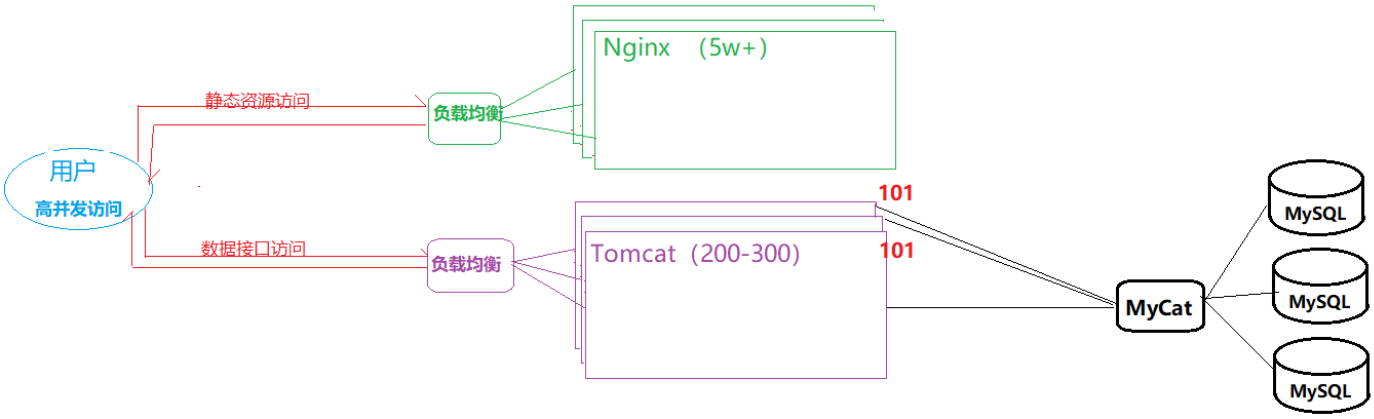
一、锋迷商城在互联网环境下存在的问题

> 一个成功的互联网项目，平台用户会不断的积累和提升，当用户量不断增多的过程中，同时使用平台系统的用户也会增多——并发访问



1.1 高并发带来的服务器访问压力问题

一个电商平台在互联网环境下首先要应对的是高并发访问的问题，高并发访问会为我们的系统带来哪些压力呢?

- 前端服务器并发访问压力
- 应用服务器并发访问压力
- 数据库服务器并发访问压力

1.2 高并发带来的业务处理问题

- 首页-数据加载效率 （redis缓存）
- 商品搜索功能—模糊查询—查询效率低问题
- 订单查询—订单的数据量会随着用户的增多大量增加—查询效率低问题
- 商品购买—分布并发访问—产品超卖问题 （redis分布式锁）

1.3 系统迭代带来的架构问题

- 一个成功的应用—系统功能不断的增加—单点故障问题

1.4 业务实现问题

- 登录功能—缓存用户信息、用户登录失效（redis 分布式会话—共享session）
- 订单超时取消—使用quartz定时任务进行轮询—性能

1.5 如何解决以上问题?

- Nginx集群\Tomcat集群 (✔)
- Redis缓存数据库(✔)
- ElasticSearch搜索引擎(✔)
- 分布式系统用户登录问题(✔)
- 分布式锁(✔)

- MyCat分布式数据库
- 微服务架构
- 分布式事务
- 消息队列（延时任务、服务通信）
- 容器化技术docker

- 优化（MySQ\Tomcat）

二、使用redis缓存商品详情

2.1 在service子工程添加Spring data redis依赖

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

2.2 在application.yml配置redis数据源

```yaml
spring:
  datasource:
    druid:
      driver-class-name: com.mysql.jdbc.Driver
      ## 如果后端项目服务器和数据库服务器不在同一台主机，则需要修改localhost为数据库服务器ip地址
      url: jdbc:mysql://localhost:3306/fmmall2?characterEncoding=utf-8
      username: root
      password: admin123
  redis:
    port: 6379
    host: 47.96.11.185
    database: 0
    password: 123456
```

2.3 在ProductServiceImpl中修改业务代码

products	1	{"productId":"101","productName":"111","productPrice":4.0}

productsimgs	1	[{ },{ },{ }]

productskus	1	[{},{},{}]

```java
@Service
public class ProductServiceImpl implements ProductService {

    @Autowired
    private ProductMapper productMapper;
    @Autowired
    private ProductImgMapper productImgMapper;
    @Autowired
    private ProductSkuMapper productSkuMapper;
    @Autowired
    private ProductParamsMapper productParamsMapper;

    @Autowired
    private StringRedisTemplate stringRedisTemplate;
    private ObjectMapper objectMapper = new ObjectMapper();
```

```java
@Transactional(propagation = Propagation.SUPPORTS)
public ResultVO getProductBasicInfo(String productId) {
    try{
        // ①根据商品id查询redis
        String productInfo = (String) stringRedisTemplate.boundHashOps( key: "products").get(productId);

        // ②如果reids中查询到了商品信息，则直接返回给控制器
        if(productInfo != null){
            Product product =objectMapper.readValue(productInfo, Product.class);
            //从redis中查询此商品的图片
            String imgsStr = (String) stringRedisTemplate.boundHashOps( key: "productImgs").get(productId);
            JavaType javaType1 = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, ProductImg.class);
            List<ProductImg> productImgs = objectMapper.readValue(imgsStr, javaType1);
            //从redis中查询此商品的套餐
            String skusStr = (String) stringRedisTemplate.boundHashOps( key: "productSkus").get(productId);
            JavaType javaType2 = objectMapper.getTypeFactory().constructParametricType(ArrayList.class, ProductSku.class);
            List<ProductSku> productSkus = objectMapper.readValue(skusStr, javaType2);
            //封装商品、图片及套餐
            HashMap<String,Object> basicInfo = new HashMap<>();
            basicInfo.put("product",product);
            basicInfo.put("productImgs",productImgs);
            basicInfo.put("productSkus",productSkus);
            return new ResultVO(ResStatus.OK, msg: "success",basicInfo);
        }else{
            //③如果redis中没有查询到商品信息，则查询数据库
            //商品基本信息
            Example example = new Example(Product.class);
            Example.Criteria criteria = example.createCriteria();
            criteria.andEqualTo( property: "productId",productId);
            criteria.andEqualTo( property: "productStatus", value: 1);//状态为1表示上架商品
            List<Product> products = productMapper.selectByExample(example);
            if(products.size() > 0 ){
                //④ 将从数据库查询的数据写入到redis
                Product product = products.get(0);
                String jsonStr = objectMapper.writeValueAsString(product);
                stringRedisTemplate.boundHashOps( key: "products").put(productId,jsonStr);   将商品信息存入到redis

                //根据商品id查询商品图片
                Example example1 = new Example(ProductImg.class);
                Example.Criteria criteria1 = example1.createCriteria();
                criteria1.andEqualTo( property: "itemId",productId);
                List<ProductImg> productImgs = productImgMapper.selectByExample(example1);
                stringRedisTemplate.boundHashOps( key: "productImgs").put(productId, objectMapper.writeValueAsString(productImgs) );   将商品图片存入到redis

                //根据商品id查询商品套餐
                Example example2 = new Example(ProductSku.class);
                Example.Criteria criteria2 = example2.createCriteria();
                criteria2.andEqualTo( property: "productId",productId);
                criteria2.andEqualTo( property: "status", value: 1);
                List<ProductSku> productSkus = productSkuMapper.selectByExample(example2);
                stringRedisTemplate.boundHashOps( key: "productSkus").put(productId,objectMapper.writeValueAsString(productSkus));   将商品套餐存入到redis

                HashMap<String,Object> basicInfo = new HashMap<>();
                basicInfo.put("product",products.get(0));
                basicInfo.put("productImgs",productImgs);
                basicInfo.put("productSkus",productSkus);
                return new ResultVO(ResStatus.OK, msg: "success",basicInfo);
            }
        }
    }catch (Exception e){
    }
    return new ResultVO(ResStatus.NO, msg: "fail", data: null);
}
```

说明：

在电商系统中，为了提高商品详情的查询速度、减少对数据库的并发访问压力，我们可以使用redis来缓存商品详情，除此以外我们还可以用 `页面静态化` 技术来达到此目的。

页面静态化：将数据库中每条数据结合模版生成单独的HTML文件进行存储（一条数据对应一个独立的HTML文件），当用户访问数据时，直接访问不同的静态HTML文件即可。

三、使用redis缓存平台首页数据

3.1 什么样的数据适合用缓存?

> 因为缓存中的数据需要进行数据一致性的维护—即：当数据库数据发生变化，要同步更新缓存中的数据
>
> 因此
>
> - 对于数据的写操作较少、但是会频繁的查询的数据适合使用缓存
> - 对于可能会发生修改，但是对数据一致性要求不高的数据也适合使用缓存

3.2 缓存首页轮播图信息

3.2 缓存首页分类列表信息

四、使用redis实现分布式会话

五、使用ES实现商品检索

5.1 在锋迷商城项目导入ES

- 在service子工程添加依赖

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

- 在api子工程的application.yml配置ES服务器地址

```yaml
spring:
  elasticsearch:
    rest:
      uris: http://47.96.11.185:9200
```

5.2 将商品信息导入到ES

> - 如果商品表中没有数据，则在平台管理系统中的商品添加功能中，当商家向商品表添加并上架一个商品时同步向ES添加一个商品；商家下架一个商品就从ES中删除一个商品。
> - 系统运行前期数据量小没有使用ES，当数据量增长之后使用ES时，需要将数据库现有的数据导入到ES（导入工作需要在项目部署到生产环境之前来完成）

5.2.1 查询所有商品信息

```
/** 根据关键字模糊搜索商品信息 ...*/
public List<ProductVO> selectProductByKeyword(@Param("kw") String keyword,
                                               @Param("start") int start,
                                               @Param("limit") int limit);

public List<ProductVO> selectProducts();
```

```
<select id="selectProductByKeyword" resultMap="ProductVOMap2"...>

<select id="selectProducts" resultMap="ProductVOMap2">
    select  product_id,
            product_name,
            category_id,
            root_category_id,
            sold_num,
            product_status,
            content,
            create_time,
            update_time
    from product
</select>
```

去掉后面的模糊查询条件

5.2.2 定义ES存储数据的对象结构

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Product4ES {

    private String productId;
    private String productName;
    private String productImg;
    private int soldNum;
    private String productSkuName;
    private double productSkuPrice;

}
```

5.2.3 代码实现

```java
@RunWith(SpringRunner.class)
@SpringBootTest(classes = ApiApplication.class)
public class ImportProductInfoIntoES {

    @Autowired
    private RestHighLevelClient restHighLevelClient;
    @Autowired
    private ProductMapper productMapper;
    @Autowired
    private ObjectMapper objectMapper;

    @Test
    public void testCreateIndex() throws IOException {
        //创建索引
        CreateIndexRequest createIndexRequest = new
CreateIndexRequest("fmmallproductsindex");
        CreateIndexResponse createIndexResponse =
restHighLevelClient.indices().create(createIndexRequest,
RequestOptions.DEFAULT);
        System.out.println(createIndexResponse.isAcknowledged());
    }

    @Test
    public void testImportData() throws IOException {
        //1.从数据库查询数据
        List<ProductVO> productVOS = productMapper.selectProducts();
```

```
        System.out.println(productVOS.size());

        //2.将查询到数据写入到ES
        for (int i = 0; i <productVOS.size() ; i++) {
            ProductVO productVO = productVOS.get(i);

            String productId = productVO.getProductId();
            String productName = productVO.getProductName();
            Integer soldNum = productVO.getSoldNum();

            List<ProductSku> skus = productVO.getSkus();

            String skuName = skus.size()==0?"":
skus.get(0).getSkuName();
            String skuImg = skus.size()==0?"":skus.get(0).getSkuImg();
            Integer sellPrice = skus.size()==0?
0:skus.get(0).getSellPrice();

            Product4ES product = new
Product4ES(productId,productName,skuImg,soldNum,skuName,sellPrice);

            IndexRequest request = new
IndexRequest("fmmallproductsindex");
            request.id(productId);
            request.source(objectMapper.writeValueAsString(product),
XContentType.JSON);
            IndexResponse indexResponse =
restHighLevelClient.index(request, RequestOptions.DEFAULT);
            System.out.println((i+1)+"---"+indexResponse);
        }

    }

}
```

5.3 从ES中进行商品的检索

5.3.1 接口实现

- ProductServiceImpl

```
@Override
```

```
public ResultVO searchProduct(String kw, int pageNum, int limit) {
    ResultVO resultVO = null;
    try {
        //1.查询搜索结果
        int start = (pageNum-1)*limit;
        //从ES查询数据
        SearchRequest searchRequest = new
SearchRequest("fmmallproductsindex");
        SearchSourceBuilder searchSourceBuilder = new
SearchSourceBuilder();
        //查询条件

 searchSourceBuilder.query(QueryBuilders.multiMatchQuery(kw,"productNam
e","productSkuName"));
        //分页条件
        searchSourceBuilder.from(start);
        searchSourceBuilder.size(limit);
        //高亮显示
        HighlightBuilder highlightBuilder = new HighlightBuilder();
        HighlightBuilder.Field field1 = new
HighlightBuilder.Field("productName");
        HighlightBuilder.Field field2 = new
HighlightBuilder.Field("productSkuName");
        highlightBuilder.field(field1);
        highlightBuilder.field(field2);
        highlightBuilder.preTags("<label style='color:red'>");
        highlightBuilder.postTags("</label>");
        searchSourceBuilder.highlighter(highlightBuilder);
        searchRequest.source(searchSourceBuilder);
        //执行检索
        SearchResponse searchResponse =
restHighLevelClient.search(searchRequest, RequestOptions.DEFAULT);

        //封装查詢結果
        SearchHits hits = searchResponse.getHits();
        //获取总记录数
        int count = (int)(hits.getTotalHits().value);
        //计算总页数
        int pageCount = count%limit==0? count/limit:count/limit+1;

        Iterator<SearchHit> iterator = hits.iterator();
        List<Product4ES> products = new ArrayList<>();
```

```
        while(iterator.hasNext()){
            SearchHit searchHit = iterator.next();
            Product4ES product4ES =
objectMapper.readValue(searchHit.getSourceAsString(),
Product4ES.class);
            //获取高亮字段
            Map<String, HighlightField> highlightFields =
searchHit.getHighlightFields();
            //productName
            HighlightField highlightField1 =
highlightFields.get("productName");
            if(highlightField1!=null){
                String highLightProductName =
Arrays.toString(highlightField1.fragments());
                product4ES.setProductName(highLightProductName);
            }
            products.add(product4ES);
        }

        //4.封装，返回数据
        PageHelper<Product4ES> pageHelper = new PageHelper<>(count,
pageCount, products);
        resultVO = new ResultVO(ResStatus.OK, "SUCCESS", pageHelper);

    } catch (IOException e) {
        e.printStackTrace();
    }
    return resultVO;
}
```

5.3.2 前端实现

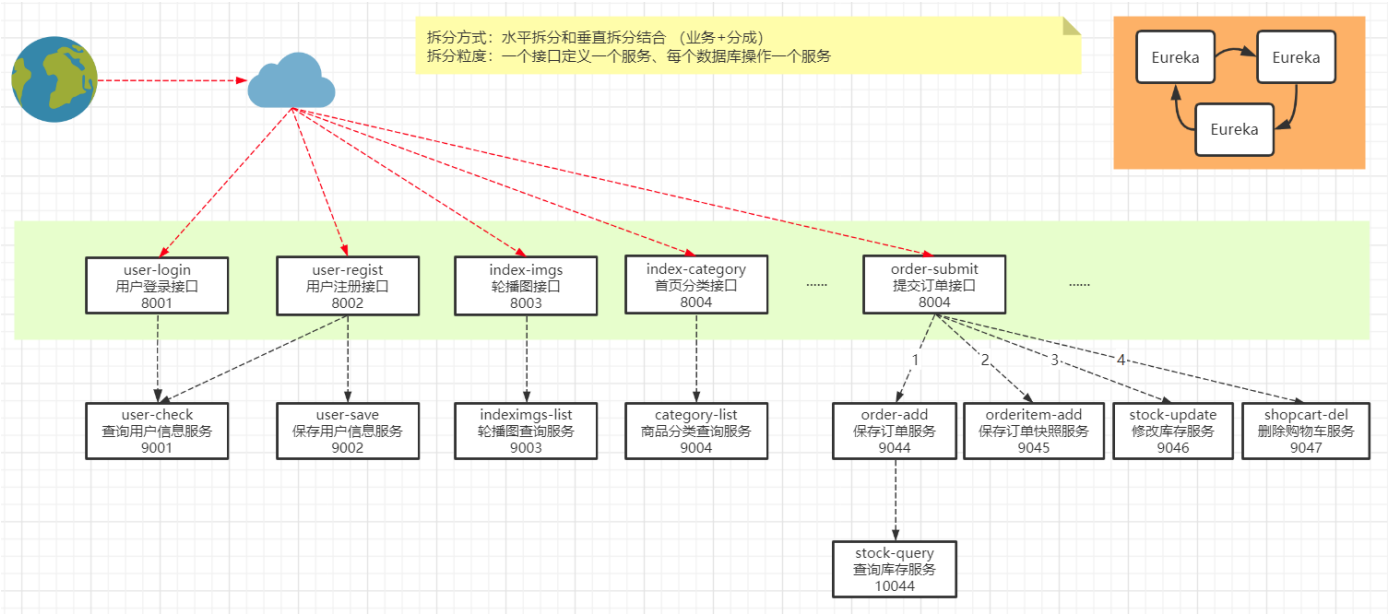
```
<template v-for="p in products"><!--p 表示的是一个 Product4ES对象-->
    <li>
        <div class="i-pic limit" @click="toIntroduction(p.productId)"
:data-id="p.productId">
            <img :src="'static/pimgs/'+p.productImg" />
            <p class="title fl">【<span v-html="p.productName">
</span>】{{p.productSkuName}}</p>
            <p class="price fl">
                <b>¥</b>
                <strong>{{p.productSkuPrice}}</strong>
```

```
            </p>
            <p class="number fl">
                销量<span>{{p.soldNum}}</span>
            </p>
        </div>
    </li>
</template>
```

六、《锋迷商城》微服务拆分

> 随着一个项目功能的增加，结构会变得越来越复杂，我们就面临`单体地狱`
>
> 为了提升单体项目的可用性、可扩展性、可维护性需要对项目进行微服务拆分

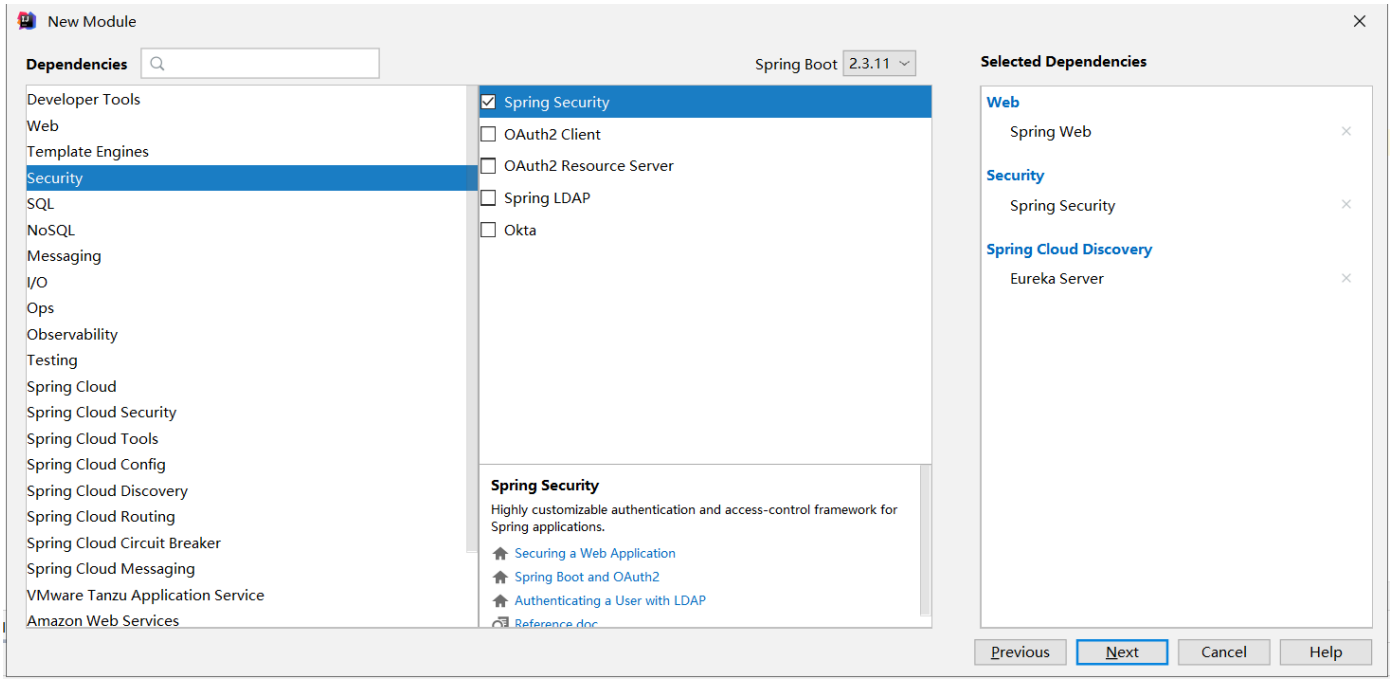
6.1 微服务拆分架构



6.2 搭建服务注册与发现中心

6.2.1 创建SpringBoot应用，添加如下依赖

> SpringBoot版本统一选择 2.3.11



6.2.2 配置application.yml

```yaml
server:
  port: 8761
spring:
  application:
    name: eureka-server
  security:
    user:
      name: zhangsan
      password: 123456
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
    service-url:
      defaultZone: http://localhost:8761/eureka
```

6.2.3 在启动类添加注解

```
@SpringBootApplication
@EnableEurekaServer
public class EurekaServerApplication {

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

}
```

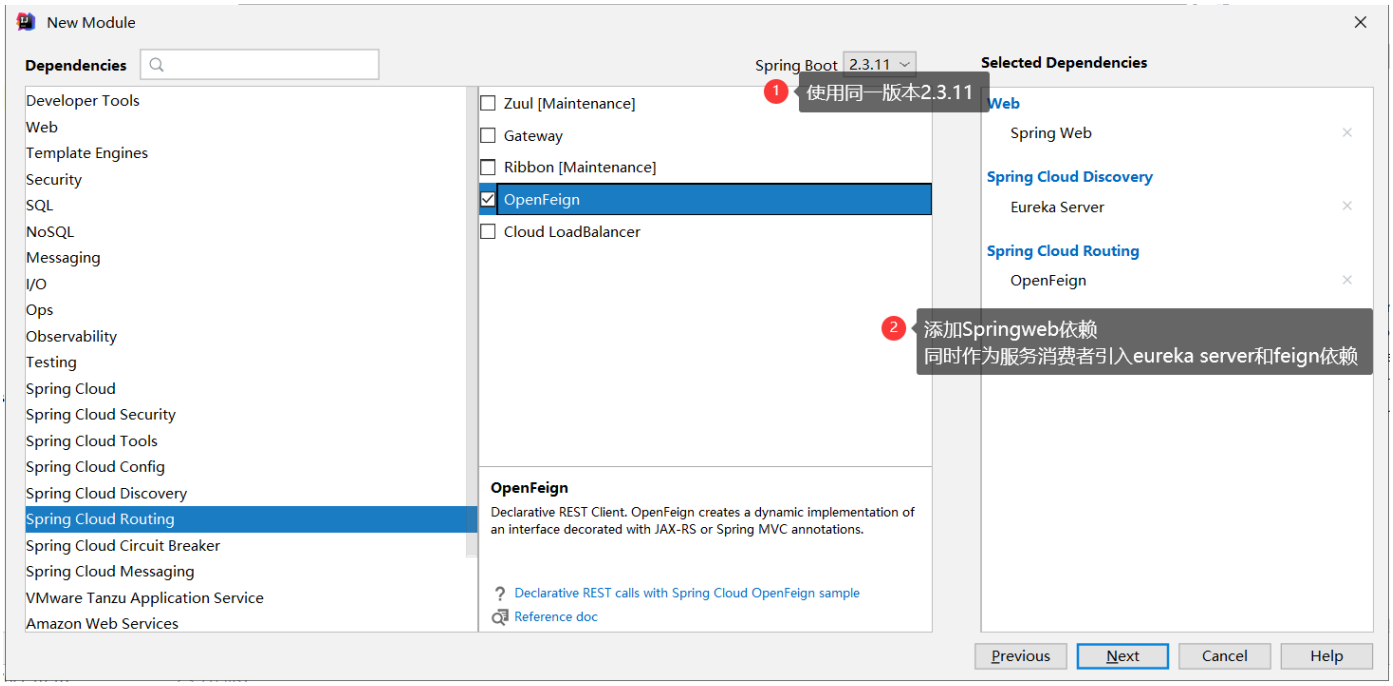
6.2.4 配置SpringSecurity

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        //设置当前服务器的所有请求都要使用spring security的认证

 http.authorizeRequests().anyRequest().authenticated().and().httpBasic(
);
    }
}
```

七、用户登录业务-微服务拆分

7.1创建用户登录接口服务

7.1.1 创建api-user-login服务



- 如果同时使用熔断器，则可以同时添加`hystrix`依赖

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-hystrix</artifactId>
</dependency>
```

7.1.2 配置application.yml

```yaml
server:
  port: 8001
spring:
  application:
    name: api-user-login
eureka:
  client:
    service-url:
      defaultZone: http://zhangsan:123456@localhost:8761/eureka

## 开启熔断器
feign:
  hystrix:
    enabled: true
```

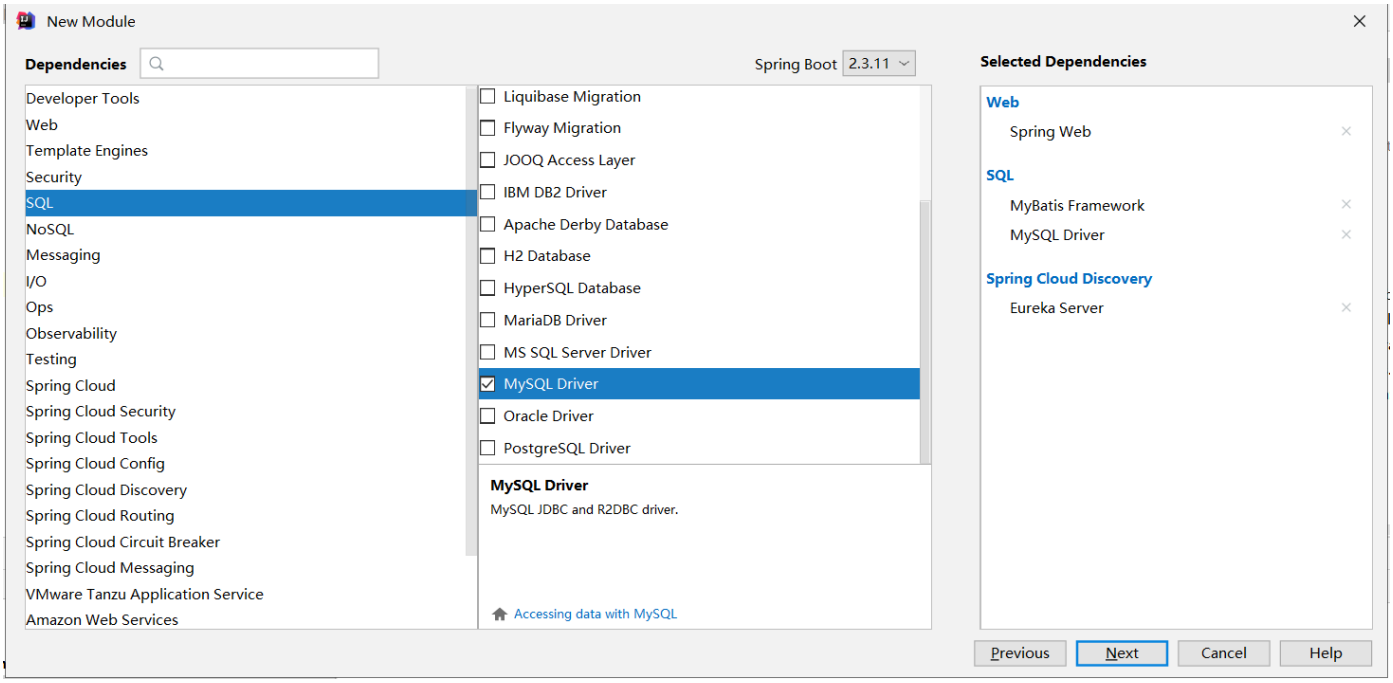
7.1.3 在启动类添加注解

```
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
@EnableHystrix
public class ApiUserLoginApplication {

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

}
```

7.2 创建用户信息查询服务

7.2.1 创建 user-check 服务

> eureka server
>
> mybatis mysql　tkmapper
>
> spring web



- 如果需要使用tkMapper完成数据库操作，则同时添加依赖：

```xml
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper-spring-boot-starter</artifactId>
    <version>2.1.5</version>
</dependency>
```

7.2.2 配置application.yml

```yaml
server:
  port: 9001
spring:
  application:
    name: user-check
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    username: root
    password: admin123
    url: jdbc:mysql://localhost:3306/fmmall2?characterEncoding=utf-8
eureka:
  client:
    service-url:
      defaultZone: http://zhangsan:123456@localhost:8761/eureka
mybatis:
  mapper-locations: classpath:mappers/*Mapper.xml
  type-aliases-package: com.qfedu.fmmall.entity
```

7.2.3 在启动类添加注解

```java
@SpringBootApplication
@EnableEurekaClient
@MapperScan("com.qfedu.user.dao")
public class UserCheckApplication {

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

}
```

7.3 完成用户查询服务的开发

八、用户注册业务-微服务拆分

8.1 创建用户注册接口服务

8.1.1 创建api-user-regist服务

- 服务调用者

8.1.2 配置application.yml

8.1.3 启动添加注解

8.2 创建保存用户信息服务

8.2.1 创建user-save服务

- 服务提供者
- 进行數據庫操作，添加tkmapper及beans依赖

```
<dependency>
    <groupId>com.qfedu</groupId>
    <artifactId>beans</artifactId>
    <version>2.0.1</version>
</dependency>

<!-- tkmapper -->
<dependency>
    <groupId>tk.mybatis</groupId>
    <artifactId>mapper-spring-boot-starter</artifactId>
    <version>2.1.5</version>
</dependency>
```

8.2.2 配置application.yml

8.2.3 在启动类添加注解

- @EnableEurekaClinet
- @MapperScan

千锋教育Java教研院 关注公众号【Java架构栈】下载所有课程代码课件及工具 让技术回归本该有的纯静!