Java 花里胡哨缩略词汇总

JDBC | 数据库连接

———— Java DataBase Connectivity（Java 数据库连接， Java 语言操作数据库）。

1. JDBC 本质：其实是官方（sun 公司）定义的一套操作所有关系型数据库的规则，即接口。各个数据库厂商去实现这套接口，提供数据库驱动 jar 包。我们可以使用这套接口（JDBC）编程，真正执行的代码是驱动 jar 包中的实现类。

2. 快速入门，使用步骤：

    1. 导入驱动 jar 包 mysql-connector-java-5.1.37-bin.jar

        1.复制 mysql-connector-java-5.1.37-bin.jar 到项目的 libs 目录下

        2.右键-->Add As Library

    2. 注册驱动

    3. 获取数据库连接对象 Connection

    4. 定义 sql

    5. 获取执行 sql 语句的对象 Statement

    6. 执行 sql，接受返回结果

    7. 处理结果

    8. 释放资源

3. 代码实现：

    ```
    //1. 导入驱动 jar 包

    //2.注册驱动

    Class.forName("com.mysql.jdbc.Driver");

    //3.获取数据库连接对象
    ```

```
        Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/db3", "root",
"root");
        //4.定义 sql 语句
        String sql = "update account set balance = 500 where id = 1";
        //5.获取执行 sql 的对象  Statement
        Statement stmt = conn.createStatement();
        //6.执行 sql
        int count = stmt.executeUpdate(sql);
        //7.处理结果
        System.out.println(count);
        //8.释放资源
        stmt.close();
        conn.close();
```

C3P0 | 数据库连接池

———— 一种数据库连接池技术。

* 使用步骤：

    1. 导入 jar 包 (两个) c3p0-0.9.5.2.jar mchange-commons-java-0.2.12.jar

        * 不要忘记导入数据库驱动 jar 包

    2. 定义配置文件：

        * 名称： c3p0.properties 或者 c3p0-config.xml

        * 路径：直接将文件放在 src 目录下即可。

    3. 创建核心对象 数据库连接池对象 ComboPooledDataSource

4. 获取连接： getConnection

* 代码：

```
//1.创建数据库连接池对象
DataSource ds  = new ComboPooledDataSource();
//2. 获取连接对象
Connection conn = ds.getConnection();
```

Druid | 数据库连接池

———— 一种数据库连接池实现技术，由阿里巴巴提供。

1. 使用步骤：
    1. 导入 jar 包 druid-1.0.9.jar
    2. 定义配置文件：
        * 是 properties 形式的
        * 可以叫任意名称，可以放在任意目录下
    3. 加载配置文件。Properties
    4. 获取数据库连接池对象：通过工厂来来获取 DruidDataSourceFactory
    5. 获取连接：getConnection

* 代码：

```
//3.加载配置文件
Properties pro = new Properties();
InputStream is = DruidDemo.class.getClassLoader().getResourceAsStream("druid.properties");
pro.load(is);
```

```
//4.获取连接池对象
DataSource ds = DruidDataSourceFactory.createDataSource(pro);
//5.获取连接
Connection conn = ds.getConnection();
```

2. 定义工具类
    1. 定义一个类 JDBCUtils
    2. 提供静态代码块加载配置文件，初始化连接池对象
    3. 提供方法
        1. 获取连接方法：通过数据库连接池获取连接
        2. 释放资源
        3. 获取连接池的方法

Spring JDBC | Spring 数据库连接池

———— Spring 框架对 JDBC 的简单封装。提供了一个 JDBCTemplate 对象简化 JDBC 的开发。

* 使用步骤：
    1. 导入 jar 包
    2. 创建 JdbcTemplate 对象。依赖于数据源 DataSource
        * JdbcTemplate template = new JdbcTemplate(ds);
    3. 调用 JdbcTemplate 的方法来完成 CRUD 的操作
        * update():执行 DML 语句。增、删、改语句
        * queryForMap():查询结果将结果集封装为 map 集合，将列名作为 key，将值作为 value 将这条记录封装为一个 map 集合
            * 注意：这个方法查询的结果集长度只能是 1

* queryForList():查询结果将结果集封装为 list 集合

    * 注意：将每一条记录封装为一个 Map 集合，再将 Map 集合装载到 List 集合中

* query():查询结果，将结果封装为 JavaBean 对象

    * query 的参数：RowMapper

    * 一般我们使用 BeanPropertyRowMapper 实现类。可以完成数据到 JavaBean 的自动封装

    * new BeanPropertyRowMapper<类型>(类型.class)

* queryForObject：查询结果，将结果封装为对象

    * 一般用于聚合函数的查询

Bootstrap | 前端框架

———— 一个半成品前端框架，来自 Twitter，是目前很受欢迎的前端框架。

- Bootstrap 是基于 HTML、CSS、JavaScript 的，它简洁灵活，使得 Web 开发更加快捷。

    * 一个半成品软件，开发人员可以在框架基础上，在进行开发，简化编码。

    1. 好处：

        1. 定义了很多的 css 样式和 js 插件。我们开发人员直接可以使用这些样式和插件得到丰富的页面效果。

        2. 响应式布局：同一套页面可以兼容不同分辨率的设备。

    2. 快速入门

        1. 下载 Bootstrap

        2. 在项目中将这三个文件夹复制

        3. 创建 html 页面，引入必要的资源文件

XML | 可扩展标记语言

———— Extensible Markup Language 可扩展标记语言。

* 可扩展：标签都是自定义的。 <user> <student>

* 功能

    * 存储数据

        1. 配置文件

        2. 在网络中传输

* xml 与 html 的区别

    1. xml 标签都是自定义的，html 标签是预定义。

    2. xml 的语法严格，html 语法松散

    3. xml 是存储数据的，html 是展示数据

* w3c:万维网联盟

* 约束：

    1. DTD ———— 一种简单的 XML 文档约束技术。

    2. Schema ———— 一种复杂的 XML 文档约束技术。

Jsoup | HTML 解析器

———— 一款 Java 的 HTML 解析器，可直接解析某个 URL 地址、HTML 文本内容。

- 它提供了一套非常省力的 API，可通过 DOM，CSS 以及类似于 jQuery 的操作方法来取出和操作数据。

* 快速入门，使用步骤：

    1. 导入 jar 包

    2. 获取 Document 对象

    3. 获取对应的标签 Element 对象

    4. 获取数据

* 代码：

```
//2.1 获取 student.xml 的 path
String path = JsoupDemo1.class.getClassLoader().getResource("student.xml").getPath();
//2.2 解析 xml 文档，加载文档进内存，获取 dom 树--->Document
Document document = Jsoup.parse(new File(path), "utf-8");
//3.获取元素对象  Element
Elements elements = document.getElementsByTag("name");
```

* 快捷查询方式：

1. selector:选择器

    * 使用的方法：Elements select(String cssQuery)

        * 语法：参考 Selector 类中定义的语法

2. XPath：XPath 即为 XML 路径语言，它是一种用来确定 XML（标准通用标记语言的子集）文档中某部分位置的语言

    * 使用 Jsoup 的 Xpath 需要额外导入 jar 包。

    * 查询 w3cshool 参考手册，使用 xpath 的语法完成查询

    * 代码：

```
//1.获取 student.xml 的 path
String path = JsoupDemo6.class.getClassLoader().getResource("student.xml").getPath();
//2.获取 Document 对象
Document document = Jsoup.parse(new File(path), "utf-8");
//3.根据 document 对象，创建 JXDocument 对象
JXDocument jxDocument = new JXDocument(document);
//4.结合 xpath 语法查询
```

Servlet | 服务器端小程序（接口）

———— 即 server applet，运行在服务器端的小程序。

- Servlet 就是一个接口，定义了 Java 类被浏览器访问到（Tomcat 识别）的规则。

Cookie | 客户端会话技术

———— 客户端会话技术，将数据保存到客户端。

* 快速入门：

* 使用步骤：

    1. 创建 Cookie 对象，绑定数据

        * new Cookie(String name, String value)

    2. 发送 Cookie 对象

        * response.addCookie(Cookie cookie)

    3. 获取 Cookie，拿到数据

        * Cookie[] request.getCookies()

Session | 服务端会话技术

———— 服务器端会话技术，在一次会话的多次请求间共享数据，将数据保存在服务器端的对象中。

* 快速入门：

    1. 获取 HttpSession 对象：

        HttpSession session = request.getSession();

2. 使用 HttpSession 对象：

Object getAttribute(String name)

void setAttribute(String name, Object value)

void removeAttribute(String name)

* 原理：

* Session 的实现是依赖于 Cookie 的。

JSP | java 服务端页面

———— Java Server Pages（ java 服务器端页面）。

1.概念：可以理解为：一个特殊的页面，其中既可以指定定义 html 标签，又可以定义 java 代码

* 用于简化书写!!!

2.原理：

* JSP 本质上就是一个 Servlet。

3.JSP 的脚本：JSP 定义 Java 代码的方式

1. <% 代码 %>：定义的 java 代码，在 service 方法中。service 方法中可以定义什么，该脚本中就可以定义什么。

2. <%! 代码 %>：定义的 java 代码，在 jsp 转换后的 java 类的成员位置。

3. <%= 代码 %>：定义的 java 代码，会输出到页面上。输出语句中可以定义什么，该脚本中就可以定义什么。

MVC | 开发模式

———— 一种比较规范的开发模式。

1. M：Model，模型。JavaBean

    * 完成具体的业务操作，如：查询数据库，封装对象

2. V：View，视图。JSP

    * 展示数据

3. C：Controller，控制器。Servlet

    * 获取用户的输入

    * 调用模型

    * 将数据交给视图进行展示

4. 优点：

    1. 耦合性低，方便维护，可以利于分工协作

    2. 重用性高

EL | 表达式

———— Expression Language（表达式语言）。

* 作用：替换和简化 jsp 页面中 java 代码的编写

* 获取值语法：${表达式}。（el 表达式只能从域对象中获取值）

    1. ${域名称.键名}：从指定域中获取指定键的值。

        * 域名称：

            1. pageScope --> pageContext

            2. requestScope --> request

            3. sessionScope --> session

            4. applicationScope --> application（ServletContext）

                * 举例：在 request 域中存储了 name=张三

        * 获取：${requestScope.name}

2. ${键名}：表示依次从最小的域中查找是否有该键对应的值，直到找到为止。

3. 获取对象、List 集合、Map 集合的值

    1. 对象：${域名称.键名.属性名}

        * 本质上会去调用对象的 getter 方法

    2. List 集合：${域名称.键名[索引]}

    3. Map 集合：

        * ${域名称.键名.key 名称}

        * ${域名称.键名["key 名称"]}

JSTL | JSP 标准标签库

———— JavaServer Pages Tag Library（JSP 标准标签库）。

1. 是由 Apache 组织提供的开源的免费的 jsp 标签

2. 作用：用于简化和替换 jsp 页面上的 java 代码

3. 使用步骤：

    1. 导入 jstl 相关 jar 包

    2. 引入标签库：taglib 指令： <%@ taglib %>

    3. 使用标签

JQuery | JS 前端框架

———— 一个 JavaScript 框架，简化 JS 开发。

* jQuery 是一个快速、简洁的 JavaScript 框架。它封装 JavaScript 常用的功能代码，提供一种简便的

JavaScript 设计模式，优化 HTML 文档操作、事件处理、动画设计和 Ajax 交互。

* JavaScript 框架：本质上就是一些 js 文件，封装了 js 的原生代码而已。

* 快速入门，使用步骤：

    1. 下载 JQuery

    2. 导入 JQuery 的 js 文件：导入 min.js 文件

    3. 使用：var div1 = $("#div1");

* 基本操作：

    1. 事件绑定

        ```
        //1.获取 b1 按钮
        $("#b1").click(function(){
            alert("abc");
        });
        ```

    2. 入口函数

        ```
        $(function () {          });
        ```

        window.onload 和 $(function) 区别

            * window.onload 只能定义一次,如果定义多次，后边的会将前边的覆盖掉

            * $(function)可以定义多次的。

    3. 样式控制：css 方法

        ```
        // $("#div1").css("background-color","red");
        $("#div1").css("backgroundColor","pink");
        ```

AJAX | 异步 JavaScript & XML

———— ASynchronous JavaScript And XML（异步的 JavaScript 和 XML）。

* JQeury 实现方式

1. $.ajax()

    * 语法：$.ajax({键值对});

    //使用$.ajax()发送异步请求

    $.ajax({

        url:"ajaxServlet1111" , // 请求路径

        type:"POST" , //请求方式

        //data: "username=jack&age=23",//请求参数

        data:{"username":"jack","age":23},

        success:function (data) {

            alert(data);

        },//响应成功后的回调函数

        error:function () {

            alert("出错啦...")

        },//表示如果请求响应出现错误，会执行的回调函数

        dataType:"text"//设置接受到的响应数据的格式

        });

2. $.get()：发送 get 请求

    * 语法：$.get(url, [data], [callback], [type])

    * 参数：

        * url：请求路径

        * data：请求参数

        * callback：回调函数

        * type：响应结果的类型

3. $.post()：发送 post 请求

    * 语法：$.post(url, [data], [callback], [type])

    * 参数：

        * url：请求路径

        * data：请求参数

        * callback：回调函数

        * type：响应结果的类型

JSON | JavaScript 对象表示法

———— JavaScript Object Notation（JavaScript 对象表示法）。

* var p = {"name":"张三","age":23,"gender":"男"};

* json 现在多用于存储和交换文本信息的语法

* 进行数据的传输

* JSON 比 XML 更小、更快，更易解析。

* 语法基本规则：

    * 数据在名称/值对中：json 数据是由键值对构成的

        * 键用引号(单双都行)引起来，也可以不使用引号

        * 值得取值类型：

            1. 数字（整数或浮点数）

            2. 字符串（在双引号中）

            3. 逻辑值（true 或 false）

            4. 数组（在方括号中）　　{"persons":[{},{}]}

            5. 对象（在花括号中）　{"address":{"province"：　"陕西"....}}

6. null

* 数据由逗号分隔：多个键值对由逗号分隔

* 花括号保存对象：使用{}定义 json 格式

* 方括号保存数组：[]

2. 获取数据:

1. json 对象.键名

2. json 对象["键名"]

3. 数组对象[索引]

4. 遍历

3. JSON 数据和 Java 对象的相互转换

* JSON 解析器：

* 常见的解析器：Jsonlib，Gson，fastjson，jackson

1. JSON 转为 Java 对象

1. 导入 jackson 的相关 jar 包

2. 创建 Jackson 核心对象 ObjectMapper

3. 调用 ObjectMapper 的相关方法进行转换

1. readValue(json 字符串数据,Class)

2. Java 对象转换 JSON

1. 使用步骤：

1. 导入 jackson 的相关 jar 包

2. 创建 Jackson 核心对象 ObjectMapper

3. 调用 ObjectMapper 的相关方法进行转换

1. 转换方法：

* writeValue(参数 1，obj):

参数 1：

File：将 obj 对象转换为 JSON 字符串，并保存到指定的文件中

Writer：将 obj 对象转换为 JSON 字符串，并将 json 数据填充到字符输出流中

OutputStream：将 obj 对象转换为 JSON 字符串，并将 json 数据填充到字节输出流中

* writeValueAsString(obj):将对象转为 json 字符串

2. 注解：

1. @JsonIgnore：排除属性。

2. @JsonFormat：属性值得格式化

* @JsonFormat(pattern = "yyyy-MM-dd")

Redis | 高速缓存数据库

———— 一款高性能的 NOSQL 系列的非关系型数据库。

* NOSQL：NoSQL(NoSQL = Not Only SQL)，意即“不仅仅是 SQL”，是一项全新的数据库理念，泛指非关系型的数据库。关系型数据库与 NoSQL 数据库并非对立而是互补的关系，即通常情况下使用关系型数据库，在适合使用 NoSQL 的时候使用 NoSQL 数据库，让 NoSQL 数据库对关系型数据库的不足进行弥补。一般会将数据存储在关系型数据库中，在 nosql 数据库中备份存储关系型数据库的数据。

* Redis：是用 C 语言开发的一个开源的高性能键值对（key-value）数据库。

Jedis | redis 数据库驱动

———— 一款 java 操作 redis 数据库的工具。

* 使用步骤：

1. 下载 jedis 的 jar 包

2. 使用

```
//1. 获取连接
Jedis jedis = new Jedis("localhost",6379);
//2. 操作
jedis.set("username","zhangsan");
//3. 关闭连接
jedis.close();
```

JedisPool | jedis 连接池

———— jedis 连接池。

* 使用:

1. 创建 JedisPool 连接池对象

2. 调用方法 getResource()方法获取 Jedis 连接

```
//0.创建一个配置对象
JedisPoolConfig config = new JedisPoolConfig();
config.setMaxTotal(50);
config.setMaxIdle(10);
//1.创建 Jedis 连接池对象
JedisPool jedisPool = new JedisPool(config,"localhost",6379);
//2.获取连接
Jedis jedis = jedisPool.getResource();
//3. 使用
```

```
jedis.set("hehe","heihei");

//4. 关闭 归还到连接池中

jedis.close();
```

Maven | 项目管理工具

———— Maven 是一个项目管理工具。

- 它包含：

1、一个项目对象模型（POM：Project Object Model）；

2、一组标准集合；

3、一个项目生命周期（Project Lifecycle）；

4、一个依赖管理系统（Dependency Management System）；

5、用来运行定义在生命周期阶段（phase）中插件（plugin）目标（goal）的逻辑。

Dubbo | 远程调用框架

———— Apache Dubbo 是一款高性能的 Java RPC 框架。

- 其前身是阿里巴巴公司开源的一个高性能、轻量级的开源 Java RPC 框架，可以和 Spring 框架无缝集成。
- Dubbo 作为一个 RPC 框架，其最核心的功能就是要实现跨网络的远程调用。
- Dubbo 三大核心能力：

1、面向接口的远程方法调用；

2、智能容错和负载均衡；

3、服务自动注册和发现。

Zookeeper | 服务注册中心

———— 服务注册中心。Dubbo 官方推荐使用 Zookeeper 作为服务注册中心。

- Zookeeper 是 Apache Hadoop 的子项目，是一个树型的目录服务，支持变更推送，适合作为 Dubbo 服务的注册中心，工业强度较高，可用于生产环境，并推荐使用。

- 流程说明：

1、服务提供者(Provider)启动时：向 /dubbo/com.foo /providers 目录下写入自己的 URL 地址；

2、服务消费者(Consumer)启动时：订阅 /dubbo/com.foo /providers 目录下的提供者 URL 地址。并向 /dubbo/com.foo.BarService/consumers 目录下写入自己的 URL 地址；

3、监控中心(Monitor)启动时：订阅 /dubbo/com.foo 目录下的所有提供者和消费者 URL 地址。

Spring Boot | 框架

———— Spring Boot 是一个便捷搭建基于 spring 工程的脚手架。

- 帮助开发人员快速搭建大型 spring 项目。简化工程配置，依赖管理；让开发人员把时间集中在业务开发上。

RestTemplate | Spring 模板工具类

———— RestTemplate 是 Spring 提供的一个模板工具类。

- 对基于 Http 的客户端进行了封装，并且实现了对象与 json 的序列化和反序列化，非常方便。

Spring Cloud | 框架（组件集合）

———— Spring Cloud 是 Spring 旗下的项目之一。

- Spring 最擅长的就是集成，把世界上最好的框架拿过来，集成到自己的项目中。

- Spring Cloud 也是一样，它将现在非常流行的一些技术整合到一起，实现了诸如：配置管理，服务发现，智能路由，负载均衡，熔断器，控制总线，集群状态等功能；协调分布式环境中各个系统，为各类服务提供模板性配置。其主要涉及的组件包括：

1、Eureka：注册中心

2、Zuul、Gateway：服务网关

3、Ribbon：负载均衡

4、Feign：服务调用

5、Hystrix 或 Resilience4j：熔断器

以上只是其中一部分。

Eureka | 注册中心

———— Eureka 是一个服务注册中心，用于服务的自动注册、发现、状态监控等。

- Eureka 就好比是滴滴，负责管理、记录服务提供者的信息。服务调用者无需自己寻找服务，而是把自己的需求告诉 Eureka，然后 Eureka 会把符合你需求的服务告诉你。同时，服务提供方与 Eureka 之间通过 “心跳”机制进行监控，当某个服务提供方出现问题，Eureka 自然会把它从服务列表中剔除。这就实现了服务的自动注册、发现、状态监控。

- 基本架构：

1、Eureka：就是服务注册中心（可以是一个集群），对外暴露自己的地址；

2、提供者：启动后向 Eureka 注册自己信息（地址，提供的服务），只要对外提供的是 Rest 风格服务即可；

3、消费者：向 Eureka 订阅服务，Eureka 会将对应服务的所有提供者地址列表发送给消费者，并定期更新；

4、心跳(续约)：提供者定期通过 http 方式向 Eureka 刷新自己的状态。

Ribbon | 负载均衡

————— Ribbon 是 Netflix 发布的负载均衡器，它有助于控制 HTTP 和 TCP 客户端的行为。

- 为 Ribbon 配置服务提供者地址列表后，Ribbon 就可基于某种负载均衡算法，自动地帮助消费者去请求。

- Ribbon 默认提供了很多的负载均衡算法，如轮询、随机等。也可实现自定义负载均衡算法。

Hystrix | 熔断器

————— 是 Netflix 开源的一个延迟和容错库，用于隔离访问远程服务、第三方库，防止出现级联失败。

- Hystrix 在英文里面的意思是 “豪猪”，它的 logo 看下面的图是一头豪猪，它在微服务系统中是一款提供保护机制的组件，和 eureka 一样也是由 netflix 公司开发。

- 雪崩问题：

* 微服务中，服务间调用关系错综复杂，一个请求，可能需要调用多个微服务接口才能实现，会形成非常复杂的调用链路，如果此时，某个服务出现异常，请求阻塞，用户请求就不会得到响应，则 tomcat 的这个线程不会释放，于是越来越多的用户请求到来，越来越多的线程会阻塞服务器支持的线程和并发数有限，请求一直阻塞，会导致服务器资源耗尽，从而导致所有其它服务都不可用，形成雪崩效应。

- Hystrix 解决雪崩问题的手段主要包括：[线程隔离] 和 [服务降级]。

1、[线程隔离] 原理：

* Hystrix 为每个依赖服务调用分配一个小的线程池，如果线程池已满调用将被立即拒绝，默认不采用排队，加速失败判定时间。

* 用户的请求将不再直接访问服务，而是通过线程池中的空闲线程来访问服务，如果线程池已满，或者请求超时，则会进行降级处理。

2、[服务降级] 原理：

* 优先保证核心服务，而非核心服务不可用或弱可用。

* 用户的请求故障时，不会被阻塞，更不会无休止的等待或者看到系统崩溃，至少可以看到一个执行结果（例如返回友好的提示信息）。

* 服务降级虽然会导致请求失败，但是不会导致阻塞，而且最多会影响这个依赖服务对应的线程池中的资源，对其它服务没有响应。

- 触发 Hystrix 服务降级的情况：

1、线程池已满；

2、请求超时。

Feign | 伪装

———— Feign 也叫伪装，伪装 rest 请求。

- Feign 可以把 Rest 的请求进行隐藏，伪装成类似 SpringMVC 的 Controller 一样。你不用再自己拼接 url，拼接参数等等操作，一切都交给 Feign 去做。

- Feign 中本身已经集成了 Ribbon 依赖和自动配置，因此不需要额外引入依赖，也不需要再注册 RestTemplate 对象。

- Feign 默认集成 Hystrix，只不过默认情况下是关闭的，需要通过参数开启。

- 注意：

1、首先这是一个接口，Feign 会通过动态代理，帮我们生成实现类。这点跟 mybatis 的 mapper 很像；

2、@FeignClient，声明这是一个 Feign 客户端，同时通过 value 属性指定服务名称；

3、接口中的定义方法，完全采用 SpringMVC 注解，Feign 会根据注解生成 URL，并访问获取结果；

4、@GetMapping 中的/user，请不要忘记；因为 Feign 需要拼接可访问的地址。

Spring Cloud Gateway | 网关

———— Spring Cloud Gateway 是加在整个微服务最前沿的防火墙和代理器。

- Spring Cloud Gateway 组件的核心是一系列的过滤器，通过这些过滤器可以将客户端发送的请求转发（路由）到对应的微服务。

- Spring Cloud Gateway 通过隐藏微服务结点 IP 端口信息，从而加强安全保护。其本身也是一个微服务，需要注册到 Eureka 服务注册中心。

- 网关的核心功能是：过滤和路由。

- 不管是来自于客户端（PC 或移动端）的请求，还是服务内部调用。一切对服务的请求都可经过网关，然后再由网关来实现鉴权、动态路由等等操作。Gateway 就是我们服务的统一入口。

- 核心概念：

1、路由（route） 路由信息的组成：由一个 ID、一个目的 URL、一组断言工厂、一组 Filter 组成。如果路由断言为真，说明请求 URL 和配置路由匹配。

2、断言（Predicate）：Spring Cloud Gateway 中的断言函数输入类型是 Spring 5.0 框架中的 ServerWebExchange。断言函数允许开发者去定义匹配来自于 Http Request 中的任何信息比如请求头和参数。

3、过滤器（Filter）：一个标准的 Spring WebFilter。Spring Cloud Gateway 中的 Filter 分为两种类型的 Filter，分别是 Gateway Filter 和 Global Filter。过滤器 Filter 将会对请求和响应进行修改处理。

Spring Cloud Config | 分布式配置中心

- 在分布式系统中，由于服务数量非常多，配置文件分散在不同的微服务项目中，管理不方便。为了方便配置文件集中管理，需要分布式配置中心组件。在 Spring Cloud 中，提供了 Spring Cloud Config，它支持配置文件放在配置服务的本地，也支持放在远程 Git 仓库（GitHub、码云）。

Spring Cloud Bus | 服务总线

———— Spring Cloud 消息代理服务总线。

- Spring Cloud Bus 是用轻量的消息代理将分布式的节点连接起来，可以用于广播配置文件的更改或者服务的监控管理。也就是消息总线可以为微服务做监控，也可以实现应用程序之间相互通信。 Spring Cloud Bus 可选的消息代理有 RabbitMQ 和 Kafka。

- 通过查看用户微服务控制台的输出结果可以发现，我们对于 Git 仓库中配置文件的修改并没有及时更新到用户微服务，只有重启用户微服务才能生效。

- 不重启微服务的情况下更新配置该如何实现呢？可以使用 Spring Cloud Bus 来实现配置的自动更新。

- 注意：Spring Cloud Bus 底层是基于 RabbitMQ 实现的，默认使用本地的消息队列服务，所以需要提前启动本地 RabbitMQ 服务（安装 RabbitMQ 以后才有）。

ElasticSearch | 搜索引擎

———— Elaticsearch，简称 “es”， es 是一个开源的高扩展的分布式全文检索引擎。

- 它可以近乎实时的存储、检索数据；本身扩展性很好，可以扩展到上百台服务器，处理 PB 级别的数据。es 也使用 Java 开发并使用 Lucene 作为其核心来实现所有索引和搜索的功能，但是它的目的是通过简单的 RESTful API 来隐藏 Lucene 的复杂性，从而让全文搜索变得简单。

Spring Data | 框架

———— Spring Data 是一个用于简化数据库访问，并支持云服务的开源框架。

- 其主要目标是使得对数据的访问变得方便快捷，并支持 map-reduce 框架和云计算数据服务。 Spring Data 可以极大的简化 JPA 的写法，可以在几乎不用写实现的情况下，实现对数据的访问和操作。除了 CRUD 外，还包括如分页、排序等一些常用的功能。

Spring Data ElasticSearch | 集成搜索引擎

- Spring Data ElasticSearch 基于 spring data API 简化 elasticSearch 操作，将原始操作 elasticSearch 的客户端 API 进行封装。Spring Data 为 Elasticsearch 项目提供集成搜索引擎。Spring Data Elasticsearch POJO 的关键功能区域为中心的模型与 Elastichsearch 交互文档和轻松地编写一个存储库数据访问层。

MQ | 消息队列

———— MQ 全称为 Message Queue，消息队列是应用程序和应用程序之间的通信方法。

- 为什么使用 MQ?

在项目中，可将一些无需即时返回且耗时的操作提取出来，进行异步处理，而这种异步处理的方式大大的节省了服务器的请求响应时间，从而提高了系统的吞吐量。

- 开发中消息队列通常有如下应用场景：

1、任务异步处理

将不需要同步处理的并且耗时长的操作由消息队列通知消息接收方进行异步处理。提高了应用程序的响应时间。

2、应用程序解耦合

MQ 相当于一个中介，生产方通过 MQ 与消费方交互，它将应用程序进行解耦合。

- AMQP 和 JMS

MQ 是消息通信的模型；实现 MQ 的大致有两种主流方式：AMQP、JMS。

1. AMQP

AMQP 是一种协议，更准确的说是一种 binary wire-level protocol（链接协议）。这是其和 JMS 的本质差别，AMQP 不从 API 层进行限定，而是直接定义网络交换的数据格式。

2. JMS

JMS 即 Java 消息服务（JavaMessage Service）应用程序接口，是一个 Java 平台中关于面向消息中间件（MOM）的 API，用于在两个应用程序之间，或分布式系统中发送消息，进行异步通信。

3. AMQP 与 JMS 区别

* JMS 是定义了统一的接口，来对消息操作进行统一；AMQP 是通过规定协议来统一数据交互的格式；

* JMS 限定了必须使用 Java 语言；AMQP 只是协议，不规定实现方式，因此是跨语言的；

* JMS 规定了两种消息模式，而 AMQP 的消息模式更加丰富。

RabbitMQ | 基于 AMQP 协议的消息队列

———— RabbitMQ 是由 erlang 语言开发，基于 AMQP（Advanced Message Queue 高级消息队列协议）协议实现的消息队列。

- 它是一种应用程序之间的通信方法，消息队列在分布式系统开发中应用非常广泛。

- RabbitMQ 提供了 6 种模式：

1、简单模式

2、work 模式

3、Publish/Subscribe 发布与订阅模式

4、Routing 路由模式

5、Topics 主题模式

6、RPC 远程调用模式

Node.js | JavaScript 运行时环境

———— Node.js 是一个可以在 js 中接收和处理 web 请求的应用平台，是 JavaScript 的运行环境。

- 简单的说 Node.js 就是运行在服务端的 JavaScript。

- Node.js 是一个基于 Chrome JavaScript 运行时建立的一个平台。

- Node.js 是一个事件驱动 I/O 服务端 JavaScript 环境，基于 Google 的 V8 引擎，V8 引擎执行 Javascript 的速度非常快，性能非常好。

ES6 | JavaScript 语法规范

———— ES6 即 “ECMAScript 第 6 版标准”，是 JavaScript 的语法规范。

- 编程语言 JavaScript 是 ECMAScript 的实现和扩展 。ECMAScript 是由 ECMA（一个类似 W3C 的标准组织）参与进行标准化的语法规范。

- ECMAScript 是前端 js 的语法规范；可以应用在各种 js 环境中。如：浏览器或者 node.js 环境。

Docker | 开源应用容器

———— Docker 是一个开源的应用容器引擎，基于 Go 语言开发。

- Docker 可以让开发者打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的 Linux 机器上，也可以实现虚拟化。容器是完全使用沙箱机制，相互之间不会有任何接口（类似 iPhone 的 app）,更重要的是容器性能开销极低。

- 使用 Docker 可以实现开发人员的开发环境、测试人员的测试环境、运维人员的生产环境的一致性。

- Docker 容器是在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统虚拟机则是在硬件层面实现虚拟化。与传统的虚拟机相比，Docker 优势体现为启动速度快、占用体积小。

- Docker 应用场景：

  1、Web 应用的自动化打包和发布；

  2、自动化测试和持续集成、发布；

3、在服务型环境中部署和调整数据库或其他的后台应用等。

Docker Compose | 多容器管理工具

———— Compose 是一个定义和运行多容器的 docker 应用工具，负责实现对 Docker 容器集群的快速编排。

- Compose 项目是 Docker 官方的开源项目，使用 compose，你能通过 YMAL 文件配置你自己的服务，然后通过一个命令，你能使用配置文件创建和运行所有的服务。

- 组成：Docker-Compose 将所管理的容器分为三层，分别是工程（project），服务（service）以及容器（container）。Docker-Compose 运行目录下的所有文件（docker-compose.yml，extends 文件或环境变量文件等）组成一个工程，若无特殊指定工程名即为当前目录名。一个工程当中可包含多个服务，每个服务中定义了容器运行的镜像、参数、依赖。一个服务当中可包括多个容器实例。

- 服务（service）：一个应用的容器，实际上可以包括若干运行相同镜像的容器实例。每个服务都有自己的名字、使用的镜像、挂载的数据卷、所属的网络、依赖哪些其他服务等等，即以容器为粒度，用户需要 Compose 所完成的任务。

- 项目（project）：由一组关联的应用容器组成的一个完成业务单元，在 docker-compose.yml 中定义。即是 Compose 的一个配置文件可以解析为一个项目，Compose 通过分析指定配置文件，得出配置文件所需完成的所有容器管理与部署操作。

- Docker-Compose 的工程配置文件默认为 docker-compose.yml，可通过环境变量 COMPOSE_FILE 或-f 参数自定义配置文件，其定义了多个有依赖关系的服务及每个服务运行的容器。

- 使用一个 Dockerfile 模板文件，可以让用户很方便的定义一个单独的应用容器。在工作中，经常会碰到需要多个容器相互配合来完成某项任务的情况。例如：要部署一个 Web 项目，除了 Web 服务容器，往往还需要再加上后端的数据库服务容器，甚至还包括负载均衡容器等。

CORS | 跨域解决方案

———— CORS 是一个 W3C 标准，全称是 "跨域资源共享"（Cross-origin resource sharing）。

- CORS 需要浏览器和服务器同时支持。目前 f，所有浏览器都支持该功能，IE 浏览器不能低于 IE10。它允许浏览器向跨源服务器，发出 XMLHttpRequest 请求，从而克服了 AJAX 只能同源使用的限制。整个 CORS 通信过程，都是浏览器自动完成，不需要用户参与。对于开发者来说，CORS 通信与同源的 AJAX 通信没有差别，代码完全一样。浏览器一旦发现 AJAX 请求跨源，就会自动添加一些附加的头信息，有时还会多出一次附加的请求，但用户不会有感觉。因此，实现 CORS 通信的关键是服务器。只要服务器实现了 CORS 接口，就可以跨源通信。

FastDFS | 分布式文件存储系统

———— FastDFS 是一个开源的轻量级分布式文件系统。

- FastDFS 对文件进行管理，功能包括：文件存储、文件同步、文件访问（文件上传、文件下载）等，解决了大容量存储和负载均衡的问题。特别适合以文件为载体的在线服务，如相册网站、视频网站等等。

- FastDFS 为互联网量身定制，充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用 FastDFS 很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

- FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 进行文件上传、下载，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。

- Tracker server 的作用是负载均衡和调度，通过 Tracker server 在文件上传时可以根据一些策略找到 Storage server 提供文件上传服务。可以将 tracker 称为追踪服务器或调度服务器。Storage server 作用是文件存储，客户端上传的文件最终存储在 Storage 服务器上，Storageserver 没有实现自己的文件系统而是利用操作系统的文件系统来管理文件。可以将 storage 称为存储服务器。

BCrypt | 加密算法

———— 一种比 MD5 更安全的加密算法。

- 目前，MD5 和 BCrypt 比较流行。相对来说，BCrypt 比 MD5 更安全，因为其内部引入了【加盐】机制。

Base64 | 编码

———— Base64 是网络上最常见的用于传输 8Bit 字节代码的编码方式之一。

- Base64 编码可用于在 HTTP 环境下传递较长的标识信息。采用 Base64 编码解码具有不可读性，即所编码的数据不会被人用肉眼所直接看到。注意：Base64 只是一种编码方式，不算加密方法。

JWT | 跨域认证解决方案

———— JSON Web Tokens，是目前流行的跨域认证解决方案，是一种基于 JSON 的、用于在网络上声明某种主张的令牌（token）。可利用 JWT 实现微服务的鉴权。

- JWT 原理

jwt 验证方式是将用户信息通过加密生成 token，每次请求服务端只需要使用保存的密钥验证 token 的正确性，不用再保存任何 session 数据了，进而服务端变得无状态，容易实现拓展。

- JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用 JWT 在用户和服务器之间传递安全可靠的信息。

- 组成：一个 JWT 实际上就是一个字符串，它由三部分组成：头部、载荷与签名。

1、头部（Header）

* 头部用于描述关于该 JWT 的最基本的信息，例如其类型以及签名所用的算法等。

2、载荷（playload）

* 载荷就是存放有效信息的地方。

3、签证（signature）

* jwt 的第三部分是一个签证信息，这个签证信息由三部分组成：

1、header (base64 后的)

2、payload (base64 后的)

3、secret

* 这个部分需要 base64 加密后的 header 和 base64 加密后的 payload 使用.连接组成的字符串，然后通过 header 中声明的加密方式进行加盐 secret 组合加密，然后就构成了 jwt 的第三部分。

- 将这三部分用 "." 连接成一个完整的字符串，构成了最终的 jwt，如：

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

JJWT | JWT 框架

———— JJWT（Java JWT）是一个提供端到端的 JWT 创建和验证的 Java 库。永远免费和开源。

- JJWT 适用于 Java 和 Android 的 JSON Web 令牌。

- JJWT 旨在成为最容易使用和理解的库，用于在 JVM 和 Android 上创建和验证 JSON Web 令牌（JWT）。

- JJWT 很容易使用和理解。它被设计成一个以建筑为中心的流畅界面，隐藏了它的大部分复杂性。

- 解析 token

* 在 web 应用中这个操作是由服务端进行然后发给客户端，客户端在下次向服务端发送请求时需要携带这个 token（这就好像是拿着一张门票一样），那服务端接到这个 token 应该解析出 token 中的信息（例如用户 id）,根据这些信息查询数据库返回相应的结果。

* 试着将 token 或签名秘钥篡改一下，会发现运行时就会报错，所以解析 token 也就是验证 token.

- 举例：

1. 用户进入网关开始登陆，网关过滤器进行判断，如果是登录，则路由到后台管理微服务进行登录；

2. 用户登录成功，后台管理微服务签发 JWT TOKEN 信息返回给用户；

3. 用户再次进入网关开始访问，网关过滤器接收用户携带的 TOKEN；

4. 网关过滤器解析 TOKEN，判断是否有权限，有则放行，没有则返回未认证错误。

Snowflake | 开源算法

———— snowflake 是 Twitter 开源的分布式 ID 生成算法，结果是一个 long 型的 ID。

- 核心思想：使用 41bit 作为毫秒数，10bit 作为机器的 ID（5 个 bit 是数据中心，5 个 bit 的机器 ID），12bit 作为毫秒内的流水号（意味着每个节点在每毫秒可以产生 4096 个 ID），最后还有一个符号位，永远是 0。

Lua | 脚本语言

———— Lua 是一个小巧的脚本语言。

- 它是巴西里约热内卢天主教大学里的一个由三人所组成的研究小组于 1993 年开发的。其设计目的是为了通过灵活嵌入应用程序中从而为应用程序提供灵活的扩展和定制功能。Lua 由标准 C 编写而成，几乎在所有操作系统和平台上都可以编译，运行。Lua 并没有提供强大的库，这是由它的定位决定的。所以 Lua 不适合作为开发独立应用程序的语言。Lua 有一个同时进行的 JIT 项目，提供在特定平台上的即时编译功能。

- 简单来说：Lua 是一种轻量小巧的脚本语言，用标准 C 语言编写并以源代码形式开放，其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。

OpenResty | 基于 NGINX 的 Web 平台

———— OpenResty（又称：ngx_openresty）是一个基于 NGINX 的可伸缩的 Web 平台，由中国人章亦春发起，提供了很多高质量的第三方模块。

- OpenResty 是一个强大的 Web 应用服务器，Web 开发人员可以使用 Lua 脚本语言调动 Nginx 支持的各种 C 以及 Lua 模块，更主要的是在性能方面，OpenResty 可以快速构造出足以胜任 10K 乃至 1000K 以上并发连接响应的超高性能 Web 应用系统。

- 360，UPYUN，阿里云，新浪，腾讯网，去哪儿网，酷狗音乐等都是 OpenResty 的深度用户。

- OpenResty 简单理解，就相当于封装了 nginx，并且集成了 LUA 脚本，开发人员只需要简单的其提供了模块就可以实现相关的逻辑，而不再像之前，还需要在 nginx 中自己编写 lua 的脚本，再进行调用了。

Canal | 数据同步解决方案

———— canal 可以用来监控数据库数据的变化，从而获得新增数据，或者修改的数据。

- canal 是应阿里巴巴存在杭州和美国的双机房部署，存在跨机房同步的业务需求而提出的。阿里系公司开始逐步的尝试基于数据库的日志解析，获取增量变更进行同步，由此衍生出了增量订阅&消费的业务。

- 数据监控微服务

  * 当用户执行数据库的操作的时候，binlog 日志会被 canal 捕获到，并解析出数据。我们就可以将解析出来的数据进行相应的逻辑处理。

Thymeleaf | 静态模板引擎

———— Thymeleaf 是一个 XML/XHTML/HTML5 模板引擎，可用于 Web 与非 Web 环境中的应用开发。

- 它是一个开源的 Java 库，Thymeleaf 提供了一个用于整合 Spring MVC 的可选模块，Thymeleaf 的主要目标在于提供一种可被浏览器正确显示的、格式良好的模板创建方式，因此也可以用作静态建模。你可以使用它创建经过验证的 XML 与 HTML 模板。相对于编写逻辑或代码，开发者只需将标签属性添加到模板中即可。接下来，这些标签属性就会在 DOM（文档对象模型）上执行预先制定好的逻辑。

- 特点：开箱即用，Thymeleaf 允许您处理六种模板，每种模板称为模板模式：

1、XML

2、有效的 XML

3、XHTML

4、有效的 XHTML

5、HTML5

6、旧版 HTML5

SSO | 单点登录

———— 单点登录（Single Sign On），简称为 SSO，是目前比较流行的企业业务整合的解决方案之一。

- SSO 的定义是在多个应用系统中，用户只需要登录一次就可以访问所有相互信任的应用系统。

Oauth2 | 授权协议

———— Oauth2 是一个标准的开放的授权协议。

- 项目中使用 Oauth2 可实现如下功能：

1、本系统访问第三方系统的资源；

2、外部系统访问本系统的资源；

3、本系统前端（客户端）访问本系统后端微服务的资源；

4、本系统微服务之间访问资源，例如：微服务 A 访问微服务 B 的资源，B 访问 A 的资源。

Spring security Oauth2 | 认证解决方案

———— Spring security 是一个强大的和高度可定制的身份验证和访问控制框架。

- Spring security 框架集成了 Oauth2 协议。

- 项目认证架构图：

1、用户请求认证服务完成认证；

2、认证服务下发用户身份令牌，拥有身份令牌表示身份合法；

3、用户携带令牌请求资源服务，请求资源服务必先经过网；

4、网关校验用户身份令牌的合法，不合法表示用户没有登录，如果合法则放行继续访问；

5、资源服务获取令牌，根据令牌完成授权；

6、资源服务完成授权则响应资源信息。

- Oauth2 有以下授权模式：

1.授权码模式（Authorization Code）

2.隐式授权模式（Implicit）

3.密码模式（Resource Owner Password Credentials）

4.客户端模式（Client Credentials）

http Basic | 认证方式

———— http 协议定义的一种认证方式。

- 将‘客户端 id’和‘客户端密码’按照：“客户端 ID：客户端密码” 的格式拼接，并用 base64 编码，放在 header 中请求服务端。