## 分布式Session一致性问题

### 什么是Session

Session 是客户端与服务器通讯会话技术， 比如浏览器登陆、记录整个浏览会话信息

### Session实现原理

客户对向服务器端发送请求后，Session 创建在服务器端，返回Sessionid给客户端浏览器保存在本地，当下次发送请求的时候，在请求头中传递sessionId获取对应的从服务器上获取对应的Sesison

### Session常见问题

#### Session 保证在那里？

答案：存放在服务器上

#### 关闭浏览器Session会失效吗

答案：不会消失

#### 相关代码

|  |
| --- |
| @SpringBootApplication  @RestController  **public** **class** TestSessionController {  // 创建session 会话  @RequestMapping("/createSession")  **public** String createSession(HttpServletRequest request, String nameValue) {  HttpSession session = request.getSession();  System.***out***.println("存入Session sessionid:信息" + session.getId() + ",nameValue:" + nameValue);  session.setAttribute("name", nameValue);  **return** "success";  }  // 获取session 会话  @RequestMapping("/getSession")  **public** Object getSession(HttpServletRequest request) {  HttpSession session = request.getSession();  System.***out***.println("获取Session sessionid:信息" + session.getId());  Object value = session.getAttribute("name");  **return** value;  }  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(TestSessionController.**class**, args);  }  } |

### 服务集群会产生那些问题

如果服务器产生了集群后，因为session是存放在服务器上，客户端会使用同一个Sessionid在多个不同的服务器上获取对应的Session，从而会导致Session不一致问题。

#### Nginx配置负载均衡

Nginx负载均衡提供上游服务器(真实业务逻辑访问的服务器),负载均衡、故障转移、失败重试、容错、健康检查等。

当上游服务器(真实业务逻辑访问的服务器)发生故障时，可以转移到其他上游服务器(真实业务逻辑访问的服务器)。

#### Upstream Server配置

upstream 主要配置如下:

IP地址和端口号：配置上游服务器的IP地址和端口

|  |
| --- |
| ###定义上游服务器(需要被nginx真实代理访问的服务器) 默认是轮训机制  upstream backServer{  server 127.0.0.1:8080;  server 127.0.0.1:8081;  }    server {  listen 80;  server\_name www.itmayiedu.com;  location / {  ### 指定上游服务器负载均衡服务器  proxy\_pass http://backServer;  index index.html index.htm;  }  } |

#### 负载均衡算法

1、轮询（默认）

每个请求按时间顺序逐一分配到不同的后端服务，如果后端某台服务器死机，自动剔除故障系统，使用户访问不受影响。

2、weight（轮询权值）

weight的值越大分配到的访问概率越高，主要用于后端每台服务器性能不均衡的情况下。或者仅仅为在主从的情况下设置不同的权值，达到合理有效的地利用主机资源。

3、ip\_hash

每个请求按访问IP的哈希结果分配，使来自同一个IP的访客固定访问一台后端服务器，并且可以有效解决动态网页存在的session共享问题。俗称IP绑定。

4、fair（第三方）

比 weight、ip\_hash更加智能的负载均衡算法，fair算法可以根据页面大小和加载时间长短智能地进行负载均衡，也就是根据后端服务器的响应时间 来分配请求，响应时间短的优先分配。Nginx本身不支持fair，如果需要这种调度算法，则必须安装upstream\_fair模块。

5、url\_hash(第三方)

按访问的URL的哈希结果来分配请求，使每个URL定向到一台后端服务器，可以进一步提高后端缓存服务器的效率。Nginx本身不支持url\_hash，如果需要这种调度算法，则必须安装Nginx的hash软件包。

|  |
| --- |
| @SpringBootApplication  @RestController  **public** **class** TestSessionController {  @Value("${server.port}")  **private** String serverPort;  @RequestMapping("/")  **public** String index() {  **return** serverPort;  }  // 创建session 会话  @RequestMapping("/createSession")  **public** String createSession(HttpServletRequest request, String nameValue) {  HttpSession session = request.getSession();  System.***out***.println(  "存入Session sessionid:信息" + session.getId() + ",nameValue:" + nameValue + ",serverPort:" + serverPort);  session.setAttribute("name", nameValue);  **return** "success-" + serverPort;  }  // 获取session 会话  @RequestMapping("/getSession")  **public** Object getSession(HttpServletRequest request) {  HttpSession session = request.getSession(**false**);  **if** (session == **null**) {  **return** serverPort + "-" + "没有找到对应的session值";  }  System.***out***.println("获取Session sessionid:信息" + session.getId() + "serverPort:" + serverPort);  Object value = session.getAttribute("name");  **return** serverPort + "-" + value;  }  **public** **static** **void** main(String[] args) {  SpringApplication.*run*(TestSessionController.**class**, args);  }  } |

### 分布式Session一致性解决方案

#### nginx或者haproxy实现IP绑定

用Nginx 做的负载均衡可以添加ip\_hash这个配置，

用haproxy做的负载均衡可以用 balance source这个配置。

从而使同一个ip的请求发到同一台服务器。

#### 利用数据库同步session

#### 使用Session集群存放Redis

使用spring-session框架，底层实现原理是重写httpsession

##### 引入maven依赖

|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.0.RELEASE</version>  <relativePath /> <!-- lookup parent from repository -->  </parent>  <properties>  <weixin-java-mp.version>2.8.0</weixin-java-mp.version>  <maven.compiler.source>1.8</maven.compiler.source>  <maven.compiler.target>1.8</maven.compiler.target>  <maven.compiler.encoding>UTF-8</maven.compiler.encoding>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.build.locales>zh\_CN</project.build.locales>  </properties>  <dependencies>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <!-- <exclusions> <exclusion> <groupId>com.fasterxml.jackson.core</groupId>  <artifactId>jackson-databind</artifactId> </exclusion> </exclusions> -->  </dependency>  <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->  <dependency>  <groupId>com.alibaba</groupId>  <artifactId>fastjson</artifactId>  <version>1.2.47</version>  </dependency>  <!-- Testing Dependencies -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-redis</artifactId>  </dependency>  <!--spring session 与redis应用基本环境配置,需要开启redis后才可以使用，不然启动Spring boot会报错 -->  <dependency>  <groupId>org.springframework.session</groupId>  <artifactId>spring-session-data-redis</artifactId>  </dependency>  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-pool2</artifactId>  </dependency>  <dependency>  <groupId>redis.clients</groupId>  <artifactId>jedis</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.apache.maven.plugins</groupId>  <artifactId>maven-compiler-plugin</artifactId>  <configuration>  <source>1.8</source>  <target>1.8</target>  </configuration>  </plugin>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  <configuration>  <maimClass>com.meiteedu.WxMpApplication</maimClass>  </configuration>  <executions>  <execution>  <goals>  <goal>repackage</goal>  </goals>  </execution>  </executions>  </plugin>  </plugins>  </build> |

##### YML配置信息

|  |
| --- |
| server:  port: 8080  redis:  hostname: 192.168.212.151  port: 6379  password: 123456 |

启动redis /usr/local/redis/bin/redis-server /usr/local/redis/etc/redis.conf

##### 创建SessionConfig

|  |
| --- |
| **import org.springframework.beans.factory.annotation.Value;**  **import org.springframework.context.annotation.Bean;**  **import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;**  **import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;**  **//这个类用配置redis服务器的连接**  **//maxInactiveIntervalInSeconds为SpringSession的过期时间（单位：秒）**  **@EnableRedisHttpSession(maxInactiveIntervalInSeconds = 1800)**  **public class SessionConfig {**  **// 冒号后的值为没有配置文件时，制动装载的默认值**  **@Value("${redis.hostname:localhost}")**  **String HostName;**  **@Value("${redis.port:6379}")**  **int Port;**  **@Bean**  **public JedisConnectionFactory connectionFactory() {**  **JedisConnectionFactory connection = new JedisConnectionFactory();**  **connection.setPort(Port);**  **connection.setHostName(HostName);**  **return connection;**  **}**  **}** |

##### 初始化Session

|  |
| --- |
| **//初始化Session配置**  **public class SessionInitializer extends AbstractHttpSessionApplicationInitializer{**  **public SessionInitializer() {**  **super(SessionConfig.class);**  **}**  **}** |

#### 最靠谱的分布式Session解决方案

基于令牌（Token）方式实现Session解决方案，因为Session本身就是分布式共享连接。

|  |
| --- |
| @Service  **public** **class** TokenService {  @Autowired  **private** RedisService redisService;  // 新增 返回token  **public** String put(Object object) {  String token = getToken();  redisService.setString(token, object);  **return** token;  }  // 获取信息  **public** String get(String token) {  String reuslt = redisService.getString(token);  **return** reuslt;  }  **public** String getToken() {  **return** UUID.*randomUUID*().toString();  }  } |

TokenController

|  |
| --- |
| @RestController  **public** **class** TokenController {  @Autowired  **private** TokenService tokenService;  @Value("${server.port}")  **private** String serverPort;  @RequestMapping("/put")  **public** String put(String nameValue) {  String token = tokenService.put(nameValue);  **return** token + "-" + serverPort;  }  @RequestMapping("/get")  **public** String get(String token) {  String value = tokenService.get(token);  **return** value + "-" + serverPort;  }  } |

## 分布式锁解决方案

## 分布式任务调度平台

## 分布式日志收集系统

## 分布式事务解决方案

## 网站跨域解决方案