1. Web开发的基本概念

> Web应用是基于 Web 技术开发的应用程序，用户可以通过浏览器访问和使用，而无需下载和安装应用。Web应用可以在不同的设备和平台上运行，无需针对不同的操作系统进行开发和维护，具有较好的跨平台兼容性。

> Java Web开发是指使用Java语言来构建和维护可以在Web浏览器上运行的应用程序的过程。这些应用程序通常由服务器端组件和客户端组件组成，服务器端负责处理业务逻辑并提供数据给客户端，而客户端则负责显示数据并允许用户与之交互。

2. web前端开发基础

HTTP协议

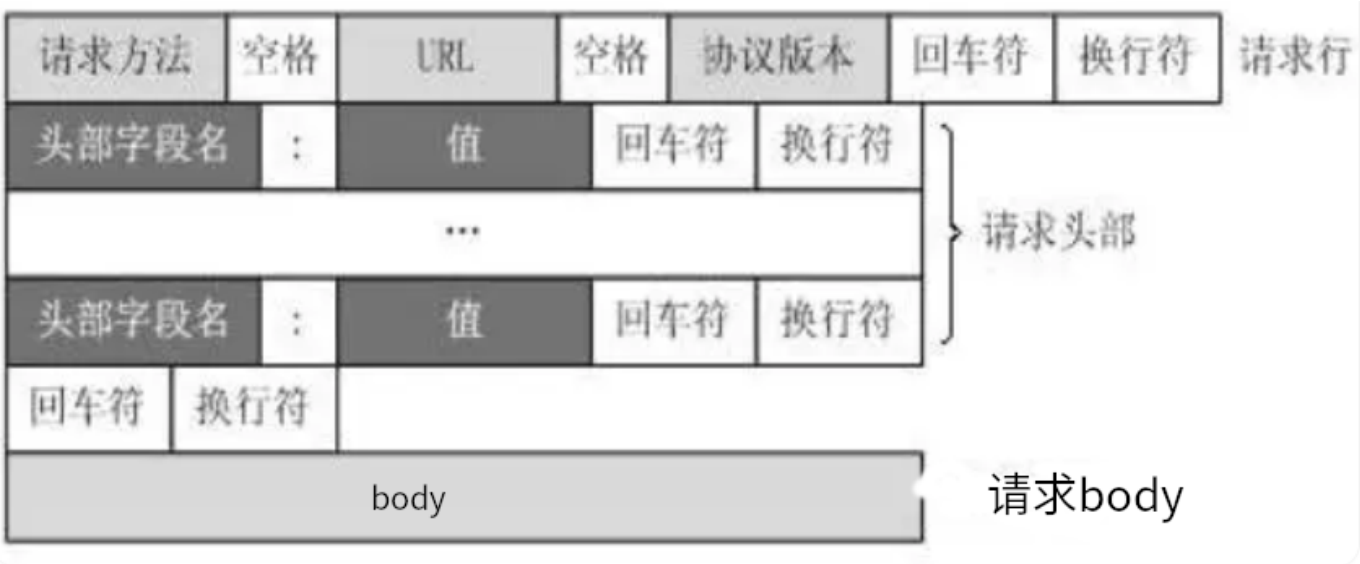
URL: Uniform Resource Locator 统一资源定位符

> URL是URI的子集，URL包含访问资源所需的协议类型（如HTTP、HTTPS、FTP）和资源的位置.

HTTP请求包结构

一个完整的HTTP请求包通常包含三部分：

- 请求行：包含请求方法Method、请求的URL和HTTP版本。
- 请求头：包含一系列键值对，提供了关于请求的额外信息，如客户端类型、接受的内容类型等。
- 请求体：请求体并不是所有HTTP请求都有的部分；主要用于包含要发送给服务器的数据,如表单数据、上传的文件内容等.



请求报文结构

HTTP请求方法

- Get: 获取资源
- Post: 创建资源
- Put: 更新资源
- Delete: 删除资源

HTTP常见状态码

- **200 OK：** 请求成功。
- **403 Forbidden：** 请求失败，服务器拒绝执行请求。
- **404 Not Found：** 请求失败，服务器上没有请求的资源。
- **500 Internal Server Error：** 请求失败，服务器出现未知的内部错误。

Cookie

HTTP 是 无状态协议。

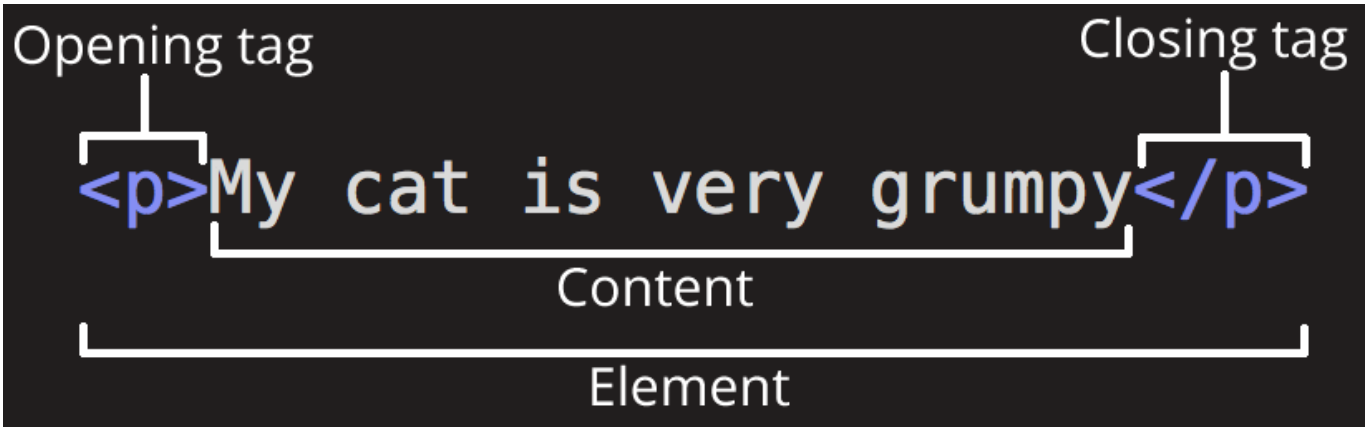
Cookie是为了解决HTTP协议无状态的问题而设计的一种持久化机制，它允许服务器在用户浏览器上存储数据。Cookie 的数据保存在用户浏览器中，服务器可以通过 Cookie 识别用户，从而实现用户个性化。

Cookie 主要用于以下三个方面：

- 会话状态管理
- 个性化设置
- 浏览器行为跟踪

HTML基础知识

HTML 元素的定义:



- 开始标签（Opening tag）：包含元素的名称（本例为 p），被左、右角括号所包围。开头标签标志着元素开始或开始生效的地方。在这个示例中，它在段落文本的开始之前。
- 内容（Content）：元素的内容，本例中就是段落的文本。
- 结束标签（Closing tag）：与开始标签相似，只是其在元素名之前包含了一个斜杠。这标志着该元素的结束。没有包含关闭标签是一个常见的初学者错误，它可能会产生奇特的结果。

元素也可以拥有属性，属性看起来像这样：

属性包含元素的额外信息，这些信息不会出现在实际的内容中。在上述例子中，`class` 属性用于设置 HTML 元素的类名。

CSS基础知识

在HTML文档中，使用`link`标签可以将外部CSS文件链接到网页中，以定义样式。 使用`style`标签，在HTML文档中直接定义样式。

css的选择器：

选择器类型	解释	示例
元素选择器	也叫标签选择器，通过标签名选取元素	`div { color: red; }` 选取所有 `<div>` 元素，将文本颜色设置为红色。
类选择器	通过类名选取元素	`.class-name { font-weight: bold; }` 选取具有类名为 `class-name` 的元素，使其字体加粗。
ID选择器	通过ID名称选取唯一元素	`#id-name { background-color: yellow; }` 选取ID为 `id-name` 的元素，背景色设为黄色。
通配符选择器	匹配所有元素	`* { margin: 0; padding: 0; }` 选取所有元素，清除默认的外边距和内边距。

css的块级元素和行内元素：

- **块级元素**：块级元素会独占一行，其宽度默认为父元素的100%。常见的块级元素有`<div>`, `<p>`等。
- **行内元素**：行内元素不会独占一行，其宽度默认为内容宽度。常见的行内元素有`<span>`, `<a>`等。

JavaScript基础知识

javascript中定义变量的关键字有：`let`, `const`, `var`. 其中`const`用于声明一个常量。

DOM操作

选择元素:

- document.getElementById(): 根据元素ID获得对应的dom对象。
- document.getElementsByClassName(): 根据元素类名获得对应的dom对象列表。
- document.querySelector(): 根据选择器获得对应的dom对象。

例如，在DOM操作中，用于获取元素的ID为example的方法是： document.getElementById("example")。

综合示例， 实现一个计数器：

```
<html>
<head>
    <meta charset="UTF-8">
    <title>计数器</title>
</head>
<body>
    <h1>计数器：<span id="counter"> 0 </span></h1>
    <button id="button">增加</button>

    <script>
        let number = 0;
```

```
        // 获取id为counter的元素
        const counter = document.getElementById('counter');
        // 获取id为button的元素
        const button = document.getElementById('button');

        // 为按钮绑定点击事件，点击后number自增并更新到span中
        button.addEventListener('click', function() {
            number++;
            counter.innerText = number;
        });
    </script>
</body>
</html>
```

JavaScript中，回调函数是一种函数，作为参数传递给另一个函数，并在这个函数内部被调用。在上面的例子中，`button.addEventListener('click', function() {})` 中第二个参数就是一个回调函数。

AJAX

AJAX 技术主要用于实现页面的部分更新。 AJAX 请求的响应数据的格式，常用的有：JSON 、XML。

通过浏览器访问服务器资源时，服务器端返回的数据很多形式，常见的有JSON、XML、HTML等。

XML 是一种用于定义文档结构和存储数据的标记语言。常用于不同系统之间的数据交换。广泛用于Web服务、配置文件、数据交换等多种场景.

JSON是一种轻量级的数据交换格式，广泛用于Web应用的数据交换格式，特别是在Ajax应用中。

在JavaScript中，可使用 `JSON.parse()` 方法将JSON字符串转换为JavaScript对象。

在js中可以使用 XMLHttpRequest 对象来发送AJAX请求. 在javascript 新版本中， `fetch` API 用来替代 `XMLHttpRequest`，它提供了更简洁的语法和基于 `Promise` 的接口。 使用Fetch API 的时候，需要注意 response.json() 方法返回的是一个 `Promise` 对象。

3. Servlet

MVC（Model-View-Controller）设计模式是一种软件架构模式，用于将应用程序的逻辑、数据和用户界面分离，以提高代码的可维护性和可扩展性。

- 模型（Model）：负责管理应用程序的数据和业务逻辑。
- 视图（View）：负责显示数据，即用户界面。
- 控制器（Controller）：负责处理用户输入，协调模型和视图之间的交互。

Java Web开发中常用的Servlet容器是 `Tomcat`，是一个开放源代码的Servlet容器。

JSP页面中包含Html代码和Java代码。 使用标签`<% %>` 在JSP页面中嵌入Java代码片段, 使用`<%= %>` 输出变量。

Servlet的doGet()方法是处理HTTP GET请求的，doPost()方法是处理HTTP POST请求的。

JavaBean的必要特征:

- 具有一个公共的无参构造器
- 属性通过公共的`getter`和`setter`方法进行访问.(JavaBean的属性应该通过标准的getter和setter方法暴露.)
- 不包含复杂的业务逻辑，主要用于保存数据

4. Spring Boot

构建工具

Maven和Gradle都可以作为Spring Boot项目的构建工具来使用。

使用Gradle构建Spring Boot项目时，配置项目依赖时，需要在gradle的配置文件`build.gradle` 中的配置块 `dependencies` 部分添加依赖.

Spring Boot基础

Spring Boot 是一个基于Spring框架的轻量级框架，它简化了Spring应用的创建和开发过程。

Spring Boot 不仅能用于开发Web应用， Spring Boot提供了丰富的功能和特性，使得开发者能够更快地构建和运行现代化的应用程序。

Spring Boot提供了内嵌的Servlet服务器`Tomcat`，开发者无需单独部署Servlet容器即可运行Spring Boot应用。 Spring Boot应用默认的监听端口是 `8080`。

Spring Boot应用默认使用的配置文件是 `application.yml` 或 `application.properties` 。

Spring Boot应用的启动类

spring boot应用的入口类（启动类），此类上必须要有注解 `@SpringBootApplication`，入口类示例如下:

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

}
```

Spring Boot中控制器类中常用注解

- @Autowired注解的作用是 自动注入依赖的`Bean`实例。
- @RestController注解用于创建RESTful风格的控制器，自动将返回对象转换为JSON。
- @RequestMapping注解用于定义请求路径和请求方法。
- @GetMapping注解用于定义GET请求路径。
- @PostMapping注解用于定义POST请求路径。
- @RequestBody注解获取HTTP请求的请求体.

- @PathVariable注解用于获取URL路径中的参数.

RestController类示例：

```
// 注解Restcontroller表明这个类是控制器类
@RestController
// 注解RequestMapping可设置父路径
@RequestMapping("/api/users")
public class UserController {
    // GetMapping注解表示此方法对应的是Get请求. {id}表示从路径中获取用户id
    @GetMapping("{id}")
    public User getUser(@PathVariable Long id) {
      // 根据用户ID获取用户信息的逻辑
    }
    //@PutMapping 注解表示对应Post请求
    @PostMapping
    public User create( @RequestBody  User user) {
      // 创建新用户的逻辑
    }
}
```

JPA

JDBC是Java编程语言中用于和数据库交互的一组API。它提供了一个通用的接口，使得Java应用程序能够连接到各种不同的数据库管理系统（DBMS），执行查询和更新操作，并处理返回的结果集。

JPA是一个Java持久化标准，用于将Java对象持久化到关系数据库中 。JPA不是一个ORM框架，而是一个规范。在spring boot中，我们使用spring data jpa来实现JPA。

ORM (对象关系映射，Object-Relational Mapping) 是一种编程技术，用于将对象模型映射到关系型数据库模型。 ORM框架的主要目的是简化数据库操作，提高开发效率。

@Entity 注解用于标记一个类为实体类，表示该类将映射到数据库中的表。

@Table 注解用于指定实体类对应的数据库表名。

JpaRepository接口提供了基本的CRUD（创建、读取、更新和删除）操作方法，以及分页和排序功能。

我们需要为实体类创建一个接口，继承JpaRepository接口，并指定实体类和主键类型，这样Spring Data JPA就会自动为该接口生成实现类，并提供相应的数据库操作方法。

以下为JPA的示例：

实体类User：

```
// 使用注解Entity声明User类是一个实体类
@Entity
public class User{
    // id是主键，需要添加注解id
    @id
    // 主键自增
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;

    //用户名，在数据库表中对应的字段名user_name,需要使用注解Column
    @Column(name = "user_name")
    String name;

    // 密码
    String password;

    // 年龄
    Integer age;
}
```

实现实体类User的数据库接口UserRepository：

```
public interface UserRepository extends JpaRepository<User, Long> {
    // 根据用户名查找用户
    User findByName(String name);
    // 查找大于指定年龄的用户
    List<User> findByAgeGreaterThan(Integer age);
}
```