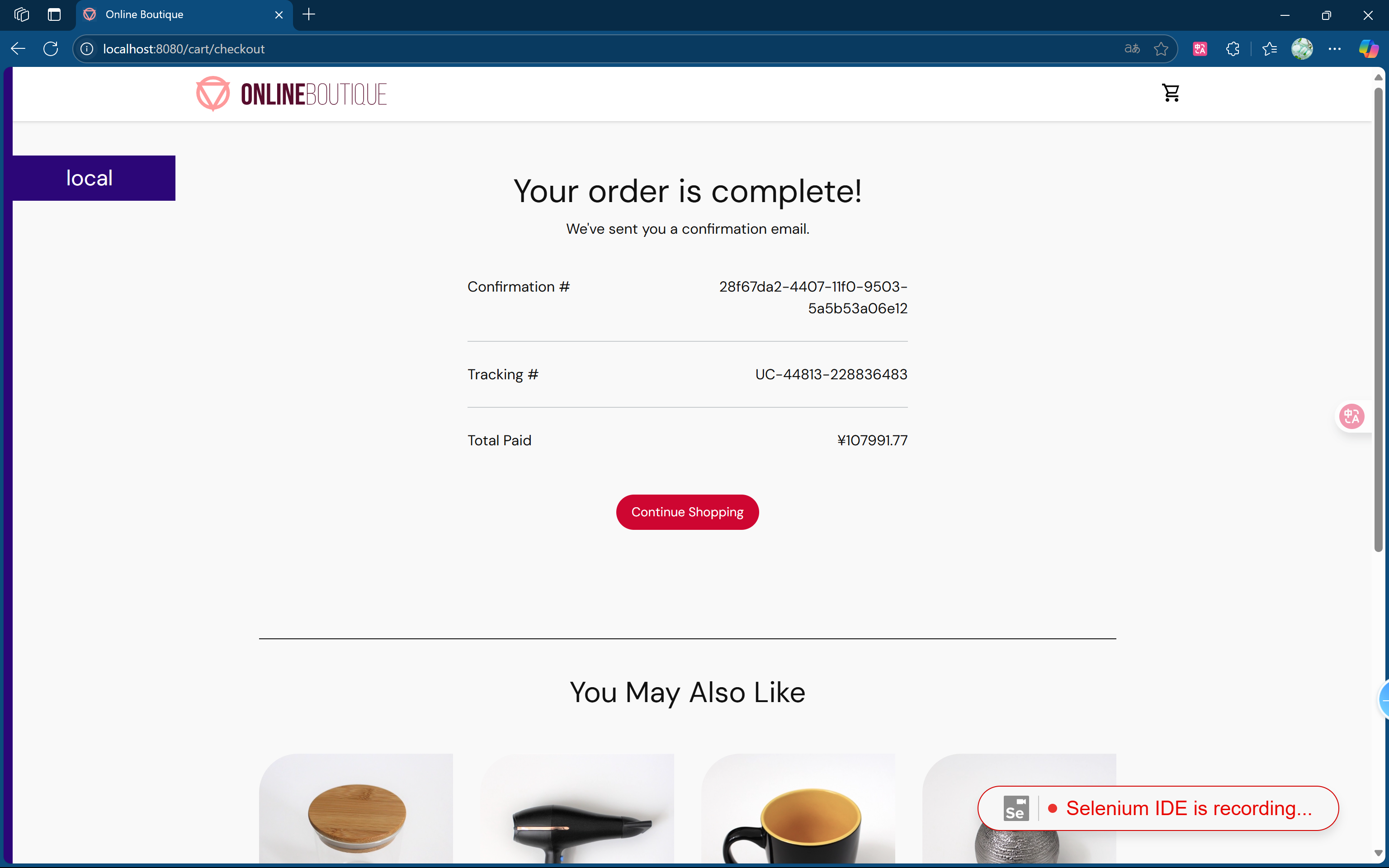
**测试文档**

**Selenium**

先后完成了

进入初始页面——>初始页面点击操作——>点击具体商品页面——>添加至购物车——>继续购买商品——>点击增加商品数量——>成功结算



**具体步骤解析**

|  |
| --- |
| Python self.driver.get("http://localhost:8080/") |

* 打开本地运行的 Web 应用主页。

|  |
| --- |
| Python self.driver.get("http://localhost:8080/") self.driver.set\_window\_size(802, 863) |

* 设置浏览器窗口大小，确保元素显示一致。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".home-mobile-hero-banner").click() |

* 点击页面中的某个横幅（通常是首页的宣传图/广告图）。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".home-mobile-hero-banner").click() self.driver.find\_element(By.NAME, "currency\_code").click() dropdown.find\_element(By.XPATH, "//option[. = 'JPY']").click() |

* 打开并选择货币类型为 **JPY（日元）**。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".col-md-4:nth-child(3) .hot-product-card-img-overlay").click() |

* 点击页面上的第3个热门商品（假设是一个卡片）。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".col-md-4:nth-child(3) .hot-product-card-img-overlay").click() self.driver.find\_element(By.ID, "quantity").click() dropdown.find\_element(By.XPATH, "//option[. = '3']").click() |

* 选择购买数量为 **3**。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".cymbal-button-primary").click() |

* 点击 **Add to Cart** 或 **Buy Now** 按钮。

|  |
| --- |
| Python self.driver.find\_element(By.LINK\_TEXT, "Continue Shopping").click() |

* 返回继续购物。

|  |
| --- |
| Python self.driver.find\_element(By.LINK\_TEXT, "Continue Shopping").click() self.driver.find\_element(By.CSS\_SELECTOR, ".col-md-4:nth-child(5) .hot-product-card-img-overlay").click() dropdown.find\_element(By.XPATH, "//option[. = '10']").click() self.driver.find\_element(By.LINK\_TEXT, "Continue Shopping").click() |

* 再次选择一个新商品（第5个），添加 **10** 件，继续购物。

|  |
| --- |
| Python self.driver.find\_element(By.LINK\_TEXT, "13").click() |

* 点击页面上显示为 “13” 的链接（可能是商品页、分页或其他 UI 元素）。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".cymbal-button-primary:nth-child(1)").click() |

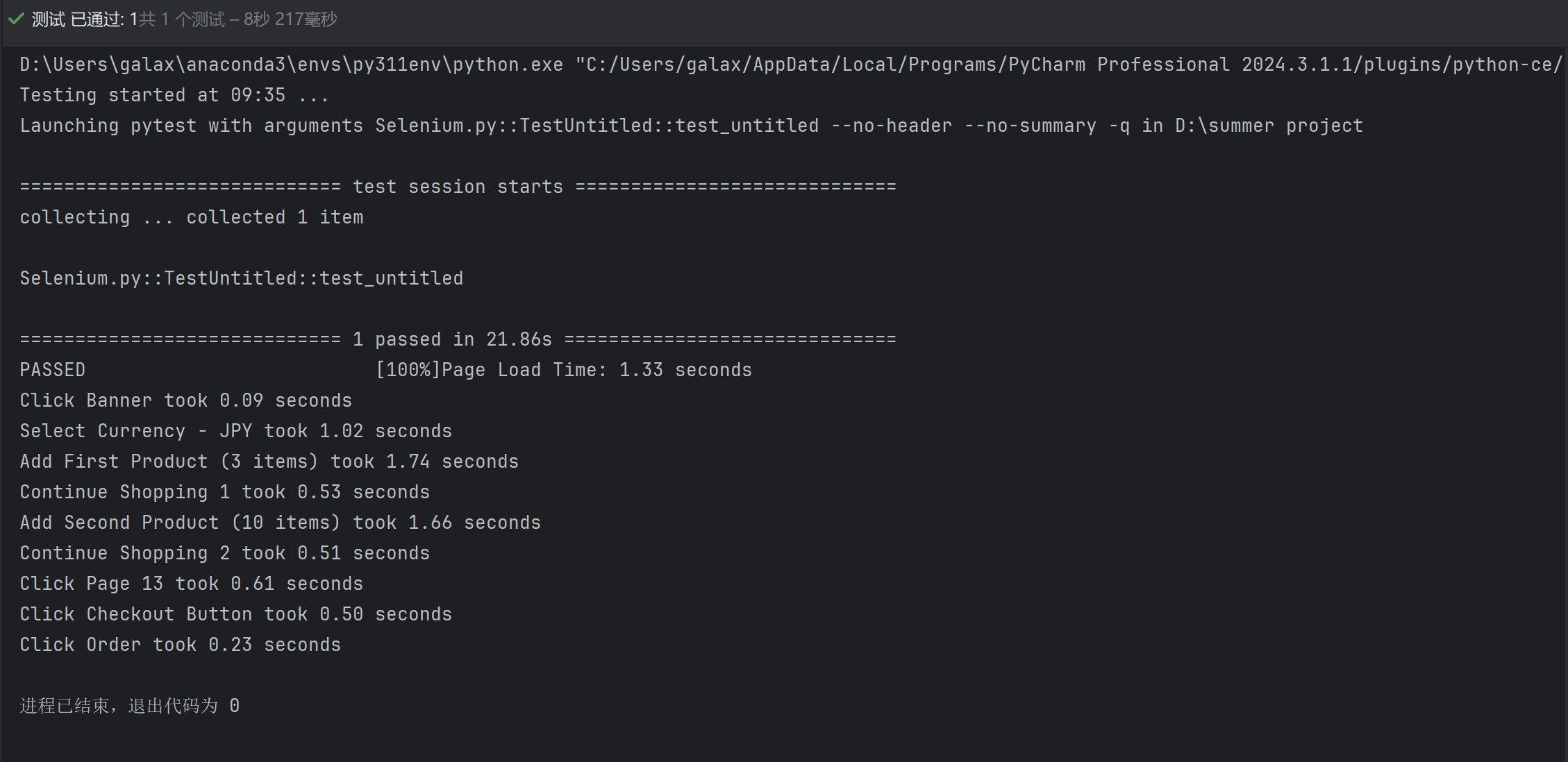
* 执行第一个主按钮点击操作（可能是“结算”或“下订单”）。

|  |
| --- |
| Python self.driver.find\_element(By.CSS\_SELECTOR, ".order").click() |

* 点击订单元素（可能是“提交订单”或“查看订单”）。

完整自动操作和性能测试的代码

|  |
| --- |
| Python import pytest import time import json from selenium import webdriver from selenium.webdriver.common.by import By from selenium.webdriver.support import expected\_conditions from selenium.webdriver.support.wait import WebDriverWait from selenium.webdriver.common.keys import Keys import os  class TestUntitled():  def setup\_method(self, method):  self.driver = webdriver.Chrome()  self.vars = {}  self.timings = []   def teardown\_method(self, method):  # 保存性能数据为 JSON 文件  with open("selenium\_performance\_report.json", "w") as f:  json.dump(self.timings, f, indent=2)  self.driver.quit()   def log\_timing(self, name, action):  start = time.time()  action()  duration = time.time() - start  print(f"{name} took {duration:.2f} seconds")  self.timings.append({"step": name, "duration": round(duration, 3)})   def test\_untitled(self):  # 1. 记录页面加载时间  start\_time = time.time()  self.driver.get("http://localhost:8080/")  self.driver.set\_window\_size(802, 863)  WebDriverWait(self.driver, 10).until(  expected\_conditions.presence\_of\_element\_located((By.CSS\_SELECTOR, ".home-mobile-hero-banner"))  )  page\_load\_time = time.time() - start\_time  print(f"Page Load Time: {page\_load\_time:.2f} seconds")  self.timings.append({"step": "Load Home Page", "duration": round(page\_load\_time, 3)})   # 2. 主页交互  self.log\_timing("Click Banner", lambda: self.driver.find\_element(By.CSS\_SELECTOR, ".home-mobile-hero-banner").click())   # 3. 切换货币  def select\_currency():  dropdown = self.driver.find\_element(By.NAME, "currency\_code")  dropdown.click()  dropdown.find\_element(By.XPATH, "//option[. = 'JPY']").click()  self.log\_timing("Select Currency - JPY", select\_currency)   # 4. 添加第一个商品（3个）  def select\_first\_product():  self.driver.find\_element(By.CSS\_SELECTOR, ".col-md-4:nth-child(3) .hot-product-card-img-overlay").click()  quantity\_dropdown = self.driver.find\_element(By.ID, "quantity")  quantity\_dropdown.click()  quantity\_dropdown.find\_element(By.XPATH, "//option[. = '3']").click()  self.driver.find\_element(By.CSS\_SELECTOR, ".cymbal-button-primary").click()  self.log\_timing("Add First Product (3 items)", select\_first\_product)   self.log\_timing("Continue Shopping 1", lambda: self.driver.find\_element(By.LINK\_TEXT, "Continue Shopping").click())   # 5. 添加第二个商品（10个）  def select\_second\_product():  self.driver.find\_element(By.CSS\_SELECTOR, ".col-md-4:nth-child(5) .hot-product-card-img-overlay").click()  quantity\_dropdown = self.driver.find\_element(By.ID, "quantity")  quantity\_dropdown.click()  quantity\_dropdown.find\_element(By.XPATH, "//option[. = '10']").click()  self.driver.find\_element(By.CSS\_SELECTOR, ".cymbal-button-primary").click()  self.log\_timing("Add Second Product (10 items)", select\_second\_product)   self.log\_timing("Continue Shopping 2", lambda: self.driver.find\_element(By.LINK\_TEXT, "Continue Shopping").click())   # 6. 跳转某页面（点击"13"）  self.log\_timing("Click Page 13", lambda: self.driver.find\_element(By.LINK\_TEXT, "13").click())   # 7. 点击结算按钮  self.log\_timing("Click Checkout Button", lambda: self.driver.find\_element(By.CSS\_SELECTOR, ".cymbal-button-primary:nth-child(1)").click())   # 8. 查看订单  self.log\_timing("Click Order", lambda: self.driver.find\_element(By.CSS\_SELECTOR, ".order").click()) |



**测试结果总结**



**点击图片可查看完整电子表格**

**📊 性能分析总结**

**🔵 页面加载时间**

* **1.33秒**，表现良好。用户通常可接受 1-3 秒范围内。

**🔵 页面交互响应时间**

* 大多数操作 **在 0.5~1.7 秒之间**，用户体验良好；
* **“切换货币”、“添加商品”** 操作稍慢（>1秒），若在生产环境应注意是否有接口瓶颈或资源加载延迟。

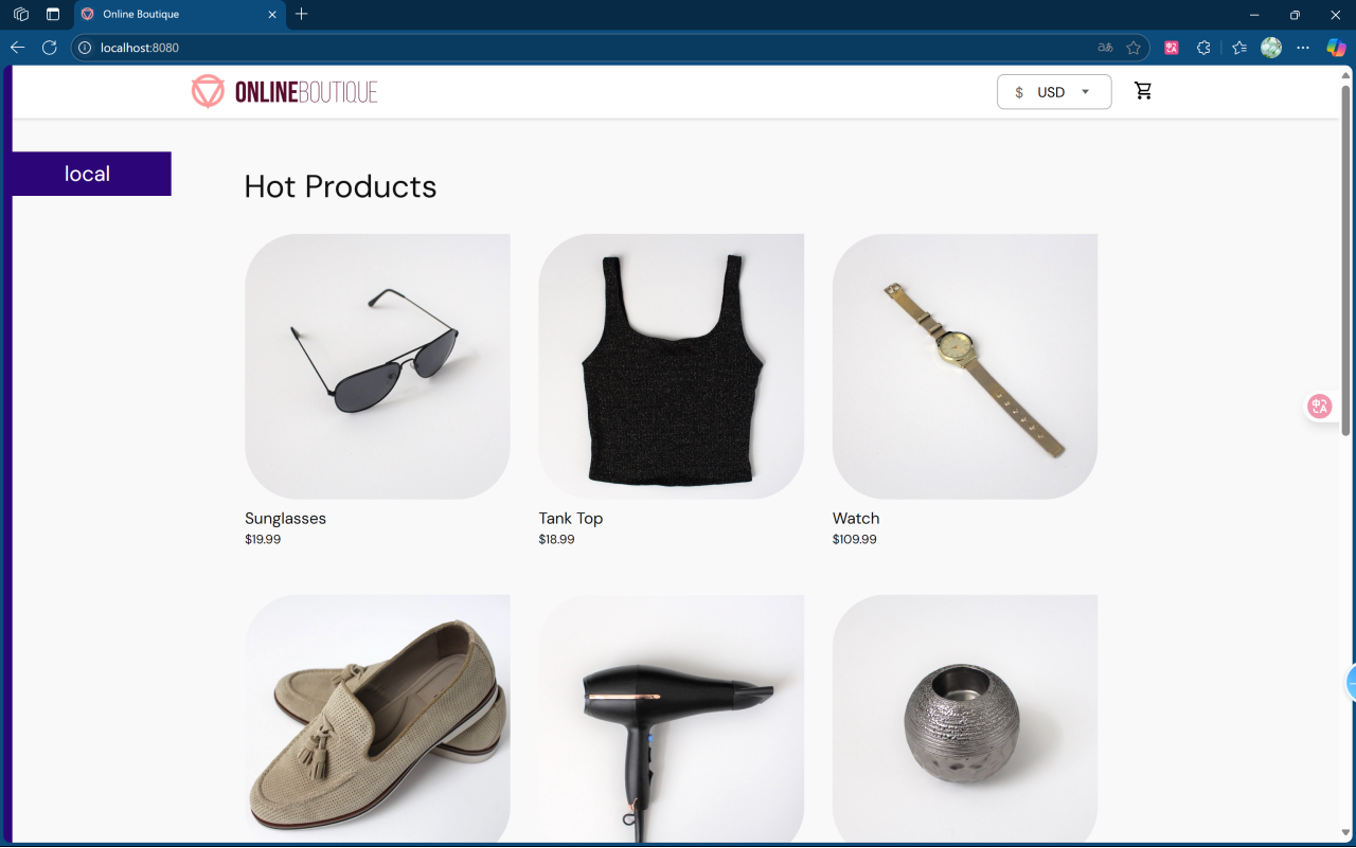
**🔵 特别关注点**

* 两次添加商品分别耗时 1.74s 和 1.66s：
* 若这些动作涉及前后端通信（如库存查询或定价刷新），建议查看后端接口耗时。
* 切换货币耗时 1.02s：
* 如触发了全站刷新或重新拉取数据，可考虑前端缓存或优化更新策略。

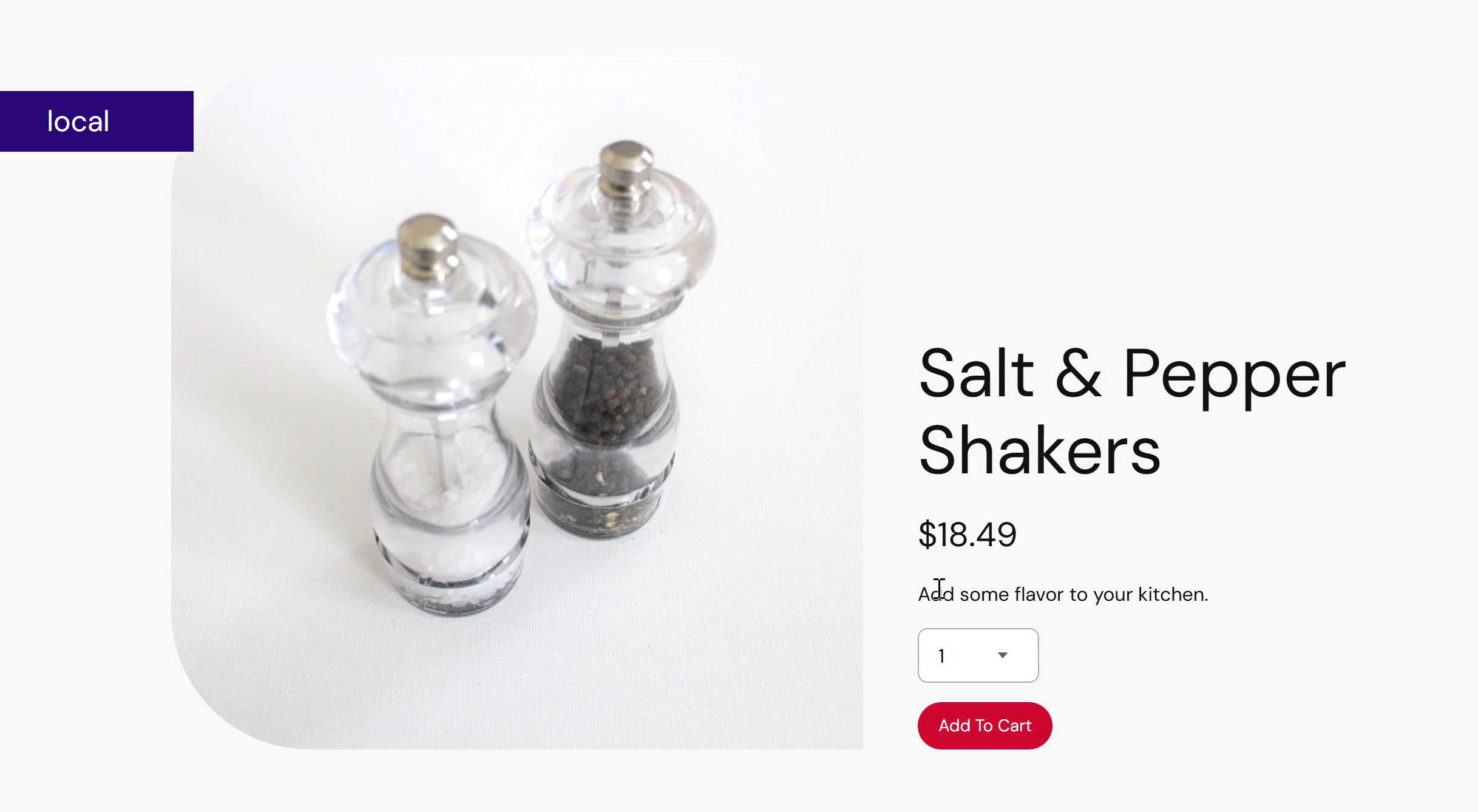
**JMeter**

* 配置好JMeter环境
* 完成两个测试：商品列表&商品详情

商品列表：



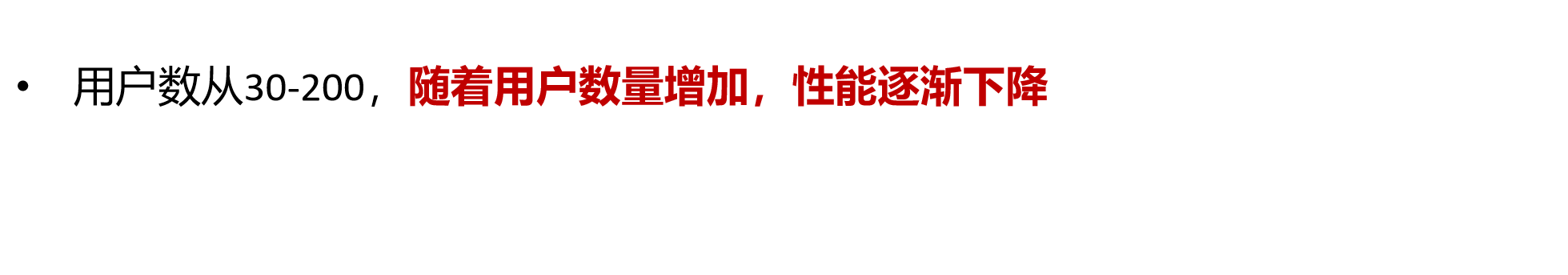
商品详情：



* 分别进行用户数30、50、150、200的测试

结果和生成的代码完整文件如下：





**以用户数为30为例（因为这个测试结果比较好）**

1. APDEX (Application Performance Index)

* 商品列表的APDEX值为0.558，属于不理想的性能，这意味着约有一部分请求未能满足期望的响应时间，表现出一定的性能问题。
* 商品详情的APDEX值为0.987，表示性能较好，大多数请求都能够在预期的时间内完成。
* 应用程序性能指标总值为0.771处于中等偏上，表明有一定的响应延迟。



1. 请求统计（Requests Summary）

* 所有的请求都通过（100% Pass），没有任何失败的请求，说明测试中的请求没有出现错误。
* 请求统计百分百通过，表现出系统的稳定性。



1. 统计信息（Statistics）

商品列表：

平均响应时间为703ms，最大值为1199ms，表明商品列表的加载时间比较长，可能需要优化。

90%的请求响应时间为1010ms，95%为1078ms，整体表现较慢。

商品详情：

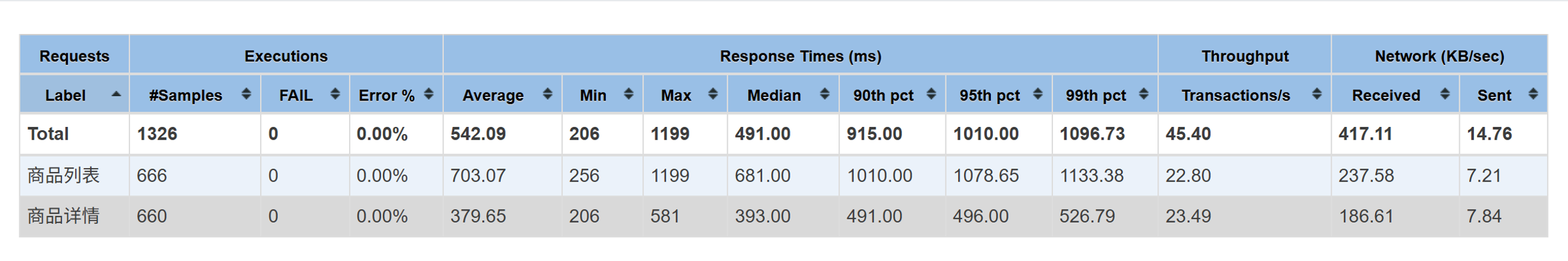
平均响应时间为379ms，相较商品列表要快一些，最大响应时间为581ms，表现良好。

90%的请求响应时间为491ms，95%为496ms，也处于较低的范围，显示出较为稳定的性能。

1. 吞吐量和网络（Throughput and Network）

吞吐量（Transactions/s）：商品列表为22.80，商品详情为23.49，表示每秒处理的事务数量较为平均。

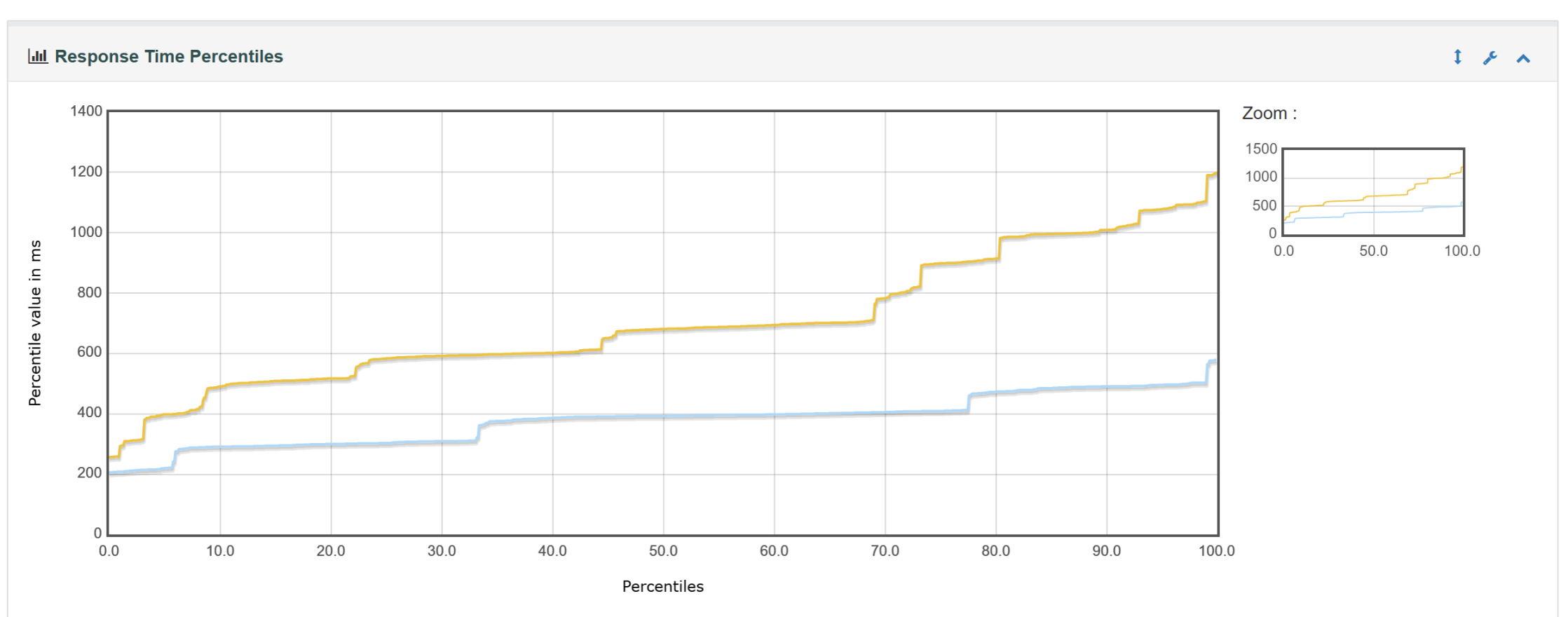
网络流量：接收的数据为237.58 KB/s（商品列表）和186.61 KB/s（商品详情），发送的数据量较小，分别为7.21 KB/s 和 7.84 KB/s，表明数据传输并未成为瓶颈。



其他图标分析：

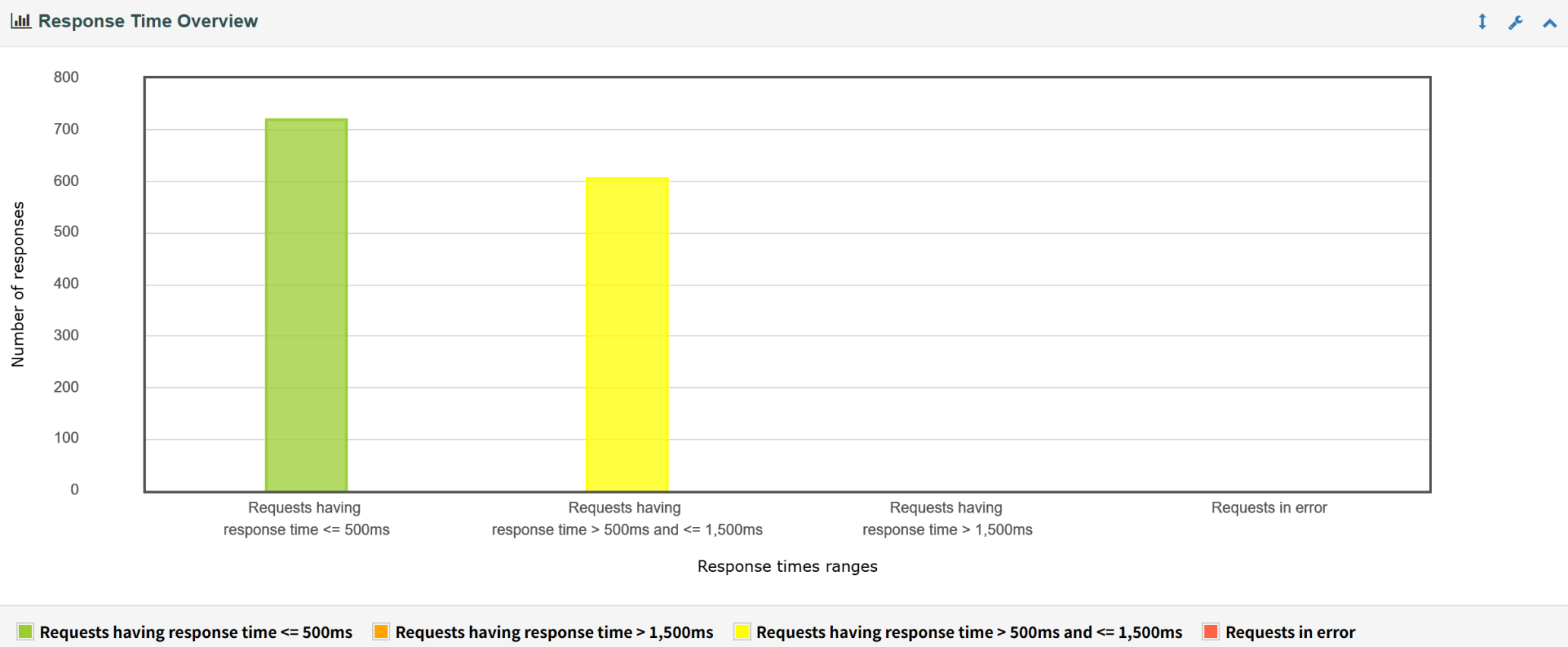
1、百分位响应时间图

* 这个图表展示了不同百分位的响应时间，从低到高展示了大部分请求的响应时间变化。
* 黄色曲线（商品列表）的响应时间显著较高，说明商品列表的页面可能存在性能瓶颈。
* 蓝色曲线（商品详情）的响应时间较低，系统能够较好地处理这些请求。



2、请求响应时间表

* 这个图表展示了大多数请求响应时间都非常迅速（500ms以内），说明系统在大多数情况下运行得很稳定，性能较好。
* 响应时间超过500ms但低于1,500ms的请求相对较少，表示在一些情况下可能有少量的延迟，但整体还是可以接受的。
* 该图表的关键是，大部分请求的响应时间都非常理想，这意味着系统在进行负载测试时表现出了较好的处理能力和性能。



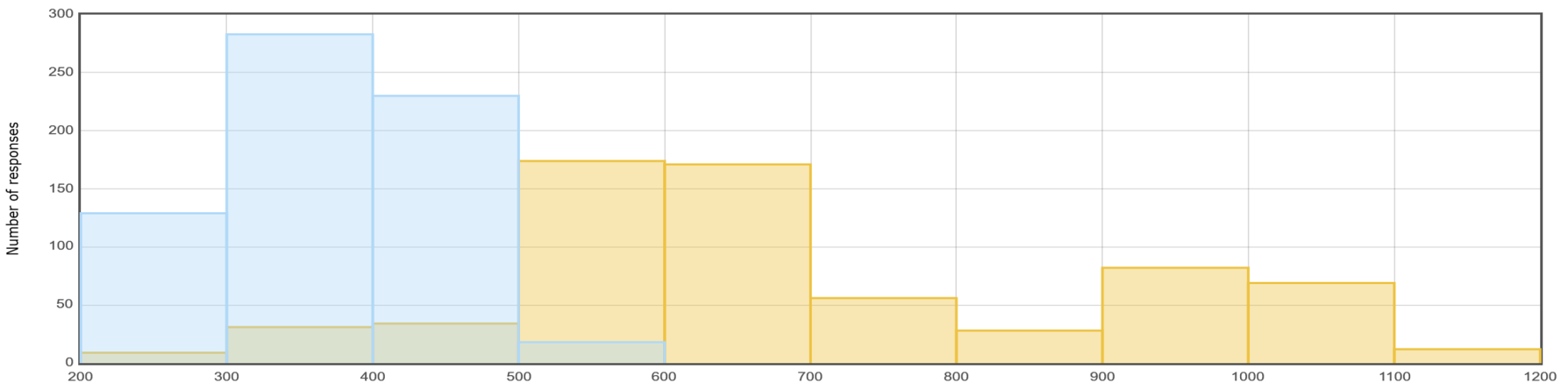
3、性能图像

* 商品列表在高并发情况下表现出性能下降，可能需要考虑优化其数据加载、缓存策略或后台处理逻辑。
* 商品详情的处理性能较好，几乎不受线程数增加的影响，显示出其在高并发场景下的较高可扩展性。



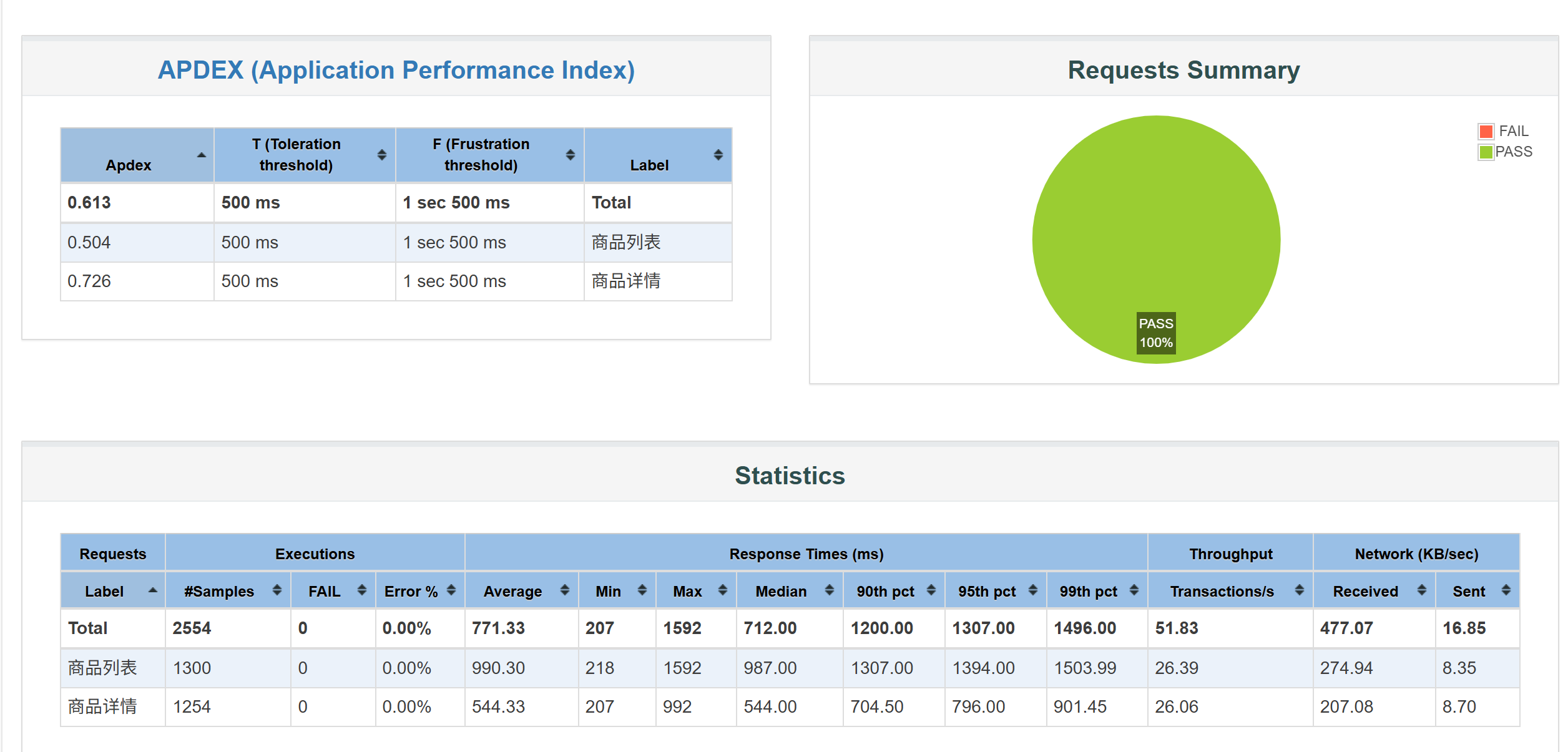
4、响应时间图像

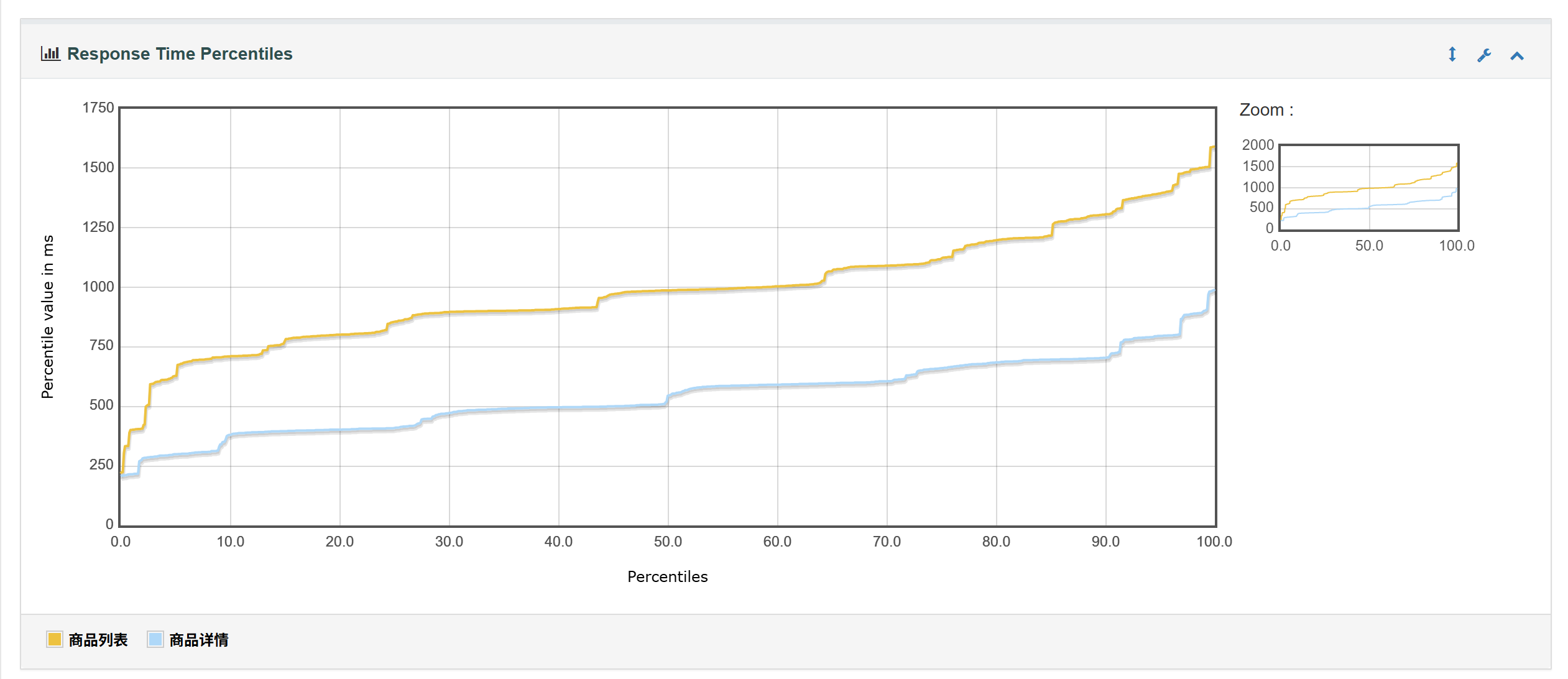
* 商品详情的响应时间较短，绝大多数请求都能在500ms以内完成，表现出良好的性能。
* 商品列表的响应时间分布较广，尤其集中在500ms到1000ms之间，说明商品列表在处理请求时相对较慢，可能存在性能瓶颈。
* 在高负载情况下，商品列表的请求响应时间会显著增加，而商品详情的性能较为稳定。

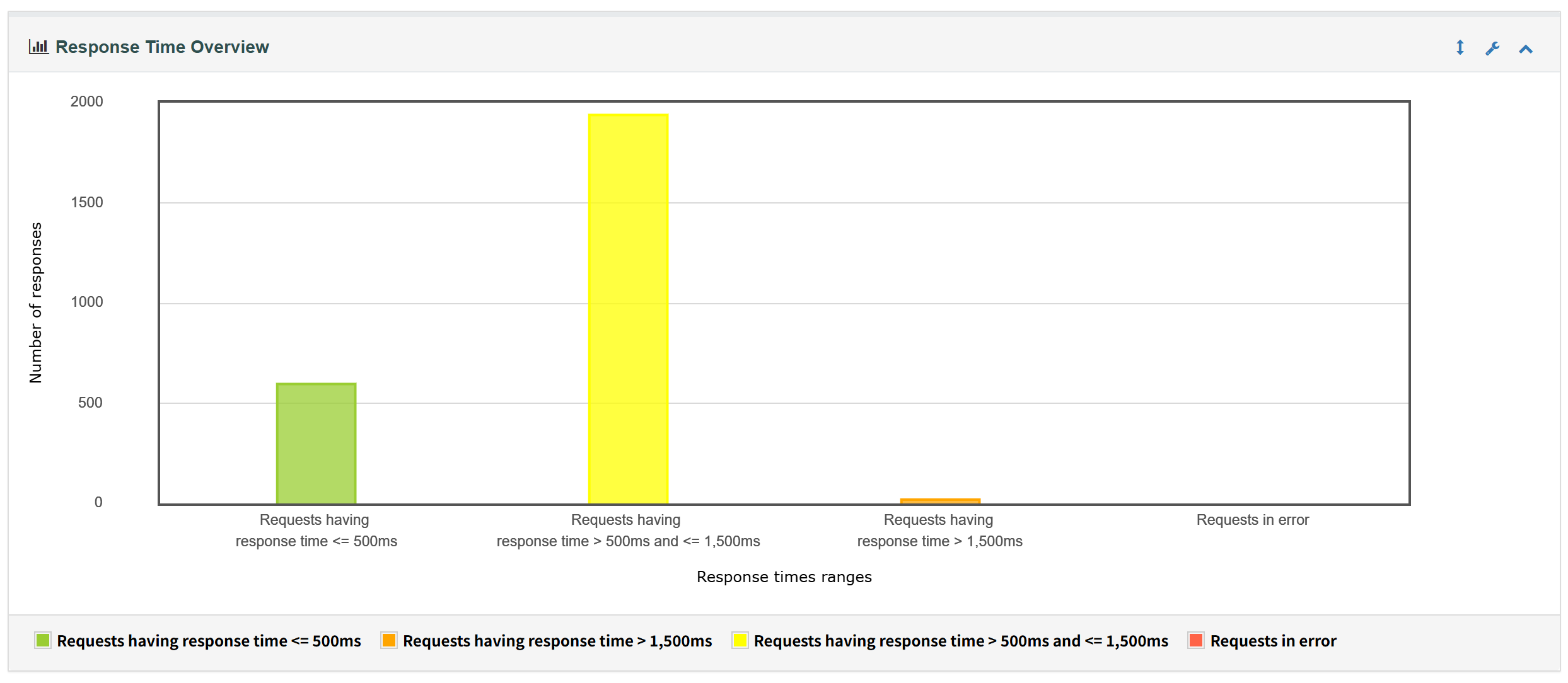


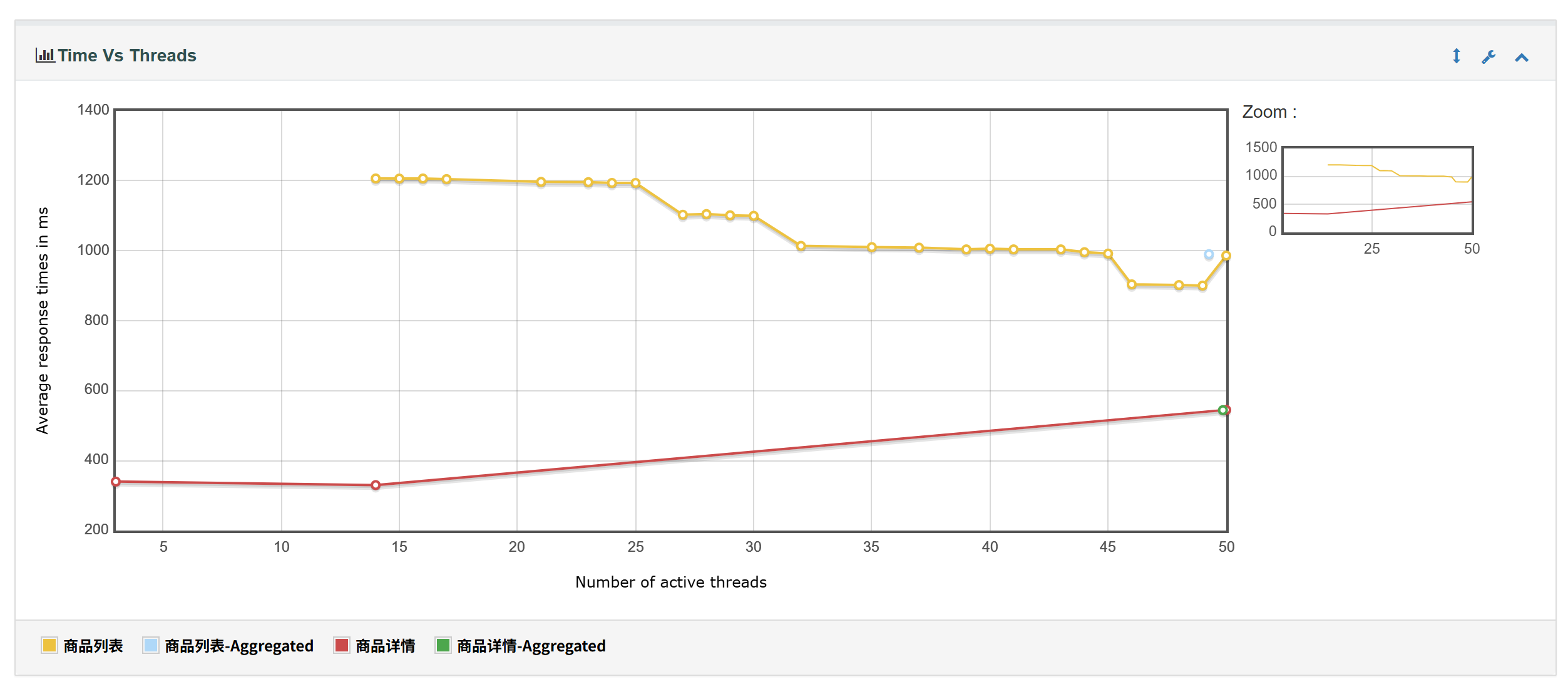
**其他结果：**

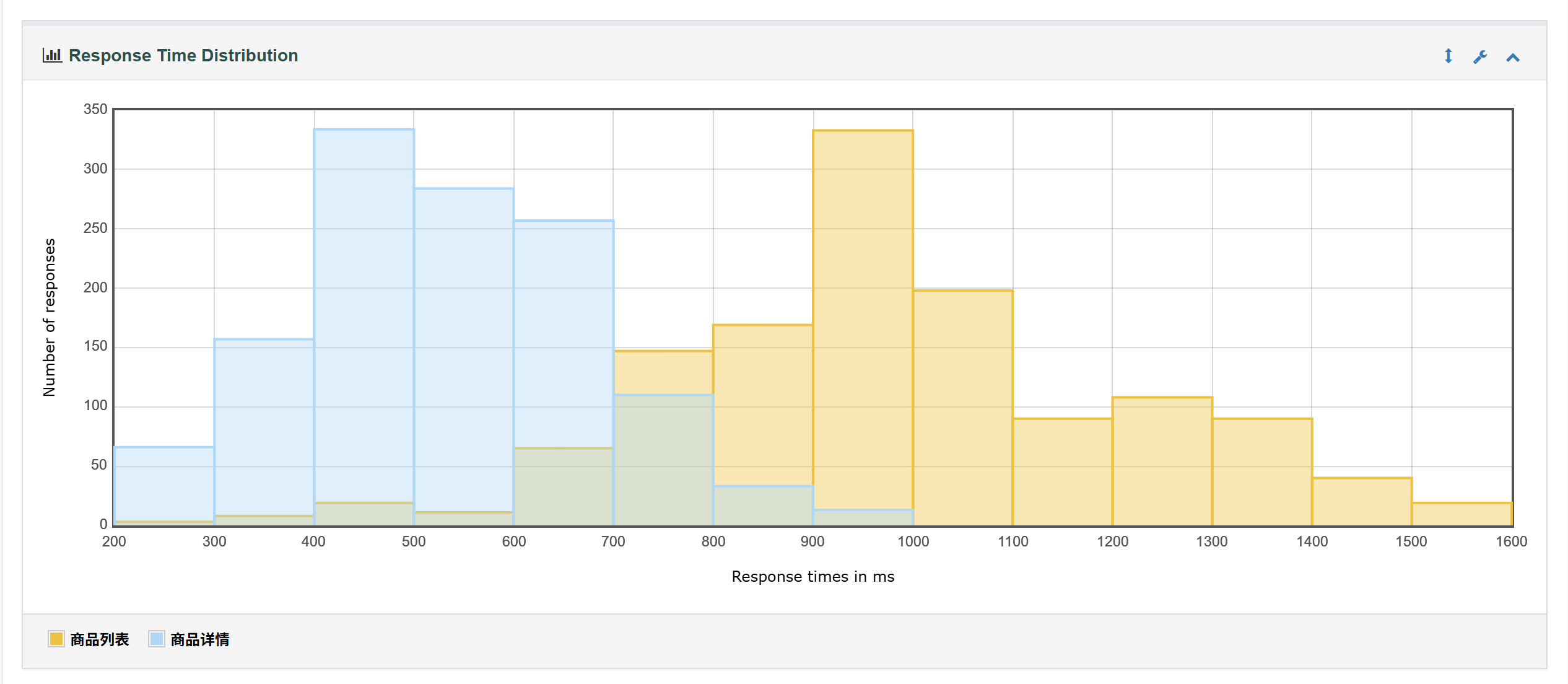
* 用户数为50的Index.html



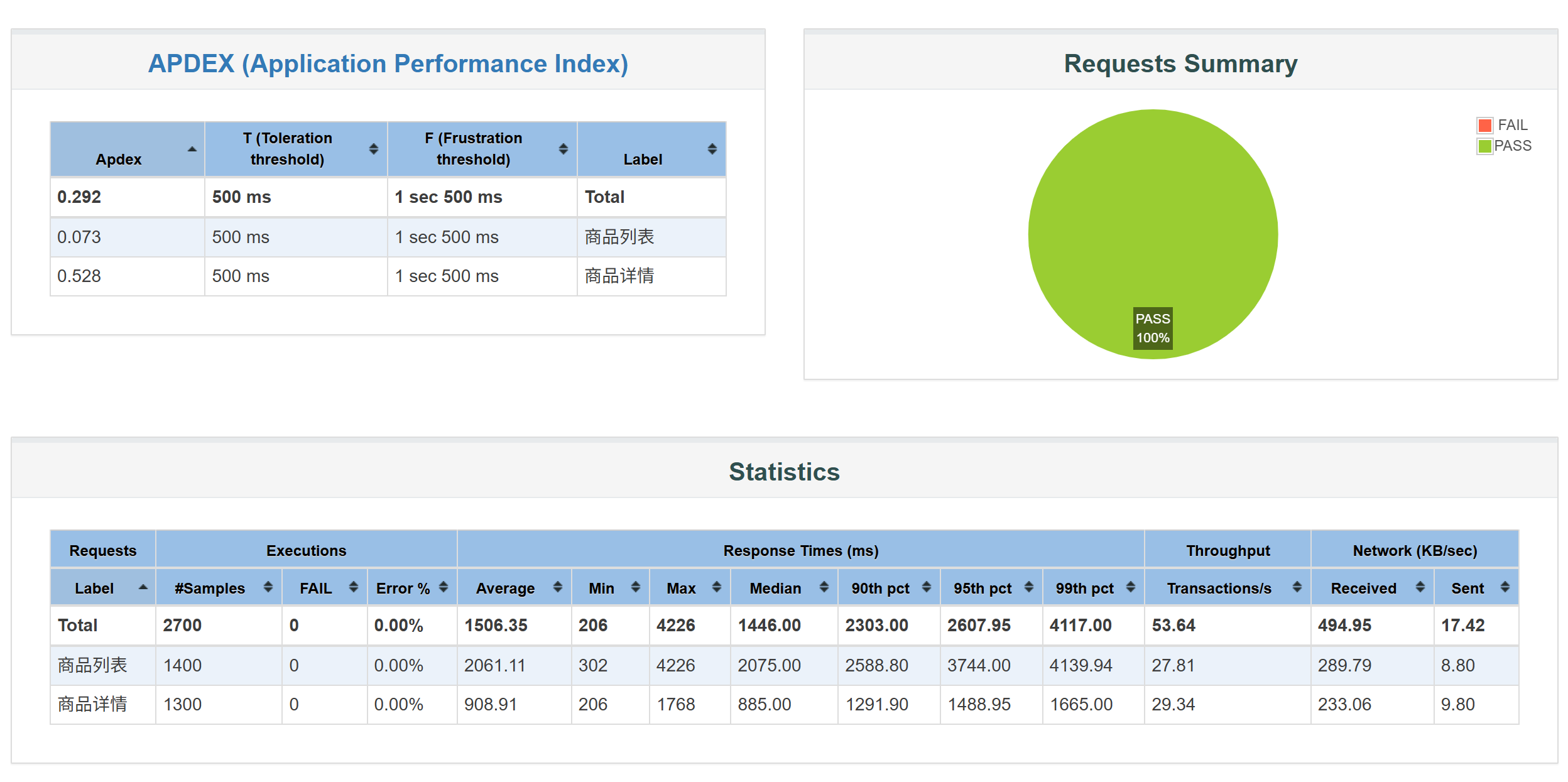


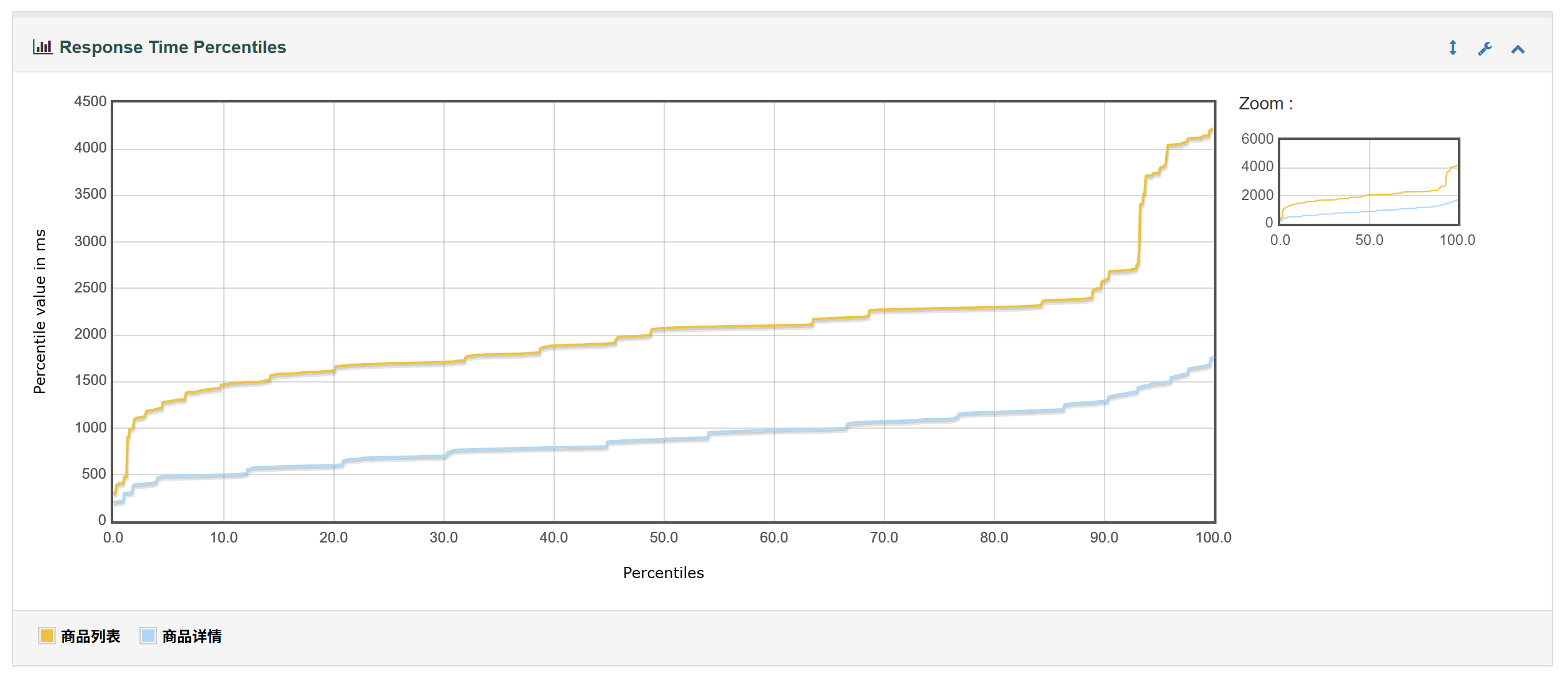


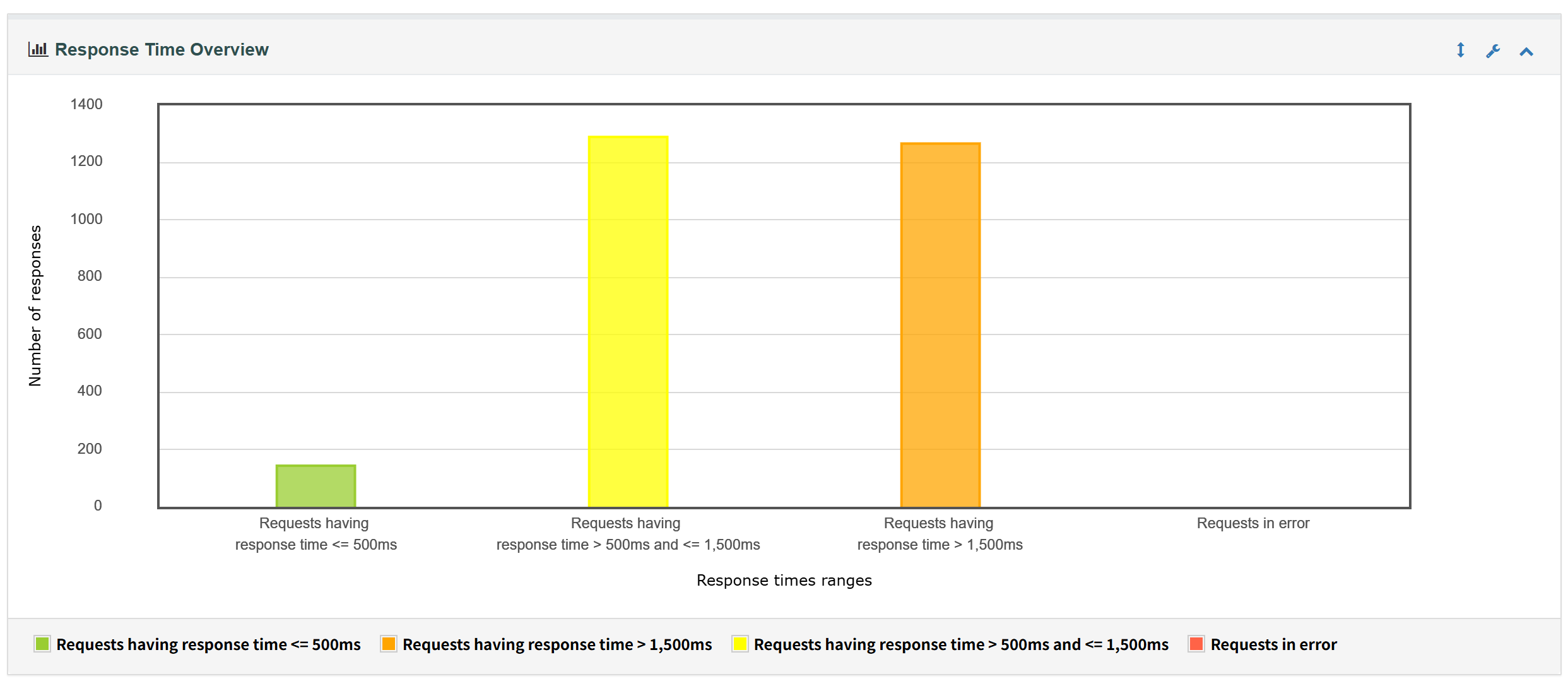


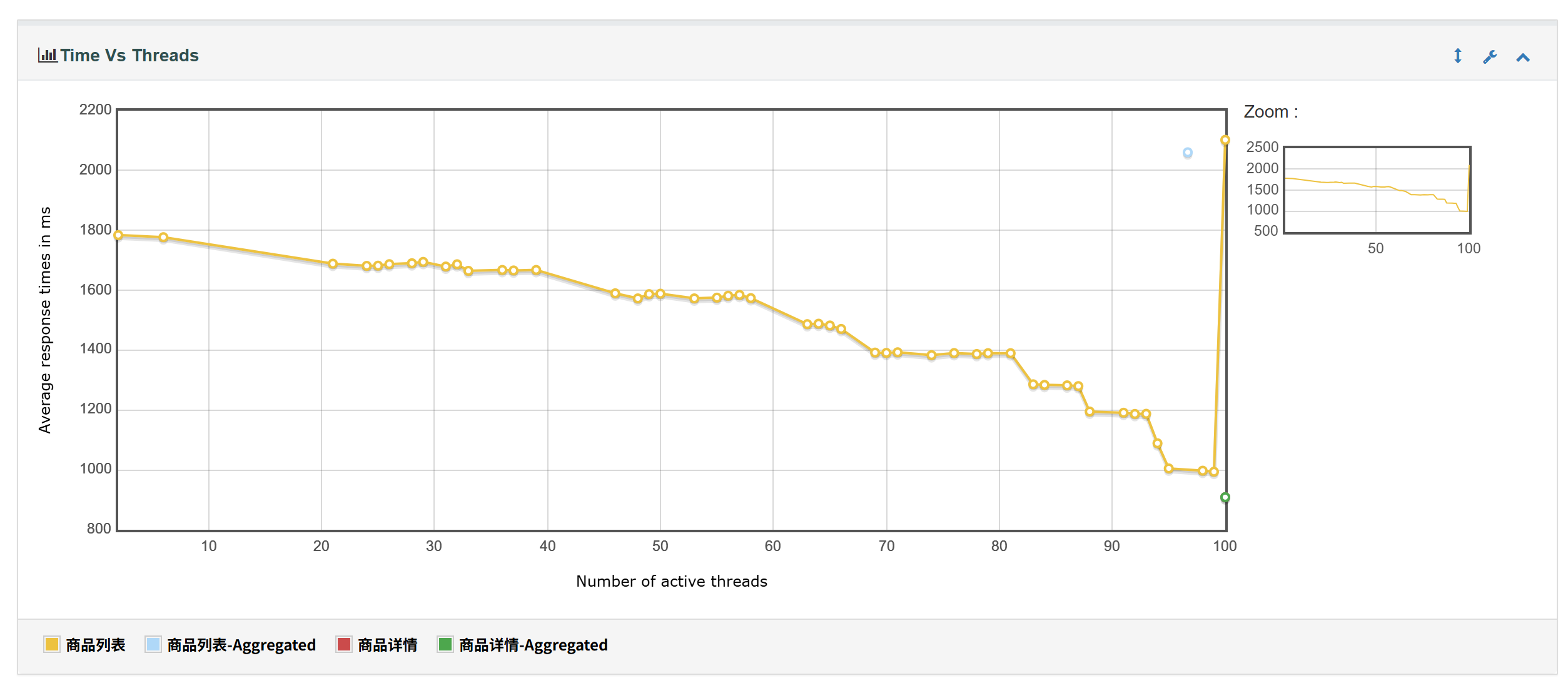


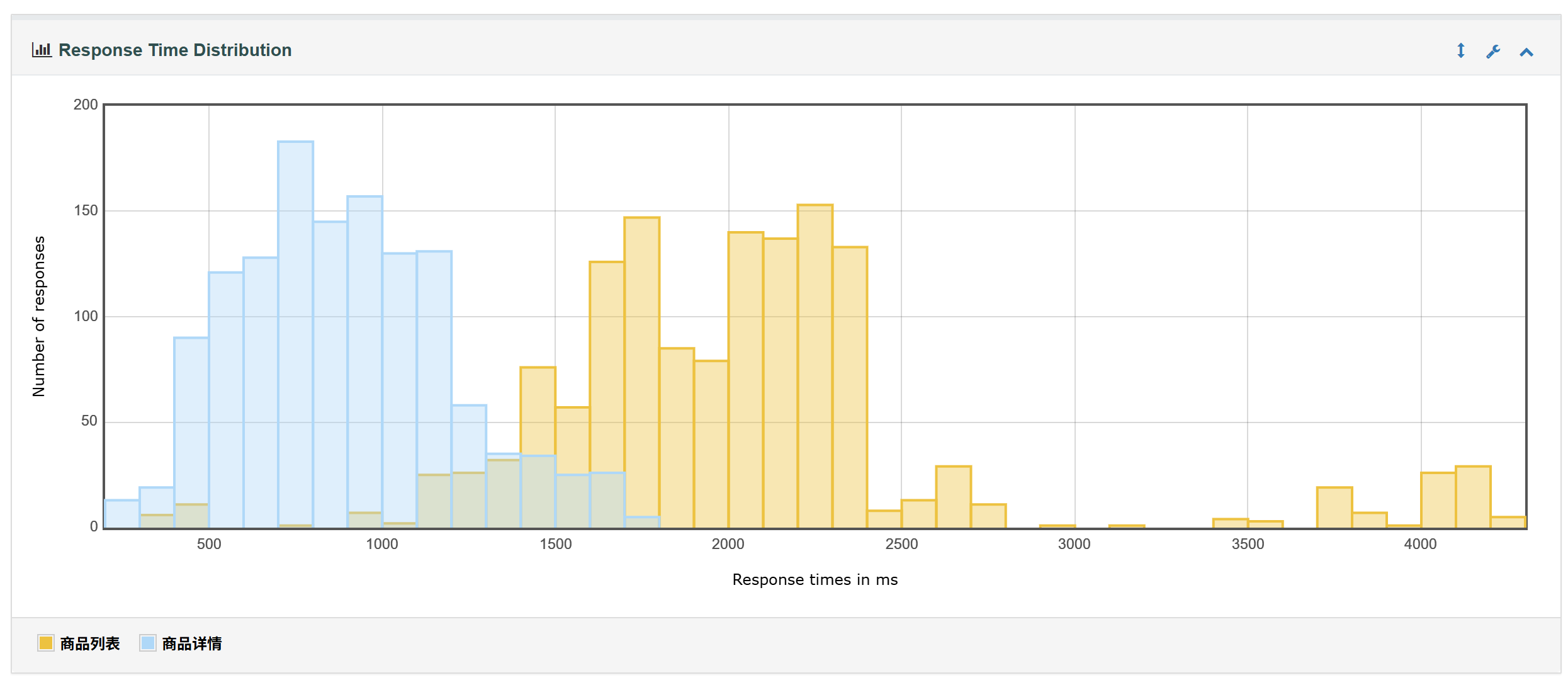
* 用户数为100的index.html



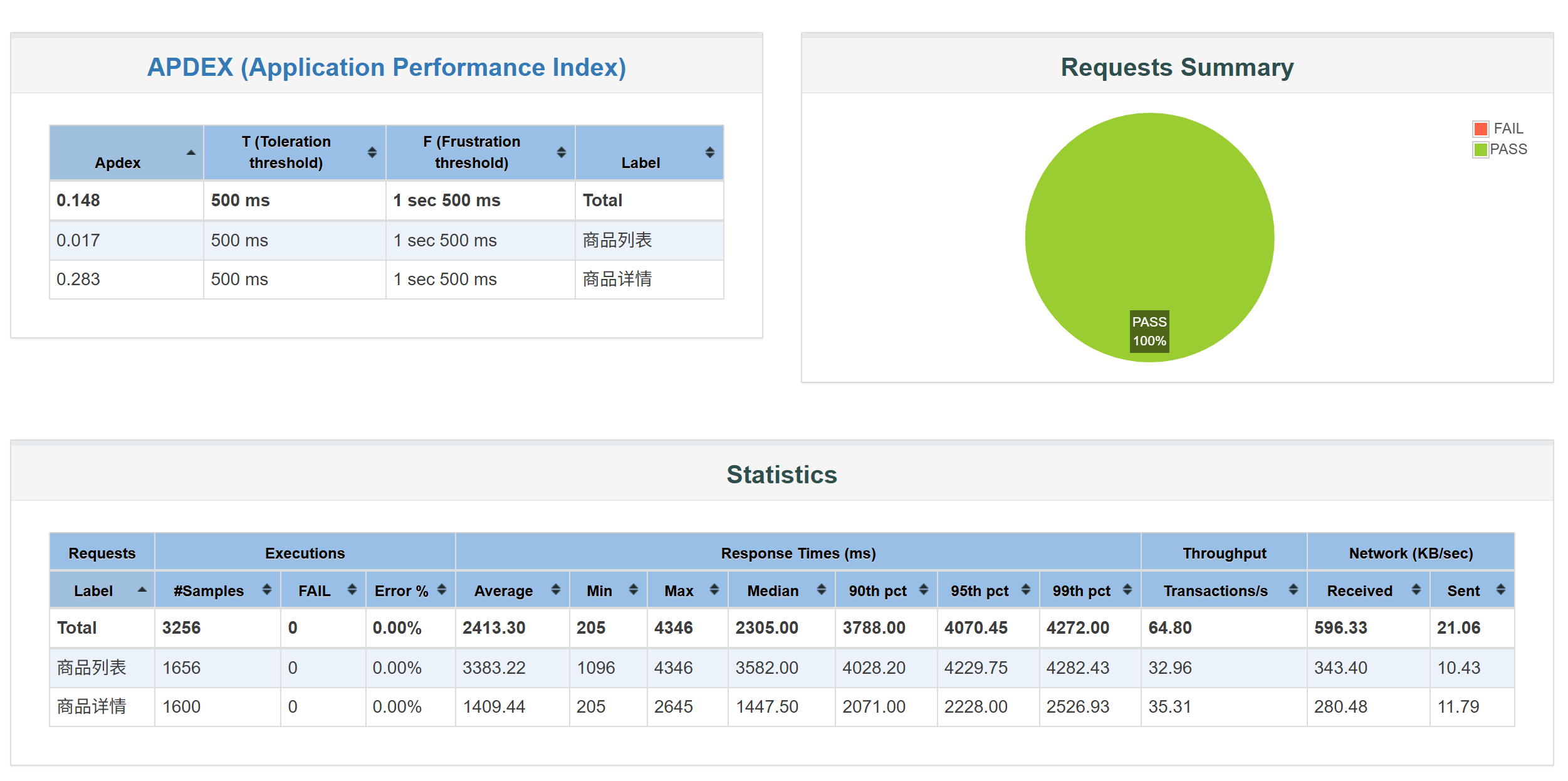


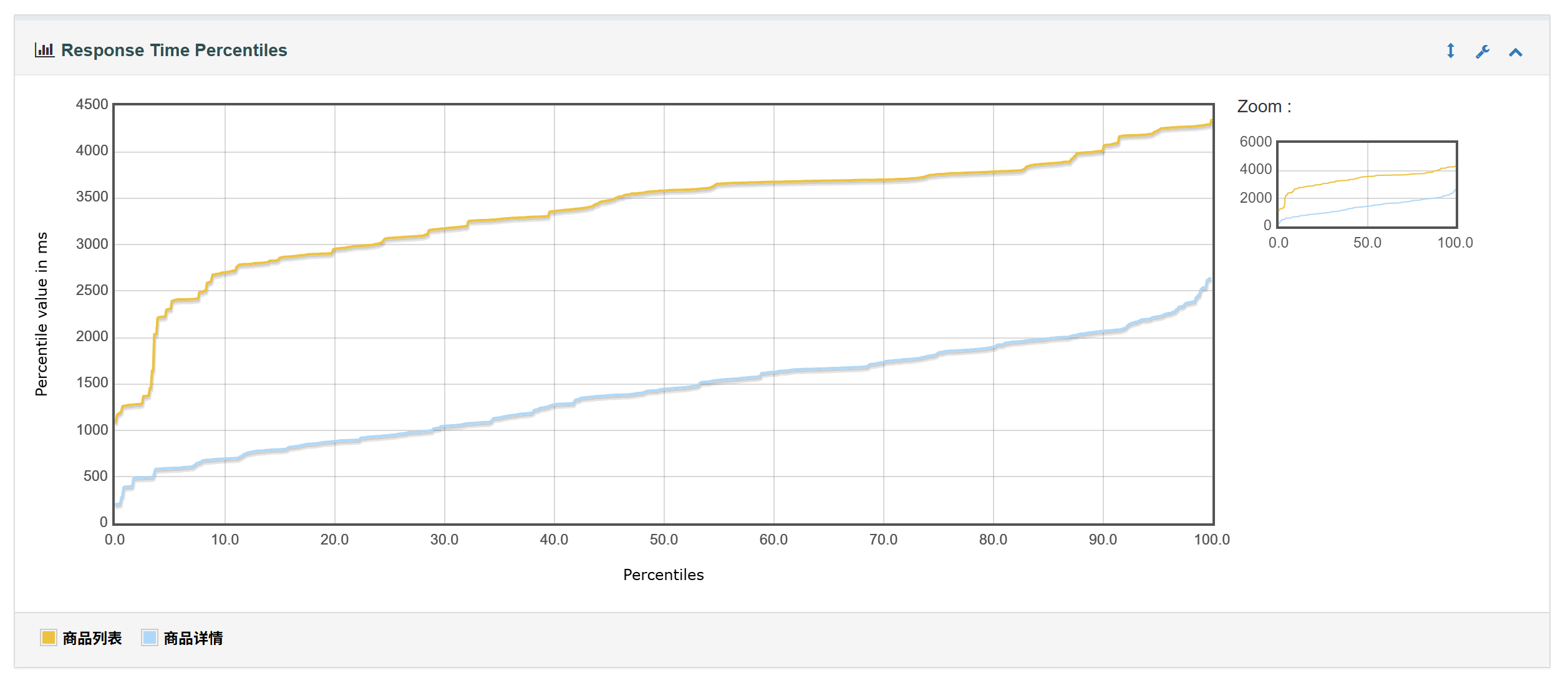


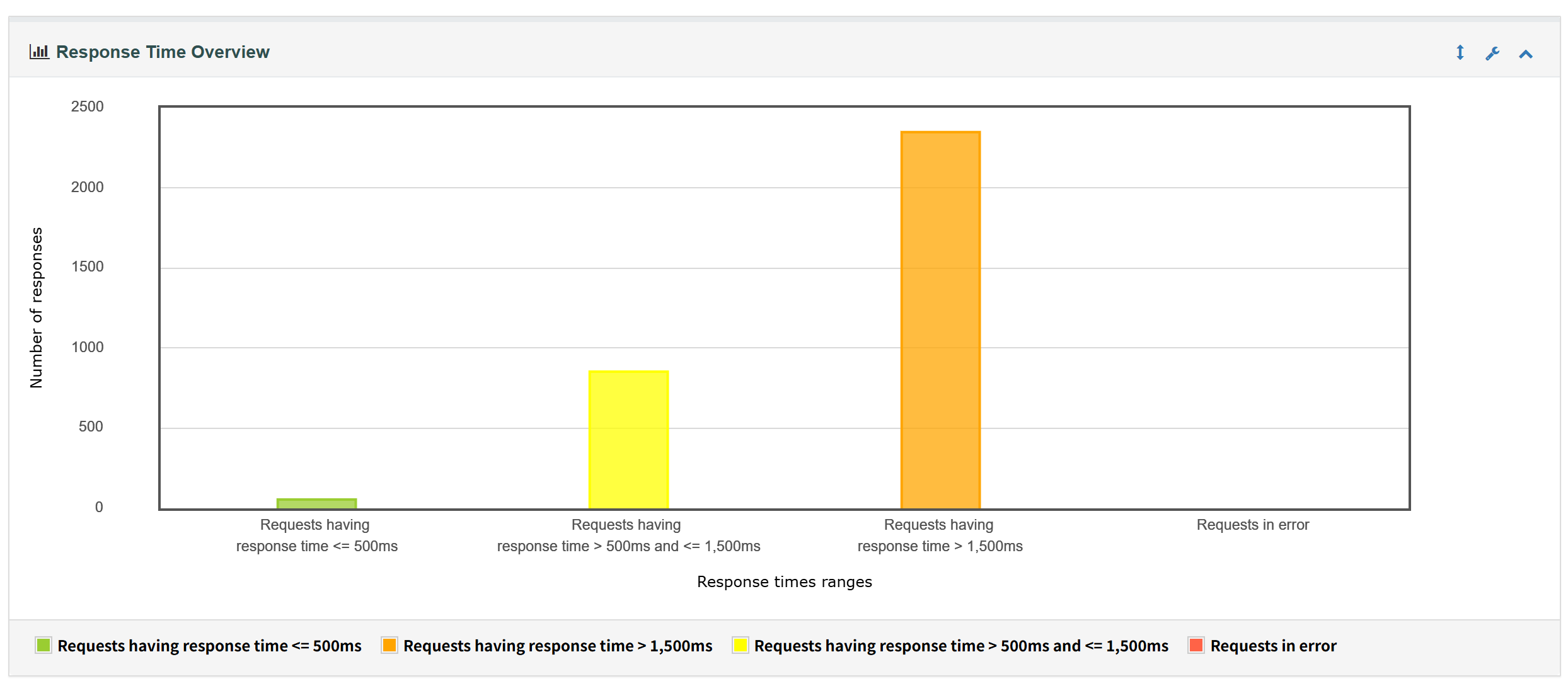


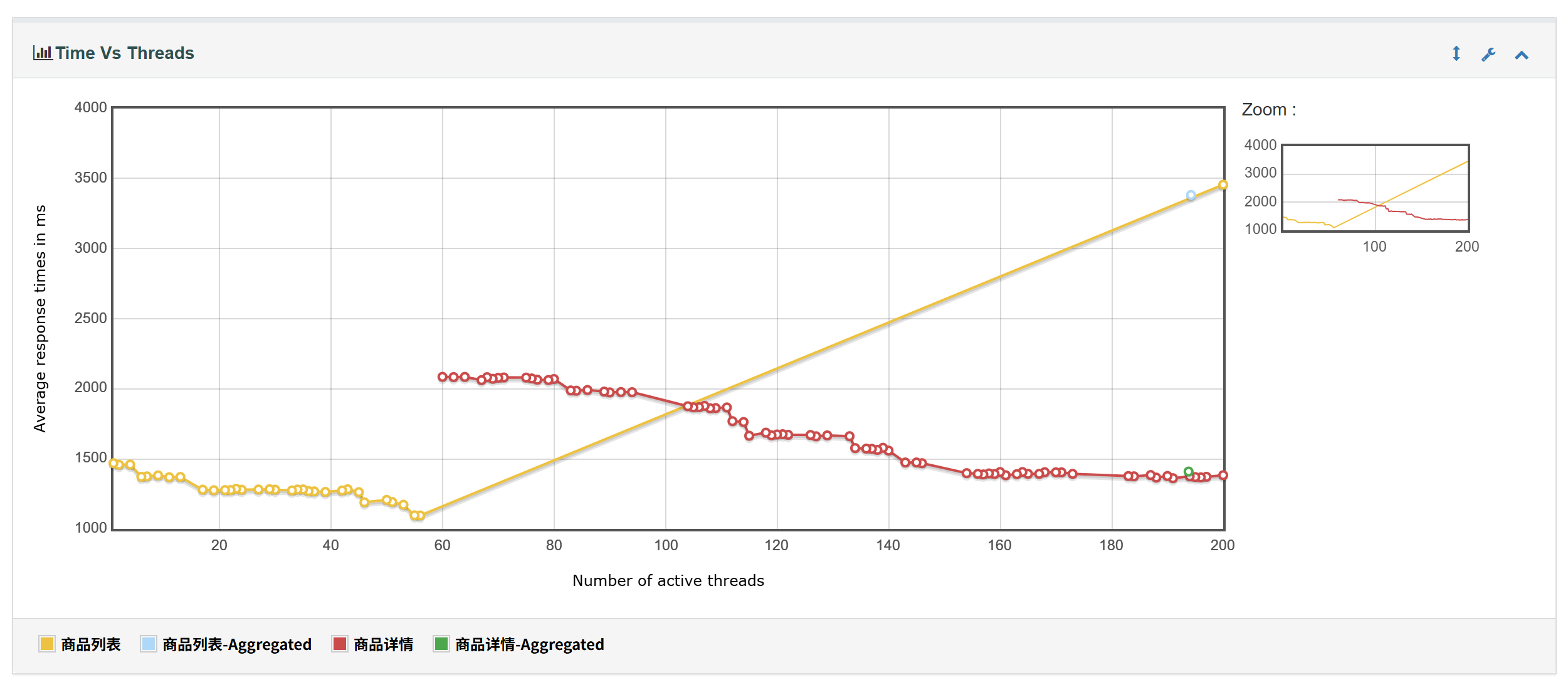


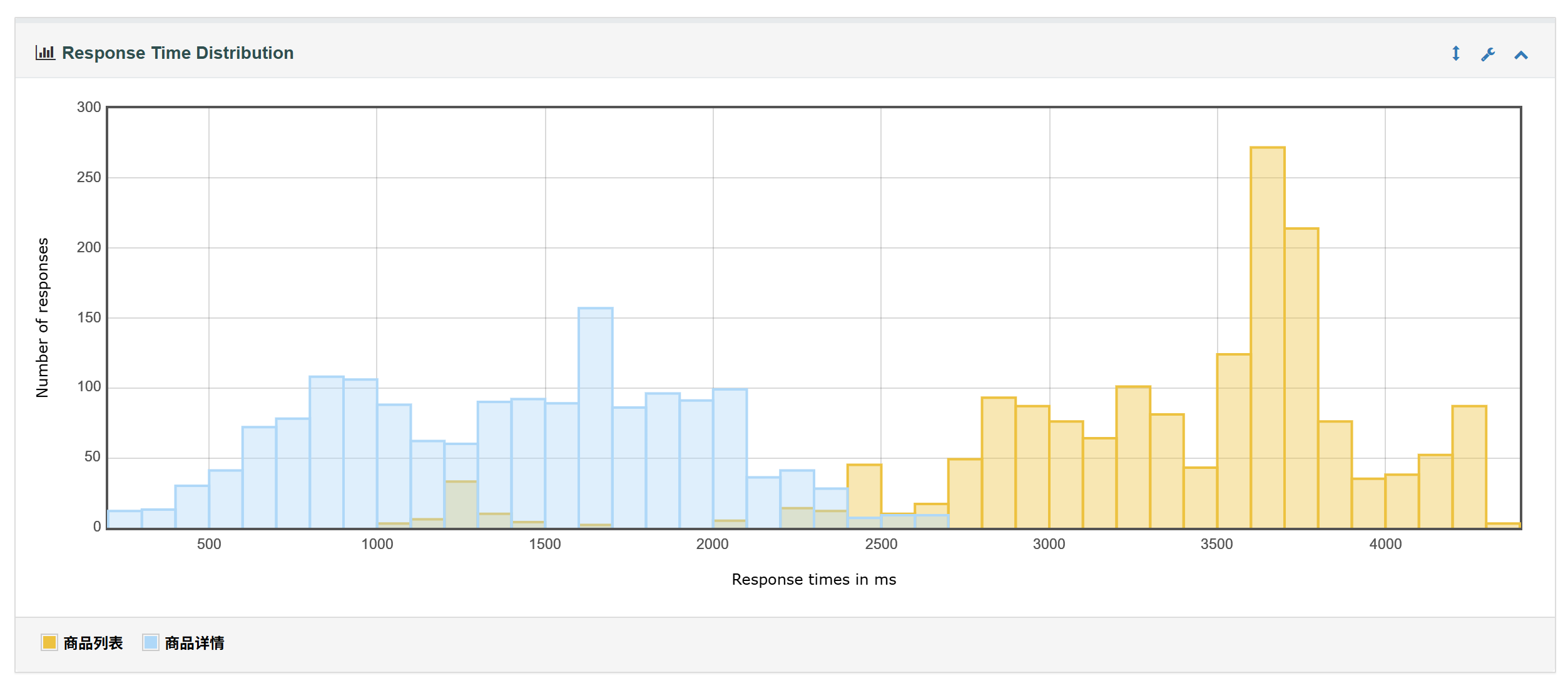
* 用户数为150的index.html











* 用户数为200的index.html

