准备工作

1. 新建一个maven项目目录结构如下

```
spring_ioc tree
├── pom.xml
└── src
    ├── main
    |   ├── java
    |   |   └── com
    |   |       └── jx
    |   |           └── spring
    |   |               └── ioc
    |    |                   ├── bean
    |    |                   |   └── Student.java
    |    |                   └── test
    |    |                       └── T1.java
    |   └── resources
    |       └── applicationContext.xml
    └── test
        └── java

```

2. 引入依赖

```
<dependencies>
    <!-- 会自动引入core、bean等模块 -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.0.2.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
         <artifactId>lombok</artifactId>
         <version>1.18.4</version>
     </dependency>
     <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>
         <version>4.12</version>
     </dependency>
</dependencies>
```

3. 新建一个实体bean

```
@Getter
@Setter
@ToString
public class Student {
    private Integer id;
    private String name;
}
```

4. 新建配置文件，放在 resource目录下

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.or

    <bean id="student" class="com.jx.spring.ioc.bean.Student"/>

</beans>
```

5. 新建测试类

```
@Test
public void t_01(){
    ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.x

    Student student = applicationContext.getBean(Student.class);
    student.setId(1);
    student.setName("a");
    System.out.println(student);
}
```

源码分析以XML的形式进行分析

入口大纲

1. 入口

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml")
Student student = applicationContext.getBean(Student.class);
```

2. ClassPathXmlApplicationContext 构造函数

```
# 执行了三个方法
# 调用父类，这里传入的parent 是 null
   super(parent);
# 设置配置
   setConfigLocations(configLocations);
# 开始启动的核心方法
   refresh();
```

3. super方法跟踪

   a. 进入到 父类 AbstractApplicationContext

```
创建了spring的资源加载器
protected ResourcePatternResolver getResourcePatternResolver() {
   return new PathMatchingResourcePatternResolver(this);
}
```

4. setConfigLocations 方法跟踪

```
进入 到 父类 AbstractRefreshableConfigApplicationContext

最终将 配置文件存入了一个字符串数组中
private String[] configLocations;
```

5. refresh方法跟踪

```
@Override
public void refresh() throws BeansException, IllegalStateException {
```

```
    synchronized (this.startupShutdownMonitor) {
       prepareRefresh();
       ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();
       prepareBeanFactory(beanFactory);
       try {
          postProcessBeanFactory(beanFactory);
          invokeBeanFactoryPostProcessors(beanFactory);
           registerBeanPostProcessors(beanFactory);
           initMessageSource();
           initApplicationEventMulticaster();
           onRefresh();
           registerListeners();
           finishBeanFactoryInitialization(beanFactory);
           finishRefresh();
        }
}
```

refresh详解

prepareRefresh —— 做一些字段初始化工作以及检查相关工作

做一些字段初始化工作以及检查相关工作：

1. 设置容器启动时间为当前系统时间
2. 设置容器的同步标识

```
protected void prepareRefresh() {
    this.startupDate = System.currentTimeMillis();
    this.closed.set(false);
    this.active.set(true);

    if (logger.isInfoEnabled()) {
       logger.info("Refreshing " + this);
    }
    initPropertySources();
     getEnvironment().validateRequiredProperties();
     this.earlyApplicationEvents = new LinkedHashSet<>();
}
```

ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory() —— *创建工厂*

```
@Override
protected final void refreshBeanFactory() throws BeansException {
    if (hasBeanFactory()) {
       destroyBeans();
       closeBeanFactory();
    }

   DefaultListableBeanFactory beanFactory = createBeanFactory();
   beanFactory.setSerializationId(getId());
    customizeBeanFactory(beanFactory);
    loadBeanDefinitions(beanFactory);
    synchronized (this.beanFactoryMonitor) {
       this.beanFactory = beanFactory;
    }
```

```
}
```

1. 如果存在 BeanFactory 就关闭
2. 创建一个 BeanFactory
3. loadBeanDefinitions 对sprng进行定制化的一些参数（基本上不用）
4. 调用 loadBeanDefinition 方法 装载 bean 定义(这个方法里面就将 bean 进行了 加载)

```
protected void loadBeanDefinitions(DefaultListableBeanFactory beanFactory) throws BeansException, IO

   XmlBeanDefinitionReader beanDefinitionReader = new XmlBeanDefinitionReader(beanFactory);

   beanDefinitionReader.setEnvironment(this.getEnvironment());
   beanDefinitionReader.setResourceLoader(this);
   beanDefinitionReader.setEntityResolver(new ResourceEntityResolver(this));

   initBeanDefinitionReader(beanDefinitionReader);
    loadBeanDefinitions(beanDefinitionReader);
}
```

a. 根据 beanFactory 创建一个 XmlBeanDefinitionReader 对象

b. 设置 环境，资源加载器，实体解析器（Sax xml解析器）

c. 初始化 bean定义的 reader

d. 加载bean

```

```

e.