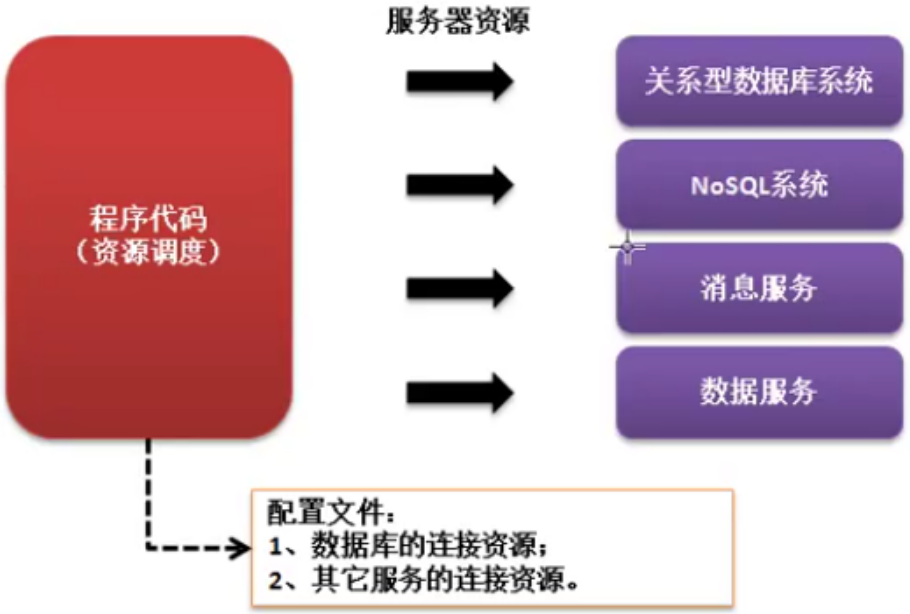
2、具体内容

Annotation是从JDK1.5以后提出的一个新的开发技术结构，利用Annotation可以有效的减少程序配置的代码，并且可以利用Annotation进行一些结构化的定义。Annotation是以一种注解的形式实现的程序开发。

程序开发



如果想要清楚Annotation的产生意义，则必须了解一下程序开发结构的历史，从历史上来讲程序的开发一共分为了三个过程；

过程一：在程序定义的时候将所有可能使用到的资源全部定义在程序代码之中；

- 如果此时服务器的相关的地址发生了改变，那么对于程序而言就需要进行源代码的修改了，维护需要由开发人员来完成，这样的做法明显是不方便的。

过程二：引入配置文件，在配置文件之中定义全部要使用的服务器资源

- 在配置项不多的情况下，此类配置非常好用，并且十分的简单，但是如果这个时候所有的项目都是采用结构开发，那么就可能出现一种可怕的场景：配置文件暴多；
- 所有的操作都需要通过配置文件完成，这样对于开发的难度提升了；

过程三：将配置信息重新写回到程序里面，理由一些特殊的标记与程序代码进行分离，这就是注解的作用，也就是说Annotation提出的基本依据。

- 如果全部都使用注解开发，难度太高了，配置文件有好处也有缺点，所以现在人们的开发基本上都是围绕着文件+注解的形式完成的。

本次锁讲解的注解实际上只是Java中提供的几个基本注解：@Override、@Deprecated、@SuppressWarnings

---

■准确的覆写：@Override

当子类继承某一个父类之后如果发现父类中的某些方法功能不足的时候往往会采用覆写的形式来对方法功能进行扩充，于是下面首先来观察一种覆写操作。

范例：观察覆写的问题

```
class Channel {
    public void connect() {
        System.out.println("********** Channel ***********") ;
```

```
    }
}
class DatabaseChannel extends Channel {
    public void connection() {
        System.out.println("子类定义的通道连接操作。") ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        new DatabaseChannel().connect() ;
    }
}
```

开发之中经常出现的两个问题：

- 虽然要明确的继承一个父类并且进行方法的覆写，但是有可能由于疏忽忘记编写extends，于是这个时候不是覆写
- 在进行方法覆写的时候单词写错了。

此时即便单词写错了，实际上程序在编译的时候也不会出现任何的错误信息，因为它认为这是一个新的方法。所以在开发之中为了避免这种问题的出现，可以在明确覆写的方法上追加有一个注解。

范例：追加注解

```
class Channel {
    public void connect() {
        System.out.println("********** Channel ************") ;
    }
}
class DatabaseChannel extends Channel {
    @Override     // 明确表示该方法是一个覆写来的方法
    public void connect() {
        System.out.println("子类定义的通道连接操作。") ;
    }
}
```

该注解主要是帮助开发者在程序编译的时候可以检查出程序的错误。

■过期操作：@Deprecated

所谓的过期操作指的是在一个软件项目迭代开发过程之中，可能有某一个方法或者是某个类，由于在最初设计的时候考虑不周（存在有缺陷），导致新版本的应用会有不适应的地方（老版本不影响），这个时候又不可能直接删除掉这些操作，那么就希望给一个过渡的时间，于是就可以采用过期的声明，目的告诉新的用户这些操作不要再用了，老的用户你用就用了，这样的方法就必须利用“@Deprecated”注解进行定义。

范例：声明过期操作

```
class Channel {
    @Deprecated // 老系统继续用，如果是新的不要用了
```

```
    public void connect() {
        System.out.println("********** Channel ************") ;
    }
    public String connection() {
        return "获取了Xxx通道连接信息。" ;
    }
}
public class JavaDemo {
    public static void main(String args[]) {
        new Channel().connect() ;
    }
}
```

在有一些开源项目里面特别讨厌：新版本出现之后，将里面的方法彻底变个样。

■压制警告：@SuppressWarings

以之前的过期程序为例，可以发现在进行程序编译的时候会出现一个错误的提示：

```
注: JavaDemo.java使用或覆盖了已过时的 API。
注: 有关详细信息, 请使用 -Xlint:deprecation 重新编译。
```

如果此时不愿意见到这些提示信息（或者已经明确的知道了错误在哪里），那么就可以进行警告信息的压制。

```
class Channel {
    @Deprecated  // 老系统继续用，如果是新的不要用了
    public void connect() {
        System.out.println("********** Channel ************") ;
    }
    public String connection() {
        return "获取了Xxx通道连接信息。" ;
    }
}
public class JavaDemo {
    @SuppressWarnings({"deprecation" })
    public static void main(String args[]) {
        new Channel().connect() ;
    }
}
```

它做的只是让警告信息不出现，不打扰你而已。