionalOnProperty：判断配置文件中有对应属性和值，才注册 bean 到 IOC 容器。

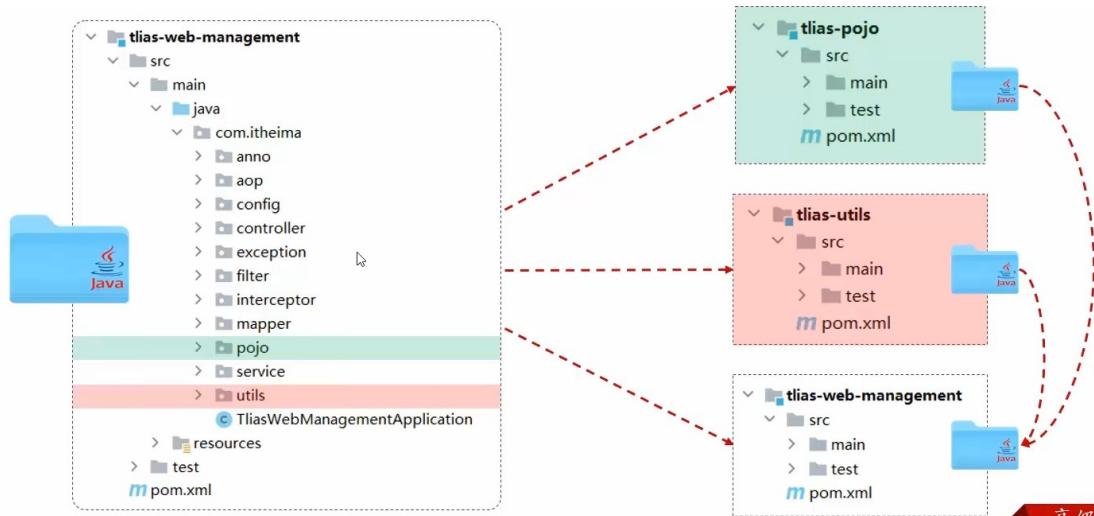
bean 不会全部自动注册为 IOC 容器的 bean，会根据条件装配



## 6 maven 高级

### 6.1 分模块设计与开发

将项目按照功能拆分成若干个子模块，方便项目的管理维护、扩展，也方便模块间的相互调用，资源共享。



分模块开发需要先针对模块功能进行设计，再进行编码。不会先将工程开发完毕，然后进行拆分。

### 6.2 继承与聚合

#### 6.2.1 继承关系实现

概念：继承描述的是两个工程间的关系，与 Java 中的继承相似，子工程可以继承父工程中的配置信息，常见于依赖关系的继承。

作用：简化依赖配置、统一管理依赖

实现：`<parent>...</parent>`

`jar`: 普通模块打包，springboot 项目基本都是 jar 包(内嵌 tomcat 运行)

`war`: 普通 web 程序打包，需要部署在外部的 tomcat 服务器中运行

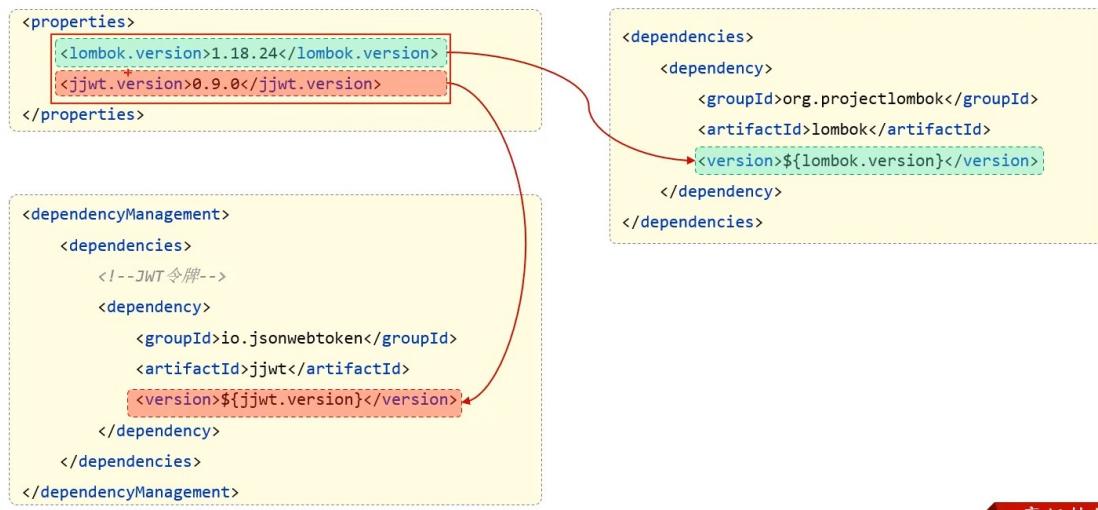
`pom`: 父工程或聚合工程，该模块不写代码，仅进行依赖管理

若父子工程都配置了同一个依赖的不同版本，以子工程的为准。

#### 6.2.2 版本锁定

- 在 maven 中，可以在父工程的 pom 文件中通过`<dependencyManagement>`来统一管理依赖版本。

- 自定义属性/引用属性



### 6.2.3 聚合实现

将多个模块组织成一个整体，同时进行项目的构建。

聚合工程：一个不具有业务功能的“空”工程(有且仅有一个 pom 文件)—父工程作用：

快速构建项目(无需根据依赖关系手动构建，直接在聚合工程上构建即可)

maven 中可以通过`<modules>`设置当前聚合工程所包含的子模块名称，聚合工程中所包含的模块，在构建时，会自动根据模块间的依赖关系设置构建顺序，与聚合工程中模块的配置书写位置无关。

继承与聚合

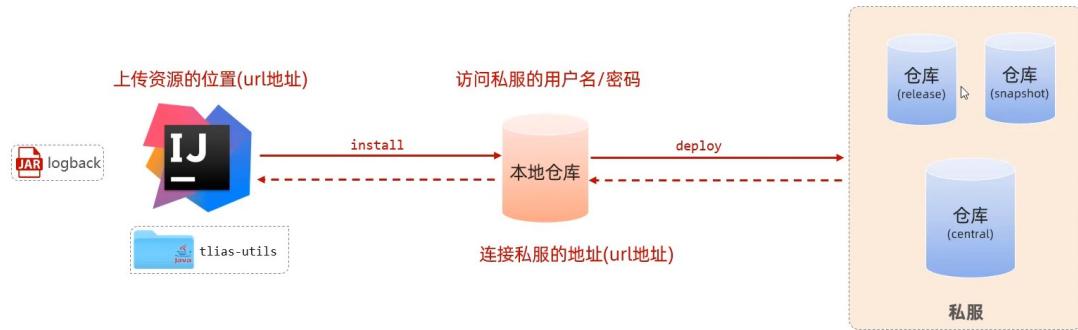
- 作用
  - 聚合用于快速构建项目
  - 继承用于简化依赖配置、统一管理依赖
- 相同点：
  - 聚合与继承的 pom.xml 文件打包方式均为 pom, 可以将两种关系制作到同一个 pom 文件
  - 聚合与继承均属于设计型模块，并无实际的模块内容
- 不同点：
  - 聚合是在聚合工程中配置关系，聚合可以感知到参与聚合的模块有哪些
  - 继承是在子模块中配置关系，父模块无法感知哪些子模块继承了自己

## 6.3 私服

私服是一种特殊的远程仓库，它是架设在局域网内的仓库服务，用来代理位于外部的中央仓库，用于解决团队内部的资源共享与资源同步问题。

依赖查找顺序：本地 → 私服 → 中央仓库

资源上传与下载：



central: 中央仓库下载的

RELEASE (发行版本): 功能趋于稳定、当前更新停止，可以用于发行的版本，存储在私服中的 RELEASE 仓库中。

SNAPSHOT (快照版本): 功能不稳定、尚处于开发中的版本，即快照版本，存储在私服的 SNAPSHOT 仓库中。

1. 设置私服的访问用户名/密码(settings.xml 中的 servers 中配置)
2. IDEA 的 maven 工程的 pom 文件中配置上传（发布）地址
3. 设置私服依赖下载的仓库组地址(settings.xml 中的 mirrors、profiles 中配置)