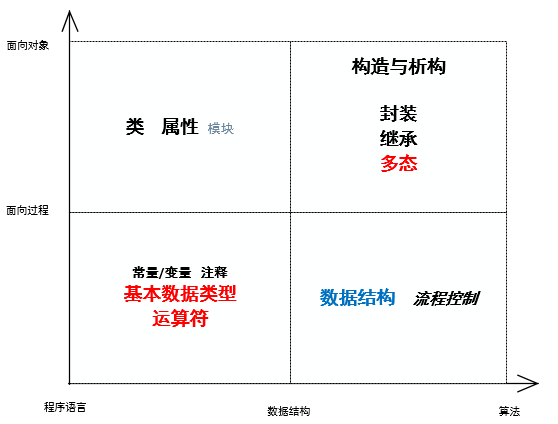
# Java EE笔记



## Java 精通

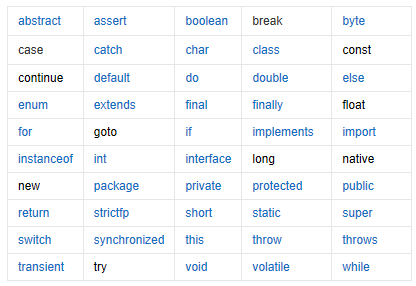
### 第一章java基础

#### 1.1 java基础语法

##### 1.1.1 java 标识符

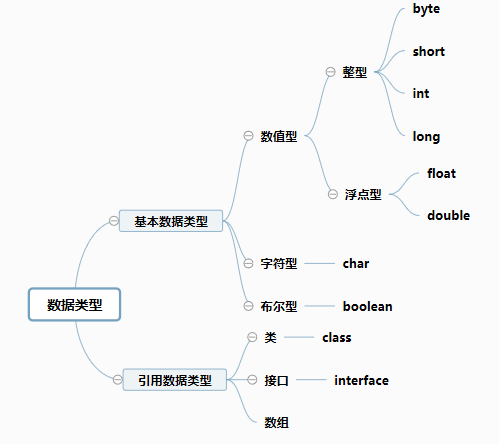
|  |  |  |
| --- | --- | --- |
| Java命名规范 | 示例 | 说明 |
| 包的命名 | scnu.cn.edu.controllor | 全部小写，由域名组成 |
| 类的命名 | GraphicsTest | 单词首字母大写 |
| 方法的命名 | drawImage | 首字母小写，其后单词首字母大写 |
| 常量的命名 | MAX\_VALUE | 全部大写，下划线分词 |
| 参数命名 | fValue | 同方法的命名 |
| 变量命名 | s\_sentence或者strSentence（常用） | 匈牙利命名，类型\_变量名|类型变量名 |

##### 1.1.2 java 关键字



#### 1.2 java数据类型

##### 1.2.1 java 数据类型分类



##### 1.2.2 java数据类型空间占用

|  |  |
| --- | --- |
| 类型 | 占内存大小 |
| byte | 1B |
| short | 2B |
| int | 4B |
| long | 8B |
| float | 4B |
| double | 8B |

#### 1.3 java 控制流程

##### 1.3.1 java运算符

算术运算符，关系运算符，逻辑运算符

|  |  |  |
| --- | --- | --- |
| 算术运算符 | 关系运算符 | 逻辑运算符 |
| + 加法 | == 等于 | && 与（短路） |
| * 减法 | != 不等于 | || 或（短路） |
| * 乘法 | > 大于 | ! 非 |
| / 除法 | < 小于 | & 与（非短路） |
| % 取模 | >= 大于等于 | | 或（非短路） |
| -- 自减 | <= 小于等于 |  |
| ++ 自加 | expression ? true:false(三元) |  |

所谓的“短路”，就是判断一半如果就能确定真假，后面的表达式就不会判断了。比如“&&”操作，如果前面的是假，整个表达式就是假，“&&”后面的表达式就不会判断。

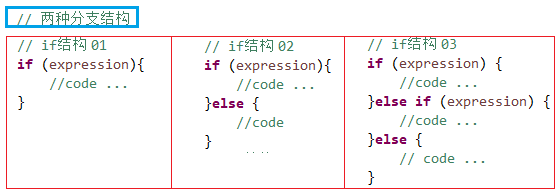
移位运算符：

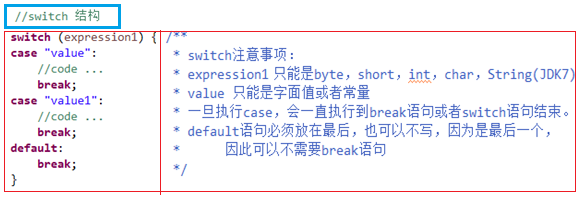
|  |  |
| --- | --- |
| & 按位与 | 1<<3 1\*23 左移 |
| | 按位或 | 8>>3 8/23 右移 |
| ^ 按位异或 | >>> 无符号右移(补0) |
| ~ 按位补(反转) |  |

##### 1.3.2 java 控制语句(6种)

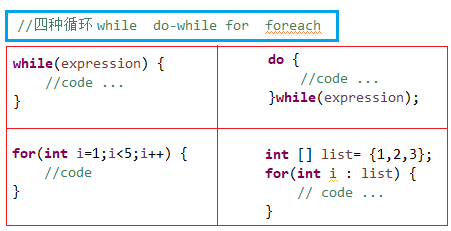
两种分支

两种分支：if语句或者switch语句。而if语句中又有三种结构。如下代码：





四种循环 while do-while for foreach



**注意**： break关键字：结束循环

continue关键字：跳出本次循环

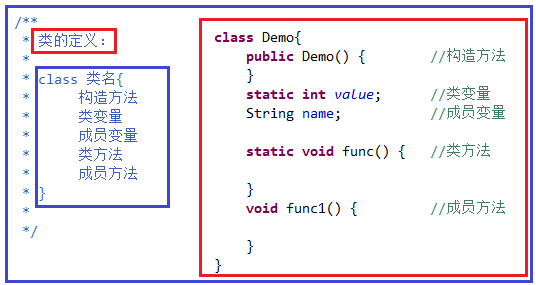
### 第二章 java进阶

#### 2.1 面向对象思想

面向对象，即是通过**消息**传递实现各个对象之间的合作，从而实现程序的功能。因此，一个对象能接收什么样的**消息**是进行面向对象设计的重心，也即是需求分析。

http://www.cnblogs.com/carsonzhu/p/5472159.html

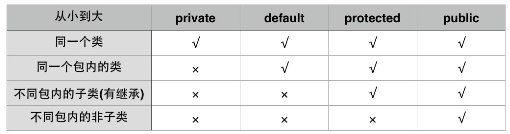
##### 2.1.1 创建类



#### 2.2 封装

##### 2.2.1 权限修饰符

***总之：private同一类内访问，default 同一包内访问，public 同一工程内访问。***

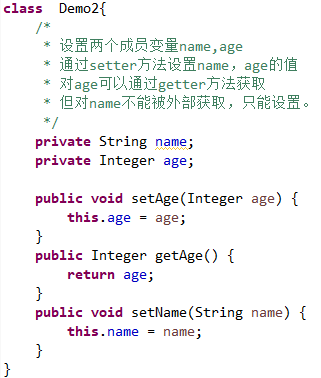


##### 2.2.2 封装性的使用

1）封装的目的：给予属性和方法权限。

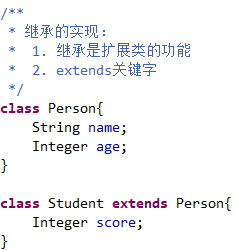
2）封装的实现（示例）：

为属性和方法通过private 进行封装，通过setter和getter方法访问。

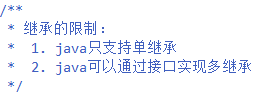


#### 2.3 继承

##### 2.3.1 继承的实现



##### 2.3.2 继承的限制

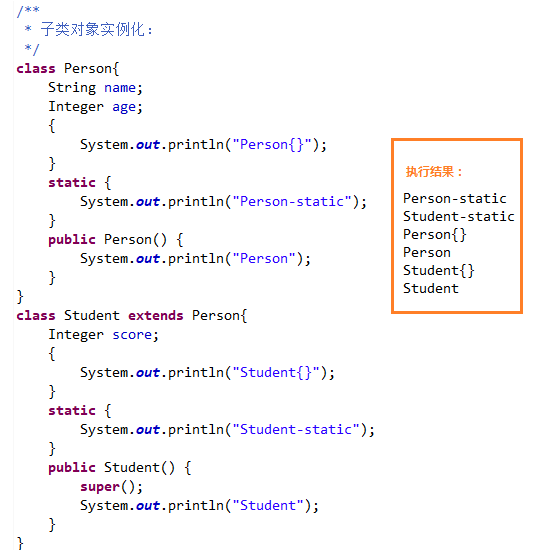


##### 2.3.3 子类对象的实例化

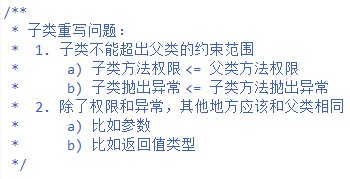
执行顺序：

1. 静态代码块 > 构造代码块 > 构造方法
2. 父类 > 子类
3. 即：

***父类静态代码块 –> 子类静态代码块 –> 父类构造代码块 –> 父类构造方法 –> 子类构造代码块 –> 子类构造方法***

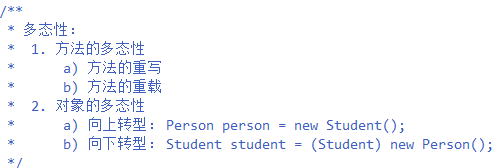


##### 2.3.4 重写



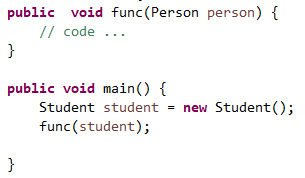
#### 2.4 多态

##### 2.4.1 多态的概念



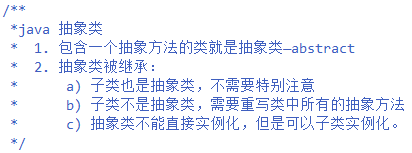
##### 2.4.2 多态的使用

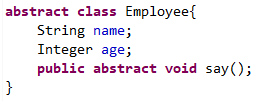
方法传递参数时，可以将参数设置为父类，传入时根据需求传入指定的子类。

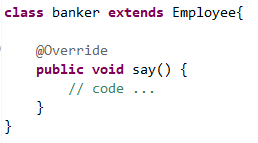


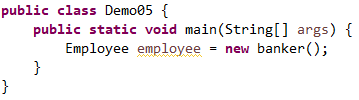
#### 2.5 接口

##### 2.5.1 java抽象类



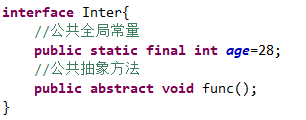




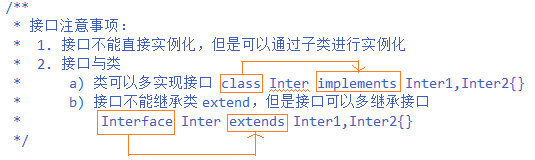


##### 2.5.2 java接口

接口的定义：



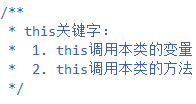
接口的特点：



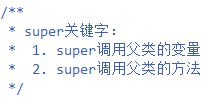
#### 2.6 类中各种关键字

|  |  |  |
| --- | --- | --- |
| 关键字 | 示例 | 说明 |
| this | this . name = name; | 本类 |
| super | super(); | 父类 |
| final | final double PI;  final Person person; | final 在指向基本类型时表示：**该基本类型是常量，比如π**  final 在指向引用类型时表示：  **该引用类型指向不变，但是内容可变。** |
| static | static int age; | **类变量，即只在类第一次实例化时创建赋值，之后的实例化不会重新创建。** |
| instanceof | 对象 instanceof 类 | 判断对象是不是该类的对象 |

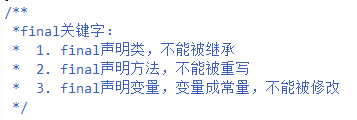
##### 2.6.1 this关键字



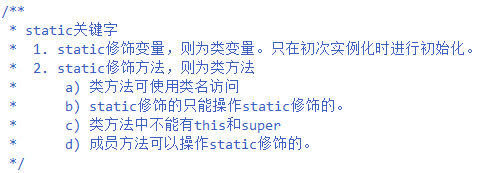
##### 2.6.2 super 关键字



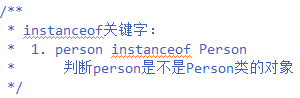
##### 2.6.3 final 关键字



##### 2.6.4 static 关键字



##### 2.6.4 instanceof 关键字



#### 2.7 其他特性与包

##### 2.7.1 匿名对象

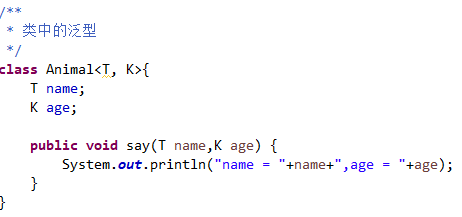


##### 2.7.2 泛型

###### 2.7.2.1 泛型的定义



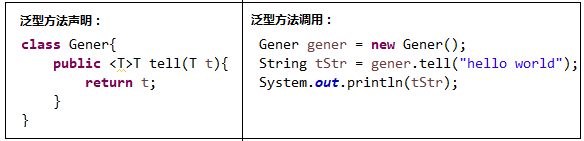
###### 2.7.2.2类/接口中的泛型



**泛型的声明与使用：**



###### 2.7.2.3 泛型方法



###### 2.7.2.4 通配符



##### 2.7.3 包



#### 2.8 常用类

##### 2.8.1 数组

###### 2.8.1.1 数组的声明



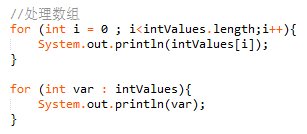
###### 2.8.1.2 数组的创建



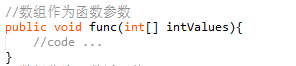
###### 2.8.1.3 数组的初始化



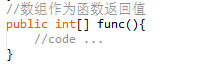
###### 2.8.1.4 数组的处理



###### 2.8.1.5 数组作为函数参数



###### 2.8.1.6 数组作为函数返回值



###### 2.8.1.7 java.util.Arrays类

Array类，其提供的方法都是类方法static

**1. 给数组赋值，fill方法**

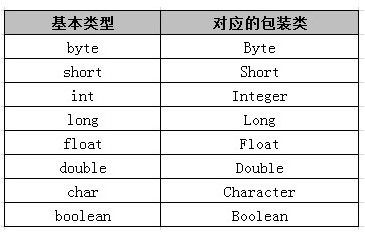
**2. 对数组排序 sort方法**

**3. 比较数组 通过equals比较是否相等**

**4. 查找数组元素：通过binarySearch执行二分法查找**

##### 2.8.2 包装类

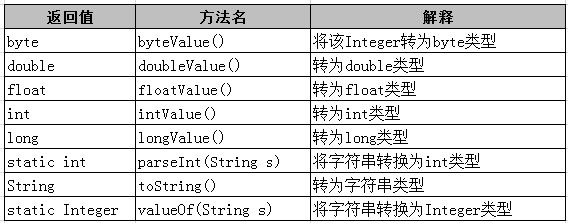
###### 2.8.2.1 基本类型与包装类的对应关系：



其中包装类主要提供了两大类型的方法：

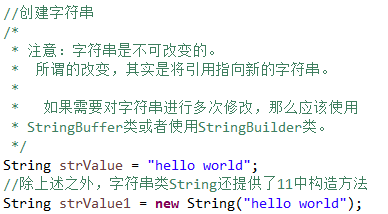
1. **将本类型和其他基本类型进行转换的方法**
2. **将字符串和本类型以及其包装类互相转换的方法**

###### 2.8.2.2 以Integer类型为例，其常用方法如下：



##### 2.8.3 String类

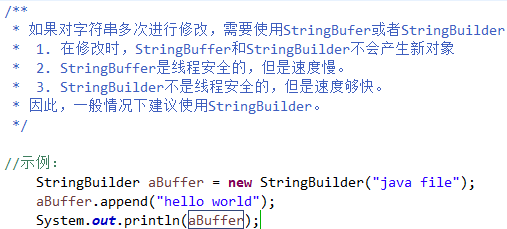
###### 2.8.3.1 String类







###### 2.8.3.2 StringBuffer和StringBuilder类



##### 2.8.4 集合类

##### 2.8.5 日期类

###### 2.8.5.1 日期处理的关键类

**Instant**——它代表的是时间戳

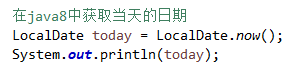
**LocalDate**——不包含具体时间的日期，比如2014-01-14。它可以用来存储生日，周年纪念日，入职日期等。

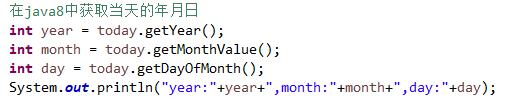
**LocalTime**——它代表的是不含日期的时间

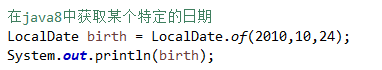
**LocalDateTime**——它包含了日期及时间，不过还是没有偏移信息或者说时区。

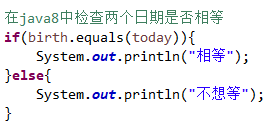
**ZonedDateTime**——这是一个包含时区的完整的日期时间，偏移量是以UTC/格林威治时间为基准的。

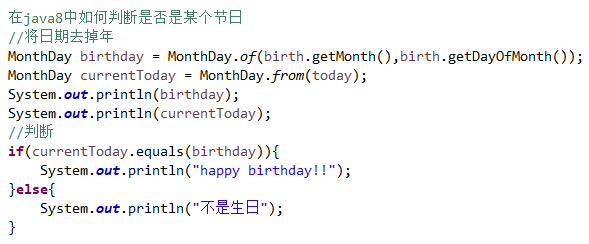
###### 2.8.5.2 日期类的测试：

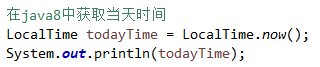


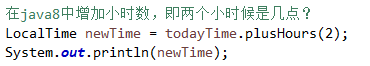


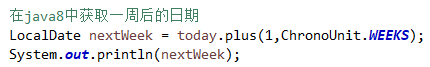


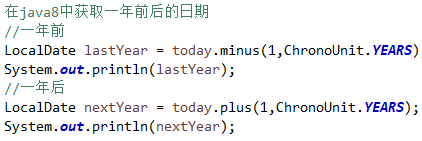












### 第三章 java高级

#### 3.1 多线程

#### 3.2 网络编程

#### 3.3 反射

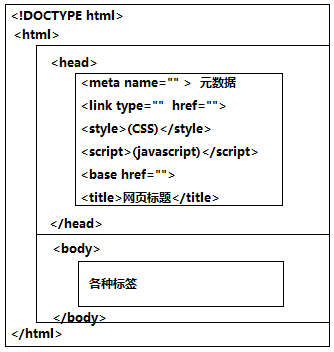
#### 3.4 设计模式

#### 3.5 GUI编程

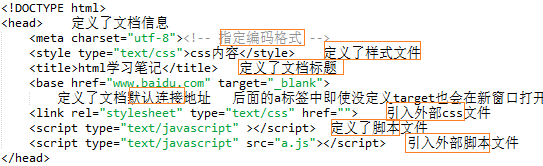
## 第二节 前端技术

### 第一章 html5

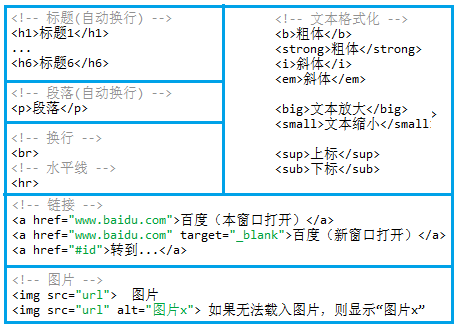
#### 1.1 html 文档结构



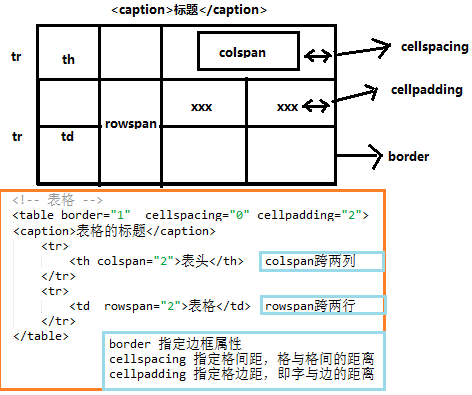
#### 1.2 head 头部



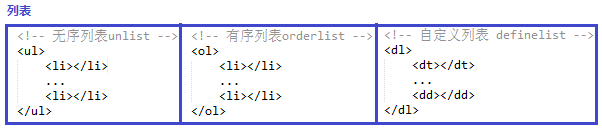
#### 1.3 常用标签



#### 1.4 表格



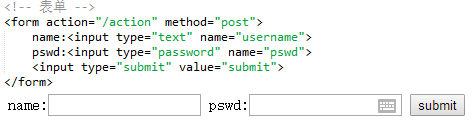
#### 1.5 列表

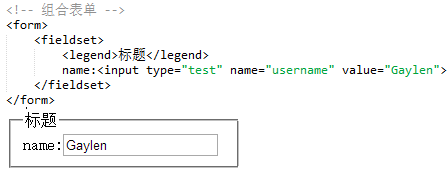


#### 1.6 容器



#### 1.7 表单



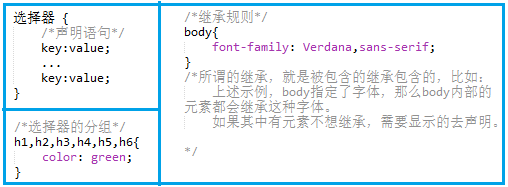




### 第二章 CSS3

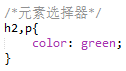
#### 2.1 CSS基础

2.1.1 CSS基础语法

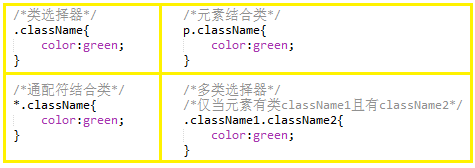


#### 2.2 CSS 选择器

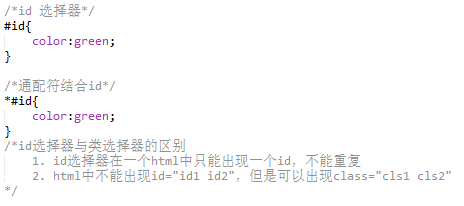
##### 2.2.1 元素选择器



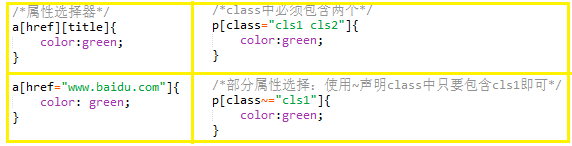
##### 2.2.2 类选择器

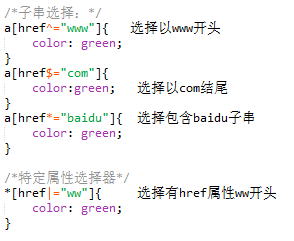


##### 2.2.3 id选择器

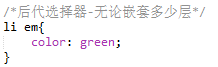


##### 2.2.4 属性选择器

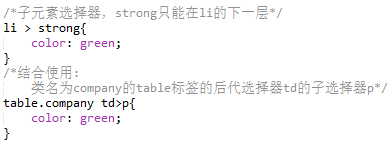




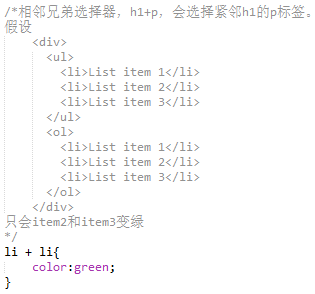
##### 2.2.5 后代选择器



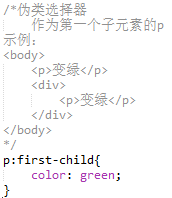
##### 2.2.6 子元素选择器



##### 2.2.7 相邻兄弟选择器



##### 2.2.8 伪类选择器





#### 2.3 CSS 高级

### 第三章 javascript

#### 3.1 JavaScript基础

##### 3.1.1 基础语法

JavaScript 是完全的面向对象脚本语言。

JavaScript 中一切都是对象，因此无论是数字还是字符串都有对应的方法进行操作。

1. 注释

// 单行注释

/\*

多行注释

\*/

2. 变量

var x =3;

var y =3.14;

var z ="hello";

var value; //value = undefined

3. 数据类型

// JavaScript拥有数字，字符串，布尔

// 数组，对象，Null，Undefined这几种数据类型

1. 数字 - JavaScript 只有一种数字类型。不区分整数和小数

var num = new Number(20);

var x = 34;

var y =3.14;

var z = 123e5; //12300000

2. 字符串

var str = new String("JavaScript");

var str = "JavaScript";

var sen = "my name is 'Gaylen'"; //引号嵌套

3. 布尔

var bool = new Boolean(false);

var bool = true;

var x =false;

4. 数组

var arr = new Array();

var arr = new Array(5);

var arr = new Array(1,2,3,4);

var arr =

5. 对象

var obj = new Object();

obj.name="mike";

var person={

name:"Bill",

age:23,

method:function{}

}

var obj = new Object();

6. null //清除

var value = null;

7. undefined //变量中不含有值

var value ;

##### 3.1.2 流程控制

运算符：

|  |  |  |  |
| --- | --- | --- | --- |
| 算术运算符 | 赋值运算符 | 比较运算符 | 逻辑运算符 |
| + 加 | = | == 等于 | && 与 |
| - 减 | += | === 全等（值和类型） | || 或 |
| \* 乘 | -= | != 不等于 | ! 非 |
| / 除 | \*= | > 大于 | (condition)?value1:value2;  三目运算符 |
| % 取余 | /= | >= 大于等于 |  |
| ++ 自增 | %= | < 小于 |  |
| -- 自减 |  | <= 小于等于 |  |

流程控制：

// if语句

//1. if语句

if(expression){

//code ...

}

//2. if-else语句

if(expression){

//code ...

}else{

//code ...

}

//3. if - else if - else语句

if(expression1){

//code ...

}else if(expression2){

//code ...

}else{

//code ...

}

//switch 语句

switch(value){

case value1:

// code ...

break;

case value2:

//code ...

break;

default:

///code ...

}

//for 语句

//1. for语句

for(var i ;i<10; i++){

//code ...

}

//2. for - in 语句

var person{

name:"mike",

age:23

};

for(var key in person){

// person[key]

}

//while 循环

//1. while 循环

var count = 0;

while(count<10){

//code

count++;

}

//2. do - while 循环

var count = 0;

do{

//code ...

count++;

}while(count<10);

//break 与 continue

break语句用于跳出整个循环

continue语句用于跳出本次循环，执行下一次循环

##### 3.1.3 函数

//JavaScript函数

function methodName(){

//code ...

}

//示例：

function myFunc(a,b){

return a+b;

}

##### 3.1.4 异常

//1. try - catch 语句

try{

// code ...

}catch(err){

// code ...

}

//2. try - catch - throw

try{

//code

if() throw "empty";

}catch(err){

// code ...

}

#### 3.2 JavaScript DOM

##### 3.2.1 获取HTML中的元素

1. 通过id找到HTML元素

var x = document.getElementById("idName");

2. 通过标签名找到HTML元素

var x = document.getElementByTagName("h1");

3. 通过类名找到HTML元素，需要自己定义函数

4. 通过name找到HTML元素，因为name不唯一，因此返回的是元素数组

var x = document.getElementByName();

##### 3.2.2 改变HTML元素的内容（innerHTML）

// 改变HTML元素的内容

document.getElementById(id).innerHTML=”new HTML”;

//改变HTML元素的属性

document.getElementById(id).attributeName=“new value“；

##### 3.2.3 改变HTML 元素的样式(CSS)

//改变HTML元素的样式

document.getElementById(id).style.propertyName=”new style”;

//示例：

<button type="button" onclick="document.getElementById('id1').style.color='red'">

##### 3.2.4 对HTML DOM事件作出反应

##### 3.2.5 添加/删除HTML 元素

1. 添加元素

/\*\*

\* 添加元素分为两步，创建新元素，追加到已有元素

\*/

var para = document.createElement("input");

//设置属性1

para.value = "这是一个按钮";

//设置属性2

para.setAttribute("type","button");

body.appendChild(para);

//或者

document.getElementByTagName("body").appendChild(pare);

2. 删除元素

/\*\*

\*JavaScript删除元素，分为两步

\* 1. 获取父节点

\* 2. 删除子节点

\*/

var parent = document.getElementById();

var chile = document.getElementById();

parent.removeChild(child);

#### 3.3 JavaScript 高级

##### 3.3.1 面向对象

1. 创建对象

//创建对象

var person = new Object();

person.name = "mike";

...

person.methodName = function{

//code ...

}

//创建直接的对象

var person = {name:"mike",age:23};

//使用函数构造对象

function person(firstname,lastname,age,eyecolor)

{

this.firstname=firstname;

this.lastname=lastname;

this.age=age;

this.eyecolor=eyecolor;

this.methodName = function(parameter){

// code …

};

}

myFather=new person("Bill","Gates",56,"blue");

2.

##### 3.3.2 常用对象

1. 数字

2. 字符串

3. 数组

4. 日期

5. Boolean

6. Math对象

7. RegExp 正则表达式

#### 3.4 JSON

JSON是指JavaScript的对象表示法

JSON 独立于编程语言

{

employees": [

{ "firstName":"Bill" , "lastName":"Gates" }, //Object1

{ "firstName":"George" , "lastName":"Bush" }, //Object2

{ "firstName":"Thomas" , "lastName":"Carter" } //Object3

]

}

### 第四章 JQuery

#### 4.1 JQuery 基本语法

##### 4.1.1 Jquery 基础语法

$(document).ready(function(){

$ (selector).action{

//code …

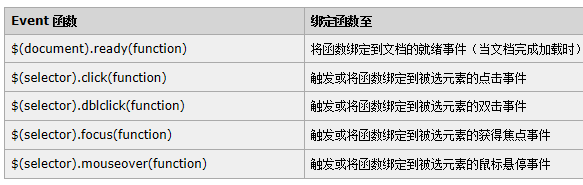
}

});

##### 4.1.2 Jquery 选择器—HTML的获取



##### 4.1.3 Jquery 事件



#### 4.2 JQuery 操作HTML

##### 4.2.1 设置内容

•text() - 设置或返回所选元素的文本内容

•html() - 设置或返回所选元素的内容（包括 HTML 标记）

•val() - 设置或返回表单字段的值

$("#btn1").click(function(){

$("#test1").text("Hello world!");

});

$("#btn2").click(function(){

$("#test2").html("<b>Hello world!</b>");

$("#test2").html(function(param){

});

});

$("#btn3").click(function(){

$("#test3").val("Dolly Duck");

});

##### 4.2.2 设置属性

$("button").click(function(){

$("#w3s").attr(

"href",<http://www.w3school.com.cn/jquery>

"href",<http://www.w3school.com.cn/jquery>

);

$("#w3s").attr("href",function(param){

// code …

});

});

##### 4.2.3 删除元素或内容

//remove 删除被选元素以及子元素

// empty 从被选元素中删除子元素

$("#div1").remove();

$("#div1").empty();

//过滤被删除元素，如删除class = italic的所有p元素

$("p").remove(".italic");

##### 4.2.4 操作CSS

•addClass() - 向被选元素添加一个或多个类

$("button").click(function(){

$("#div1").addClass("important blue");

});

•removeClass() - 从被选元素删除一个或多个类

$("button").click(function(){

$("h1,h2,p").removeClass("blue");

});

•toggleClass() - 对被选元素进行添加/删除类的切换操作

$("button").click(function(){

$("h1,h2,p").toggleClass("blue");

});

•css() - 设置或返回样式属性

//设置样式属性

$("p").css("background-color","yellow");

//返回样式属性

$("p").css("background-color");

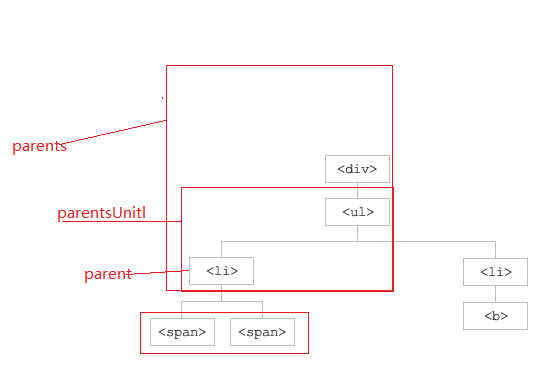
#### 4.3 JQuery 实现遍历HTML

4.3.1 祖先

•parent() 返回span的直接父元素

•parents() 返回span的所有祖先

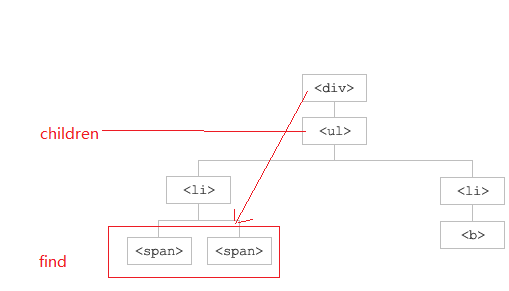
•parentsUntil() 返回span与div之间的祖先



4.3.2 后代

•children() 返回div的直接子元素

•find() 查找div所有后代中的span元素



4.3.3 同胞

•siblings() h2的所有同胞

•next() h2的下一个同胞

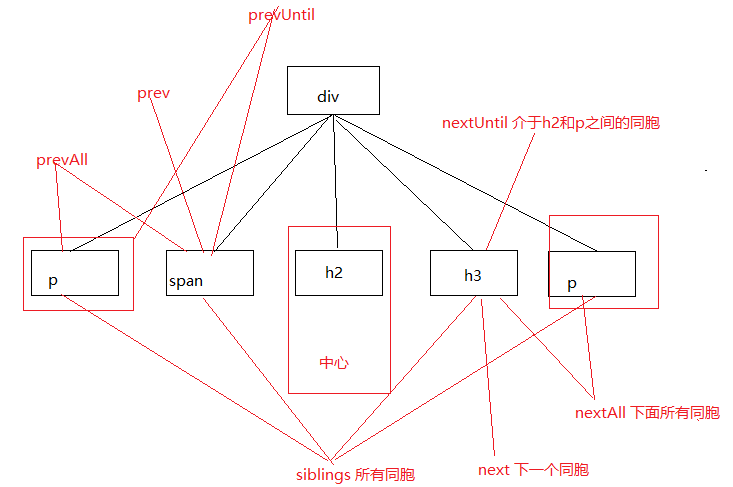
•nextAll() h2后面所有同胞

•nextUntil() h2和p之间的同胞

•prev() h2前一个同胞

•prevAll() h2前面所有同胞

•prevUntil() h2和前面p之间的同胞



4.3.4 过滤

//选取首个div元素内部的第一个p元素

$(document).ready(function(){

$("div p").first();

});

//选取最后一个div的最后一个p元素

$(document).ready(function(){

$("div p").last();

});

//选取第二个p元素。索引号从0开始

$(document).ready(function(){

$("p").eq(1);

});

#### 4.4 JQuery 中的Ajax

#### 4.5 JQuery 中的特效

### 第五章 Angular.js

5.1 Angular 历史

快速开发

5.2 Angular核心概念

5.3 Angular 基础

### 第六章 Node.js

6.1 Node.js 基础

##### 6.1.1 创建第一个应用——helloworld

创建server服务器

var http = require('http');

http.createServer(function (request, response) {

// 发送 HTTP 头部

// HTTP 状态值: 200 : OK

// 内容类型: text/plain

response.writeHead(200, {'Content-Type': 'text/plain'});

// 发送响应数据 "Hello World"

response.end('Hello World\n');

}).listen(8888);

// 终端打印如下信息

console.log('Server running at <http://127.0.0.1:8888/>');

执行命令 node Server.js

在网页打开：<http://127.0.0.1:8888/>

##### 6.1.2 npm 的使用

// npm 版本查看

npm -v

//npm 的升级

npm install npm -g

//npm 安装Nodejs模块

npm install express //本地安装

npm install express -g //全局安装

//npm 卸载模块

npm uninstall express

//npm 更新模块

npm updata express

//npm 搜索模块

npm search express

//npm 创建模块

npm init //生成package.json文件

npm adduser //注册用户

//npm 发布模块

npm publish

6.2 Node.js 进阶

6.3 Node.js 高级

### 第七章 Ajax

#### 7.1 Ajax简介

Ajax 是一步的JavaScript和XML。

Ajax并不是新的编程语言。

Ajax是页面与服务器进行数据交换的艺术。不加载整个页面的情况下进行局部刷新。

#### 7.2 Ajax 基础

##### 7.2.1 创建XHR对象

创建XMLHttpRequest对象，并且兼容IE5，IE6.

var xmlhttp;

if (window.XMLHttpRequest)

{ // code for IE7+, Firefox, Chrome, Opera, Safari

xmlhttp=new XMLHttpRequest();

}

else

{ // code for IE6, IE5

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

##### 7.2.2 向服务器发送请求

Get请求示例：

var xmlhttp;

if (window.XMLHttpRequest)

{// code for IE7+, Firefox, Chrome, Opera, Safari

xmlhttp=new XMLHttpRequest();

}

else

{// code for IE6, IE5

xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");

}

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

}

xmlhttp.open("GET","../ajax/demo\_get.asp.htm",true);

xmlhttp.send();

##### 7.2.3 获得来自服务器的响应



示例：

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

##### 7.2.4 onreadystatechange 事件



示例：

xmlhttp.onreadystatechange=function()

{

if (xmlhttp.readyState==4 && xmlhttp.status==200)

{

document.getElementById("myDiv").innerHTML=xmlhttp.responseText;

}

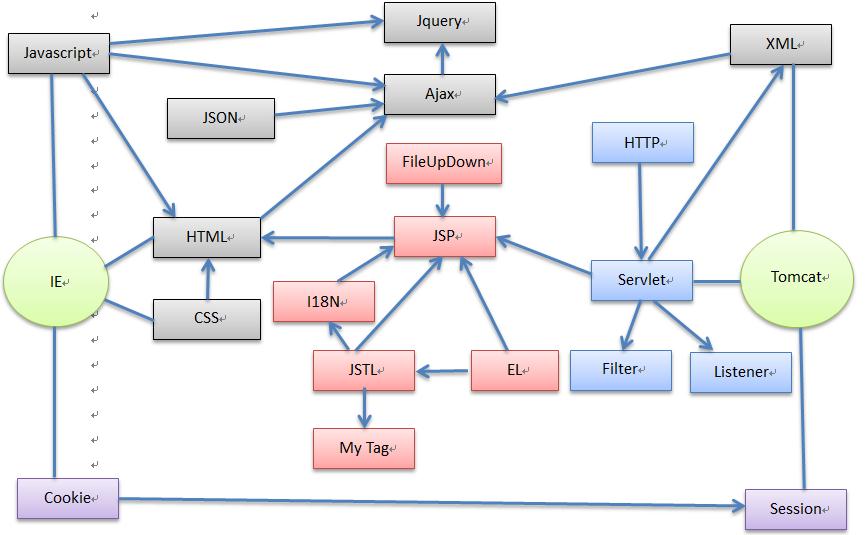
}

#### 7.3 Ajax 进阶

#### 7.4 Ajax 高级

## JAVA Web精通

JavaWEB 知识结构图



### HTTP

#### HTTP的主要特点

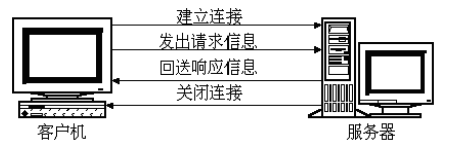
简单，快速，灵活，无连接，无状态

所谓的无连接，就是在交互后关闭连接，下一次交互会发送新的请求

所谓的无状态，就是下一次交互的页面与上一次的无关，每次请求都是独立的，这时如果下一个页面需要上一个页面的信息，就需要重新传送，这样就会导致数据量比较大，但是如果不需要上一个页面的信息，那么会反应比较快。

#### HTTP 的会话方式

##### 1.2.1 浏览器与客户端至少需要四个步骤



##### 1.2.2 HTTP请求方式1.0 与 1.1

在1.0 版本中，如果网页中带有图片之类，会开启一个新的链接下载图片。但是1.1之后都是使用同一个连接

#### HTTP请求

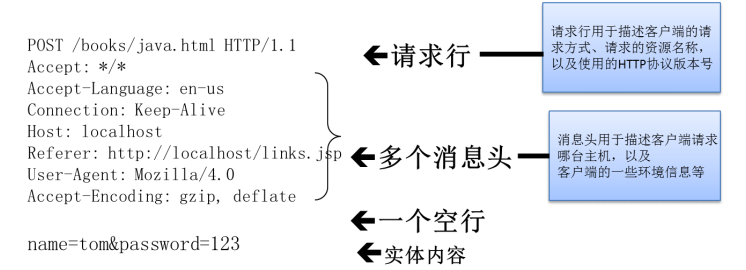
##### 一个完整的HTTP请求

一个完整的HTTP请求包括：

①请求行

②请求头

③请求体



##### 请求方法

①HTTP中定义了7中请求方式：POST，GET，HEAD，OPTIONS，DELETE，TRACE，PUT

②GET请求

GET请求是比较常用的，无论是在浏览器中输入url地址还是超链接等都是以GET的方式进行的

GET对于参数的携带都是在URL地址之后的<http://xxx.htm?name> = abc & pwd=123

从上可知，GET在携带数据上大小是有限制的，因为数据放在请求头中。

GET请求由于将数据放在请求头中，因此本身没有请求体

③POST请求

POST一般用在表单提交或者在Ajax中指定使用POST，由于POST方式是将数据放在请求体中，因此可以携带大量数据，并且大小是没有限制的。

POST请求，是将form表单提交的数据放在请求体中，发送给服务器的，因此会有请求体。

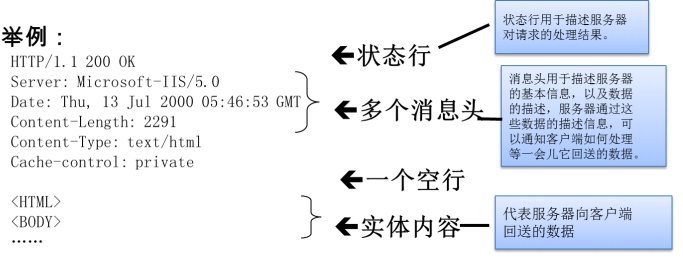
##### HTTP的响应

一个HTTP的响应也是由三部分组成的：

①响应状态行，包括协议版本，状态码等

②响应消息头，响应头

③实体内容：响应体



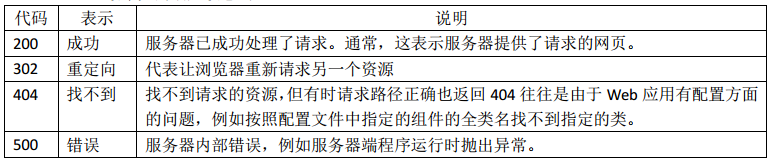
常见的响应状态码：

响应状态码以2开头通常表示成功

响应状态码以3开头通常表示转移，重定向

响应状态码以4开头通常表示无法访问，找不到或者没权限，如404

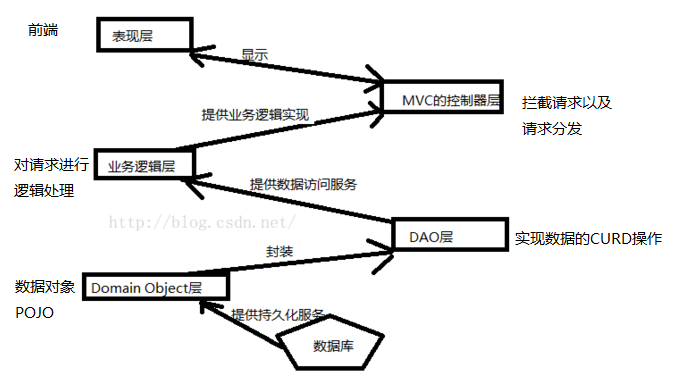
响应状态码以5开头通常表示服务器错误



### 第二章 Tomcat

### 第三章 Servlet

#### 3.1 JavaEE的分层模型



其中Servlet主要实现的是业务逻辑层

### 第四章 JSP

## 第四节 Spring 精通

### 第一章 Spring框架

### 第二章 spring 核心容器IOC

#### 2.1 bean的实例化

#### 2.2 bean的管理

#### 2.3 bean属性的设置

#### 2.4 使用注解完成IOC配置

### 第三章 spring 的AOP机制

#### 3.1 spring AOP的原理

#### 3.2 spring AOP的几种通知

#### 3.3 AOP 的注解配置

### 第四章 spring的数据库操作

#### 4.1 spring集成jdbc—jdbcTemplet

#### 4.2 spring的事务控制

## 第五节 Spring MVC精通

### 第一章 spring mvc原理

### 第二章 springMVC的各个组件

#### 2.1 handlerMapping

#### 2.2 控制器

#### 2.3 基于注解的控制器

#### 2.4 ModelAndViewResolver

### 第三章 springMVC 实现功能

#### 3.1 异常处理

#### 3.2 拦截器

#### 3.3 数据传递

#### 3.4 国际化

#### 3.5 类型转换

#### 3.6 验证

## 第六节 MyBatis 精通

### 第一章 MyBatis原理

### 第二章 全局mybatis-config配置

### 第三章 映射XML文件

### 第四章 动态SQL