necos

版本信息：
SpringBoot：2.1.4.RELEASE
nacos: 2.0.3

下载necos

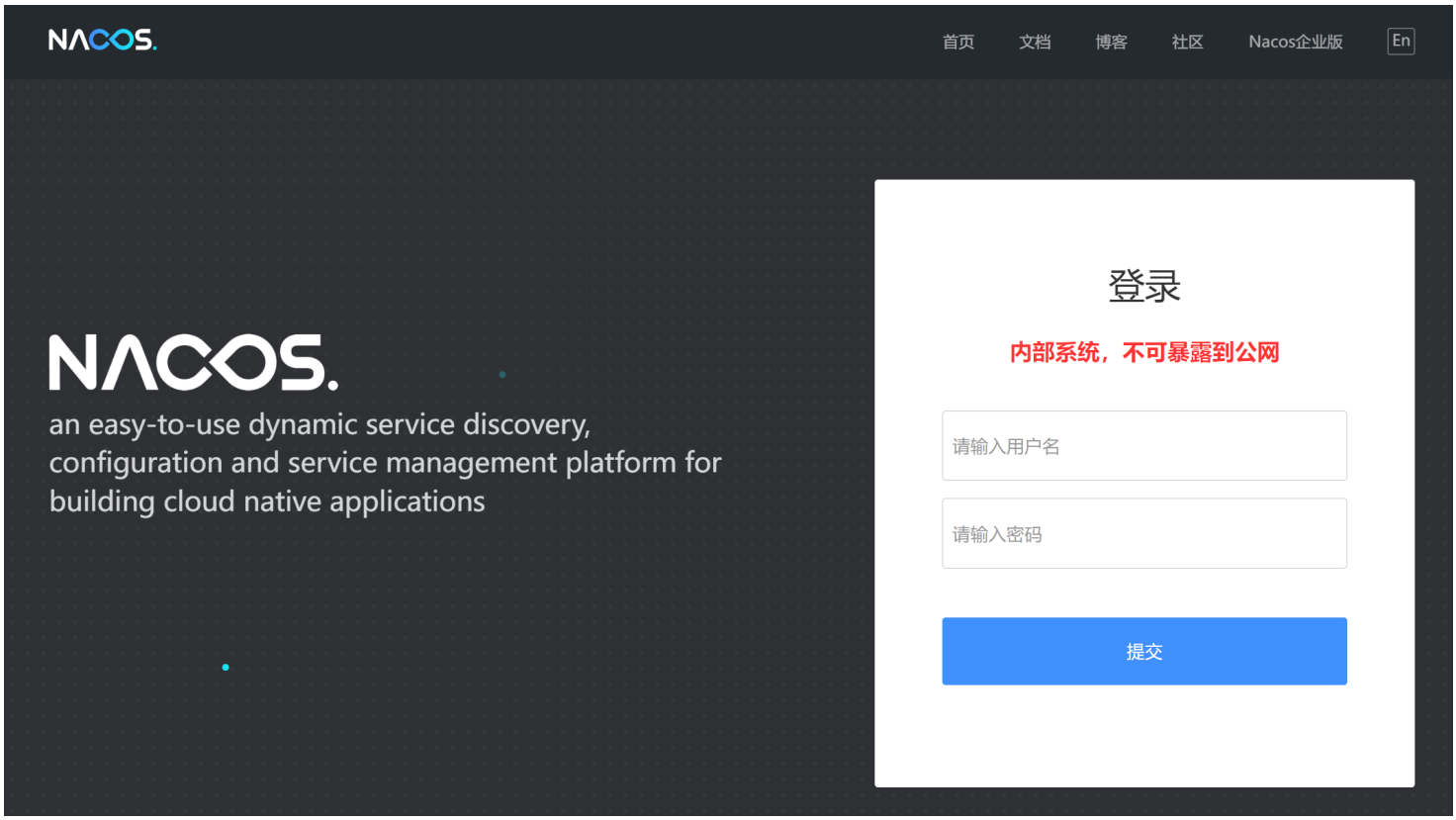
下载地址

necos版本2.0.3

启动服务器

进入 `bin` 目录,以单机模式运行 `startup.cmd -m standalone`

浏览器访问 `localhost:8848/nacos` 即可看到页面



报错:

1.
```
D:\OneDrive - 东南大学\实习\单点登录\nacos-server-2.0.3(1)\nacos\bin>startup.cmd -m standalone
"nacos is starting with standalone"
此时不应有 \nacos"\logs\java_heapdump.hprof -XX:-UseLargePages"。
```

解决

注释掉startup.cmd中下图中代码

```
if %MODE% == "cluster" (
    echo "nacos is starting with cluster"
      if %EMBEDDED_STORAGE% == "embedded" (
          set "NACOS_OPTS=-DembeddedStorage=true"
      )

    rem set "NACOS_JVM_OPTS=-server -Xms2g -Xmx2g -Xmn1g -XX:MetaspaceSize=128m -XX:MaxMetaspaceSize=320m -XX:-OmitStackTraceInFastThrow -XX:+HeapDumpOnOutOfMemoryE
)
```

2. 2022-08-09 12:56:54,379 ERROR Nacos failed to start, please see D:\nacos-server-2.0.3(1)\nacos\logs\nacos.log for more details.

解决
将startup.cmd中修改set MODE="standalone"

与SpringBoot结合

pom.xml中添加依赖

properties

```
<properties>
    <nacos-config-spring-boot.version>0.2.1</nacos-config-spring-boot.version>
</properties>
```

dependencies

```
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-test</artifactId>
                <scope>test</scope>
        </dependency>


        <dependency>
                <groupId>com.alibaba.cloud</groupId>
                <artifactId>spring-cloud-alibaba-dependencies</artifactId>
                <version>2.2.5.RELEASE</version>
                <type>pom</type>
                <scope>import</scope>
        </dependency>

        <dependency>
                <groupId>com.alibaba.boot</groupId>
                <artifactId>nacos-config-spring-boot-starter</artifactId>
                <version>${nacos-config-spring-boot.version}</version>
        </dependency>

        <dependency>
                <groupId>com.alibaba.boot</groupId>
                <artifactId>nacos-config-spring-boot-actuator</artifactId>
                <version>${nacos-config-spring-boot.version}</version>
        </dependency>
```

服务注册

在工程的配置文件 `application.yml` 做相关的配置，配置如下：

```
server:
  port: 8081
spring:
  application:
    name: nacos-provider
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848
```

然后在Spring Boot的启动文件 `NacosProviderApplication` 加上 `@EnableDiscoveryClient` 注解，代码如下：

```
// nacos-provider NacosProviderApplication.java
@SpringBootApplication
@EnableDiscoveryClient
public class NacosProviderApplication {

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

通过以上方法可以注册nacos-provider和nacos-consumer两个服务。

服务调用

提供服务

在nacos-provider工程，写一个Controller提供API服务，用来打印hi。

```
// nacos-provider ProviderController.java
@RestController
public class ProviderController {

Logger logger= LoggerFactory.getLogger(ProviderController.class);

@GetMapping("/hi")
public String hi(@RequestParam(value = "name",defaultValue = "forezp",required = false)String name){

        return "hi "+name;
    }
}
```

消费服务

在这里使用2种方式消费服务，一种是RestTemplate，一种是Feign。

RestTemplate服务

添加依赖

```
<!-->nacos-consumer pom.xml<-->
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-ribbon</artifactId>
</dependency>
```

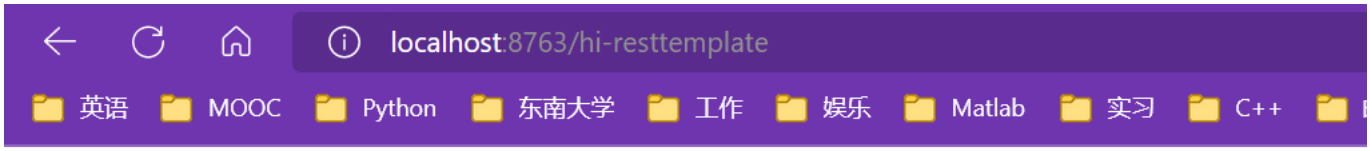
在 `NacosConsumerApplication` 启动文件注入 `RestTemplate` 的Bean:

```
// nacos-consumer NacosConsumerApplication.java
        @LoadBalanced
        @Bean
        public RestTemplate restTemplate(){
                return new RestTemplate();
        }
```

写一个消费服务的ConsumerController:

```
// nacos-consumer ConsumerController.java
@RestController
public class ConsumerController {

    @Autowired
    RestTemplate restTemplate;

 @GetMapping("/hi-resttemplate")
    public String hiResttemplate(){
        return restTemplate.getForObject("http://nacos-provider/hi?name=resttemplate",String.class);
    }
```

重启工程，在浏览器上访问 `http://localhost:8763/hi-resttemplate` ，可以在浏览器上展示正确的响应，这时nacos-consumer调用nacos-provider服务成功。



hi resttemplate consumer (from nacos-provider)

FeignClient服务

添加依赖

```
<!-->nacos-consumer pom.xml<-->
<dependency>
        <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

在NacosConsumerApplication启动文件上加上@EnableFeignClients注解开启FeignClient的功能。

```
// nacos-consumer NacosConsumerApplication.java
@EnableFeignClients
public class NacosConsumerApplication {

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

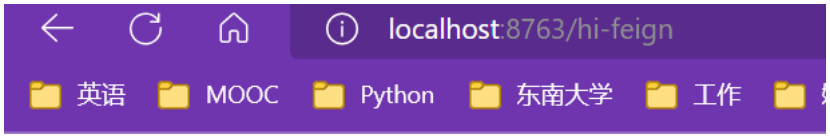
写一个FeignClient，调用nacos-provider的hi服务:

```
// nacos-consumer ProviderClient.java
@FeignClient("nacos-provider")
public interface ProviderClient {

    @GetMapping("/hi")
    String hi(@RequestParam(value = "name", defaultValue = "forezp", required = false) String name);
}
```

写一个消费API，该API使用ProviderClient来调用nacos-provider的API服务:

```
// nacos-consumer ConsumerController.java
@RestController
public class ConsumerController {


    @Autowired
    ProviderClient providerClient;

    @GetMapping("/hi-feign")
    public String hiFeign(){
       return providerClient.hi("feign");
    }
}
```

重启工程，在浏览器上访问 `http://localhost:8763/hi-feign` ，可以在浏览器上展示正确的响应，这时nacos-consumer调用nacos-provider服务成功.



hi feign (from nacos-provider)

Nacos作为配置中心

添加依赖

```
<!-->nacos-provider pom.xml<-->
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-alibaba-nacos-config</artifactId>
        <version>0.9.0.RELEASE</version>
</dependency>
```

在 `bootstrap.yml` (一定是 `bootstrap.yml` 文件，不是 `application.yml` 文件)文件配置:

```
<!-->nacos-provider bootstrap.yml<-->
spring:
  application:
    name: nacos-provider
  cloud:
    nacos:
      config:
        server-addr: 127.0.0.1:8848
        file-extension: yaml
        prefix: nacos-provider
  profiles:
    active: dev
```

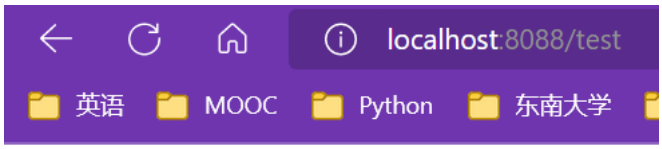
在nacos网站(`localhost:8848/nacos`)添加配置。
其中Data ID对应于bootstrap.yml中的 `${prefix}-${spring.profile.active}.${file-extension}` 。在上例中是 `nacos-provider-dev.yaml` 。
`Group` 默认为 `DEFAULT_GROUP` 。
配置格式选择 `YAML` 。
配置内容与 `ConfigController.java` 中的字段保持一致。

配置内容(?):

```
server.port: 8088
username: LIU
x: 30
test: Test!
```

```
// ConfigController.java
@RestController
@RefreshScope
public class ConfigController {
    @Value("${username:Luck}")
    private String username;

    @Value("${x:20}")
    private Integer x;

    @Value("${test:No}")
    private String test;

    @RequestMapping("/username")
    public String get() {
        return username;
    }

    @RequestMapping("/x")
    public Integer get1() {return x;}

    @RequestMapping("/test")
    public String get2() {return test;}
}
```

访问 `localhost:8088/username` 可以得到对应字段的值。



Test!

SpringBoot连接MySQL数据库

添加依赖

向 `pom.xml` 中添加MySQL、JDBC、mybatis-plus依赖。

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.29</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>

<dependency>
  <groupId>com.baomidou</groupId>
  <artifactId>mybatis-plus-boot-starter</artifactId>
  <version>3.5.1</version>
</dependency>
```

连接MySQL数据库

在配置文件(yml或properties文件)中连接数据库

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    type: com.zaxxer.hikari.HikariDataSource
    url: jdbc:mysql://localhost:3306/nacos?useUnicode=true&characterEncoding=utf-8&useSSL=true
    username: root
    password:
```

测试

1.创建UserPo实体类

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@TableName("nacos1")
public class UserPO {
    @TableId(value = "id",type = IdType.AUTO)
    private int id;
    @TableField("name")
    private String name;
    @TableField("age")
    private int age;
}
```

2.在Mapper包下创建UserMapper

```
@Repository
public interface UserMapper extends BaseMapper<UserPO> {
}
```

3.在启动类增加注解

```
@SpringBootApplication
@MapperScan("com.forezp.Mapper")
public class NacosProviderApplication {
    public static void main(String[] args) {
        SpringApplication.run(NacosProviderApplication.class, args);
    }
}
```

4.测试

在测试类中查询所有用户

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class NacosProviderApplicationTests {
        @Autowired
        private UserMapper userMapper;

        @Test
        public void contextLoads() {
                for (UserPO userPO : userMapper.selectList(null)){
                        System.out.println(userPO.toString());
                }
        }
}
```

实现配置中心修改MySQL数据库中变量

nacos-config

Zuul网关

功能：
路由转发+过滤

- 动态网关路由
  1. 路由信息不再配置在配置文件中，将路由信息配置在Nacos的配置中。
  2. 在服务网关Spring Cloud Gateway中开启监听，监听Nacos配置文件的修改。
  3. Nacos配置文件一旦发生改变，则Spring Cloud Gateway重新刷新自己的路由信息。
- 身份验证

Filter过滤器

类型：

1. PRE：这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现身份验证、在集群中选择请求的微服务、记录调试信息等。
2. ROUTING：这种过滤器将请求路由到微服务。这种过滤器用于构建发送给微服务的请求，并使用Apache HttpClient或Netfilx Ribbon请求微服务。
3. POST：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的HTTPHeader、收集统计信息和指标、将响应从微服务发送给客户端等。
4. ERROR：在其他阶段发生错误时执行该过滤器。

流程：

1. 正常流程：
   请求到达首先会经过pre类型过滤器，而后到达routing类型，进行路由，请求就到达真正的服务提供者，执行请求，返回结果后，会到达post过滤器。而后返回响应。

2. 异常流程：
   整个过程中，pre或者routing过滤器出现异常，都会直接进入error过滤器，再error处理完毕后，会将请求交给POST过滤器，最后返回给用户。
   如果是error过滤器自己出现异常，最终也会进入POST过滤器，而后返回。如果是POST过滤器出现异常，会跳转到error过滤器，但是与pre和routing不同的是，请求不会再到达POST过滤器了。
3. 不同过滤器的场景：
   请求鉴权：一般放在pre类型，如果发现没有访问权限，直接就拦截了
   异常处理：一般会在error类型和post类型过滤器中结合来处理。
   服务调用时长统计：pre和post结合使用。

CAS-SSO

cas 服务端 （tomcat）

1. 建立 `tomcat https` 支持
2. 搭建cas server服务器
3. 进入cas登录页面

cas 客户端 （Spring Boot）

参考网址

修改 `HTTPSandIMAPS-10000001.json`文件 来使 `cas` 支持 `http`
参考网址

需要使用IE浏览器或Chrome打开CAS登录页面

- Cas服务端 `https://localhost:8443/cas/login`
- Cas客户端1 `http://127.0.0.1:8890/casTest2/user2`
- Cas客户端2 `http://127.0.0.1:9990/casTest3/user3`

报错:

1. `java: 程序包org.junit.jupiter.api不存在`
   解决:使用IDEA自带的将相关的依赖添加进Maven，并导入适当的包

2. `java.lang.IllegalArgumentException: no server available`
   解决

3. `org.apache.catalina.LifecycleException: Protocol handler start failed`
   端口复用，重启一下就能用。

4. `java.lang.IllegalStateException: failed to req API:/nacos/v1/ns/instance aft`
   打开 `startup.cmd` 就行

5. 主java程序中 `import org.springframework.boot.SpringApplication; import org.springframework.boot.autoconfigure.SpringBootApplication;` 导入包标红。
   在 `pom.xml` 中给 `SpringBoot` 插件添加版本信息(version)。

```
<build>
        <plugins>
                <plugin>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-maven-plugin</artifactId>
                        <version>2.6.10</version>
                </plugin>
        </plugins>
</build>
```

6. `java.lang.IllegalStateException: Error processing condition on org.springframework.boot.autoconfigure.task.TaskExecutionAutoConfiguration.taskE`
   去掉下述依赖

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot</artifactId>
        <version>2.0.4.RELEASE</version>
</dependency>
```

并去掉下述依赖的版本

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-autoconfigure</artifactId>
</dependency>
```

7. `org.codehaus.plexus.component.repository.exception.ComponentLookupException:`
   在IDEA设置中修改Maven主路径

8. `Non-resolvable import POM: Failure to find org.springframework.cloud:spring-cloud-dependencies:pom:${spring-cloud.version} in https://repo.mave`
   解决

9. 无法使用注解 `@EnableZuulProxy`
   `dependencymanagement` 和 `dependencies` 的区别

10. Zuul网关启动报错： `The bean 'proxyRequestHelper', defined in class path resource [org/springframework/cloud`
    `SpringBoot` 和 `SpringCloud` 版本不匹配
    解决

版本

- `jdk：1.8`
- `Maven：3.6.3`
- `SpringBoot：2.1.4.RELEASE`
- `SpringCloud：Greenwich.RELEASE`
- `nacos：2.0.3`
- `tomcat：8.5.81`
- `cas：5.3`

端口

- nacos 8848
- tomcat 8443