University of Beer

Final year project

Lager brewing techniques

Author:
John Smith

Supervisor:
Dr. Mark Brown

May 4, 2017

Part I

Framework

Spring

http://stackoverflow.com/questions/31045955/using-configurationproperties-to-fill-map-in-generic-way

JDBC 数据库重连

在数据库重启后，程序无法重新连接到数据库。程序使用的是 Hikari，找到了如下属性：

```
# 使用 Hikari pool 时，是否允许连接池暂停，默认为: false
spring.datasource.allow-pool-suspension=true
```

可以将连接池到 c3p0，c3p0 连接池本身具有数据库重连机制。目前常见到连接池如下表所示：

序号	名称	协议	备注
1	c3p0	LGPL v.2.1	C3P0 是一个开源数据连接池，Hibernate3.0 默认自带的数据连接池，性能比较稳定。
2	HikariCP	Apache 2.0	Fast, simple, reliable. HikariCP is a "zero-overhead" production ready JDBC connection pool. At roughly 130Kb, the library is very light.
3	Druid	Apache 2.0	Druid 是 Java 语言中最好的数据库连接池。Druid 能够提供强大的监控和扩展功能。
4	DBCP	Apache 2.0	DBCP(DataBase connection pool), 是 apache 上的一个 java 连接池项目，也是 tomcat 使用的连接池组件。单独使用 dbcp 需要 2 个包：commons-dbcp.jar,commons-pool.jar
5	BoneCP	Apache 2.0	使用 HikariCP 替代

登陆服务器，使用命令查看正常的数据库连接：

```
lsof -i:5236
```

输出的结果如下：

```
COMMAND   PID USER   FD   TYPE  DEVICE SIZE/OFF NODE
          NAME
java    20026   hl   38u  IPv4 2197171      0t0  TCP localhost:58220->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   42u  IPv4 2197527      0t0  TCP localhost:58222->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   43u  IPv4 2197176      0t0  TCP localhost:58224->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   44u  IPv4 2197177      0t0  TCP localhost:58226->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   45u  IPv4 2197178      0t0  TCP localhost:58228->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   46u  IPv4 2197179      0t0  TCP localhost:58230->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   47u  IPv4 2197180      0t0  TCP localhost:58232->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   48u  IPv4 2197181      0t0  TCP localhost:58234->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   49u  IPv4 2197182      0t0  TCP localhost:58236->
          localhost:padl2sim (ESTABLISHED)
java    20026   hl   50u  IPv4 2197536      0t0  TCP localhost:58238->
          localhost:padl2sim (ESTABLISHED)
```

在程序启动时，并没有真正建立 TCP 连接。只有真正的接受到请求查询数据库时，程序才与数据库建立 TCP 连接。此处建立了 10 个 TCP 连接。而 Hikari 默认的最大连接池数量也为 10 个。如下配置设启连接池连接池时，初始建立的连接数量为 5 个：

```
spring.datasource.initial-size=5
```

在 Spring 中配置数据源类型如下：

```
# 指定数据源类型
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
```

Hikari 默认的数据源默认的连接池数量为 10 个，默认的数量在 HikariConfig 类中进行的设置。为什么修改 Spring 的 Initial Size 数量会影响 Hikari 的连接数量呢？如果不配置数据源，Spring Boot 默认的数据源是：

```
org.apache.tomcat.jdbc.pool.DataSource
```

但是在实际开发中，可能需要使用自己是熟悉的数据源或者其他性能比较高的数据源。此时就可以通过制定 Spring 的数据源类型来实现。

停止数据库：

```
nohup sudo /opt/dmdbms/bin/dmserver /opt/dmdbms/data/DAMENG/
        dm.ini -noconsole &
```

在反复研究后发现，程序并未使用 HikariCP 连接池，而是使用的 Spring JDBC 连接池，所以在初始化 Datasource 时指定连接池，如下代码片段所示：

```
@Configuration
@Data
@ConfigurationProperties(prefix = "spring.datasource")
public class DataSourceConfig {

    private String jdbcUrl;

    private String username;

    private String driverClassName;

    private String password;

    @Bean
    @Primary
    public DataSource primaryDataSource() {
        HikariConfig hikariConfig = new HikariConfig();
        hikariConfig.setDriverClassName(driverClassName);
        hikariConfig.setJdbcUrl(jdbcUrl);
        hikariConfig.setUsername(username);
        hikariConfig.setPassword(password);
        hikariConfig.setMaximumPoolSize(5);
        hikariConfig.setConnectionTestQuery("SELECT 1");
        hikariConfig.setPoolName("springHikariCP");
        hikariConfig.addDataSourceProperty("dataSource.cachePrepStmts
         ", "true");
```

```
26          hikariConfig.addDataSourceProperty("dataSource.
             prepStmtCacheSize", "250");
27          hikariConfig.addDataSourceProperty("dataSource.
             prepStmtCacheSqlLimit", "2048");
28          hikariConfig.addDataSourceProperty("dataSource.
             useServerPrepStmts", "true");
29          HikariDataSource dataSource = new HikariDataSource(
             hikariConfig);
30          return dataSource;
31      }
32  }
```

0.0.1 读取 properties 属性

读取自定义属性

有时需要读取自定义属性，以配置 HikariCP 连接池为例，在 application.properties 文件指定指定配置：

```
1  spring.datasource.hikari.jdbc−url=jdbc:dm://dn4:5236/DMSERVER
```

在类中获取配置相应的值，注意需要添加 Data 注解，否则需要手写 get 和 set 方法：

```
1  @Configuration
2  @Data
3  @ConfigurationProperties(prefix = "spring.datasource.hikari")
4  public class DataSourceConfig {
5      private String jdbcUrl;
6  }
```

由于此处注解是默认写在 application.properties 配置文件中，所以在 ConfigurationProperties 中可以不指定路径。否则需要使用 locations 指定配置文件路径。

0.0.2 jps

jps(Java Virtual Machine Process Status Tool) 是 JDK 1.5 提供的一个显示当前所有 java 进程 pid 的命令。jdk 中的 jps 命令可以显示当前运行的 java 进程以及相关参数，它的实现机制如下：

java 程序在启动以后，会在 java.io.tmpdir 指定的目录下，就是临时文件夹里，生成一个类似于 hsperfdata_User 的文件夹，这个文件夹里（在 Linux 中为/tmp/hsperfdata_{userName}/），有几个文件，名字就是 java 进程的 pid，因此列出当前运行的 java 进程，只是把这个目录里的文件名列一下而已。至于系统的参数什么，就可以解析这几个文件获得。hsperfdata 的含义就是 HotSpot Performance Data。

```
1 nohup java -jar -Xmx2g credit-system-web-boot-1.0.0.jar --spring.
        config.location=application-jenkins.properties &
```

10

Groovy

Part II

Tool

0.1 OS

0.1.1 Linux

查看 Linux 信息

```
# Linux 查看版本当前操作系统内核信息
uname -a
```

Gradle

0.1.2 执行流程

There is a one-to-one relationship between a Project and a ”build.gradle” file. During build initialisation, Gradle assembles a Project object for each project which is to participate in the build, as follows:

Create a Settings instance for the build. Evaluate the ”settings.gradle” script, if present, against the Settings object to configure it. Use the configured Settings object to create the hierarchy of Project instances. Finally, evaluate each Project by executing its ”build.gradle” file, if present, against the project. The projects are evaluated in breadth-wise order, such that a project is evaluated before its child projects. This order can be overridden by calling evaluationDependsOnChildren() or by adding an explicit evaluation dependency using evaluationDependsOn(String).

0.1.3 repositories

在 Gradle 构建标本 build.gradle 里，经常会看到如下脚本：

```
repositories {
    maven {
        url 'http://www.eveoh.nl/files/maven2'
```

```
    }
    maven {
        url 'http://repox.gtan.com:8078'
    }
    mavenCentral()
    jcenter()
    maven { url 'http://repo.spring.io/plugins-release' }
}
```

总的来说，只有两个标准的 Android library 文件服务器：Jcenter 和 Maven Central。起初，Android Studio 选择 Maven Central 作为默认仓库。如果你使用老版本的 Android Studio 创建一个新项目，mavenCentral() 会自动的定义在 build.gradle 中。但是 Maven Central 的最大问题是对开发者不够友好。上传 library 异常困难。上传上去的开发者都是某种程度的极客。同时还因为诸如安全方面的其他原因，Android Studio 团队决定把默认的仓库替换成 jcenter。正如你看到的，一旦使用最新版本的 Android Studio 创建一个项目，jcenter() 自动被定义，而不是 mavenCentral()。mavenCentral() 表示依赖是从 Central Maven 2 仓库中获取的，库的地址是`https://repo1.maven.org/maven2`。jcenter 表示依赖是从 Bintary’s JCenter Maven 仓库中获取的，仓库的地址是`https://jcenter.bintray.com`，bintray 是一家提供全球企业软件开发包托管的商业公司。

0.1.4 属性 (Properties)

Extra Properties

extra 属性一般用于定义常量，All extra properties[1] must be defined through the ”ext” namespace. Once an extra property has been defined, it is available directly on the owning object (in the below case the Project, Task, and sub-projects respectively) and can be read and updated. Only the initial declaration that needs to be done via the namespace.

```
buildscript {
```

[1] `https://docs.gradle.org/current/javadoc/org/gradle/api/Project.html#extraproperties`

```
    ext {
        springBootVersion = '1.4.5.RELEASE'
        jacksonVersion = '2.8.7'
        springfoxVersion = '2.6.1'
        poiVersion = "3.14"
        aspectjVersion = '1.7.4'
    }
}
```

Reading extra properties is done through the ”ext” or through the owning object.

ext.isSnapshot = version.endsWith(”-SNAPSHOT”) if (isSnapshot) // do snapshot stuff

0.1.5 同时连接内外网

在工作中，有时需要同时连接封闭的网络和互联网，开发环境需要依赖封闭的内网，与远程的同事交流需要依赖互联网，而封闭的内网与互联网切换是一大痛点，不仅影响效率，同时也影响心情，宝贵的时间就在这样无意义的开关中白白浪费掉了。制造障碍很容易，清除障碍很艰难。

解决双网卡问题，既能保证内网访问的安全，又能避免频繁切换网卡带来的时间开销，可谓一举两得。一般的计算机有有线网卡，也有无线网卡。一般情况下要么使用有线网卡，要么使用无线网卡，在使用无线时，连接上了有线，因为有线网卡的优先级高，故此时仅有有线能够工作，无线网卡可连接但是无法传送数据。实现双网卡的基本思路是删除默认网关，配置各自的网关即可。

Mac 同时连内网外网

输入如下命令查看 Mac 的所有网络连接方式：

```
networksetup −listallnetworkservices
```

输出的结果如下：

```
An asterisk (*) denotes that a network service is disabled.
```

```
Apple USB Ethernet Adapter
Wi−Fi
Bluetooth PAN
Thunderbolt Bridge
```

可以看出 Mac 可以通过 Wi-Fi 联网，也可以通过 USB 有线联网。给指定的网络连接方式设定 DNS 服务器代码如下：

```
sudo networksetup −setdnsservers AirPort 192.168.10.200
```

清空 DNS 缓存代码如下：

```
dscacheutil –flushcache
```

输入如下命令查看 MacBook 的路由表：

```
netstat −nr
```

Fedora 同时连接内外网

给有线网卡配置地址信息时，内网网卡不要加默认网关，外网网卡加默认网关，并查看无线网卡分配到的地址信息中是否有默认网关。在这里内网是有线网络，外网是无线网络。使用 route 命令查看默认路由，输出如图1所示：

```
[dolphin@localhost project]$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         gateway         0.0.0.0         UG    600    0        0 wlp3s0
172.26.0.0      0.0.0.0         255.255.0.0     U     600    0        0 wlp3s0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0 virbr0
```

Figure 1: 查看 Kernal 路由表

其中 wlp3s0 是无线网卡的名称，说明此时的默认路由是无线网卡的路由。metric 设置路由跳数，metric 值越高，优先级越低。添加-p(permanent) 参数永久保存此条路由信息，系统重启后路由也规则也不会丢失。三删除默认网关的命令如下：

```
sudo route del default
```

删除一条路由：

```
sudo route del -net 192.168.122.0 netmask 255.255.255.0
```

如果默认路由是外网网关。那么只需要单独为内网设置转发特例，所有 59.214.*.* 开头的，全部走 enp0s26u1u2:

```
sudo route add -net 59.214.0.0 netmask 255.255.255.0 gw 10.55.10.1 dev
        enp0s26u1u2
```

其中 59.214.0.0 为内网 IP 的起始网段，255.255.0.0 为内网的子网掩码，内网网关是 10.36.40.12，eth0 为内网网卡的名称。路由添加最好是添加到开机启动中：

```
vim /etc/rc.local
```

添加删除默认网关：

```
#添加默认网关
sudo route add default gw 10.55.10.1
#删除默认网关
sudo route del default gw 10.55.10.1
```

重启网络服务：

```
/etc/init.d/networking restart
```

此时可以同时访问互联网和局域网中的主机，但是无法访问 A 网络。访问 A 网络可通过局域网代理的方式访问，将局域网的一台主机作为代理服务器，本机访问局域网中的代理服务器，代理服务器再将请求转发给 A 网络。在本机只需要在浏览器中配置好代理服务器即可。以 FireFox 浏览器为例，在 Preference->Advance->Network->Connection->Settings 中，增加手动代理配置，地址填写代理主机的地址，端口填写 8888，代理服务器运行的是 Fiddler。以后，访问内网时就使用 FireFox 浏览器，访问外网时就使用 Google Chrome 浏览器。

Ubuntu 同时连接内外网

Windows 同时连接内外网

http://blog.csdn.net/jay329106193/article/details/8129374

http://www.jianshu.com/p/b64da1d93e3d

0.2 Ansible

0.3 Fiddler

0.3.1 Fiddler 替换 Host

0.4 OpenVPN

OpenVPN 从 2001 年开始开发，使用的是 C 语言。此处使用的 OpenVPN 版本是 2.4.1。如果使用 Mac 下的 brew 工具安装，则 OpenVPN 目录在：/usr/local/Cellar/openvpn/2.4.1，OpenVPN 的配置文件在：/usr/local/etc/openvpn。目前 OpenVPN 能在 Solaris、Linux、OpenBSD、FreeBSD、NetBSD、Mac OS X 与 Microsoft Windows 以及 Android 和 iOS 上运行，并包含了许多安全性的功能。

0.4.1 安装

安装基础包：

```
1 sudo yum -y install openssl openssl-devel lzo openvpn easy-rsa -y
```

生成客户端证书

新建 client 文件夹，文件夹可随意命名，然后拷贝 easy-ras 文件夹到 client 文件夹，进入下列目录：

```
cp -R /etc/openvpn/easyrsa/ .
```

初始化:

```
./easyrsa init-pki
```

PKI：Public Key Infrastructure 公钥基础设施。生成请求：

```
./easyrsa gen-req dolphinfedora
```

输入 PEM 验证码。PEM - Privacy Enhanced Mail, 打开看文本格式, 以”——BEGIN...” 开头, ”——END...” 结尾, 内容是 BASE64 编码. 查看 PEM 格式证书的信息:openssl x509 -in certificate.pem -text -noout Apache 和 *NIX 服务器偏向于使用这种编码格式. 签约：

```
#切换到服务端生成 rsa 的目录
#导入 req
./easyrsa import-req ~/client/easyrsa/easy-rsa-master/easyrsa3/pki/
        reqs/dolphinfedora.req dolphinfedora
#用户签约，根据提示输入服务端的 ca 密码
./easyrsa sign client dolphinfedora
```

服务端生成的文件有：

序号	名称
ca.crt	根证书文件
reqs/server.req	C3P0 是一个开源数据连接池，Hibernate3.0 默认自带的数据连接池，性能比较稳定。
reqs/dolphin.req	C3P0 是一个开源数据连接池，Hibernate3.0 默认自带的数据连接池，性能比较稳定。
private/ca.key	根证书私钥文件
private/server.key issued/server.crt issued/dolphin.crt dh.pem	

客户端生成的文件有：

序号	名称
private/dolphinclient.key	C3P0 是一个开源数据连接池，Hibernate3.0 默认自带的数据连接池，性能比较稳定。
reqs/sdolphinclient.req	C3P0 是一个开源数据连接池，Hibernate3.0 默认自带的数据连接池，性能比较稳定。

拷贝出客户端证书文件：

```
cp easyrsa/easy-rsa-master/easyrsa3/pki/ca.crt ~/dolphinfedora/
cp easyrsa/easy-rsa-master/easyrsa3/pki/issued/dolphinfedora.crt ~/
        dolphinfedora/
cp ~/client/easyrsa/easy-rsa-master/easyrsa3/pki/private/dolphinfedora.
        key ~/dolphinfedora/
```

启动 OpenVPN：

```
sudo openvpn server.conf
# Mac 下启动 OpenVPN
sudo /usr/local/Cellar/openvpn/2.4.1/sbin/openvpn /usr/local/etc/
        openvpn/client.conf
```

客户端端配置如下：

```
client          #指定当前 VPN 是客户端
dev tun         #必须与服务器端的保持一致
proto udp       #必须与服务器端的保持一致
#指定连接的远程服务器的实际 IP 地址和端口号
remote 192.168.1.106 1194
#断线自动重新连接
#在网络不稳定的情况下 (例如：笔记本电脑无线网络) 非常有用
resolv-retry infinite
nobind          #不绑定特定的本地端口号
persist-key
persist-tun
ca ca.crt       #指定 CA 证书的文件路径
cert client1.crt        #指定当前客户端的证书文件路径
key client1.key     #指定当前客户端的私钥文件路径
ns-cert-type server     #指定采用服务器校验方式
#如果服务器设置了防御 DoS 等攻击的 ta.key
```

```
#则必须每个客户端开启；如果未设置，则注释掉这一行；
tls-auth ta.key 1
comp-lzo              #与服务器保持一致
#指定日志文件的记录详细级别，可选 0-9，等级越高日志内容越详细
verb 3
```

0.4.2 常见错误

ssl3_get_server_certificate:certificate verify failed

0.5 ssh

0.5.1 Session 时间

在使用 ssh 的过程中，经常会遇到一会儿没有操作就自动断开了，不是非常方便。

ClientAliveInterval 修改/etc/ssh/sshd_config 配置文件 ClientAliveInterval 300（默认为 0），参数的是意思是每 5 分钟，服务器向客户端发一个消息，用于保持连接，使用 service sshd reload 让其修改后生效。如果发现还是有问题，可以试着把 300 设置小一点，例如 60。

ClientAliveCountMax 另外, 至于 ClientAliveCountMax, 使用默认值 3 即可.ClientAliveCountMax 表示服务器发出请求后客户端没有响应的次数达到一定值, 就自动断开。

ControlPersist 4h 在./ssh/config 中添加一行：

```
ControlPersist 4h
```

When used in conjunction with ControlMaster, specifies that the master connection should remain open in the background (waiting for future client connections) after the initial client connection has been closed. If set to no,

then the master connection will not be placed into the background, and will close as soon as the initial client connection is closed. If set to yes or 0, then the master connection will remain in the background indefinitely (until killed or closed via a mechanism such as the "ssh -O exit"). If set to a time in seconds, or a time in any of the formats documented in sshd_config, then the backgrounded master connection will automatically terminate after it has remained idle (with no client connections) for the specified time[2]. 现在你每次通过 SSH 与服务器建立连接之后，这条连接将被保持 4 个小时，即使在你退出服务器之后，这条连接依然可以重用，因此，在你下一次（4 小时之内）登录服务器时，你会发现连接以闪电般的速度建立完成，这个选项对于通过 scp 拷贝多个文件提速尤其明显，因为你不在需要为每个文件做单独的认证了。

0.6 Curl

```
curl 'http://59.214.215.6:3800/inapi/united/detail?param=%E8%B0%8A
    %E5%BE%B7%E5%AE%9E%E4%B8%9A&reds=&blacks=
    TS_F_HEIMINGDAN_C500033,TS_F_HEIMINGDAN' -H '
    Pragma: no-cache' -H 'HL-APP-KEY:
    CreditChongqingSharePortal' -H 'DNT: 1' -H 'Accept-
    Encoding: gzip, deflate, sdch' -H 'Accept-Language: zh-CN,zh;q
    =0.8,en;q=0.6,zh-TW;q=0.4' -H 'User-Agent: Mozilla/5.0 (X11
    ; Fedora; Linux x86_64) AppleWebKit/537.36 (KHTML, like
    Gecko) Chrome/55.0.2883.87 Safari/537.36' -H 'Accept:
    application/json, text/plain, */*' -H 'Cache-Control: no-cache'
    -H 'HL-CURRENT-URL: http://59.214.215.6:3800/main/
    search/unitedListQuery' -H 'Referer: http://59.214.215.6:3800/
    main/search/unitedListQuery' -H 'Cookie: UM_distinctid=15
    b667c97f65a-0afddd1bed9cb8-1421150f-15f900-15b667c97f737f;
     CNZZDATA1257579122=576897936-1492070602-http%253A
    %252F%252F59.214.215.6%253A8081%252F%7C1492393083; cc-o
    -t=
    ZnNoeEpnMXZUcklpRXAyTm9ReXhwWTdNSFFUMnptZ2NOdmFDWlhPajhlakRvYVk1eEtRem53enJ
    ' -H 'Connection: keep-alive' --compressed
```

[2] http://man.openbsd.org/ssh_config.5

0.7 Git

0.7.1 merge

配置 Meld 为默认合并工具：

```
git config --global merge.tool meld
```

0.8 LaTex

0.8.1 字体

Computer Modern 是自由软件 TeX 的默认字体，为美国计算机科学家高德纳（Donald Knuth）使用 METAFONT 软件创造。但是此字体不是很漂亮，所以考虑换一个字体。

Part III

Network

0.9 PAC

0.9.1 PAC 简介

PAC 是 proxy auto-config 的缩写。PAC 文件是纯文本格式的，实际上就是 JavaScript 文件。PAC 最简单的格式就是包含一个叫 FindProxyForURL 的 JavaScript 函数，IE 通过传入两个变量来调用这个函数。一个是用户流量的地址的 URL 全路径，一个是这个 URL 中的主机名 host。在日常生活中浏览器中设置代理很简单，但是当你来回切换时总是觉得很烦，你可以使用 pac 脚本自动判断是否走代理，方便省去了来回手动切换的烦恼。

0.9.2 PAC 实例

一个最简单的 PAC 脚本：

```
function FindProxyForURL(url,host){
    return "DIRECT";
}
```

参数 url 是用户输入的 url，参数 host 是 url 中的主机名。PAC 文件返回值有三种类型：

- DIRECT 直连不通过代理
- PROXY www.lybbn.cn:8080 http 通过 8080 端口代理上网，也可以使用 ip:port 的形式
- SOCKS5 www.lybbn.cn:8080 socks 通过 8080 端口代理上网，可以使用 ip:port 形式

```
var FindProxyForURL = function(init, profiles) {
    return function(url, host) {
        "use strict";
        var result = init, scheme = url.substr(0, url.indexOf(":"));
        do {
            result = profiles[result];
```

```
            if (typeof result === "function") result = result(url, host,
             scheme);
        } while (typeof result !== "string" || result.charCodeAt(0) ===
         43);
        return result;
    };
}("+dolphin2", {
    "+dolphin2": function(url, host, scheme) {
        "use strict";
        if (/^59\.214\.215\.6$/.test(host)) return "+dolphin-proxy";
        if (/^10\.10\.1\.11$/.test(host)) return "+dolphin-proxy";
        return "DIRECT";
    },
    "+dolphin-proxy": function(url, host, scheme) {
        "use strict";
        if (/^127\.0\.0\.1$/.test(host) || /^::1$/.test(host) || /^localhost$
         /.test(host)) return "DIRECT";
        return "PROXY 10.55.10.2:8888";
    }
});
```

以上脚本说明，当 IP 为 59.214.215.6 或 10.10.1.11 时，使用代理 dolphin-proxy，而代理 dolphin-proxy 设置的是代理机器的相关信息，代理机器的 IP 为 10.55.10.2，代理的端口是 8888。