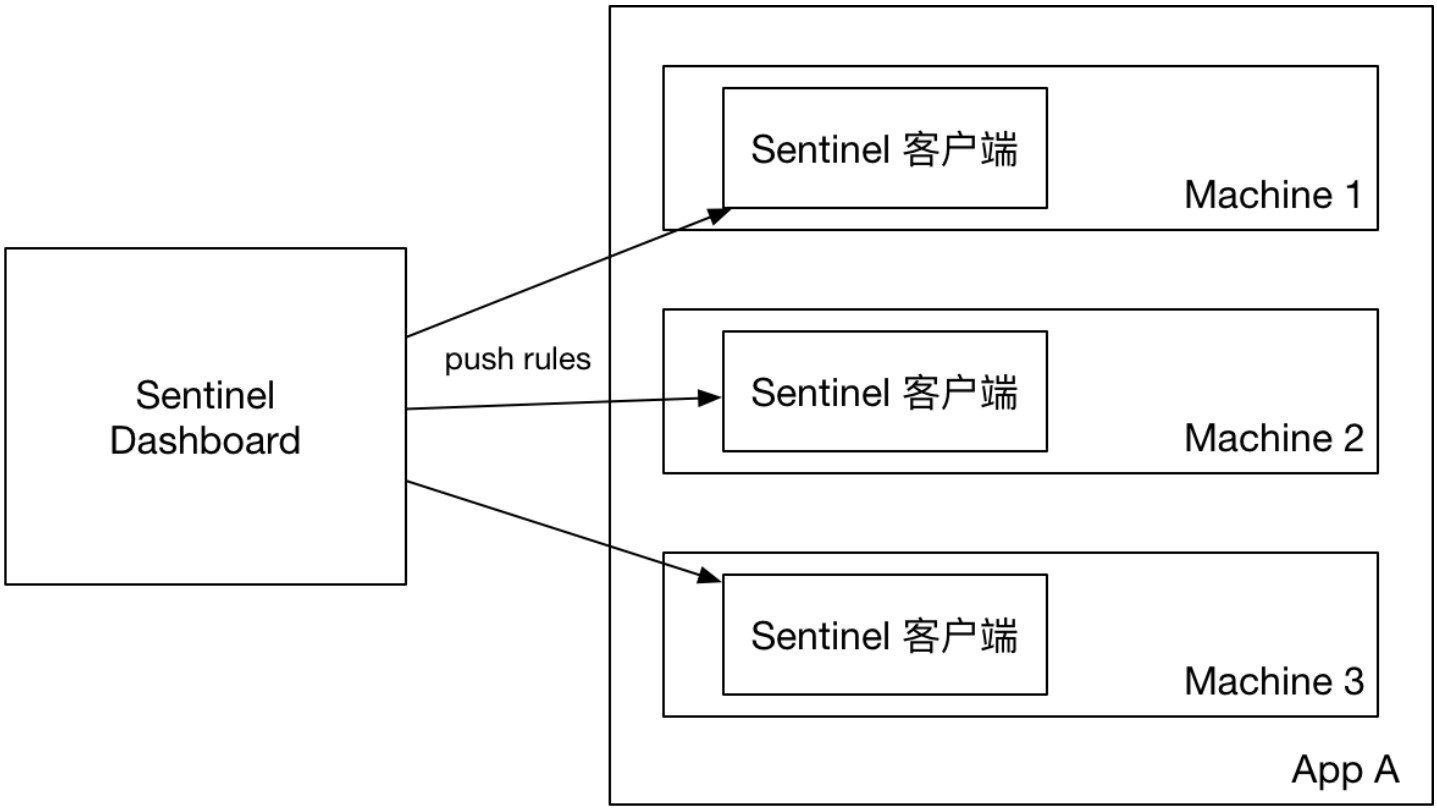
1. Sentinel持久化模式

Sentinel规则的推送有下面三种模式:

推送模式	说明	优点	缺
原始模式	API 将规则推送至客户端并直接更新到内存中，扩展写数据源（WritableDataSource）	简单，无任何依赖	不 重 境
Pull 模式	扩展写数据源（WritableDataSource），客户端主动向某个规则管理中心定期轮询拉取规则，这个规则中心可以是 RDBMS、文件 等	简单，无任何依赖；规则持久化	不 过
Push 模式	扩展读数据源（ReadableDataSource），规则中心统一推送，客户端通过注册监听器的方式时刻监听变化，比如使用 Nacos、Zookeeper 等配置中心。这种方式有更好的实时性和一致性保证。生产环境下一般采用 push 模式的数据源。	规则持久化；一致性；快速	引

1.1 原始模式

如果不做任何修改，Dashboard 的推送规则方式是通过 API 将规则推送至客户端并直接更新到内存中：



这种做法的好处是简单，无依赖；坏处是应用重启规则就会消失，仅用于简单测试，不能用于生产环境。

1.2 拉模式

pull 模式的数据源（如本地文件、RDBMS 等）一般是可写入的。使用时需要在客户端注册数据源：将对应的读数据源注册至对应的 RuleManager，将写数据源注册至 transport 的 `WritableDataSourceRegistry` 中。

1.3 推模式

生产环境下一般更常用的是 push 模式的数据源。对于 push 模式的数据源,如远程配置中心（ZooKeeper, Nacos, Apollo等等），推送的操作不应由 Sentinel 客户端进行，而应该经控制台统一进行管理，直接进行推送，数据源仅负责获取配置中心推送的配置并更新到本地。因此推送规则正确做法应该是 **配置中心控制台/Sentinel 控制台 → 配置中心 → Sentinel 数据源 → Sentinel**，而不是经 Sentinel 数据源推送至配置中心。这样的流程就非常清晰了：

1.3.1 基于Nacos配置中心控制台实现推送

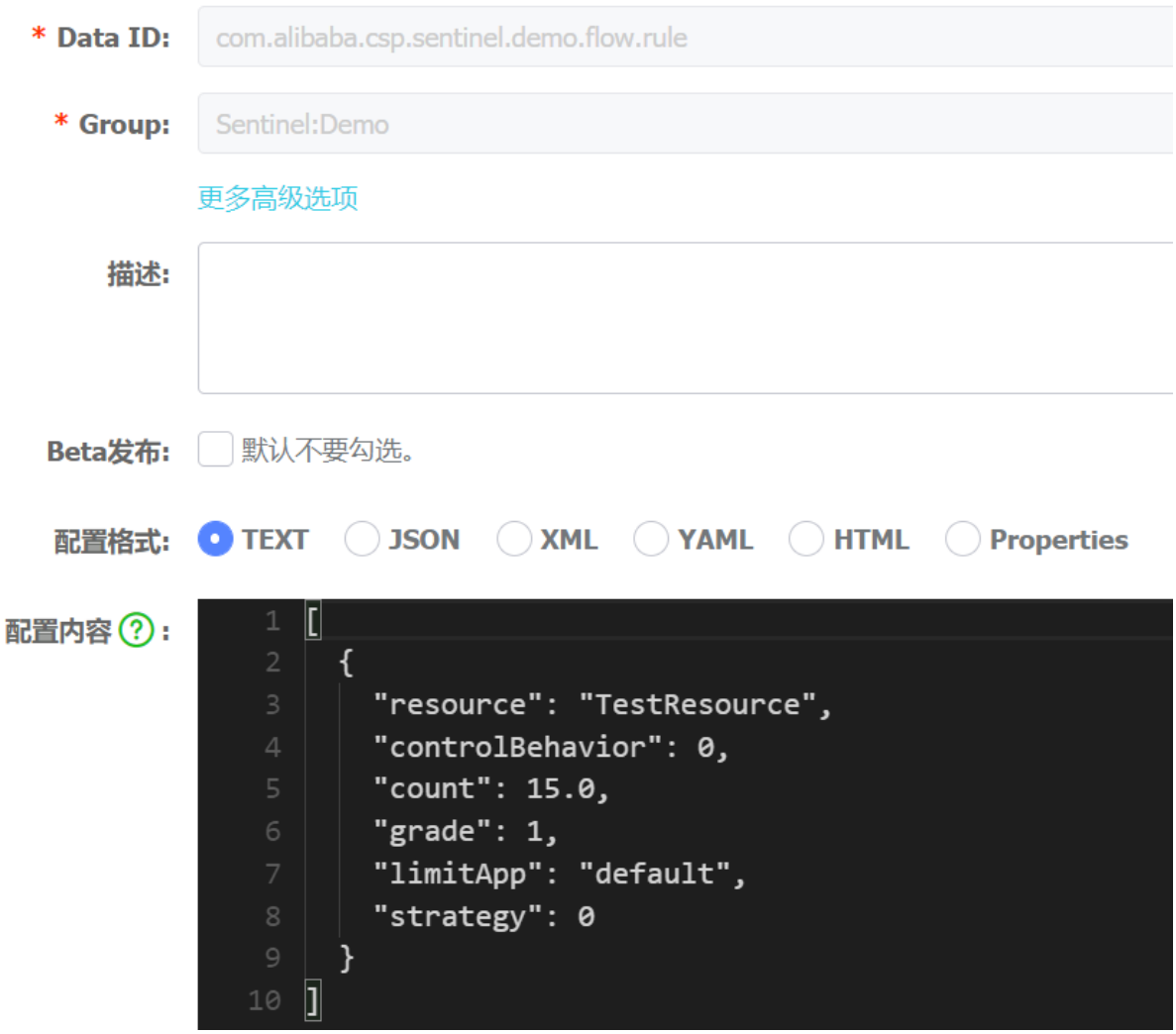
官方demo: sentinel-demo-nacos-datasource

引入依赖

```
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
</dependency>
```

nacos配置中心中配置流控规则

```
[
  {
    "resource": "TestResource",
    "controlBehavior": 0,
    "count": 10.0,
    "grade": 1,
    "limitApp": "default",
    "strategy": 0
  }
]
```



2）yml中配置

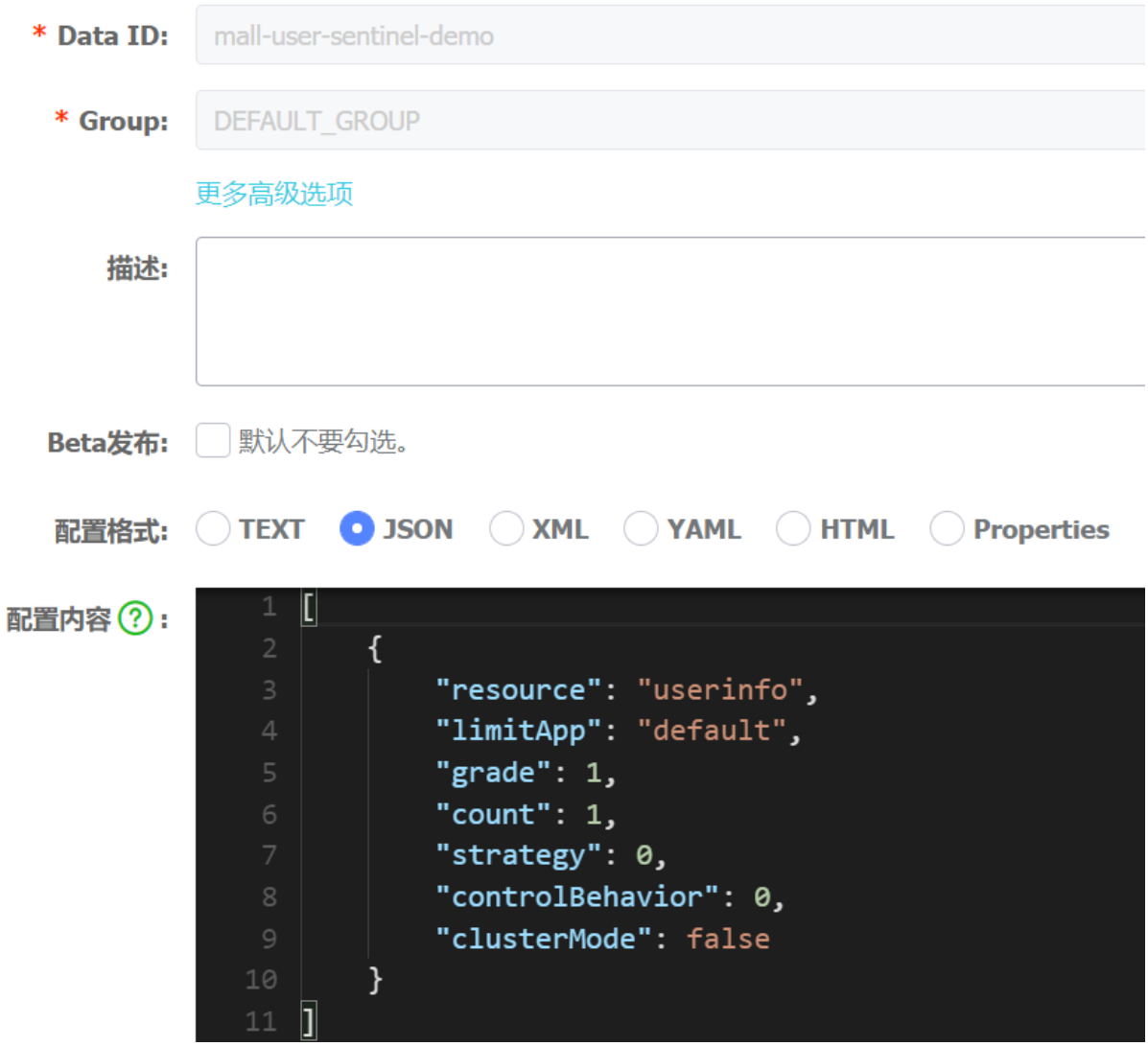
```
spring:
  application:
```

```yaml
    name: mall-user-sentinel-demo
  cloud:
    nacos:
      discovery:
        server-addr: 127.0.0.1:8848

    sentinel:
      transport:
        # 添加sentinel的控制台地址
        dashboard: 127.0.0.1:8080
        # 指定应用与Sentinel控制台交互的端口，应用本地会起一个该端口占用的HttpServer
        port: 8719
      datasource:
        ds1:
          nacos:
            server-addr: 127.0.0.1:8848
            dataId: ${spring.application.name}
            groupId: DEFAULT_GROUP
            data-type: json
            rule-type: flow
```

3）nacos配置中心中添加

```json
[
    {
        "resource": "userinfo",
        "limitApp": "default",
        "grade": 1,
        "count": 1,
        "strategy": 0,
        "controlBehavior": 0,
        "clusterMode": false
    }
]
```



引入依赖

```
 <!--sentinel持久化 采用 Nacos 作为规则配置数据源-->
<dependency>
    <groupId>com.alibaba.csp</groupId>
    <artifactId>sentinel-datasource-nacos</artifactId>
</dependency>
```

增加yml配置

```
server:
  port: 8806

spring:
  application:
    name: mall-user-sentinel-rule-push-demo  #微服务名称

  #配置nacos注册中心地址
  cloud:
    nacos:
```

```yaml
    discovery:
      server-addr: 127.0.0.1:8848

  sentinel:
    transport:
      # 添加sentinel的控制台地址
      dashboard: 127.0.0.1:8080
      # 指定应用与Sentinel控制台交互的端口，应用本地会起一个该端口占用的HttpServer
      #port: 8719
    datasource:
#       ds1:   #名称自定义，唯一
#         nacos:
#           server-addr: 127.0.0.1:8848
#           dataId: ${spring.application.name}
#           groupId: DEFAULT_GROUP
#           data-type: json
#           rule-type: flow
      flow-rules:
        nacos:
          server-addr: 127.0.0.1:8848
          dataId: ${spring.application.name}-flow-rules
          groupId: SENTINEL_GROUP   # 注意groupId对应Sentinel Dashboard中的定义
          data-type: json
          rule-type: flow
      degrade-rules:
        nacos:
          server-addr: 127.0.0.1:8848
          dataId: ${spring.application.name}-degrade-rules
          groupId: SENTINEL_GROUP
          data-type: json
          rule-type: degrade
      param-flow-rules:
        nacos:
          server-addr: 127.0.0.1:8848
          dataId: ${spring.application.name}-param-flow-rules
          groupId: SENTINEL_GROUP
          data-type: json
          rule-type: param-flow
      authority-rules:
        nacos:
          server-addr: 127.0.0.1:8848
```

```
            dataId: ${spring.application.name}-authority-rules
            groupId: SENTINEL_GROUP
            data-type: json
            rule-type: authority
        system-rules:
          nacos:
            server-addr: 127.0.0.1:8848
            dataId: ${spring.application.name}-system-rules
            groupId: SENTINEL_GROUP
            data-type: json
            rule-type: system
```

以流控规则测试，当在sentinel dashboard配置了流控规则，会在nacos配置中心生成对应的配置。