第27部分 高并发秒杀的设计精要与实现(上) (双语)

第01章—开篇：写给你的小册

🌹 **开篇提示：本小册源码基于JAVA编写，请注意是否与你的技术栈相符。**

Opening note: The source code of this booklet is written in JAVA. Please check if it is compatible with your technical stack.

提起秒杀，你可能已经耳熟能详，甚至可以说，没吃过猪肉也见过猪跑。比如，年中的**618大促**和年底的**双11狂欢**，都是经典的秒杀场景，全国人民在同一时刻抢购有限的库存。此外，我们节假日期间在12306购票也是秒杀场景，疫情初期领口罩也是秒杀。

When it comes to "seckilling", you may already be familiar with it, or even say that even if you haven't eaten pork, you've seen a pig run. For example, the mid-year 618 shopping festival and the end-of-year Double 11 shopping carnival are classic seckilling scenarios, where people across the country rush to purchase limited inventory at the same time. Additionally, purchasing tickets on 12306 during holidays is also a seckilling **scenario**, as is the initial stage of the pandemic when masks were distributed.

事实上，秒杀最早的确起源于电商，但经过近十年的发展，秒杀的应用已经渗透到我们生活中的方方面面了。

In fact, the concept of "seckilling" originally originated from e-commerce, but after nearly a decade of development, its application has **penetrated** into every aspect of our lives.

这是因为，**秒杀本质上是高并发技术在特定场景下的综合应用**。高并发的场景你肯定不陌生，而高并发架构设计能力则是开发者不可或缺的关键能力。

This is because "seckilling" is essentially the comprehensive application of high concurrency technology in specific scenarios. High concurrency scenarios are certainly familiar to you, and the ability to design high concurrency architecture is a crucial capability that developers cannot do without.

因此，即使我们的业务中没有秒杀场景，我们也应该去学学秒杀架构，甚至**越早学习越好**。

Therefore, even if there are no flash sales scenarios in our business, we should still learn about flash sale architecture, and the sooner we start learning, the better.

学秒杀，我们究竟在学什么？

那么，学秒杀，我们究竟是在学什么？回答这个问题之前，我们必须先解决的问题是，**大家为什么要学秒杀?**

So, what are we learning from flash sale? Before answering this question, we must first solve the question: why do you want to learn flash sale?

这个朴素的问题背后，无非是两个朴素的答案：**一是实际生产需要，二是面试升职加薪需要。**

Behind this simple question, there are only two simple answers: one is the need for actual production, and the other is the need for interview, promotion and salary increase.

秒杀系统在设计过程中，需要用到高并发、缓存、数据库、架构和性能优化等多方面的技术知识，这很考验设计者的**思维的严谨度**和**技术的广度与深度**。

In the design process of flash sale system, technical knowledge of high concurrency, cache, database, architecture and performance optimization is needed, which tests the preciseness（严谨） of designers' thinking and the breadth and depth of technology.

比如说，我们需要关注**全链路**在高并发**极限压力**下的表现。要知道，**量变引起质变**，很多我们看起来正确的常规设计和方案，在极限压力和高并发下，**会变得极其脆弱以至于濒临崩溃**。

For example, we need to pay attention to the performance of the whole link under high concurrency limit pressure. You should know that quantitative change leads to qualitative change. Many conventional designs and schemes that we seem to be correct will become extremely vulnerable to collapse under extreme pressure and high pressure.

这要求我们不仅要横向考虑应用功能、限流、安全等问题，还要在纵向深入思考极致性能和数据的一致性等问题。

This requires us not only to consider cross-functional issues such as application functions, rate limiting, and security, but also to delve deeper into vertical issues such as ultimate performance and data consistency.

因此，**秒杀架构可以说是技术水平的试金石，而学习秒杀技术，我们本质上是要获得技术的广度和深度的双向扩展。**

Therefore, the flash sale architecture can be said to be a test of technical proficiency, and by learning flash sale techniques, we are essentially seeking to expand our technical breadth and depth in both directions.

至于秒杀在面试、升职中的作用，确实有很多面试官热衷于考察候选人秒杀类问题，当然也并不完全是出于场面需要，而是基于秒杀背后的技术确实在生产中应用广泛，也大有文章，就像我们上面说的一样。

As for the role of flash sales in interviews and promotions, there are indeed many interviewers who are keen on examining candidates' flash sales questions. Of course, this is not entirely based on the needs of the situation, but on the fact that the technology behind flash sales is widely used in production and has great potential, as we mentioned earlier.

而套路化面试是行业的基本现状，掌握秒杀架构无疑是化解套路的有效举措。因此，**学习秒杀，我们也是在学习一劳永逸地解决套路化面试的方法。**

The routine interview is the basic situation in the industry, and mastering the flash sale structure is undoubtedly an effective measure to resolve the routine. Therefore, learning flash sales is also learning a way to solve routine interviews once and for all.

小册有何与众不同

梳理清楚了以上这些之后，我就萌生了写秒杀和高并发相关小册的想法。六月份，我在掘金上写了一个叫做《王者并发课》的专栏，慢慢地整理自己的思路，收集大家的反馈建议。积累到现在，秒杀架构设计与实现的小册才慢慢成型。

After sorting out the above, I had the idea of writing a booklet related to flash sales and high concurrency. In June, I wrote a column called "King Concurrent Course" on Nuggets, slowly organizing my thoughts and collecting feedback and suggestions from everyone. Accumulating until now, the brochure on the design and implementation of the flash sale architecture is slowly taking shape.

在写作的过程中，我发现市面上关于秒杀的学习资料随处可见，但大部分鱼龙混杂质量参差不齐，且多侧重纯理论连篇累牍、脱离实践，无法给读者从理论到实践的立体呈现。

因此，**如何从理论结合实践、如何把理论讲述清楚、从何从实践中看破理论是我们这份小册的重点和难点**。

In the process of writing, I found that learning materials about flash sales can be seen everywhere in the market, but most of them are mixed with good and bad quality, and mostly focus on pure theory and are detached from practice, unable to provide readers with a three-dimensional presentation from theory to practice.

那么，在这本小册中，我准备用16个章节的内容，带着大家一起**实践**。

So, in this booklet, I plan to use the content of 16 chapters to lead everyone in practice.

在文章的前两篇，我们主要讲解秒杀架构的理论的部分，也就是秒杀架构的目标挑战和相应的设计原则和方法。而从第三篇文章开始，我们将转入实战，从需求分析到技术方案设计，再转入编码实现，力求给大家讲清楚，怎么从 0 实现**核心领域**健全的秒杀系统，以及从实践中真切地认知理论的意义。

In the first two articles, we mainly explained the theoretical part of the flash sale architecture, namely the target challenges and corresponding design principles and methods of the flash sale architecture. Starting from the third article, we will move on to practical applications, from requirement analysis to technical solution design, and then to coding implementation, striving to explain clearly how to implement a sound flash sale system in core areas from scratch, and to truly understand the significance of theory through practice.

在这个过程中，**我们也将为读者提供源码，对于有兴趣的读者我们也欢迎加入共同实现**。

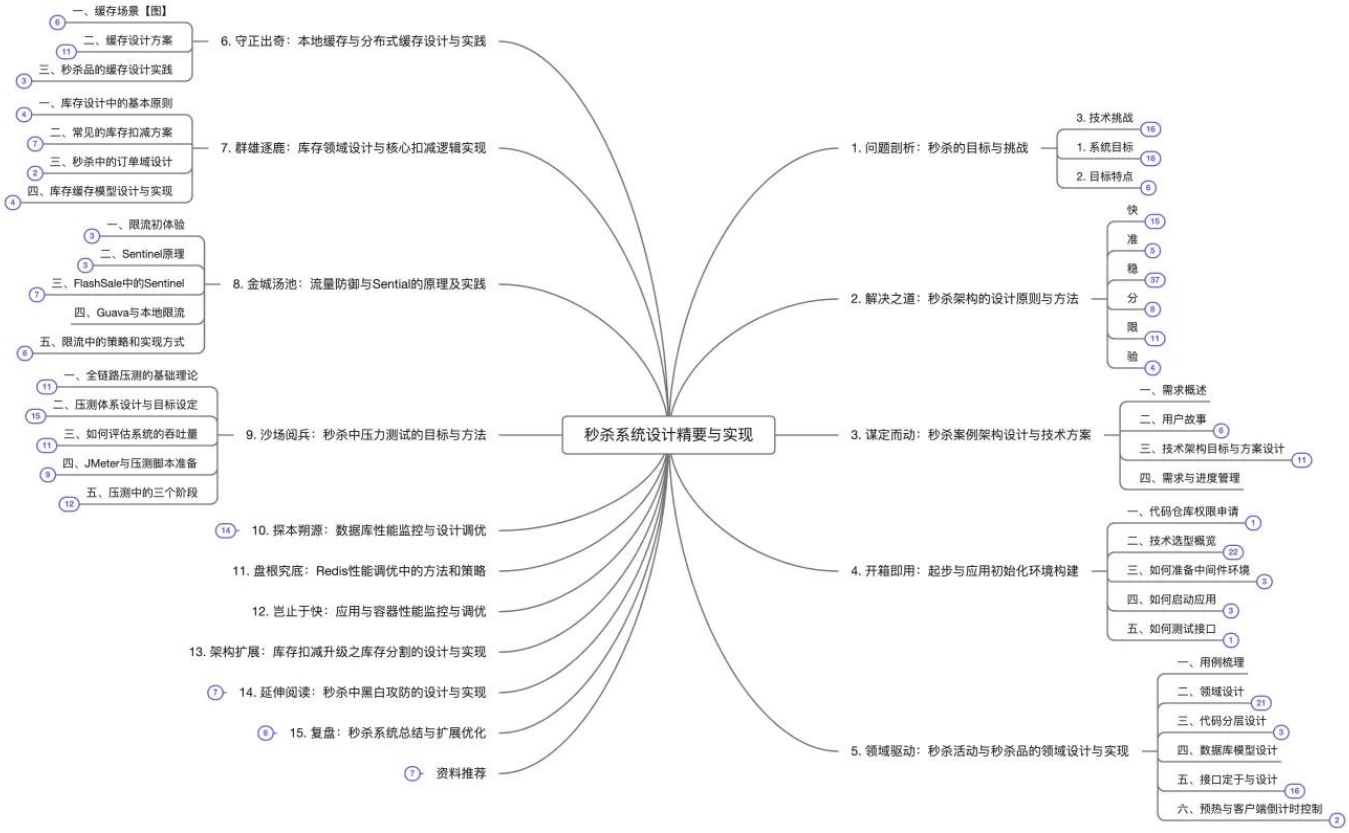
In this process, we will also provide the source code for readers, and we welcome interested readers to join us in implementing it together.

此外，**在内容设定和边界方面，出于内容篇幅和读者水平不一等考虑，这份小册并非面面俱到**。我们遵循领域驱动设计的理念，在有限的篇幅里，**只讲解最关键的核心领域**，对于非核心领域我们采用接口的方式预留出扩展点。

In addition, in terms of content setting and boundaries, due to differences in content length and reader level, this booklet is not comprehensive. We follow the concept of domain driven design, and in limited space, we only explain the most critical core domains. For non core domains, we reserve extension points through interfaces.

下面这幅思维导图体现的是小册的基本内容组织结构。当然，它并不是小册的全部内容。**在小册更新结束后，我们会为读者提供完整、详实且高清的思维导图**。

The following mind map illustrates the basic content organization structure of the booklet. Of course, it is not the entire content of the booklet. After the update of the booklet, we will provide readers with a complete, detailed, and high-definition mind map.



总的来说，**如果你希望鄙弃碎片化的泛泛而谈，转而系统性、结构化学习秒杀架构，那么这份小册可能适合你。**

In general, if you wish to abandon fragmented and general discussions and instead focus on systematic and structured learning of flash sale architectures, then this booklet may be suitable for you.

除此之外，你还能收获：

In addition, you can also gain:

- 认知完整的秒杀架构设计原则与方法论；
- 认知秒杀架构中的关键技术点如何落地；
- 从源码中掌握本地缓存与分布式缓存的设计技巧；
- 从源码中掌握秒杀系统的核心之库存扣减如何实现；
- 从源码中掌握限流系统的接入和配置；
- 从实践中掌握压力测试的方法和技巧；
- 从原理理解MYSQL读写瓶颈所在；
- 从架构上认知Redis的能与不能；
- Recognize the complete second-kill architecture design principles and methodologies;

- How to implement the key technical points in the cognitive second-kill architecture;
- Master the design skills of local cache and distributed cache from the source code;
- How to realize the inventory deduction of the core of the second-kill system from the source code;
- Master the access and configuration of the current-limiting system from the source code;
- Master the methods and skills of stress testing from practice;
- UNDERSTAND THE BOTTLENECK OF MYSQL READING AND WRITING FROM THE PRINCIPLE;
- Recognize the ability and inability of Redis from the architecture;
- 其他：
  - 体验领域驱动设计中的分层架构如何落地；
  - How to implement the hierarchical architecture in the experience field-driven design;
  - 体验如何通过看板管理项目的进度；
  - Experience how to manage the progress of the project through Kanban;
  - 体验如何设计RESTful API；
  - Experience how to design RESTful API;
  - 体验如何通过容器化管理应用和简单的集群管理。
  - Experience how to manage applications and simple cluster management through containerization.

关于答疑

如果你在阅读小册的过程中，有任何疑问，你可以选择以下四种方式提问：

If you have any questions in the process of reading the booklet, you can choose the following four ways to ask questions:

- 评论区直接留言；
- Leave a message directly in the comment area;
- 加入学习群或作者微信：
- Join the learning group or the author's WeChat:

- 发送问题到邮thoughts.beta+qa@gmail.com.
- Send the question to the mailbox

在提问时，请尽量附上问题的上下文，比如图片或代码以及你的个人看法，我会在每天的固定时间统一回复。

When asking questions, please try to attach the context of the question, such as pictures or codes and your personal opinions. I will reply at a fixed time every day.

最后我想说，**知是行之始，行是知之成。这是一份小册，更是一次知与行的检验。** 那么，出于我本人的水平限制，这份小册难免会有错误和BUG，欢迎批评指正，也欢迎向源码提交PR，我们会持续完善这份小册。

Finally, I want to say that knowledge is the beginning of action, and action is the success of knowledge. This is a booklet, and it is also a test of knowledge and action. Well, due to my own level limit, this booklet will inevitably have errors and bugs. You are welcome to criticize and correct, and you are also welcome to submit PR to the source code. We will continue to improve this booklet.

希望在秒杀系统这条妙趣横生的学习之路上，能与你同行~

I hope to walk with you on the interesting learning road of the second-kill system~

内容更新记录

- 👉 2021年12月8日
- December 8, 2021
    - 发布14章节：《集群部署：熔断、降级与容器化》

- Release Chapter 14: Cluster Deployment: Meltdown, Downgrade and Containerization
- 新增网关服务并提交源码；
- Add new gateway services and submit the source code;
- 新增集群部署脚本并提交源码；
- Add a cluster deployment script and submit the source code;
- 新增集群化测试API，并提交POSTMAN测试脚本。
- Add a clustered test API and submit the POSTMAN test script.

- **2021年11月27日**
- **November 27, 2021**
  - 发布第13章节：《魔高一丈：黑白攻防与安全风控策略》
  - Release Chapter 13: "The Devil is One Foot Higher: Black and White Attack and Defense and Security Risk Control Strategy"
  - 提交风控源码；
  - Submit the risk control source code;
  - 提交章节相关的图片脚本；
  - Submit chapter-related picture scripts;
  - 优化各模块参数校验；
  - Optimize the parameter verification of each module;
  - 优化配置文件和说明；
  - Optimize configuration files and instructions;
  - 增加统一鉴权并重构Controller API入口参数；
  - Add unified authentication and reconstruct Controller API entry parameters;
  - 文章内增加阅读引导。
  - Add reading guidance in the article.
- **2021年11月22日**
- **November 22, 2021**
  - 发布第12章节：《去中心化：分库分表与分桶策略和实现》
  - Release Chapter 12: "Decentralization: Strategy and Implementation of Sub-DB Sub-table and Sub-bucket"
  - 提交去中心化章节源码；

    - Submit the decentralized chapter source code;
    - 提交去中心化章节相关的容器化环境配置；
    - Submit the containerized environment configuration related to the decentralized chapter;
    - 提交章节相关的图片脚本；
    - Submit the containerized environment configuration related to the decentralized chapter;
    - 重构代码。
    - Reconstruct the code.
- **2021年11月15日**
- **November 15, 2021**
    - 发布第11章节：《日志治理：可观测性的概念、原则与落地》
    - Release Chapter 11: "Log Governance: Concepts, Principles and Implementation of Observability"
    - 提交日志优化章节源码；
    - Submit the log to optimize the chapter source code;
    - 优化部分代码。
    - Optimize part of the code.
- **2021年11月08日**
- **November 8th, 2021**
    - 发布第10章节：《异步下单：队列设计与任务闭环处理》
    - Release Chapter 10: "Asynchronous Ordering: Queue Design and Task Closed-loop Processing"
    - 提交**异步下单**章节源码；
    - Submit the source code of the asynchronous order chapter;
    - 新增异步下单相关的PlantUML脚本和压测脚本；
    - Newly added PlantUML scripts and pressure test scripts related to asynchronous ordering;
    - 优化章节内容组织。
    - Optimize the organization of chapter content.
- **2021年10月31日**

- **October 31, 2021**
  - 发布第9章节：《压力测试：目标、方法与实践》
  - Release Chapter 9: Stress Test: Goals, Methods and Practices
  - 源码中增加PlantUML图片脚本，方便读者自主生成高清大图；
  - PlantUML picture script is added to the source code, which is convenient for readers to independently generate high-definition large pictures;
  - 源码中增加压测脚本；
  - Add a pressure test script to the source code;
  - 优化部分文章内容。
  - Optimize the content of some articles.
- **2021年10月22日**：首发初版，包含源码及前八章节。
- October 22, 2021: The first edition, including the source code and the first eight chapters.

第02章一直面问题：秒杀系统目标与挑战

从本章开始至下一章节，是秒杀架构的理论讲述部分。说实话，这部分内容似乎着实**无聊且枯燥**。乍看起来，这部分内容读者懒得看、作者懒得写，忽略完事不好吗?

From the beginning of this chapter to the next chapter, it is the theoretical part of the second-kill structure. To be honest, this part of the content seems to be really boring. At first glance, readers are too lazy to read this part of the content, and the author is too lazy to write. Isn't it bad to ignore the end?

其实不然，须知理论指导实践。只不过，当我们尚未深入实践时，便觉得理论枯燥乏味，可一旦“**躬身入局**”后再回头，就会发现其中之妙。

In fact, it is not. It is important to know the theory to guide practice. However, when we have not deeply practiced it, we feel that the theory is boring, but once we "bow down into the game" and look back, we will find the beauty of it.

所以，如果这部分内容令你烦躁或头疼时，你完全可以直接跳过去看源码实践部分，哪天你开心了再回头来看完全没问题。

Therefore, if this part of the content makes you irritable or headache, you can skip it directly to see the source code practice part. It's no problem to look back one day when you are happy.

当然，如果你选择继续阅读，那我们就进入正题。

Of course, if you choose to continue reading, then we will get to the point.

从用户角度来理解秒杀其实是一件非常容易的事情，似乎不用我赘述，你可能早已了然于心，甚至对如何得手可能还颇有心得。

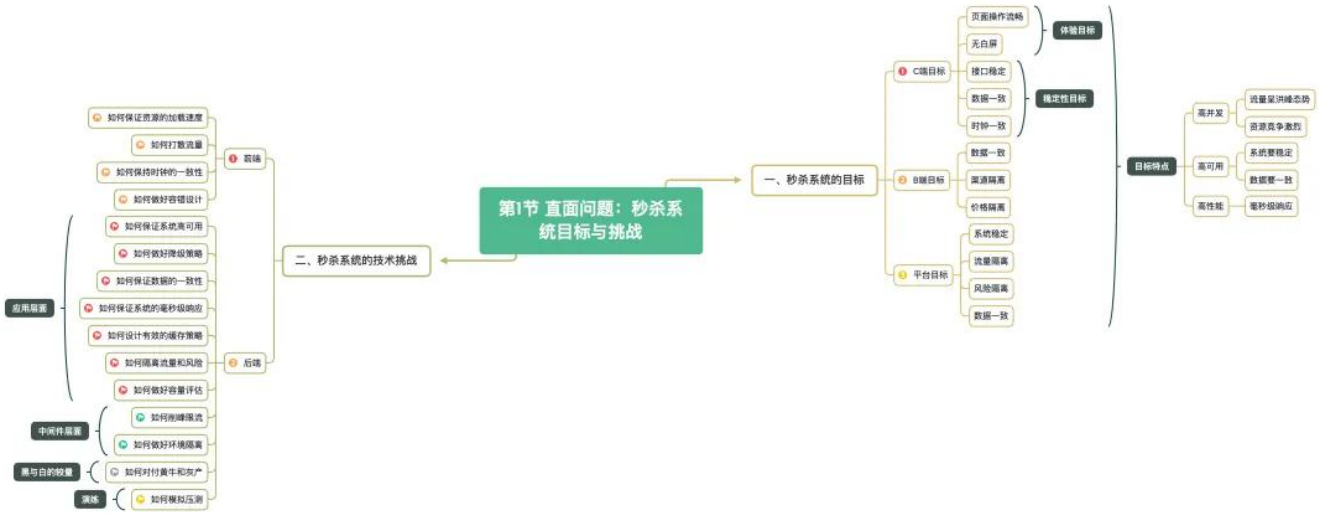
It is actually very easy to understand the second kill from the user's perspective. It seems that I don't need to elaborate. You may already know it, and even have a lot of experience on how to do it.

可是，**我们不仅是普通的用户，而且是技术人员，甚至是秒杀架构的缔造者**。我们的理解的角度怎么可以和普通用户一样?

However, we are not only ordinary users, but also technicians, and even the creators of the second-kill architecture. How can our understanding be the same as that of ordinary users?

所以，**如何从技术的角度，洞见秒杀系统的内涵，理解秒杀系统的精要，设计出可靠的秒杀系统**才是我们关心的重点所在，也是我们在此共同探讨的目的。

Therefore, how to understand the connotation of the second-kill system from a technical perspective, understand the essence of the second-kill system, and design a reliable second-kill system is the focus of our concern, and it is also the purpose of our joint discussion here.



在动机上，秒杀分为**主动秒杀**和**被动秒杀**。所谓主动秒杀，即我们主动设计秒杀活动，借助秒杀活动实现去库存、制造热点效应、拉新促活等目的，一般电商活动就是主动的秒杀。

In terms of motivation, second kill is divided into active second kill and passive second kill. The so-called active second-kill, that is, we take the initiative to design second-kill activities to achieve the purpose of de-inventory, creating hot spots, attracting new people and promoting activities with the help of second-kill activities. Generally, e-commerce activities are active second-kill.

而所谓被动秒杀，即场景非人为设计但由于供给关系紧张导致了不可避免的资源争夺，比如火车票抢购就属于被动秒杀。

The so-called passive second kill, that is, the scene is not humanely designed, but due to the tight supply relationship, it leads to inevitable resource competition. For example, the rush to buy train tickets belongs to passive second kill.

但是，无论是主动秒杀还是被动秒杀，只要出现了不可避免的秒杀场景，就必须要从**业务**和**技术**两个层面抓住秒杀的目标，因为**目标是设计与架构的靶向**。

However, whether it is active second kill or passive second kill, as long as there is an inevitable second kill scenario, the goal of second kill must be grasped from both business and technical levels, because the goal is the target of design and architecture.

一、秒杀系统的目标

通常，我们把秒杀看作是高并发的典型场景，是**大量请求**对**有限资源**的**瞬时争夺**，是用户对系统发起的压力测试。然而，这话其实并不完全准确，秒杀与高并发的关系可疏可密 。

Usually, we regard second kill as a typical scenario of high concurrency, which is the instantaneous competition of a large number of requests for limited resources, and a pressure test initiated by users on the system. However, this is actually not completely accurate, and the relationship between second kill and high concurrency can be sparse or dense.

因为，秒杀系统的核心目标是实现**大流量对有限资源的瞬时争夺**。围绕这个核心目标，我们可以进一步将其拆解为三个关键目标，分别是**C端目标**、**B端目标**和**平台目标**。

Because the core goal of the flash sale system is to realize the instantaneous(瞬间的) competition of large traffic for limited resources. Focusing on this core goal, we can further break it down into three key goals, namely, the C-end goal, the B-end goal and the platform goal.

（一）C端目标

C端用户是上帝。在秒杀体验过程中上帝的**体验是否流畅**、**数据是否无误**至关重要，所以我们要首先保障用户的体验。为什么这么说呢?

The C end user is God. In the process of flash sale experience, it is crucial to see whether God's experience is smooth and the data is correct, so we should first ensure the user's experience. Why do you say that?

对于大部分上帝而言，体验是最直观的感受，**突然的卡顿**、**白屏**、**宕机**等糟糕的体验会导致上帝逐渐失去对我们的眷顾，甚至对我们的技术能力产生怀疑。而能否秒杀成功倒是其次，大部分上帝在秒杀时都默认自己抢不到，他们不会因为抢不到而大为恼火。

For most of God, experience is the most intuitive(直觉的) feeling. Sudden bad experiences such as stuck, white screen, downtime will lead to God gradually losing his favor on us, and even doubt our technical ability. The success of flash sale is the second. Most of the gods acquiesce that they can't grab flash sale, and they won't be angry because they can't grab it.

（二）B端目标

如果说C端用户是上帝，那么B端用户就是我们的金主（如果B端就是自己的公司，那我们要更为谨慎，因为公司不仅是金主，更是雇主）。对于金主而言，在**价格**、**数量**和**资金**上都不能出错。

If the C-end user is God, the B-end user is our owner (if the B-end user is our own company, we should be more cautious, because the company is not only the owner, but also the employer). For the gold owner, there should be no mistakes in price, quantity and funds.

首先是价格，秒杀品的价格一般有别于其他渠道的商品，哪怕是同样的商品，价格也会存在差异，所以在价格上我们要做到渠道隔离。其次是数量，也就是不能**超卖**，超卖的后果很严重，会直接造成资损或服务无法兑现，客诉会接踵而至，甚至引发舆情。

The first is the price. The price of flash sale products is generally different from that of products from other channels. Even the same products have different prices, so we need to isolate channels in terms of price. The second is the quantity, that is, it cannot be oversold. The consequences of oversold are very serious, which will directly lead to asset loss or service failure, customer complaints will follow, and even cause public opinion.

最后是资金安全，除了超卖会导致资损外，还有一种情况也会造成资损，那就是**恶意下单**。如果你的竞争对手或者某些恶意用户在瞬间下单了你的所有商品，但就是不付款，这种情况会导致B端因

在秒杀活动中无法进账而引发资损，不仅活动前期的推广费用、活动费用无法回收，可能还会导致更为严重的连锁反应。

（三）平台目标

至于平台的首要目标则是保障C端目标和B端目标的顺利实现。 其次，平台的业务往往不止秒杀一种，所以在秒杀活动期间，要保证不能影响到平台其他业务的正常开展。比如，如果我们把秒杀的流量洪峰直接引到其他非重保的应用上，轻则受牵连的应用宕机，重则因雪崩效应而导致整个平台宕机。

The primary goal of the platform is to ensure the smooth realization of the C end goal and the B end goal. Secondly, the business of the platform is often more than flash sale, so during the flash sale activity, ensure that it does not affect the normal development of other business of the platform. For example, if we direct the traffic peak of flash sale to other non heavily insured applications, the affected applications will go down, or the entire platform will go down due to avalanche(雪崩) effect.

在设计秒架构时，以上三个目标都必须全部实现。如果其中某个目标未能实现，可能会导致一地鸡毛，提桶走人也并非耸人听闻。

When designing the second architecture, all three goals must be achieved. If one of these goals is not achieved, it may lead to chicken feathers everywhere. It is not sensational to leave with bucket.

从秒杀系统的目标来看，我们可以总结出秒杀过程中的几个明显的特征：

From the goal of flash sale system, we can summarize several obvious characteristics in the process of flash sale:

- **瞬时大流量**：从秒杀开始的前期，流量会逐渐聚集，在秒杀开始的瞬间，流量会瞬间涌入，监控图上也会体现出瞬间跳起，竖起一根芒刺；
- **Instantaneous large flow**: from the early stage of the start of flash sale, the flow will gradually gather. At the moment of the start of flash sale, the flow will flow in instantaneously, and the monitoring chart will also show an instant jump and a prick;
- **高并发**：虽然大流量并不一定意味着高并发，但在秒杀场景中，大流量将直接导致高并发。高并发系统的设计与普通系统的设计有着很多不同，即使提供同样的服务，两者在设计上也会有着天壤之别；
- High concurrency: Although large traffic does not necessarily mean high concurrency, in the flash sale scenario, large traffic will directly lead to high concurrency. There are

many differences between the design of high concurrency system and that of ordinary system. Even if the same service is provided, the design of the two systems will be very different;

- **高可用**：在任何情况下，我们不能让系统宕机，不能让C端用户失望，不能让兄弟应用跪下；
- High availability: in any case, we should not let the system down, let the C-end users down, or let brother applications kneel down;
- **高性能**：即使在洪峰流量下，也要保证C端用户的流畅体验，不能出现白屏、明显卡顿等情况；
- High performance: even under the peak flow, it is necessary to ensure the smooth experience of C-end users without white screen, obvious jamming, etc;
- **读多写少**：大流量是"读"，只有在并发中竞争成功的才会涉及到"写"。然而，成功者毕竟是极少数，所以“读多写少”也是秒杀系统的关键特征；
- Read more and write less: large traffic is "read". Only those who compete successfully in concurrency will be involved in "write". However, there are very few winners after all, so "reading more and writing less" is also the key feature of flash sale system;
- **数据强一致性**：数据一致性体验在两个方面，一是所有C端用户所看到的秒杀开始时间、库存等信息一致，二是最终成单数量与库存一致，不可以存在超卖的情况。
- Strong data consistency: the data consistency experience is in two aspects: one is that the flash sale start time, inventory and other information seen by all C-end users are consistent; the other is that the final order quantity is consistent with the inventory, and there can be no oversold.

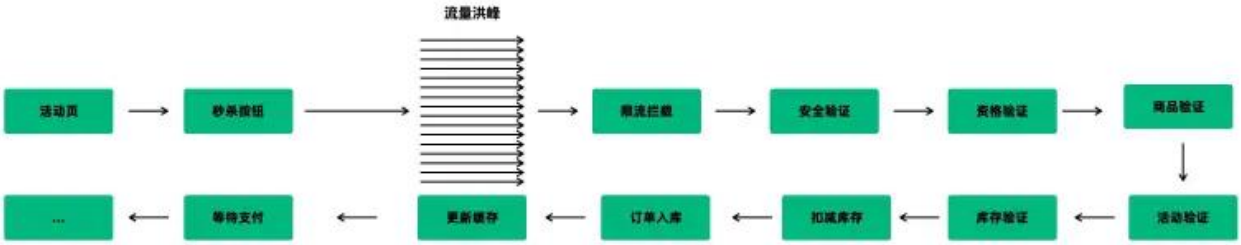
二、秒杀系统的技术挑战

正如我们所见，**秒杀系统具有多维度的目标要求，以及高流量、高并发和高性能等“三高”特征，而正是这些决定了技术的难点和挑战。**

As we can see, the flash sale system has multi-dimensional objectives and requirements, as well as the "three highs" characteristics of high traffic, high concurrency and high performance, which determine the technical difficulties and challenges.

与普通系统不同，高并发所导致的系统质变，将影响整个链路上的各个环节，其中任何一个环节的疏忽和故障都将会造成全局的失败。所谓质变，指的是原本体验良好、功能正常的系统，在瞬时高压下会变得极度脆弱，乃至于瞬时崩溃。

Different from ordinary systems, the qualitative change of the system c aused by high concurrency will affect all links of the whole link, and the negligence and failure of any link will cause global failure. The so-called qualitative change refers to that the system with good experience and normal function will become extremely fragile under transient high pressure, and even collapse instantaneously.



所以，如何解决上述的各种问题，满足多维度的目标要求，正是我们设计秒杀系统的挑战所在。根据秒杀系统所呈现的关键特征，我们可以梳理出如下的技术挑战。

Therefore, how to solve the above problems and meet the requirements of multi-dimensional goals is the challenge for us to design flash sale system. According to the key features of flash sale system, we can sort out the following technical challenges.

在客户端，我们面临两类挑战：**一是如何保证资源的加载速度，二是如何打散流量。**

On the client side, we face two kinds of challenges: one is how to ensure the loading speed of resources, and the other is how to disperse traffic.

首先，客户端的资源包括HTML页面、JS和相关的图片。这些资源与服务端计算是独立的，所以无论服务端面临的流量如何，这部分资源都要保证流畅加载。对于大部分中大型公司而言，静态资源都会走CDN，这也是推荐的做法，将资源放在离用户近的地方。而对于小型和自建机房的公司，可能会将静态资源存放在自己的机房，在流量洪峰下这会存在风险，瞬时的带宽占用可能会导致客户端资源加载困难。

First, the resources of the client include HTML pages, JS and related images. These resources are independent of the server-side computing, so no matter what traffic the server is facing, these resources should be loaded smoothly. For most large and medium-sized companies, static resources will go through CDN, which is also the recommended practice. Put resources close to users. For small and self built computer rooms, companies may store static resources in their own computer rooms, which will be at risk under the peak flow. Instantaneous bandwidth occupation may lead to difficulties in loading client resources.

其次，流量洪峰的特点是流量比较集中，集中的流量会导致拥堵和高并发。因此，我们需要考虑在客户端将流量打散，避免所有用户在同一时刻发出请求。比如，常见的做法有滑块、验证码和问答等，每个人的操作速度和问答速度不可能完全一致，所以我们可以利用这点来打散流量。不要小瞧几秒钟的差异，它可能会将流量峰值降低好几倍。

Secondly, the characteristic of the flow peak is that the flow is relatively concentrated, which will lead to congestion and high concurrency. Therefore, we need to consider breaking up traffic at the client to avoid all users sending requests at the same time. For example, common practices include slider, verification code and question and answer. Everyone's operation speed and question and answer speed cannot be exactly the same, so we can use this to disperse traffic. Don't underestimate the difference of a few seconds. It may reduce the peak traffic several times.

而在服务端我们面临的挑战最多，几乎是客户端的3倍。

On the server side, we face the most challenges, almost three times as many as the client side.

首先，如何保证系统的高可用。 服务端的首要目标是保证在活动期间系统的高可用，秒杀系统是对整个链路的压力测试，我们需要确保每个环节都不会出纰漏。

First, how to ensure the high availability of the system. The primary goal of the server is to ensure the high availability of the system during the activity. The flash sale system is a stress test of the entire link. We need to ensure that there are no mistakes in each link.

其次，如何削峰限流。 当客户端的流量洪峰过来后，需要对流量进行逐层地拦截和过滤。所以，我们需要在每一层设置相应的策略，识别出对应的垃圾流量和需要过滤的用户，确保最后达到服务的只有少量的合法流量。

Secondly, how to cut peak and limit current. When the traffic peak of the client comes, the traffic needs to be intercepted and filtered layer by layer. Therefore, we need to set corresponding policies at each layer to identify the corresponding garbage traffic and users who need to be filtered, so as to ensure that only a small amount of legal traffic finally reaches the service.

再次，如何保证系统的响应速度。 随着流量的激增，系统的负载可能会线性增加。但是，我们不能因此而降低系统的响应速度。一般而言，当接口的响应速度在10ms以内时，用户在操作时不会有明显的卡顿体感。

Thirdly, how to ensure the response speed of the system. As traffic surges, the load on the system may increase linearly. However, we cannot reduce the response speed of the

system because of this. Generally speaking, when the response speed of the interface is within 10ms, users will not experience significant lag during operation.

然后，如何保证秒杀品不被超卖。 正如前文所述，超卖的后果很严重，在这方面我们需要慎重设计下单减库存的策略。一旦用户冲破了层层关卡，进入最终的下单环节，性能将不再是主要的考量，数据一致性才是。所以，在减库存操作方面，需要多方面的校验，确保万无一失。

Then, how to ensure that flash sale products are not oversold. As mentioned earlier, the consequences of overselling are serious, and we need to carefully design the strategy of reducing inventory when placing orders in this regard. Once users break through the layers of hurdles and enter the final order placement process, performance will no longer be the main consideration, but data consistency. Therefore, in terms of inventory reduction operations, it is necessary to conduct multi-faceted verification to ensure foolproofness.

接着，如何隔离流量和风险。 非秒杀场景的流量如溪水潺潺，而秒杀的流量则是大江大河波涛汹涌。在错误设计或未考虑流量隔离的情况下，秒杀所调用的其他应用的接口将遭遇水漫金山的灭顶之灾。当然，也可能存在不幸中的万幸，比如所调用的系统有自己的限流，这个时候对方不会死，至少整个应用不会宕机，但是对方所触发的限流会让我们就地跪下。

Next, how to isolate traffic and risks. The traffic in non-flash sale scenarios is like a gentle stream, while the traffic in flash sales is like a raging river. In the case of incorrect design or without considering traffic isolation, the interfaces of other applications called by flash sales will encounter a catastrophic disaster like a flood. Of course, there may be a stroke of luck in misfortune. For example, if the called system has its own current limiting, then the other party will not die. At least the entire application will not crash. However, the current limiting triggered by the other party will make us kneel on the spot.

还有，如何设计有效的缓存策略。 秒杀系统的一个关键特征是“读多写少”，因此缓存的使用必不可少。可问题是，虽然缓存可以提高响应速度，但在数据的一致性方面如何保证又会是新的挑战，我们需要保证过程中中心化缓存、本地缓存和数据库之间的数据一致。

Moreover, how to design effective caching strategies. One of the key features of the flash sale system is "more reads and fewer writes", so the use of caching is essential. However, the problem is that although caching can improve the response speed, ensuring data consistency will be a new challenge. We need to ensure the data consistency among the centralized cache, local cache and database during the process.

最后，如何对付黄牛和秒杀器。 没有参与过秒杀活动设计的同学，对黄牛和秒杀器可能没有深刻的体会。而事实上，由于秒杀品一般存在着较大的利益空间，很多用户为了增加成功的概率，会使用多设备、自动化脚本等手段参与秒杀，这就催生了职业化的代秒选手的出现。这些选手有着**成熟**

的、丰富的秒杀设备，他们的存在会严重扰乱秒杀活动的正常进行，设计不完善的秒杀系统可能会被黄牛瞬间抢购一空，其后果可想而知。所以，如何对黄牛和**恶意**攻击也是秒杀系统中的关键考量；

Finally, how to deal with scalpers and bots for flash sales. Those who have never been involved in the design of flash sale activities may not have a deep understanding of scalpers and bots. In fact, since there is usually a large profit margin for the products in flash sales, many users will use multiple devices, automated scripts and other means to participate in the flash sales in order to increase their chances of success. This has given rise to the emergence(发生) of professional proxy buyers for flash sales. These people have **sophisticated** and abundant equipment for flash sales, and their existence will seriously disrupt the normal progress of flash sale activities. An imperfectly designed flash sale system may be snapped（抢购一空） up by scalpers in an instant, and the consequences can be imagined. Therefore, how to deal with scalpers and **malicious** attacks is also a key consideration in the flash sale system.

在中间件部分，我们也会面临两类挑战。

In the middleware part, we will also face two types of challenges.

一是如何应对突增的网络带宽。 大型公司的同学可能对网络带宽没有切身的体会，但是对于经历过自建机房的同学来说，可能会印象深刻。由于所有的服务共享带宽，当其中某个服务需要大流量时，往往要进行限流，否则突增的网络带宽可能拖垮整个公司的服务；

First, how to cope with the sudden increase in network bandwidth. Students from large companies may not have a personal experience of network bandwidth, but for those who have experienced building their own computer rooms, they may have a deep impression. Since all services share the bandwidth, when a certain service requires a large amount of traffic, traffic limiting often has to be carried out. Otherwise, the sudden increase in network bandwidth may bring down the services of the entire company.

二是如何监控各环节的状态和数据。 如果我们不能度量系统，我们将无法设计系统。因此，虽然监控是最后的挑战，但这并不影响它至关重要的地位。对于秒杀系统链路上的所有环节，都应该具有实时的、量化的指标可查看。同时，可以预先评估的流量和环节设计的**阈值**，设置相应的报警水位。

Second, how to monitor the status and data of each link. If we cannot measure the system, we will not be able to design it. Therefore, although monitoring is the final challenge, it does not affect its crucial position. For all the links in the flash sale system chain, there should be real-time and quantifiable indicators available for viewing. Meanwhile, the

thresholds of the traffic that can be evaluated in advance and the link design can be set, and corresponding alarm levels can be established.

小结

总体上，在不同公司的秒杀场景中，虽然秒杀的目标大同小异，但是秒杀系统在设计时的技术挑战，往往存在较大差距。

Generally speaking, in the flash sale scenarios of different companies, although the goals of flash sales are largely the same, there are often significant differences in the technical challenges during the design of flash sale systems.

比如，当公司具备完善的基础设施时，可以提供开箱即用的限流能力或防黄牛能力，而小公司却并不具备。

For example, when a company has a complete infrastructure, it can provide out-of-the-box capabilities for traffic limiting or anti-scalper measures, while small companies do not have such capabilities.

因此，在思考秒杀架构中的技术挑战时，我们要结合自己的业务背景和公司的基础设施能力做出**全面的**评估。

Therefore, when considering the technical challenges in the flash sale architecture, we should make a **comprehensive** assessment by combining our own business background and the infrastructure capabilities of the company.

回答本文开头的问题，秒杀架构和高并发一定存在亲密关系吗？其实未必。

To answer the question raised at the beginning of this article, is there necessarily a close relationship between the flash sale architecture and high concurrency? In fact, not necessarily.

通过对秒杀架构整体链路的梳理，我们可以发现如果我们的业务只是少量商品的秒杀，那么我们需要保证的有两点，一是要实现业务上的成功，即通过秒杀促活或者拉新；二是保证不要超卖。

By sorting out the overall link of the flash sale architecture, we can find that if our business only involves the flash sale of a small number of products, then there are two points that we need to ensure. First, we should achieve business success, that is, to promote user activity or acquire new customers through flash sales. Second, we should ensure that overselling does not occur.

对于此类业务，虽然有高并发的场景，但技术上可以把限流作为主要手段，把绝对流量挡在门外。

For this kind of business, although there are scenarios of high concurrency, technically, traffic limiting can be used as the main means to keep the absolute traffic out.

但是，如果像双十一那样的大量商品秒杀，但限流只能是辅助手段，而不能是**主要**手段，在这种场景下，我们不仅要限流，还要同时提高整个链路，尤其是数据库、缓存等底层的并发能力。

However, in the case of flash sales involving a large number of products like the Double Eleven Shopping Festival, traffic limiting can only be an **auxiliary** means rather than the primary one. In such a scenario, we not only need to implement traffic limiting but also **simultaneously（同时）** improve the concurrent capabilities of the entire link, especially those of the underlying parts such as databases and caches.

以上就是本文关于秒杀系统的目标和技术挑战的全部内容。在下一篇文章中，我们将结合秒杀架构中的技术挑战，聊一聊它的设计原则和方法。

The above is all the content of this article about the goals and technical challenges of the flash sale system. In the next article, we will discuss its design principles and methods in combination with the technical challenges in the flash sale architecture.

本章节到此结束，欢迎在评论区留下你的收获和疑问。对于本文未提及或未讲清楚的地方，也欢迎评论区批评指正，我会及时调整。如果你对小册有内容建议，请移步填写表单。
This chapter comes to an end. You are welcome to leave your gains and questions in the comment section. For the parts that are not mentioned or not explained clearly in this article, you are also welcome to criticize and correct them in the comment section. I will make adjustments in a timely manner. If you have content suggestions for this booklet, please go to fill in the form.

第03章—解决之道：秒杀架构的设计原则和方法

在前面的文章中，我们介绍了秒杀架构的目标和特征，以及由此所带来的一系列的技术挑战。在本文中，我们将探讨攻克这些技术挑战的设计原则与方法。

In the previous articles, we introduced the goals and characteristics of the flash sale architecture, as well as a series of technical challenges brought about by it. In this article,

we will explore the design principles and methods for overcoming these technical challenges.

秒杀架构的技术挑战众多，在设计时不能东一榔头西一棒槌，或者头痛医头脚痛医脚，而是要对特征和挑战进行全面的总结归纳，找出其中的共性，这样才能多维度有的放矢。

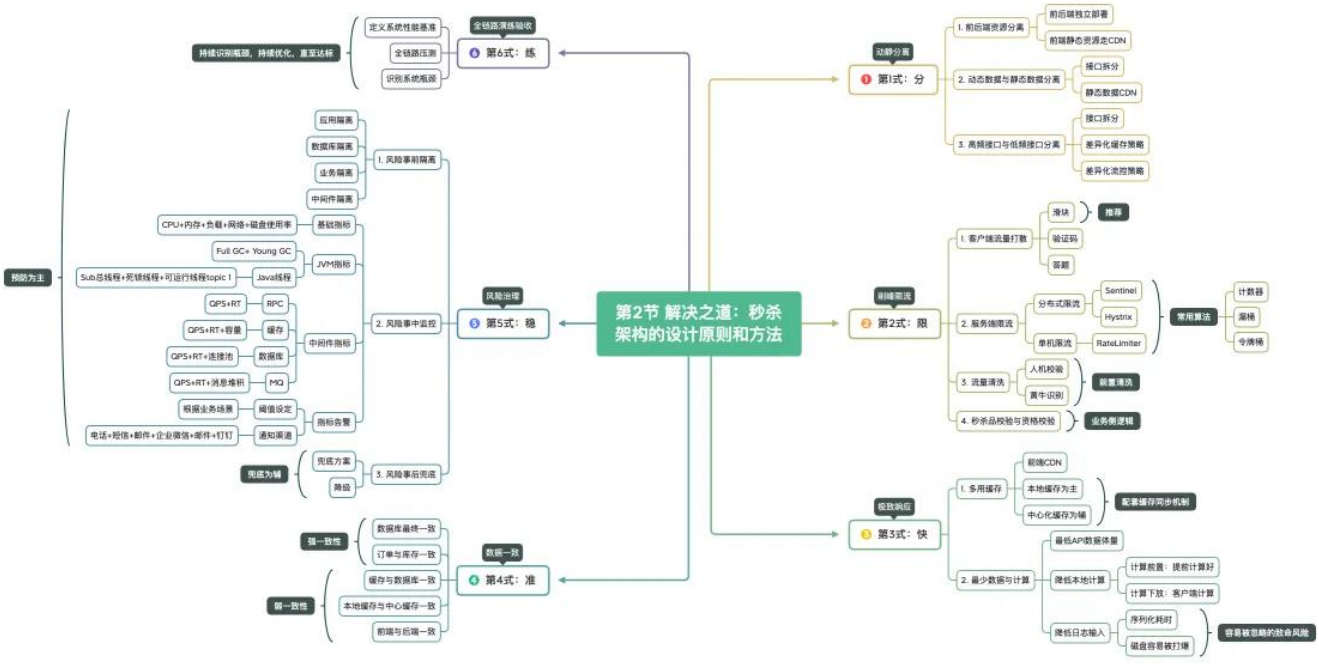
There are numerous technical challenges in the flash sale architecture. When designing it, we shouldn't work in a haphazard(随意的) way or just treat the symptoms rather than the root causes. Instead, we should comprehensively summarize and analyze the characteristics and challenges to find out the commonalities among them. Only in this way can we take targeted measures from multiple aspects.

概括地说，在设计秒杀架构时要牢牢记住**分-限-快-准-稳-练**六字箴言。为了方便记忆，我们姑且给它们取个可以上天的名字：**六脉神剑**。

Generally speaking, when designing the flash sale architecture, we should firmly keep in mind the six-word motto of "Divide – Limit – Speed – Accuracy – Stability – Practice". For the convenience of memory, let's give them a rather impressive name for the time being: Six Meridians Divine Sword.

不要误会，我们没有任何想**碰瓷**某大厂的意思。其剑式即是秒杀架构设计的原则，理解并掌握它们可以让我们避免犯一些原则性错误。接下来，我们将围绕着六个字逐一讲解。

Don't misunderstand. We don't have any intention of trying to **piggyback** on a certain big company. Its sword forms are the principles for the design of the flash sale architecture. Understanding and mastering them can help us avoid making some fundamental mistakes. Next, we will explain them one by one around these six words.



第1式：分–动静分离

所谓“**分**”，主要包含三个方面：**一是前后端资源分离，二是动态数据与静态数据的分离，三是高低频数据分离**。

The so-called "Divide" mainly includes three aspects: first, the separation of front-end and back-end resources; second, the separation of dynamic data and static data; and third, the separation of high-frequency and low-frequency data.

其中，第二点和第三点异曲同工，核心都是对不同类型的数据区分识别，并通过设计独立的接口和其他数据进行隔离，以采取不同的治理措施。

Among them, the second and third points share the same idea. The core of both is to distinguish and identify different types of data, and isolate them from other data by designing independent interfaces so as to adopt different governance measures.

1. 前后端资源分离

除了一些较为传统且老旧的系统之外，前后端分离已经是目前较为普遍的做法。

Apart from some relatively traditional and outdated systems, the separation of the front end and the back end is already a fairly common practice nowadays.

前端资源走CDN，一般在前端构建的时候就可以将构建完的静态资源推送到CDN服务器，这一过程完全可以**嵌入**在持续交付流程中。

Front-end resources are routed through CDNs (Content Delivery Networks). Generally, when the front end is being built, the completed static resources can be pushed to the CDN servers, and this process can be fully **embedded** in the continuous delivery workflow.

另外，CDN也不必自建，各大云计算厂商都提供了开箱即用的CDN服务。

Additionally, there is no need to build your own CDN. Major cloud computing vendors all offer out-of-the-box CDN services.

2. 动态数据与静态数据分离

所谓动态数据指的是秒杀期间**容易**变更的数据，比如秒杀品的品类、库存和上下架状态等。

The so-called dynamic data refers to the data that is **prone** to change during the flash sale period, such as the categories of the flash sale products, inventory, and the on-shelf or off-shelf status, etc.

动态数据对时间极为敏感，变化只在一瞬间。而所谓静态数据的变化相对较慢，比如秒杀结果等个人相关的数据。

Dynamic data is extremely sensitive to time, and changes can occur in an instant. On the other hand, the so-called static data changes relatively slowly, such as personal-related data like the results of the flash sale.

之所以要将动态数据和静态数据进行分离，是因为不同数据的更新频率会涉及到不同的架构设计和性能优化，分离后可以做到有的放矢。

The reason for separating dynamic data from static data is that the update frequencies of different data involve different architectural designs and performance optimizations. After separation, targeted measures can be taken.

在某些场景下，服务端返回的静态数据也可以考虑推送到CDN；

In some scenarios, the static data returned by the server can also be considered to be pushed to the CDN.

3. 高频接口与低频接口分离

高频与低频指的是调用频次。在所有的数据整合到一个接口中，并不总是好的主意。

High frequency and low frequency refer to the frequency of invocation. It's not always a good idea to integrate all the data into one interface.

虽然这种做法可以降低API的请求次数，但是会提高单个接口的计算耗时，并增加它的脆弱性。

Although this approach can reduce the number of API requests, it will increase the computational time consumption of a single interface and make it more vulnerable.

在秒杀架构中，我们要注意对高频接口的识别，高频接口的数据量相对较少；

In the flash sale architecture, we should pay attention to the identification of high-frequency interfaces. The amount of data in high-frequency interfaces is relatively small.

第2式：限-削峰限流

1. 客户端流量打散

客户端包含对外透出的H5、APP和小程序。作为流量的直接来源入口，前端可以通过**答题**、**滑块**和验证码等方式将流量打散。

The client side includes externally exposed H5, APPs and mini-programs. As the direct entrance for traffic sources, the front end can disperse the traffic by means of answering questions, using sliders and verifying codes, etc.

在削峰限流中，这是非常有用的方式。我们不要小看一秒或者几百毫秒的时间，在秒杀过程中，毫秒的差距也足以造成量级的影响。

In peak shaving and flow limiting, this is a very useful method. We shouldn't underestimate the time of one second or several hundred milliseconds. During the process of flash sales, a difference of even just a few milliseconds is enough to have an impact on a large scale.

比如 ，在客户端不采取任何措施的情况下，在同一时刻到达服务端的流量是100万QPS，而如果在客户端采取流量打散的措施，这个值则很可能由100万降低到80万、50万以及30万或者更低。

For example, if no measures are taken on the client side, the traffic reaching the server side at the same moment could be 1 million QPS. However, if measures for traffic dispersion are adopted on the client side, this value is very likely to be reduced from 1 million to 800,000, 500,000, 300,000 or even lower.

在设计时，答题相对来说成本要高一些，我们需要设计动态的题库和答案。相比之下，滑块和验证码则较为简单。

During the design process, answering questions is relatively more costly as we need to design dynamic question banks and answers. In contrast, sliders and verification codes are relatively simpler.

当然，无论是何种方式，都需要在接口设计上体现，而不是仅限于页面使用，要防止**绕过**页面直接请求接口。

Of course, no matter which method is used, it needs to be reflected in the interface design rather than being limited to page usage. It is necessary to prevent requests from directly accessing the interface by **bypassing** the page.

2. 服务端限流

无论客户端是否采取了流量打散的措施，服务端都需要设置限流保护，有条件的还要进行流量清洗。

Whether or not the client side has adopted measures for traffic dispersion, the server side needs to set up flow-limiting protection, and if conditions permit, traffic cleaning should also be carried out.

根据限流的不同时机和链路位置，我们可以将服务端限流通常分为**分布式限流**和**单机限流**两种。

According to different timings of flow limiting and the positions in the link, we can usually divide server-side flow limiting into two types: distributed flow limiting and single-machine flow limiting.

所谓分布式限流指的是通过独立的中间件在流量入口对系统整体流量进行拦截和管控，只有符合配置的流量才能通过，比如通过Sentinel在Spring Cloud Gateway中设置限流。

The so-called distributed flow limiting refers to intercepting and managing the overall traffic of the system at the traffic entrance through independent middleware. Only the traffic that meets the configuration can pass through. For example, flow limiting can be set in Spring Cloud Gateway through Sentinel.

Sentinel由阿里开源，提供了流控规则、降级规则和热点规则等功能，配置也较为灵活。

Sentinel is open-sourced by Alibaba and provides functions such as flow control rules, degradation rules and hotspot rules, and its configuration is also relatively flexible.

通过独立的中间件限流是比较通用的做法，当然也需要一定的维护成本，小公司可能不具备这样的条件。

Limiting the flow through independent middleware is a relatively common practice. Of course, it also requires certain maintenance costs, and small companies may not have such conditions.

相比之下，单机限流则简单很多，就是在应用内自主限流。比如，Guava中提供的RateLimiter就是一款不错的限流器。如果你的流量不是特别大，或者在应用内需要独立的限流处理，那么可以考虑这种方式。

In contrast, single-machine flow limiting is much simpler, which means implementing flow limiting independently within the application. For example, the RateLimiter provided by Guava is a nice flow limiter. If your traffic is not particularly large or you need independent flow-limiting processing within the application, then this approach can be considered.

当然，分布式限流和单机层限流两种在使用时并不冲突，事实上两者结合相得益彰。

Certainly, there is no conflict when using distributed flow limiting and single-machine level flow limiting. In fact, the combination of the two can bring out the best in each other.

在算法上，限流组件一般采用的是计数器、漏桶和令牌桶等算法，如果我们有特殊的限流需求，也可以考虑根据算法自主实现，比如我们可以业务规则将概率向特定用户倾斜等。

In terms of algorithms, flow-limiting components generally adopt algorithms such as counters, leaky buckets and token buckets. If we have special flow-limiting requirements, we can also consider implementing them independently according to the algorithms. For example, we can tilt(倾斜) the probability towards specific users based on business rules.

3. 安全校验

如果说限流是随机拒绝的话，那么安全校验则是有针对性地识别和拒绝特定的流量，也就是流量清洗。通俗地讲，就是在秒杀活动中，我们要识别出哪些是人，哪些是“鬼”。

If flow limiting is random rejection, then security verification is targeted identification and rejection of specific traffic, that is, traffic cleaning. In plain terms, in flash sale activities, we need to identify who are real people and who are "ghosts".

恶意竞对或者不怀好意的用户可能会通过CC攻击破坏活动，造成服务资源枯竭；而一些普通用户为了提高成功率则可能会采用自动脚本强刷，不仅给服务端带来较大压力，对其他用户也不公平，比如成群结队的“黄牛党”便是。所以，人机识别和CC攻击是安全校验的主要功能。

Malicious(恶意) competitors or malicious users may disrupt activities through CC attacks, causing service resources to be depleted. Some ordinary users may use automatic scripts to forcefully refresh in order to increase their success rate, which not only brings greater

pressure to the server but is also unfair to other users. For example, groups of "scalpers" are like this. **Therefore**, human–machine identification and CC attacks are the main functions of security verification.

4. 秒杀品校验与资格校验

通过限流和安全校验两道关卡之后，并不意味着用户此刻就可以高枕无忧，它至少还需要闯过秒杀品校验与资格校验。**比如**，当前的秒杀活动是否已经结束、秒杀品是否依然有效等，因为用户在客户端时秒杀品依然有效，但是当请求到达服务端之后，活动可能已经结束，所以这一步的校验必不可少。

After passing through the two checkpoints of flow limiting and security verification, it doesn't mean that users can sit back and relax at this moment. They still need to get through the verification of flash sale items and eligibility(资格) verification at least. **For example**, whether the current flash sale activity has ended, whether the flash sale items are still valid, etc. Because the flash sale items were still valid when users were on the client side, but the activity might have already ended when the requests reach the server side. So the verification at this step is essential.

而所谓资格校验，则是根据秒杀活动具体的业务背景所做的校验。**比如**，当前用户此前已经存在秒杀记录，那么此时是否还允许进行秒杀。**又或者**，部分用户可能比较“狡猾”，虽然此前已经秒杀过，但现在换了马甲再次秒杀，那么此刻可能就需要加强身份证等唯一性校验。**对于**一些重要的秒杀品，如果忽略对用户唯一身份的甄别，不仅对用户不公平，对业务运营或合作方也会造成麻烦。

And the so-called eligibility verification refers to the verification carried out according to the specific business background of the flash sale activity. **For example**, if the current user already has a flash sale record before, whether they are still allowed to participate in the flash sale at this time. **Or,** some users may be rather "sly". Although they have already participated in the flash sale before, they change their identities and try to participate in the flash sale again. Then it may be necessary to strengthen the verification of the uniqueness of identity cards and so on at this moment. **For some important** flash sale items, if the screening of users' unique identities is ignored, it will not only be unfair to users but also cause troubles for business operations or partners.

第3式：快–极致响应

1. 多用缓存

作为性能提升的利器，在秒杀活动中**缓存**绝对是不可或缺的存在。**无论**是前端资源等静态资源，还是服务端的动态数据，缓存不仅可以大幅提升性能，也是我们对系统防护的屏障。

As a powerful tool for performance improvement, cache is absolutely indispensable in flash sale activities. **Whether** it's static resources like front-end resources or dynamic data on the server side, cache can not only greatly improve performance but also act as a protective barrier for our system.

试想，在没有缓存或者发生缓存雪崩或穿透的情况下，大量请求打入数据库将会是怎样的画面？**毫**无疑问，数据库将迅速面临连接池被耗尽的风险，**而如果**恰好某个SQL查询的性能不佳，在这种屋漏偏逢连夜雨的情况下，数据库大概率会被拖垮，进而会导致整个应用崩坏以及关联应用都会被殃及。**由此可见**，对于分布式系统来说，尤其是秒杀类的系统来说，缓存是何等的重要。

Just imagine, what would it be like if a large number of requests hit the database when there was no cache or when cache avalanche or cache penetration occurred? **There** is no doubt that the database would quickly face the risk of the connection pool being exhausted. **And if,** unfortunately, the performance of a certain SQL query was poor, under such circumstances where misfortunes never come singly, the database would most likely be brought down, which would then lead to the breakdown of the entire application and even affect related applications. **From this**, it can be seen how important cache is for distributed systems, especially for systems like those for flash sales.

2. 中心化缓存与本地缓存

在分布式系统中，缓存的使用一般以**中心化缓存**和**本地缓存**两种形式存在。所谓中心化缓存，指的是将缓存集中存储于独立的系统，多个应用间共享一份缓存数据，比如Redis即使如此。

In distributed systems, the use of cache generally exists in two forms: centralized cache and local cache. The so-called centralized cache means that the cache is centrally stored in an independent system, and multiple applications share a set of cache data. For example, Redis is like this.

中心化缓存的优势在于数据物理结构简单，维护成本较低，我们只要保证数据库和中心缓存之间的数据一致性即可。

The advantage of the centralized cache lies in its simple physical data structure and relatively low maintenance cost. All we need to do is to ensure the data consistency between the database and the central cache.

当然，中心化缓存的优点也正是它的缺点。**在高并发的情况下，缓存系统将承受较大压力，轻则影响到缓存RT，重则直接拖垮缓存系统。**

Certainly, the advantages of the centralized cache are also its disadvantages. Under high-concurrency circumstances, the cache system will bear relatively large pressure. In mild cases, it will affect the cache response time (RT), and in severe cases, it will directly bring down the cache system.

因此，当秒杀场景的并发量较大时，我们需要通过本地缓存设计为一级缓存，也就是在应用本地设置缓存。

Therefore, when the concurrency volume in the flash sale scenario is relatively large, we need to design a local cache as the first-level cache, that is, set up a cache locally within the application.

3. 最少数据与计算

当我们致力于设计极致性能的API时，很显然缓存并不可少，但仅有缓存依然是不够的。所以，我们还要讲究另外一个字：**少**。

When we are committed to designing APIs with ultimate performance, it's obvious that cache is indispensable. However, cache alone is still not enough. Therefore, we also need to focus on another word: less.

所谓“少”，有两层含义：**一是数据量要少，二是计算量要少，二者缺一不可**。不要忘记，我们的瓶颈不是仅网络IO，CPU、内存和磁盘等所有设备都是。

The so-called "less" has two meanings: first, the amount of data should be small, and second, the amount of computation should be small. Both are indispensable. Don't forget that our bottlenecks are not just network I/O. All devices such as the CPU, memory and disk can also be bottlenecks.

虽然我们通过缓存解决了IO问题，但是即使数据全部在本地，在高并发情况下，复杂的计算可能会耗尽CPU，而大体量数据的则可能会耗尽内存。

Although we have solved the I/O problem through caching, even if all the data is stored locally, under high-concurrency circumstances, complex computations may exhaust the CPU, while a large amount of data may exhaust the memory.

在更为极端的情况下，我们高并发下输出的日志都可以瞬间将磁盘打爆。不要认为这些都是耸人听闻，当我们设计不当的时候，它们就会即刻发生。

In more extreme cases, the logs output under high concurrency can instantly fill up the disk. Don't think these are just alarmist remarks. They will happen immediately when our design is improper.

第4式：准–数据一致

在分布式系统架构中，数据一致性是个老生常谈的话题。在秒杀架构中，中心化缓存和本地缓存可以极大地提升系统性能。

In the architecture of distributed systems, data consistency is a topic that has been discussed time and time again. In the architecture of flash sale systems, centralized caches and local caches can greatly improve the performance of the system.

但是，我们需要为此付出数据一致性的维护代价，即维护数据库、中心化缓存和本地缓存之间的一致性，主要的一致性问题集中在下面四个方面：

However, we need to pay the price for maintaining data consistency, that is, to maintain the consistency among the database, the centralized cache and the local cache. The main consistency issues mainly focus on the following four aspects:

- 缓存与数据库一致
- Cache is consistent with the database
- 本地缓存与中心缓存一致
- Local cache is consistent with central cache
- **订单与库存一致**
- Orders are consistent with inventory
- 前端与后端一致
- The front end is consistent with the back end

在上面这四种一致性关系中，订单与库存属于**强一致性**关系，即决不允许超卖的情况出现。而其他场景则属于**弱一致性**关系，一般允许毫秒级的延迟，以降低维护的复杂度。

Among the above four consistency relationships, the relationship between orders and inventory belongs to a strong consistency relationship, which means that overselling is never allowed to occur. While the other scenarios belong to weak consistency relationships, and generally allow delays at the millisecond level in order to reduce the complexity of maintenance.

关于本地缓存一致性的维护问题，可简单也可以很复杂，我们在此不打算展开陈述，推荐一篇文章仅供参考。

Regarding the issue of maintaining the consistency of local caches, it can be either simple or very complicated. We don't intend to elaborate on it here. Instead, we recommend an article for your reference only.

第5式：稳–风险管控

稳定性是秒杀架构中压倒一切的前提。**我们绝对不能容忍在唱着歌、吃着火锅的时候，车却翻了。甭管来者是张麻子还是王麻子，动不动就翻车的那是狗头师爷，真正的猛男从来不翻车。**

Stability is the overriding prerequisite in the flash sale architecture. We can never tolerate the situation where the vehicle overturns while we are singing and enjoying hot pot. No matter whether it's Zhang Mazi or Wang Mazi who comes, those who overturn the vehicle easily are nothing but despicable advisors. A real tough guy never has such accidents.

既然稳定性如此重要，我们该从哪些方面进手落实？答案是两个词：**隔离**和**监控**。

Since stability is so important, from which aspects should we start to implement it? The answer lies in two words: isolation and monitoring.

1. 风险隔离

在风险无法被完全规避时，**隔离**是一种非常有效的手段。

When risks cannot be completely avoided, isolation is a very effective means.

在新冠疫情肆虐全球的时候，我们之所以比很多国家更好地控制住了疫情，**“隔离”**在其中起到了非常关键的作用。通过隔离，我们将风险限定在可控的范围内，阻止了进一步地扩散和蔓延。目前国内仍有不少地方会偶发新一波的疫情，但通过隔离我们依然可以有效控制。此外，隔离在火灾防范方面也应用广泛，比如森林中有防火隔离带，现代建筑中也有防火隔离墙。

When the COVID-19 pandemic was raging across the globe, the reason why we managed to control the epidemic better than many countries was that "isolation" played a very crucial role in it. Through isolation, we confined the risks within a controllable range and prevented further spread and proliferation. At present, there are still quite a number of places in China where new waves of the epidemic break out occasionally, but we can still effectively control it through isolation. In addition, isolation is also widely used in fire

prevention. For example, there are firebreaks in forests and fireproof partition walls in modern buildings.

同样的，在分布式架构系统中，隔离也是防止灾难蔓延的有效手段，我们常见的“**熔断**”正是如此。

秒杀架构中的风险隔离主要体现在以下几个方面：

Similarly, in distributed architecture systems, isolation is also an effective means to prevent the spread of disasters, and the commonly seen "circuit breaker" is exactly like this.

The risk isolation in the flash sale architecture is mainly reflected in the following aspects:

- **应用隔离**：秒杀类应用需要承载更高的QPS，同时为了不影响其他业务，可以将秒杀类应用独立部署。隔离后的应用不仅可以防范灾难蔓延，在应用的伸缩方面也可以更为灵活，比如在活动期间增加机器数据量，活动结束后再回收，减少资源浪费。常见的应用隔离有**宿主机隔离**和**同应用分组**两种方式。
- Application isolation: Flash sale applications need to handle a higher QPS. Meanwhile, in order not to affect other business operations, flash sale applications can be deployed independently. Isolated applications can not only prevent the spread of disasters but also be more flexible in terms of application scaling. For example, the number of machines can be increased during the activity period and then recycled after the activity ends to reduce resource waste. Common application isolation methods include host isolation and the same application grouping.
- **数据库隔离**：数据库的连接池有限，在没有做数据库隔离的情况下，如果系统存在设计缺陷导致瞬间耗尽连接池，那么全站都将面临被拖垮的风险；
- Database isolation: The connection pool of the database is limited. In the absence(缺席) of database isolation, if there are design flaws in the system that lead to the instant exhaustion of the connection pool, then the entire website will face the risk of being brought down.
- **业务隔离**：在设计秒杀业务场景时，原则上路径要简单，不要依赖于其他业务场景；
- Business isolation: When designing the business scenarios for flash sales, in principle, the paths should be simple and should not rely on other business scenarios.
- **中间件隔离**：对于大部分公司来说，实现中间件隔离比较奢侈，使用更多的是**中间件降级**，即在秒杀活动期间，降低其他非核心应用对中间件资源的保障力度，比如延迟投递消息等。
- Middleware isolation: For most companies, implementing middleware isolation is rather luxurious. What is used more often is middleware degradation. That is, during the flash

sale activities, the guarantee level of middleware resources for other non-core applications is reduced, such as delaying the delivery of messages.

2. 风险监控

管理学大师彼得德鲁克曾经说过：**你如果无法度量它，就无法管理它**。同样的，如果我们不能定义风险的指标，我们将无法管理风险。因此，我们需要对系统中的**关键指标**进行梳理和定义：

Peter Drucker, a master of management, once said, "If you can't measure it, you can't manage it." Similarly, if we can't define the indicators of risks, we won't be able to manage risks. Therefore, we need to sort out and define the key indicators in the system.

- **基础指标**：CPU+内存+负载+网络+磁盘使用率；
- Basic indicators: CPU, memory, load, network and disk usage.
- **JVM指标**：Full GC和Young GC的次数和耗时，以及Java线程中的总线程、死锁线程和可运行线程等；
- JVM indicators: The number of times and the time consumption of Full GC and Young GC, as well as the total number of threads, deadlock threads and runnable threads in Java threads, etc.
- **中间件指标**：接口的QPS和RT、缓存的QPS和RT以及容量、数据库的QPS和RT以及连接池、MQ的QPS和RT以及消息堆积等。
- Middleware indicators: QPS and RT of interfaces, QPS and RT as well as capacity of caches, QPS and RT as well as connection pools of databases, QPS and RT as well as message accumulation of MQ.

指标定义好之后，接着就是实现对这些指标的监控和多渠道报警，而这方面已经有很多开源的成熟产品，Grafana。

After the indicators are defined, the next step is to implement the monitoring of these indicators and multi-channel alarming. And there are already many mature open-source products in this regard, such as Grafana.

第6式：练-全链路演练验收

所谓**练**，即通过压力测试给系统施加一百个点的暴击，看它能不能扛得住。

The so-called "practice" means applying a critical strike of one hundred points to the system through stress testing to see if it can withstand it.

俗话说，是骡子是马，拉出来溜溜。我们设计的秒杀系统究竟如何？系统的负载能力究竟怎样？配套的监控指标的采集和报警是否奏效？回答这些问题，不能凭直觉，而是要实打实地通过**全链路压测**进行验证。

As the saying goes, "You can tell the quality of something only when you put it to the test." How exactly is the flash sale system we've designed? What's the load capacity of the system like? Are the collection and alarming of the supporting monitoring indicators effective? To answer these questions, we can't rely on intuition. Instead, we need to verify them through end-to-end stress testing(全链路压力测试) in a down-to-earth manner.

注意，我们刚才说的是**全链路压测**，木桶原理告诉我们只有桶壁上的所有木板都足够高，那木桶才能盛满水。通过压测，我们可以掌握包括中间件在内的整个链路的性能表现，以及系统的瓶颈在哪里，对各方面做到心中有数。

Note that what we just mentioned is end-to-end stress testing. The barrel principle tells us that only when all the boards on the barrel wall are high enough can the barrel be filled with water. Through stress testing, we can grasp the performance of the entire link including middleware and figure out where the bottlenecks of the system are, thus having a clear understanding of all aspects.

小结

以上就是关于秒杀系统的设计原则和方法。总结来说，就是要记住**分-限-快-准-稳-练**这六脉神剑。

The above are the design principles and methods for the flash sale system. To sum up, we should remember the "Six Meridians Divine Sword" which consists of six aspects: isolation, limitation, speed, accuracy, stability and practice.

当然，系统架构没有放之四海而皆准的方法，秒杀架构也是如此。不同的用户规模、不同的业务目标、不同的组织和基础设施，秒杀系统的落地也有着不同的表现形式。

Of course, there is no one-size-fits-all method in system architecture, and the same is true for the flash sale architecture. With different user scales, different business objectives, different organizations and infrastructure, the implementation of the flash sale system also has different manifestations.

因此，**在设计秒杀系统的时候，我们不要生搬硬套别人的做法，而是要从业务目标、现有的基础设施水平和团队技术能力出发，基于秒杀系统的设计原则，有针对性地采纳和调整，才能设计出符合自己组织的秒杀系统**。

Therefore, when designing a flash sale system, we should not blindly copy what others have done. Instead, we should start from the business objectives, the existing level of infrastructure and the technical capabilities of the team. Based on the design principles of the flash sale system, we should adopt and adjust them in a targeted manner so as to design a flash sale system that suits our own organization.

到本文为止，关于秒杀系统设计的**理论部分**已经结束。在下篇文章中，我们将着手进行秒杀系统的需求分析和技术架构设计。

Up to this point in this article, the theoretical part regarding the design of the flash sale system has come to an end. In the next article, we will start with the demand analysis and technical architecture design of the flash sale system.

本章节到此结束，欢迎在评论区留下你的收获和疑问。对于本文未提及或未讲清楚的地方，也欢迎评论区批评指正，我会及时调整。如果你对小册有内容建议，请移步填写表单。

This chapter comes to an end. You are welcome to leave your gains and questions in the comment section. For the parts that are not mentioned or clearly explained in this article, you are also welcome to offer criticisms and corrections in the comment section, and I will make timely adjustments. If you have content suggestions for this booklet, please go to fill in the form.

第04章—谋定而动：案例分析与技术方案

在前两篇文章中，我们介绍了秒杀架构中的技术挑战，以及系统的设计原则和方法。然而，有了原则和方法还远远不够，正所谓**光说不练假把式**。

In the previous two articles, we introduced the technical challenges in the flash sale architecture as well as the design principles and methods of the system. However, having principles and methods alone is far from enough. As the saying goes, "It's all talk and no action."

未经实践的文字不仅读起来如同走马观花，而且难以消化理解，刚才还是“**马冬梅**”，转头就成了“**马什么梅**”。为此，我们从本文开始进入实战环节，将前文所述的原则与方法应用到实践中。

Texts that haven't been put into practice are not only read in a cursory manner but also difficult to digest and understand. One moment it's "Ma Dongmei", and the next moment it

becomes "What's Ma's name?". For this reason, starting from this article, we will enter the practical stage and apply the principles and methods mentioned before to practice.

在本文中，我们将从秒杀需求出发，探讨如何分析需求以及秒杀系统中的一些总体设计。**当然，限于篇幅，本文在阐述设计方案时将不会呈现完整细节，细节部分会在后续对应的文章中详细描述。**

In this article, we will start from the requirements of flash sales and explore how to analyze the requirements as well as some overall designs in the flash sale system. Of course, due to the limitation of space, this article will not present the complete details when elaborating on the design schemes. The detailed parts will be described in detail in the corresponding subsequent articles.

✨***阅读提示****文章篇幅较长且大图较多，阅读时可以浏览器右键新窗口查看高清大图，或在IDE中查看图片脚本及预览；此外，还可以安装Smart TOC浏览器插件完美体验可拖拽目录阅读，源码下载请参考第五章节。*

✨*Reading tips: This article is quite long and contains a large number of big pictures. When reading, you can right-click in the browser to open a new window to view the high-definition pictures, or view the picture scripts and previews in an IDE. Besides, you can also install the Smart TOC browser plugin to enjoy the reading experience with a draggable table of contents. For source code download, please refer to Chapter Five.*

一、需求概述

周五的傍晚，夕阳晚照，一抹余晖预示着美好的周末即将开始。你哼着小曲收起渔网，躁动着等待着下班时刻的到来。那时，你便可以箭步上前冲出公司，大步走向你风雨无阻、价值千万的专列：**地铁5号线**。

On Friday evening, the setting sun bathed everything in its warm glow, and the last rays of sunlight heralded the beginning of a wonderful weekend. You were humming a tune as you put away the fishing net, feeling restless and eagerly waiting for the moment to get off work. At that time, you would be able to dash out of the company and stride towards your exclusive train that never fails you regardless of wind or rain: Subway Line 5.

然而，突然间你隐约地察觉到一阵凉风落在了你的后背。莫非又是他？你迅速地把眼睛转向桌角的水杯，在玻璃杯的映射中你看到了背后的产品经理，果然是他。

However, all of a sudden, you vaguely felt a cool breeze on your back. Could it be him again? You quickly turned your eyes to the water cup at the corner of the desk. In the reflection of the glass, you saw the product manager behind you. Sure enough, it was him.

“老弟，我们老大说需要一个秒杀系统。可以让用户在中秋节秒杀月饼，在双十一秒杀特价商品，在圣诞节也可以秒杀圣诞老人”。还没等你转过头来，产品经理已经先开了口。

"Buddy, our boss said we need a flash sale system. It can allow users to snap up mooncakes during the Mid-Autumn Festival, snap up special-priced goods on Double Eleven, and even snap up Santa Claus at Christmas." Before you could turn your head around, the product manager had already started speaking.

“呃...还有吗？”

“Er... Anything else?”

“还有就是最好下周末能上线。”

"And another thing is that it would be best if it can be launched next weekend."

作为职场老兵，你深知无论产品经理怎么说以及说了什么，你都必须保持足够的淡定，即使你听完很生气，即使**“滚犊子”**三个字到了嘴边也得咽下去。对于这样的需求，三言两语断然不行。于是，你故作镇静地把飘到九霄云外的心往回收了收，试图**引导产品经理说出关于产品的更多想法细节**...

As a veteran in the workplace, you are well aware that no matter what the product manager says or how he says it, you must remain calm enough. Even if you feel angry after listening to him, even if the words "get lost" are on the tip of your tongue, you have to swallow them back. For such requirements, a few words will definitely not suffice. So, you pretended to be calm and pulled back your mind that had wandered far away, trying to guide the product manager to share more detailed ideas about the product...

经过了半个小时的挖掘，对于产品经理嘴里所说的秒杀系统，你已经帮助产品经理理清了眉目，并确认了其中的一些关键信息：

After half an hour of in-depth exploration, regarding the flash sale system mentioned by the product manager, you have already helped the product manager sort things out and confirmed some of the key information.

- **运营可以组织不同的秒杀场次，多场次可以同时进行；**
- Operations can organize different flash sale sessions, and multiple sessions can be carried out simultaneously..
- **每个秒杀活动场次可以上线不同的秒杀品；**
- Each flash sale session can have different flash sale products online..
- **运营可以灵活地控制场次和秒杀品的状态，以及上下线时间；**
- Operations can flexibly control the status of sessions and flash sale products, as well as

their online and offline times..

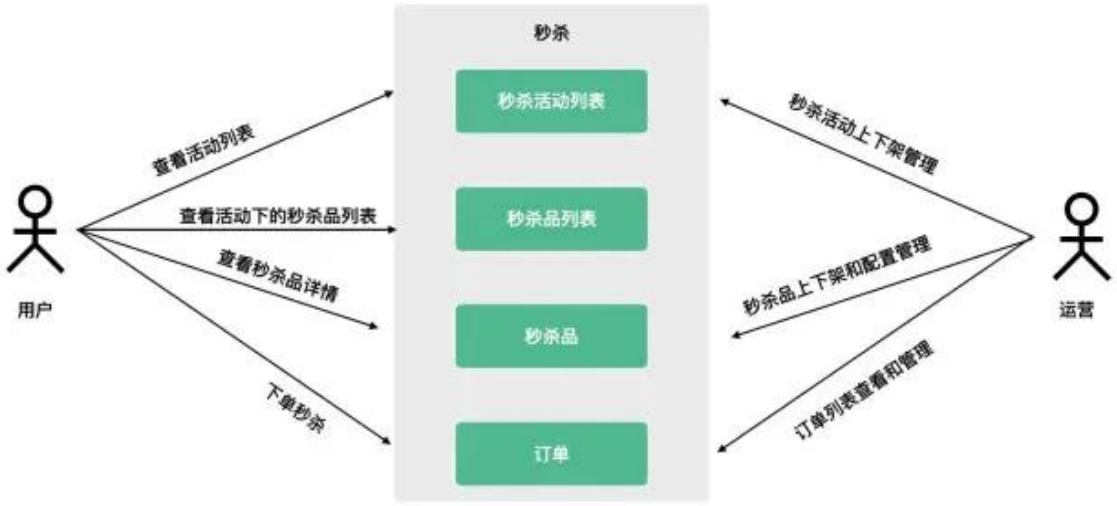
- **用户可以按照活动场次查看对应的秒杀品和详情；**
- Users can view corresponding flash sale products and details according to activity sessions..
- **在秒杀品详情页，用户可以看到库存和商品介绍等信息，并可以执行下单操作；**
- On the detail page of flash sale products, users can see information such as inventory and product descriptions, and can perform order placement operations..
- **前期并发流量大概10万，后面有可能会增加，主要看效果和运营投入。**
- In the early stage, the concurrent traffic is approximately 100,000. It may increase later, mainly depending on the effect and operational investment..

二、用户故事

业务是技术的出发点，也是落脚点。所以，即使你胸有成竹跃跃欲试，也要保持克制，不要上来就着手技术方案设计或直接撸代码，而是要首先将需求理解透彻。

Business is both the starting point and the end point of technology. Therefore, even if you are confident and eager to have a try, you should still exercise restraint. Don't rush to start designing technical solutions or directly write code. Instead, you should first fully understand the requirements.

如何将产品的思考条理性地表述出来，有很多种处理方式，我们这里采用**用户故事**的方式。通过分析，我们将与产品的沟通记录转换成以下的表述。



1. 创建秒杀活动

👉 **用户故事**

👉 User Story

作为**运营**，可以**创建新的秒杀活动**，以便可以通过活动**组织和管理秒杀品**。

As an operator, I can create new flash sale activities so that I can organize and manage flash sale products through these activities.

👉 **验收标准**

- 创建秒杀活动时，需要提供活动**名称**、**描述**、**活动起止时间**等关键字段；
- 秒杀活动字段校验应满足规则：
  - 名称：非空，字符串，长度不超过20；
  - 描述：可空，文本，长度不限；
  - 起止时间：非空，时间类型，起止时间不能晚于当前时间，结束时间不能早于开始时间；
- 当运营输入的字段符合校验规则时，创建活动成功；
- 当运营输入的字段不符合校验规则时，活动创建失败；
- 创建活动时需要进行权限校验，未授权运营无权创建活动。

👉 Acceptance Criteria

● When creating a flash sale activity, key fields such as the activity name, description, start and end time of the activity need to be provided.

● The field validation for the flash sale activity should meet the following rules:

○ Name: It should not be empty, be in string format, and the length should not exceed 20 characters.

○ Description: It can be empty, be in text format, and there is no limit on the length.

○ Start and end time: It should not be empty, be in time format. The start and end time cannot be later than the current time, and the end time cannot be earlier than the start time.

● When the fields entered by the operator **meet** the validation rules, the activity is created successfully.

● When the fields entered by the operator do not meet the validation rules, the activity creation fails.

● Permission verification is required when creating an activity. Unauthorized operators have no right to create activities.

2. 上线秒杀品

👉 用户故事

作为**运营**，在创建完秒杀活动后可以上线新的秒杀品，以便让用户进行秒杀下单。

👉 User Story

As an operator, I can launch new flash sale products after creating flash sale activities so that users can place orders for the flash sales.

👉 验收标准

👉 Acceptance Criteria

- 上线的秒杀品应包含**商品编码**、**标题**、**副标题**、**描述**、**库存**、**原价**、**秒杀价**、**起止时间**等必要字段信息；
- The launched flash sale products should contain necessary field information such as product code, title, subtitle, description, inventory quantity, original price, flash sale price, start and end time.
- 秒杀品字段应该满足规则：
- The fields of flash sale products should meet the following rules:
  - 商品编码：非空，字符串，长度不超过20；
  - Product code: It should not be empty, be in string format, and the length should not exceed 20 characters.
  - 标题：非空，字符串，长度不超过50；
  - Title: It should not be empty, be in string format, and the length should not exceed 50 characters.
  - 副标题：非空，字符串，长度不超过100；
  - Subtitle: It should not be empty, be in string format, and the length should not exceed 100 characters.
  - 描述：可空，文本，长度不限；
  - Description: It can be empty, be in text format, and there is no limit on the length.
  - 库存：非空，数字，范围大于0；

    - Inventory quantity: It should not be empty, be in numeric format, and the value should be greater than 0.
    - 原价：非空，金额，范围大于0；
    - Original price: It should not be empty, be in monetary amount format, and the value should be greater than 0.
    - 秒杀价：非空，金额，范围大于0，且不得高于原价；
    - Flash sale price: It should not be empty, be in monetary amount format, the value should be greater than 0, and it must not be higher than the original price.
    - 起止时间：非空，时间类型，起止时间不能晚于当前时间，结束时间不能早于开始时间；
    - Start and end time: It should not be empty, be in time format. The start and end time cannot be later than the current time, and the end time cannot be earlier than the start time.
    - 所属活动：非空，指定所属活动的ID。
    - Associated activity: It should not be empty, and the ID of the specified associated activity should be provided.
- 秒杀品必要信息校验未通过时，应拒绝上架；
- When the necessary information verification of the flash sale products fails, the products should be rejected from being put on the shelf.
- 秒杀活动存在相同商品编码的秒杀品且处于进行中的状态时，应拒绝上架；
- When there are flash sale products with the same product code in the flash sale activity and they are in progress, the new products should be rejected from being put on the shelf.
- 运营上线秒杀品时，应指定该秒杀品所属活动；
- When the operator launches flash sale products, the associated activity of these products should be specified.
- 如果秒杀品所指定的活动不存在或者已经下架，应拒绝上架；
- If the specified activity for the flash sale products does not exist or has already been taken off the shelf, the products should be rejected from being put on the shelf.

3. 查看秒杀活动

👉 **用户故事**

👉 User Story

作为**用户**，可以在活动页**查看进行中的秒杀活动**，以便**找到并参与**适合自己的秒杀活动。

As a user, I can view the ongoing flash sale activities on the activity page so that I can find and participate in the flash sale activities that suit me.

👉 **验收标准**

👉 Acceptance Criteria

- 进入秒杀活动列表页面，可以查看对应的秒杀活动；
- When entering the flash sale activity list page, the corresponding flash sale activities can be viewed.
- 活动列表中应包含活动名称、活动状态等信息；
- The activity list should contain information such as the activity name and activity status.
- 查看秒杀活动无需权限校验，对所有用户开放。
- There is no need for permission verification when viewing flash sale activities, and it is open to all users.

4. 查看秒杀品

👉 **用户故事**

👉 User Story

作为用户，可以通过点击活动列表页的活动，查看活动下的秒杀品列表，在可以通过点击具体的秒杀品，查看秒杀品详情，以便准确了解秒杀品是否符合自己需要。

As a user, I can view the list of flash sale products under an activity by clicking on the activity on the activity list page. And then I can view the details of a specific flash sale product by clicking on it, so that I can accurately understand whether the flash sale product meets my needs.

👉 **验收标准**

👉 Acceptance Criteria

- 进入秒杀活动列表页面，可以查看对应的秒杀活动；
- When entering the flash sale activity list page, the corresponding flash sale activities

can be viewed.

- 点击指定的秒杀活动时，进入秒杀品列表页；
- When clicking on a specified flash sale activity, enter the flash sale product list page.
- 秒杀品列表页应包含的信息：秒杀品标题、秒杀价、剩余库存；
- The information that should be included on the flash sale product list page: the title of the flash sale product, the flash sale price, and the remaining inventory.
- 点击具体的秒杀品，进入秒杀品详情页；
- When clicking on a specific flash sale product, enter the flash sale product details page.
- 秒杀品详情页包含的信息：**标题**、**副标题**、**描述**、**剩余库存**、**原价**、**秒杀价**、**起止时间**。
- The information included on the flash sale product details page: title, subtitle, description, remaining inventory, original price, flash sale price, start and end time.

5. 下单秒杀品

👉 用户故事

👉 User Story

作为**用户**，可以在秒杀品详情页通过秒杀按钮提交订单，以便抢购心仪的秒杀品。

As a user, I can submit an order through the flash sale button on the flash sale product details page so that I can rush to purchase the flash sale products I like.

👉 验收标准

👉 Acceptance Criteria

- 当秒杀品有库存时，秒杀按钮可用；
- When there is inventory for the flash sale product, the flash sale button is available.
- 当秒杀品没有库存或秒杀时间已经结束时，秒杀按钮置灰不可用；
- When the flash sale product is out of stock or the flash sale time has ended, the flash sale button is grayed out and unavailable.
- 当用户点击秒杀按钮时，给出loading提示，直到请求结束或超时；
- When the user clicks on the flash sale button, a loading prompt will be shown until the request is completed or times out.

- 当下单成功或失败时，loading消失并给出具体的成功或失败的提示说明。
- When the order is placed successfully or fails, the loading prompt disappears and specific success or failure prompt descriptions are given.

6. 查看我的订单

👉 用户故事

👉 User Story

作为**用户**，可以在“我的订单”中查看下单成功的订单，以便了解和管理秒杀成功的订单。

As a user, I can view the orders that have been successfully placed in "My Orders" so that I can understand and manage the orders for which the flash sales were successful.

👉 验收标准

👉 Acceptance Criteria

- 进入“我的订单”，按分页展示下单成功秒杀品列表；
- Enter "My Orders" and display the list of successfully ordered flash sale products in pages.
- 列表中应包含的信息：订单编号、秒杀品标题、下单时间、状态；
- The information that should be included in the list: order number, title of the flash sale product, order placement time, and status.
- 秒杀品应包含的按钮：取消按钮、删除按钮。
- The buttons that the flash sale product should include: a cancel button and a delete button.

7. 管理我的订单

👉 用户故事

👉 User Story

作为**用户**，可以在“我的订单”中对已下单成功的订单进行删除和取消，以实现用户对自己订单的管理。

As a user, I can delete and cancel the orders that have been successfully placed in "My Orders", so as to enable users to manage their own orders.

👉 **验收标准**

👉 Acceptance Criteria

- 进入“我的订单”，按分页展示下单成功秒杀品列表；
- Enter "My Orders" and display the list of successfully ordered flash sale products in pages.
- 点击对应订单的“取消”按钮，可以取消当前订单；
- Click on the "Cancel" button of the corresponding order to cancel the current order.
- 取消订单时将恢复库存，并将仍然存在于订单列表中；
- When canceling an order, the inventory will be restored and the order will still remain in the order list.
- 点击对应订单的“删除”按钮，可以删除当前订单；
- Click on the "Delete" button of the corresponding order to delete the current order.
- 已删除的订单不会恢复库存，并从订单列表中消失。
- The deleted orders will not restore the inventory and will disappear from the order list.

三、技术架构目标与方案设计

1. 方案目标

目标	关键点
业务目标	实现用户故事中的各类场景目标，交付无延期
技术目标	1. 秒杀品详情页接口：**100,000**; 2. 秒杀接口：**10,000**
架构目标	系统高可用：指标可监控、故障可预警、资源可伸缩

Objectives	Key Points
Business Objectives	To achieve various scenario objectives in user stories and deliver without delay.

Technical Objectives	1. Interface of flash sale product details page: 100,000; 2. Flash sale interface: 10,000.
Architectural Objectives	High availability of the system: Metrics can be monitored, failures can be predicted in advance, and resources can be scaled.

备注：由于QPS目标与硬件规模相关，在忽略服务器硬件配置和集群规模的情况下谈QPS毫无意义。因此，**在系统的开发中，更重要的是着眼于如何评估系统的整体性能表现，以及如何发现瓶颈并优化。**

Remarks: Since the QPS (Queries Per Second) target is related to the hardware scale, it makes no sense to talk about QPS without considering the server hardware configuration and the scale of the cluster. Therefore, in the development of the system, it is more important to focus on how to evaluate the overall performance of the system and how to identify bottlenecks and optimize them.

2. 总体架构

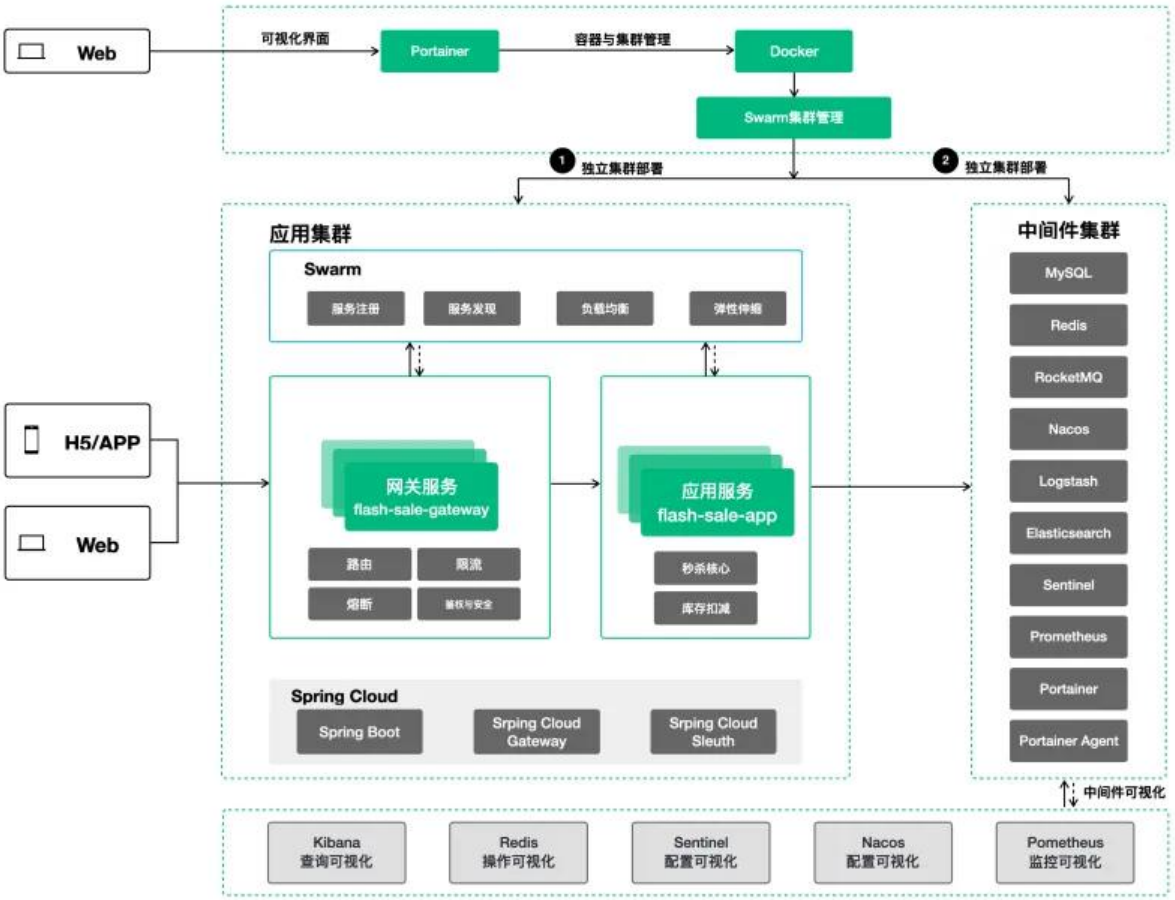
系统架构总体设计如下图所示，它展示了我们秒杀系统设计所涉及到的技术选型和拓扑结构。除了Spring Boot等基础框架外，在架构上也采用了容器化等方案。

The overall design of the system architecture is shown in the following figure, which demonstrates the technology selection and topological structure involved in the design of our flash sale system. Besides basic frameworks such as Spring Boot, solutions like containerization are also adopted in the architecture.

分层架构视角

Perspective of Layered Architecture

链路视角



为了兼顾不同层次的读者，降低对技术理解的门槛和成本，在具体的技术选型上我们主要采用目前市面上主流的技术产品和方案：

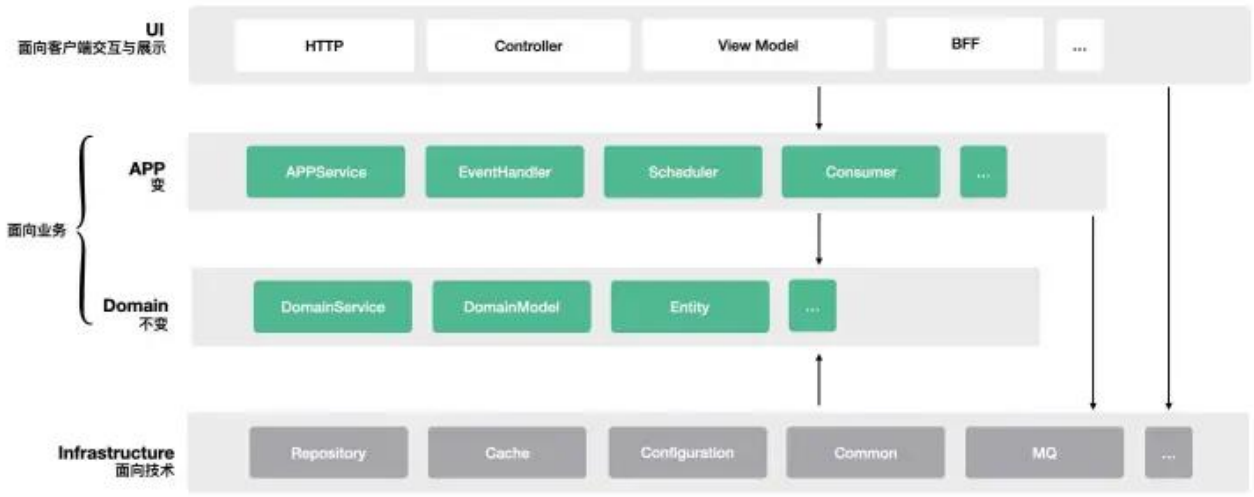
In order to cater to readers at different levels and reduce the threshold and cost of understanding technology, in terms of specific technology selection, we mainly adopt mainstream technology products and solutions currently available on the market:

- 应用框架：Spring Boot
- Application Framework: Spring Boot
- 数据库：MySQL+Mybatis
- Database: MySQL + Mybatis
- 缓存：Redis+本地缓存
- Cache: Redis + Local Cache
- 单元测试：Junit
- Unit Testing: Junit
- 容器化：Docker
- Containerization: Docker
- 容器化管理：Swarm、Portainer
- Containerization Management: Swarm, Portainer
- 可观性与可视化监控：Prometheus、Grafana
- Observability and Visual Monitoring: Prometheus, Grafana
- 限流：Sentinel
- Rate Limiting: Sentinel
- 压测工具：Jmeter
- Performance Testing Tool: Jmeter

3. 分层架构

在应用架构上，我们主要借鉴DDD的分层架构思想，将应用分为展示层、应用层、领域层和基础设施层四个层次，各层的职责定义和依赖关系如下图所示。

In the application architecture, we mainly draw on the idea of the layered architecture of Domain-Driven Design (DDD) and divide the application into four layers, namely the presentation layer, application layer, domain layer, and infrastructure layer. The definitions of responsibilities and the dependency relationships of each layer are shown in the following figure.



（1）用户界面层（UI）

用户界面层是应用程序的最上层，负责向用户显示信息和解释用户指令，它所承担的其实是类似于**BFF**的职责。在我们平日所见的一般性代码结构里，它是API的最外层。

The user interface layer is the top layer of the application, responsible for displaying information to users and interpreting user instructions. In fact, it undertakes responsibilities similar to those of a Backend for Frontend (BFF). In the general code structures we usually see, it is the outermost layer of the API.

在我们即将设计的秒杀系统中，用户界面层将主要**负责处理API请求**，以及相关的**模型定义**和**异常处理**。

In the flash sale system we are about to design, the user interface layer will be mainly responsible for handling API requests, as well as relevant model definitions and exception handling.

（2）应用层

在软件系统的设计中，软件系统的业务逻辑有两个主要部分构成：**应用程序逻辑**和**领域逻辑**。前者千变万化，后者则稳如泰山，而应用层就属于前者。

In the design of software systems, the business logic of software systems consists of two main parts: application logic and domain logic. The former is constantly changing, while the latter remains as stable as a mountain, and the application layer belongs to the former.

应用层面向用户故事，处理多种多样的**前端场景**，以及相关的**事件处理**、**调度处理**和**其他聚合处理**等。

The application layer is oriented(面向) towards user stories, handling a wide variety of front-end scenarios, as well as relevant event handling, scheduling handling and other aggregation handling.

（3）领域层

领域层是DDD分层架构中的精髓，它是业务系统中**相对不变**的部分，是领域模型以及所有与其直接相关的设计元素的表现。在MODEL-DRIVEN DESIGN中，领域层的软件构造反映了模型概念。

The domain layer is the essence of the Domain-Driven Design (DDD) layered architecture. It is the relatively unchanging part in the business system and represents the domain model and all the design elements directly related to it. In Model-Driven Design, the software construction of the domain layer reflects the model concepts.

在我们秒杀系统中，领域层没有依赖基础设施层，甚至于它没有任何的**直接**依赖，它是**纯净**、**干脆的**存在。

In our flash sale system, the domain layer has no dependence on the infrastructure layer. In fact, it doesn't have any direct dependencies at all. It exists in a pure and straightforward manner.

（4）基础设施层

基础设施层为上面各层提供**通用的技术能力**，包括**消息处理**、**持久化机制**、**缓存处理**等，基础设施层还可以通过软件架构来支持各层次间的交互模式。

The infrastructure layer provides common technical capabilities for the upper layers, including message processing, persistence mechanisms, cache processing, etc. The infrastructure layer can also support the interaction patterns among various layers through the software architecture.

需要注意的是，如果你细看上面的分层架构图，你可能会发现领域层没有依赖基础设施层，反而是基础设施层依赖了领域层。

It should be noted that if you take a closer look at the above layered architecture diagram, you may find that the domain layer does not depend on the infrastructure layer. Instead, it is the infrastructure layer that depends on the domain layer.

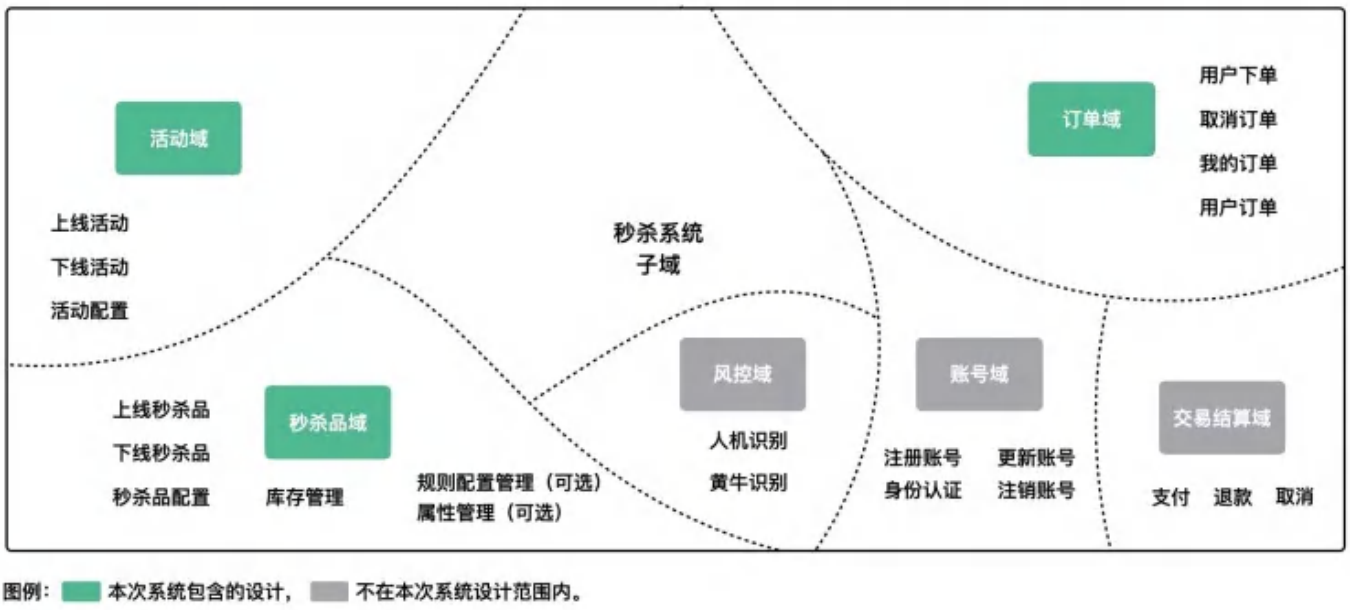
初看起来，这与我们平时所见的分层架构图不同，但这是我们对系统审慎评估后的方案，我们认为**领域层应该保持足够的纯净，它对外部是抽象依赖，而不是直接的实现依赖**。如果领域层依赖于基础设施层，那么领域层将变得臃肿，同时我们需要增加模块处理持久层的实现等。

At first glance, this is different from the layered architecture diagrams we usually see. However, this is the solution after our careful assessment of the system. We believe that the domain layer should remain pure enough. It has an abstract dependence on the outside rather than a direct implementation dependence. If the domain layer depends on the infrastructure layer, then the domain layer will become bloated. Meanwhile, we will need to add modules to handle the implementation of the persistence layer and so on.

4. 限界上下文

在一个DDD项目中，会有很多的限界上下。这些限界上下文中，有一个或多个会成为核心域，而其他的限界上下文中则会出现许多不同的子域。

In a Domain-Driven Design (DDD) project, there will be numerous bounded contexts. Among these bounded contexts, one or more will become the core domain, while many different sub-domains will emerge(出现) in the other bounded contexts.



秒杀系统包含了多个领域，包括**核心子域**、**支撑域**和**通用子域**等。

The flash sale system encompasses multiple domains, including the core sub-domains, supporting domains, and general sub-domains, etc.

比如，**秒杀系统中的活动域、秒杀品域等属于核心子域，是我们需要重点设计并亲自实现的**；

For example, the activity domain and the flash sale product domain in the flash sale system belong to the core sub-domains, which are the key parts that we need to focus on in design and implement by ourselves.

而账号子域、交易结算子域等则属于支撑子域，它们具有复用属性，在使用上并不局限于秒杀系统，可以和其他团队和业务共享，这部分可以和其他团队合作完成；

While the account sub-domain and the transaction settlement sub-domain belong to the supporting domains. They have the property of being reusable and are not limited to the flash sale system in terms of usage. They can be shared with other teams and businesses, and this part can be completed in cooperation with other teams.

最后的风控、缓存、限流等技术型的通用子域则可以通过采用开源产品或购买等引入，不需要我们自己开发。

Finally, the general technical sub-domains such as risk control, caching, and rate limiting can be introduced by adopting open-source products or through purchasing, without the need for us to develop them on our own.

有效地识别出不同的子域，**可以更加合理地分配组织的资源，把优秀资源投入到重要的事情中**，这也是DDD的重要价值体现。在我们秒杀系统中，当我们把核心域、支撑域识别出来后，我们就可以知道哪些是需要我们讲解和实现的重点，而哪些我们是可以采用其他集成方案并一笔带过的。

Effectively identifying different sub-domains enables a more rational allocation of an organization's resources, allowing excellent resources to be invested in important matters. This is also an important manifestation of the value of Domain-Driven Design (DDD). In our flash sale system, once we identify the core domain and the supporting domain, we can figure out which aspects are the key points that we need to elaborate on and implement, and which ones we can adopt other integration solutions for and just mention briefly.

关于核心域、支撑子域和通用子域的补充说明

Supplementary Explanations on the Core Domain, Supporting Sub-domains and General Sub-domains

- ***核心域（Core Domain）：****所谓核心域，它是一个唯一的、定义明确的领域模型，不需要太过复杂的理解，你只需要知道对待核心域，要对它进行**战略投资**，并在一个明确的限界上下文中投入大量资源去精心打磨通用语言。它是组织中最重要的项目，某种程度上说，它是你与竞争者的区别所在。换句话说，任何组织都无法在所有领域出类拔萃，所以必须把核心领域打造成组织的核心竞争力。*
- *Core Domain: The so-called core domain is a unique and clearly defined domain*

model. There's no need for overly complicated understanding. All you need to know is that when it comes to the core domain, strategic investment should be made in it, and a large amount of resources should be devoted to carefully refining the ubiquitous language within a specific bounded context. It is the most important project in an organization. To some extent, it is what distinguishes you from your competitors. In other words, no organization can excel in all areas, so the core domain must be built into the core competitiveness of the organization.

- ***支撑子域（Supporting Subdomain）**：对于支撑子域的建模场景，一般提倡的是**定制开发**。也就是说，它也挺重要，但可能没有现成的方案，又不是核心领域，所以可以定制开发，或者说外包开发。但要注意的是，虽说是支撑子域，在战略投资上不如核心域，但它是核心域成功的基础，也需要认真对待。*
- *Supporting Subdomain: For the modeling scenarios of the supporting subdomain, custom development is generally advocated. That is to say, it is also quite important, but there may not be ready-made solutions available, and it is not the core domain either. So it can be developed through customization or outsourced for development. However, it should be noted that although it is a supporting subdomain and receives less strategic investment than the core domain, it is the foundation for the success of the core domain and also needs to be taken seriously.*
- ***通用子域（Generic Subdomain）**：所谓通用子域，你可以简单理解为可以购买现成的，也可以外包实现。总之，它没那么重要，又不可或缺，不必投入太多资源。*
- *Generic Subdomain: As for the so-called generic subdomain, you can simply understand that it can be purchased as ready-made products or achieved through outsourcing. In short, it is not that important yet indispensable, and there is no need to invest too many resources in it.*

5. 时序图

根据前面梳理的用户故事，共有7个时序图。限于篇幅，我们这里只展现两个时序图，其他的时序图在后续介绍相关模块时会有详细说明。

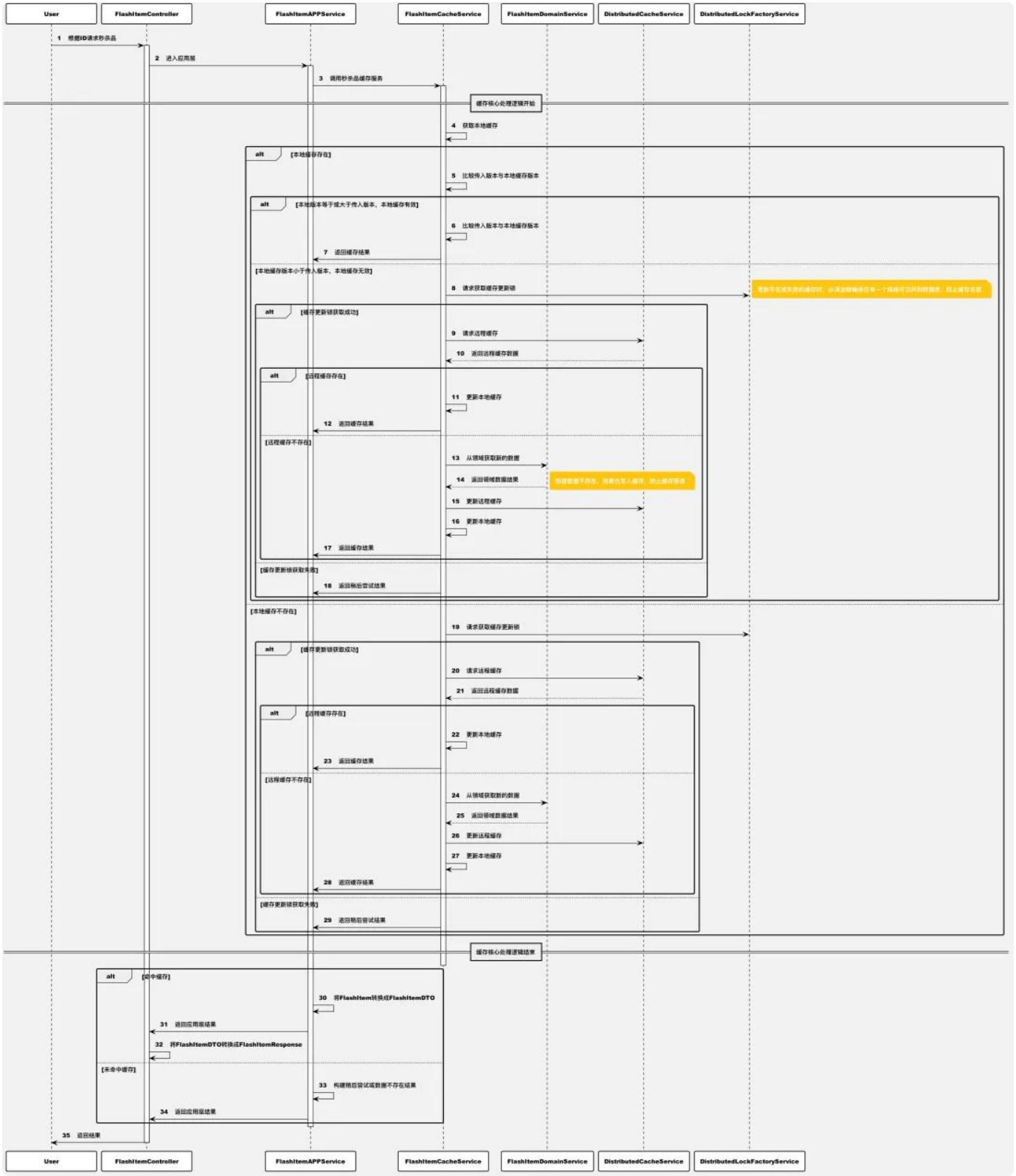
According to the user stories sorted out previously, there are a total of 7 sequence diagrams. Due to space limitations, only two sequence diagrams are presented here. The other sequence diagrams will be explained in detail when the relevant modules are introduced later.

秒杀品详情示例时序图

Sequence Diagram Example for Flash Sale Product Details

查看高清原图

View the High-Definition Original Image

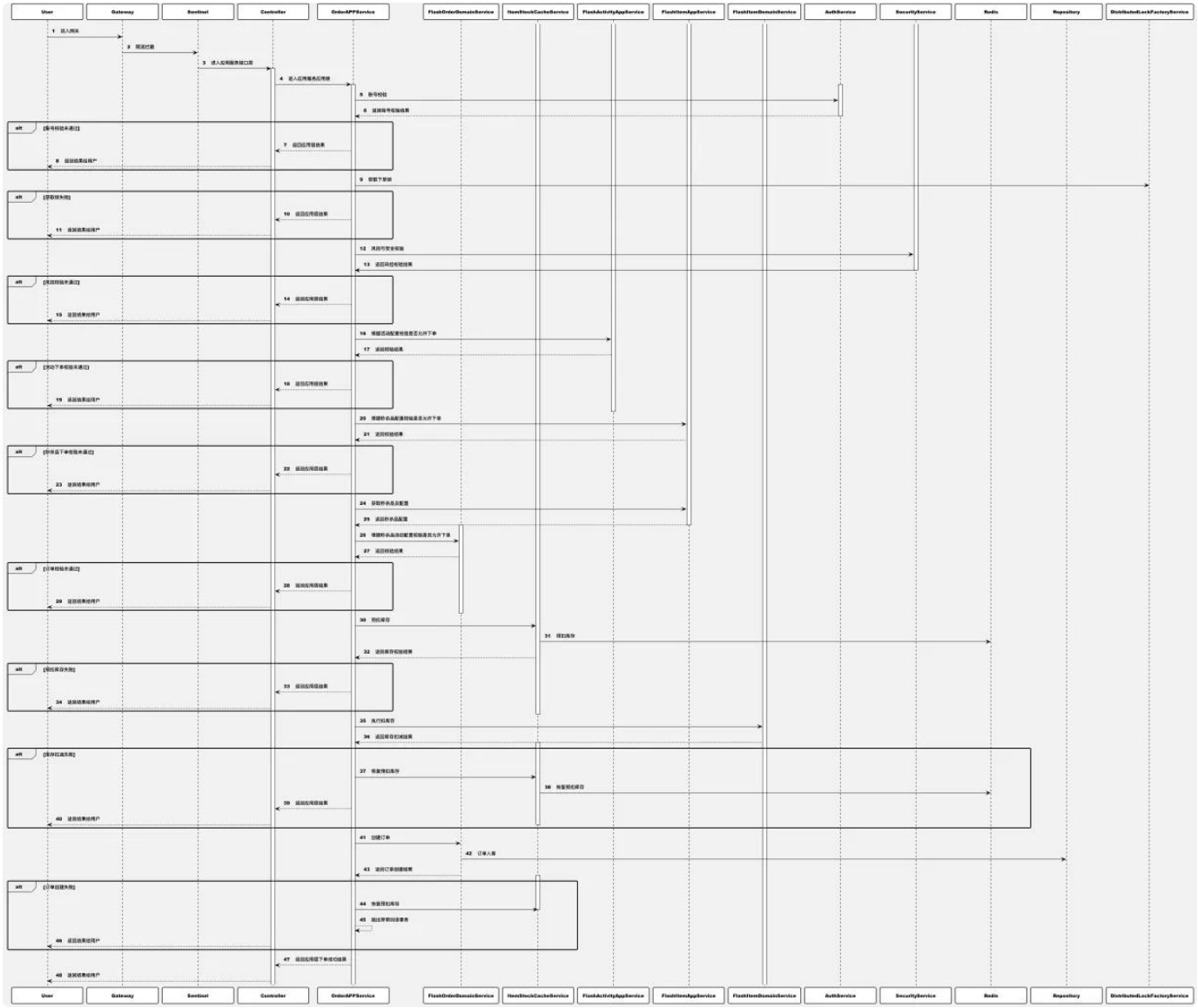


同步下单示例时序图

Sequence Diagram Example for Synchronous Order Placement

查看高清原图

View the High-Definition Original Image



6. 数据库设计

（1）秒杀活动表

Flash Sale Activity Table

```
CREATE TABLE IF NOT EXISTS flash_sale.`flash_activity` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `activity_name` varchar(50) NOT NULL COMMENT '秒杀活动名称',
  `activity_desc` text COMMENT '秒杀活动描述',
  `start_time` datetime NOT NULL COMMENT '秒杀活动开始时间',
  `end_time` datetime NOT NULL COMMENT '秒杀活动结束时间',
  `status` int(11) NOT NULL DEFAULT '0' COMMENT '秒杀活动状态',
  `modified_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '更新时间',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`),
  KEY `flash_activity_end_time_idx` (`end_time`),
  KEY `flash_activity_start_time_idx` (`start_time`),
  KEY `flash_activity_status_idx` (`status`)
) ENGINE = InnoDB CHARSET = utf8mb4 COMMENT '秒杀活动表';
```

（2）秒杀品表

Flash Sale Product Table

```
Plain Text
```

```
CREATE TABLE IF NOT EXISTS flash_sale.`flash_item` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `item_title` varchar(50) NOT NULL COMMENT '秒杀品名称标题',
  `item_sub_title` varchar(50) NULL COMMENT '秒杀品副标题',
  `item_desc` text COMMENT '秒杀品介绍富文本文案',
  `initial_stock` int(11) NOT NULL DEFAULT '0' COMMENT '秒杀品初始库存',
  `available_stock` int(11) NOT NULL DEFAULT '0' COMMENT '秒杀品可用库存',
  `stock_warm_up` int(11) NOT NULL DEFAULT '0' COMMENT '秒杀品库存是否已经预热',
  `original_price` bigint(20) NOT NULL COMMENT '秒杀品原价',
  `flash_price` bigint(20) NOT NULL COMMENT '秒杀价',
  `start_time` datetime NOT NULL COMMENT '秒杀开始时间',
  `end_time` datetime NOT NULL COMMENT '秒杀结束时间',
  `rules` text COMMENT '秒杀可配规则，JSON格式',
  `status` int(11) NOT NULL DEFAULT '0' COMMENT '秒杀品状态',
  `activity_id` bigint(20) NOT NULL COMMENT '所属活动id',
  `modified_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '更新时间',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`),
  KEY `flash_item_end_time_idx` (`end_time`),
  KEY `flash_item_start_time_idx` (`start_time`),
  KEY `flash_item_status_idx` (`status`)
) ENGINE = InnoDB CHARSET = utf8mb4 COMMENT '秒杀品';
```

（3）秒杀订单表

Flash Sale Order Table

```
Plain Text
```

```
CREATE TABLE IF NOT EXISTS flash_sale.`flash_order` (
  `id` bigint(20) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `item_id` bigint(20) NOT NULL COMMENT '秒杀品ID',
  `activity_id` bigint(20) NOT NULL COMMENT '秒杀活动ID',
  `item_title` varchar(50) NOT NULL COMMENT '秒杀品名称标题',
  `flash_price` bigint(20) NOT NULL COMMENT '秒杀价',
  `quantity` int(11) NOT NULL COMMENT '数量',
  `total_amount` bigint(20) NOT NULL COMMENT '总价格',
  `status` int(11) NOT NULL DEFAULT '0' COMMENT '订单状态',
  `user_id` bigint(20) NOT NULL COMMENT '用户ID',
  `modified_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '更新时间',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP COMMENT '创建时间',
  PRIMARY KEY (`id`),
  UNIQUE `flash_order_id_uk` (`id`),
  KEY `flash_order_user_id_idx` (`user_id`)
) ENGINE = InnoDB CHARSET = utf8mb4 COMMENT '秒杀订单表';
```

注意，以上建表语句均适用于未分库分表情况，分库分表相关建表语句请参考源码enviroment下的初始化建表语句和相关章节。

Note that the above table creation statements are all applicable to the situation where database and table sharding has not been implemented. For the table creation statements related to database and table sharding, please refer to the initialization table creation statements under the "enviroment" in the source code and the relevant sections.

7. 代码结构

Plain Text

```
.
├── README.md
├── environment  --> 应用依赖的中间件初始化脚本及数据
│   ├── config
│   ├── data
│   ├── docker-compose.yml
│   └── grafana-storage
├── postman  --> Postman测试脚本
│   └── flash-sale-postman.json
├── flash-sale-app  --> 应用层
│   ├── pom.xml
│   └── src
│       ├── main
│       │   └── java
│       │   │   └── com.actionworks.flashsale
│       └── test
│           ├── java
│           │   └── com
│           │       └── actionworks
│           │           └── flashsale
│           └── resources
│               └── logback-test.xml
├── flash-sale-controller   --> UI层
│   ├── pom.xml
│   └── src
│       └── main
│           └── java
│               └── com.actionworks.flashsale
├── flash-sale-domain   --> 领域层
│   ├── pom.xml
│   └── src
│       ├── main
│       │   └── java
│       │       └── com.actionworks.flashsale
│       └── test
│           └── java
│               └── com.actionworks.flashsale
├── flash-sale-infrastructure  --> 基础设施层
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   └── com.actionworks.flashsale
│       │   └── resources
│       │       ├── mybatis
```

```
46  |       |       └── mybatis-config.xml
47  |       └── test
48  |           ├── java
49  |           |   └── com.actionworks.flashsale
50  |           └── resources
51  |               ├── logback-test.xml
52  |               └── mybatis-config-test.xml
53  ├── mvnw
54  ├── mvnw.cmd
55  ├── pom.xml
56  └── start  --> 系统启动入口
57      ├── pom.xml
58      └── src
59          ├── main
60          |   ├── java
61          |   |   └── com.actionworks.flashsale
62          |   └── resources
63          └── test
64              ├── java
65              |   └── com.actionworks.flashsale
66              └── resources
67                  └── logback-test.xml
```

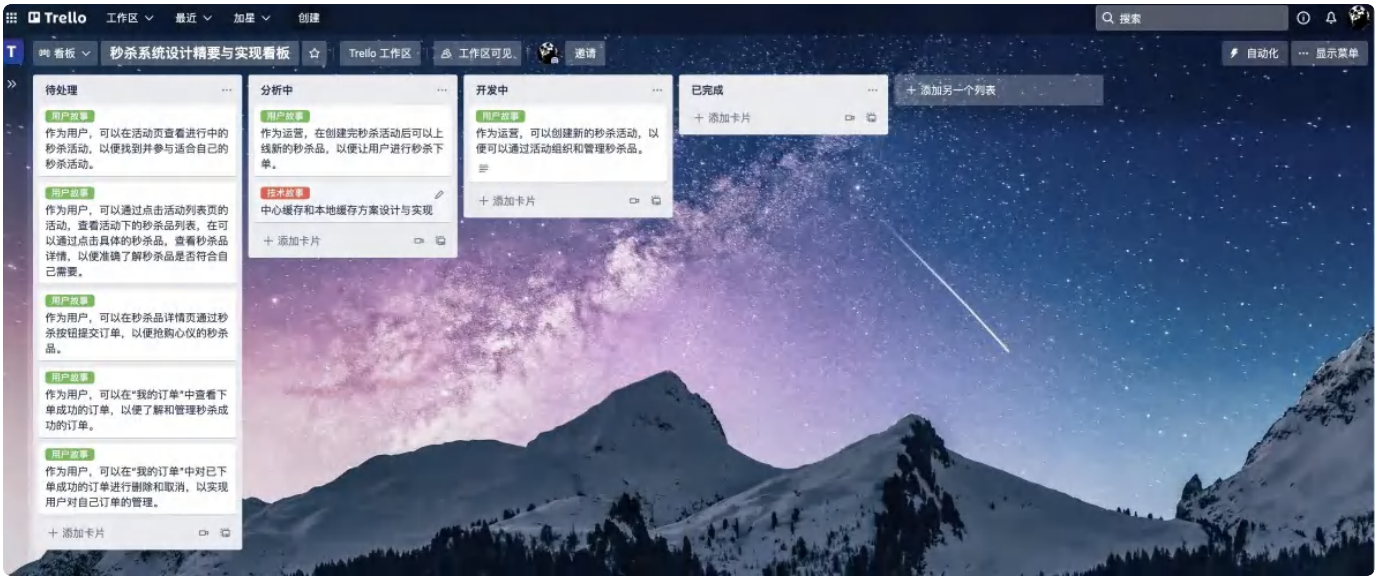
四、需求与进度管理

因部分读者反馈Trello存在国内访问困难问题，所以Trello暂停维护，直接使用Github的集成看板。

Due to the feedback from some readers that there are difficulties in accessing Trello within China, the maintenance of Trello has been suspended and the integrated kanban of Github will be used directly instead.

~~设计和实现秒杀系统是一个阶段性的持续性工作，因为我们将通过Trello中的看板来做需求和进度管理。看板方法是一个非常有趣且有用的团队协作方法，有兴趣的可以阅读书籍《看板方法》。~~

~~你可以通过访问《秒杀系统设计精要与实现看板》加入看板并浏览我们的项目概况和和当前进度，在那里你可以了解我们**已完成**、**正在进行**的和**未开始**的业务需求和技术需求。~~



小结

在本文中，我们确定了秒杀系统的业务需求和技术目标，并根据目标制定了对应的技术方案。在技术方案的选型上，我们选取了市面上主流的技术类型，并借鉴了DDD的分层架构思想。此外，在梳理技术方案之前，我们应仔细理解并梳理产品需求。

In this article, we have identified the business requirements and technical objectives of the flash sale system, and formulated corresponding technical solutions based on these objectives. In terms of the selection of technical solutions, we have chosen the mainstream technical types available on the market and drawn on the idea of the layered architecture of Domain–Driven Design (DDD). Besides, before sorting out the technical solutions, we should carefully understand and analyze the product requirements.

需要注意的是，**本文并非标准的技术方案文稿，而是为了讲清楚我们后面要做的事情，对于相关技术细节并没有展开描述**。此外，接口设计、类图设计等其他技术细节会在讲解是实现具体模块时提供。

It should be noted that this article is not a standard technical solution document. Instead, it is intended to clearly explain what we will do later, and relevant technical details are not elaborated. In addition, other technical details such as interface design and class diagram design will be provided when the specific modules are implemented during the explanation.

在下一篇文章中，我们将介绍并实现**上线秒杀活动和秒杀品的管理**。

In the next article, we will introduce and implement the management of online flash sale activities and flash sale products.

本章节到此结束，欢迎在评论区留下你的收获和疑问。对于本文未提及或未讲清楚的地方，也欢迎评论区批评指正，我会及时调整。如果你对小册有内容建议，请移步填写表单。

This chapter comes to an end. You are welcome to leave your gains and questions in the comment section. For the parts that are not mentioned or not clearly explained in this article, you are also welcome to offer criticism and corrections in the comment section. I will make adjustments in a timely manner. If you have any suggestions regarding the content of this booklet, please go to fill in the form.

第05章—开箱即用：源码获取与应用环境初始化

开箱即用：源码获取与应用环境初始化

在前面的文章中，我们已经了解秒杀架构的原理，并介绍了我们示例秒杀系统的目标。本文将为你介绍如何准备秒杀系统的环境，包括如何获取源码、源码的特性、如何部署中间件和启动应用，以及如何通过现成的测试脚本来测试应用。在进入秒杀系统复杂的的细节之前，推荐你先获取源码并最好在本地运行，以此来快速建立起对系统的整体感知。

✨***阅读提示****文章篇幅较长且大图较多，阅读时可以浏览器右键新窗口查看高清大图，或在IDE中查看图片脚本及预览；此外，还可以安装Smart TOC浏览器插件完美体验可拖拽目录阅读。*

一、源码快速上手指南

为了方便读者快速上手源码，我们制作了视频指南，请移步B站查看：
https://www.bilibili.com/video/BV1EF41187ij?share_source=copy_web

二、如何源码获取

从21年12月9日开始，小册源码已开源，面向所有读者开放。配套源码旨在帮助小册读者从源码解构高并发设计的核心要义，**你可以直接Clone、Fork以及Star，也欢迎你把它分享给周边未购买小册但可能需要它的同学、朋友和同事**。源码包含两个部分：

- **核心应用：**https://github.com/ThoughtsBeta/flash-sale
- **网关应用：**https://github.com/ThoughtsBeta/flash-sale-gateway

源码核心特性

- 基于Spring Boot和Spring Cloud的完整分布式架构应用实践；
- 本地缓存与分布式缓存的设计技巧；
- 同步下单和高并发库存扣减方式；
- 异步队列下单和库存扣减；
- 去中心化分库分表和分桶库存扣减；
- 限流的原理和应用；
- 黑白攻防和安全风控策略；
- 领域驱动设计方法与实践；
- 限流、降级与熔断落地与实践；

- 容器化技术与Swarm集群部署；
- Redis+Nacos+RocketMQ+ELK等10+中间件的应用与实践；
- 应用动态配置方法和实践；
- 分布式架构的度量与监控；
- RESTful APIs设计与体验。

此外，在获取源码之后，如果你对源码架构与设计有更好的理解，或者发现有错误的地方，欢迎提交PR.

~~由于掘金对小册的源码交由作者自行管理，而且在访问上有一定限制，在我们把FlashSale的源码托管在Github上并设置为**私有项目**之后，新读者的访问将会受限。麻烦在于，当读者购买完小册需要获取源码时，必须经由代码仓库的拥有者手动添加后才能访问。~~

~~因此，为了能让进入小册的读者可以在第一时间获取源码，我花了点时间写了个微信小程序。读者可以扫描下方的小程序码进入小程序，并在小程序中填入GItHub账号名称和邀请码（**ES350H**），即可完成**自助申请**加入仓库。在申请成功后，检查邮箱会收到如最右图所示的邀请邮件，通过邮件指引即可加入仓库并下载源码。~~

~~如果你按照流程却并未收到邀请邮件，也不要着急，可以加我微信（微信号：ThoughtsBeta）或发送邮件到~~thoughts.beta+flashsale@gmail.com~~，回复可能会稍有延迟，但会在看到消息后第一时间回复你。~~

三、 技术选型概览

在技术选型上，我们选取的都是主流技术方案，并尽量降低理解门槛。对于示例系统所使用的技术体系，如果有某些你可能未曾了解过的中间件，也不必担心，对于必要的关键技术我们会在需要的时候重点讲解，对于支撑型但非必要的技术体系可以选择性忽略。

当然，面对如此多的中间件，你也不用担心安装麻烦的问题，我们在下文提供了完整的一键安装方案。

四、如何启动并运行应用

对于FlashSale所使用到的中间件，我们提供了基于Docker-compose的完整方案，读者可以在Docker环境下一键安装。下载并打开源码之后，在项目的根目录下，你会看到enviroment目录。这个目录有四个相关的额文件和文件夹：

- **docker-compose.yml**：【完整版】中间件部署脚本。相关中间件的安装脚本，你可以通过执行`docker-compose -f docker-compose.yml up`命令安装所依赖的全部中间件；
- **docker-compose-light.yml**：【轻量版】中间件部署脚本（**本地开发推荐**），移除了非必要的中间件服务。相关中间件的安装脚本，你可以通过执行`docker-compose -f docker-compose-light.yml up`命令安装所依赖的全部中间件；
- **docker-cluster-middlewares.yml**：【集群化】的中间件部署方案，适用于Swarm网络下的集群部署，具体部署方式可以参考第15章节；
- **docker-cluster-apps.yml**：【集群化】的应用部署方案，适用于Swarm网络下的集群部署，

具体部署方式可以参考第15章节；

- **config**:相关中间件的配置文件所在位置，包括Prometheus和MYSQL等配置；
- **grafana-storage**: grafana存储的配置数据。之所以要把这个文件也提供出来，主要是因为Grafana的数据源和报表配置相对比较麻烦，自行配置可能需要倒腾好一阵子，我们提供出来便可以直接加载使用；
- **data**:部分中间件的数据存储位置，包括MYSQL等。这个目录的数据是由中间件系统运行时产生，数据多且杂乱，我们并没有把它放在git中，因此你下载源码后不会看到它，但是运行时就会出现。那为什么要把它放在这个位置，而不是计算机系统的其他位置？这个主要是为了方便数据查看和管理，你可以随时清除所有数据重新来过，当然也可以把它放在任何地方。

第一步：启动中间件

1. 下载源码后进入`environment`目录，执行`docker-compose -f docker-compose-light.yml up`启动中间件；
2. 如果你对Docker命令不熟悉，建议安装Docker Desktop简化容器管理，可以直观看到容器的启动状态和日志输出；
3. 需要停止所有容器时，请执行`docker-compose -f docker-compose-light.yml down`；
4. 需要重新创建所有容器时，请执行`docker-compose -f docker-compose-light.yml up --force-recreate`.

关于数据库中库表的初始化

对于FlashSale所使用的业务表，我们已经将初始化脚本放在`enviroment/config/mysql` 中，**docker-compose在安装完MYSQL之后，便会执行数据表初始化动作，实现数据库的开箱即用。**

Plain Text

```
.
├── config
│   └── mysql
│       └── init
│           ├── flash_sale_init.sql // 默认主库初始化语句
│           ├── flash_sale_init_sharding_0.sql // #0号数据库初始化语句
│           ├── flash_sale_init_sharding_1.sql // #1号数据库初始化语句
│           └── nacos_init.sql //Nacos持久化语句
├── docker-cluster-apps.yml
├── docker-cluster-middlewares.yml
├── docker-compose-light.yml
└── docker-compose.yml
```

需要稍微注意的时，我们为MYSQL提供了两份初始化脚本：flash_sale_init.sql和nacos_init.yml，前者是业务表初始化脚本，后者是Nacos的初始化脚本，因为FlashSale需要借助Nacos所提供的动态配置功能，但是Nacos默认是内存存储，所以我们为它实现了基于MYSQL的持久化存储方案。

第二步：通过IDE启动应用运行

在启用应用前，请务必确保已成功执行第一步，并且各中间件容器启动成功。

1. 下载源码后执行./mvnw clean install完成系统依赖包的安装；
2. 选择start模块中的com.actionworks.flashsale.FlashSaleApplication作为程序入口运行。

***特别提醒**本地启动时请在IDE中指定properties为local.*

在调试阶段，推荐使用这种方式。FlashSale启动时，将会连接到前面所安装的中间件。

可选：通过Docker启动运行

除了在IDE启动FlashSale之外，通过Docker启动也是一种非常便捷的方案。

1. 通过下面的命令构建FlashSale本地镜像：

```
Plain Text
1  docker build -t flash-sale-app .
```

构建完成后，通过docker images查看镜像是否已经存在。

1. 将下面的配置添加到前面所说的docker-compose.yml中，在运行中间的时候，也将同时启动系统。当然，我们也可以通过独立的文件运行。需要注意的是，在通过docker运行时，FlashSale将和中间件共处同一个网络中，我们为此创建了独立的配置文件，在运行时需要指定docker 配置。

```
services:
  flash-sale-app:
    image: flash-sale-app
    container_name: flash-sale-app
    environment:
      - SPRING_PROFILES_ACTIVE=docker
    ports:
      - 8090:8090
    networks:
      - thoughts-beta-cluster-apps
    restart: on-failure
  flash-sale-gateway:
    image: flash-sale-gateway
    container_name: flash-sale-gateway
    environment:
      - SPRING_PROFILES_ACTIVE=docker
    ports:
      - 8080:8080
    networks:
      - thoughts-beta-cluster-apps
    restart: on-failure
```

五、 如何测试接口

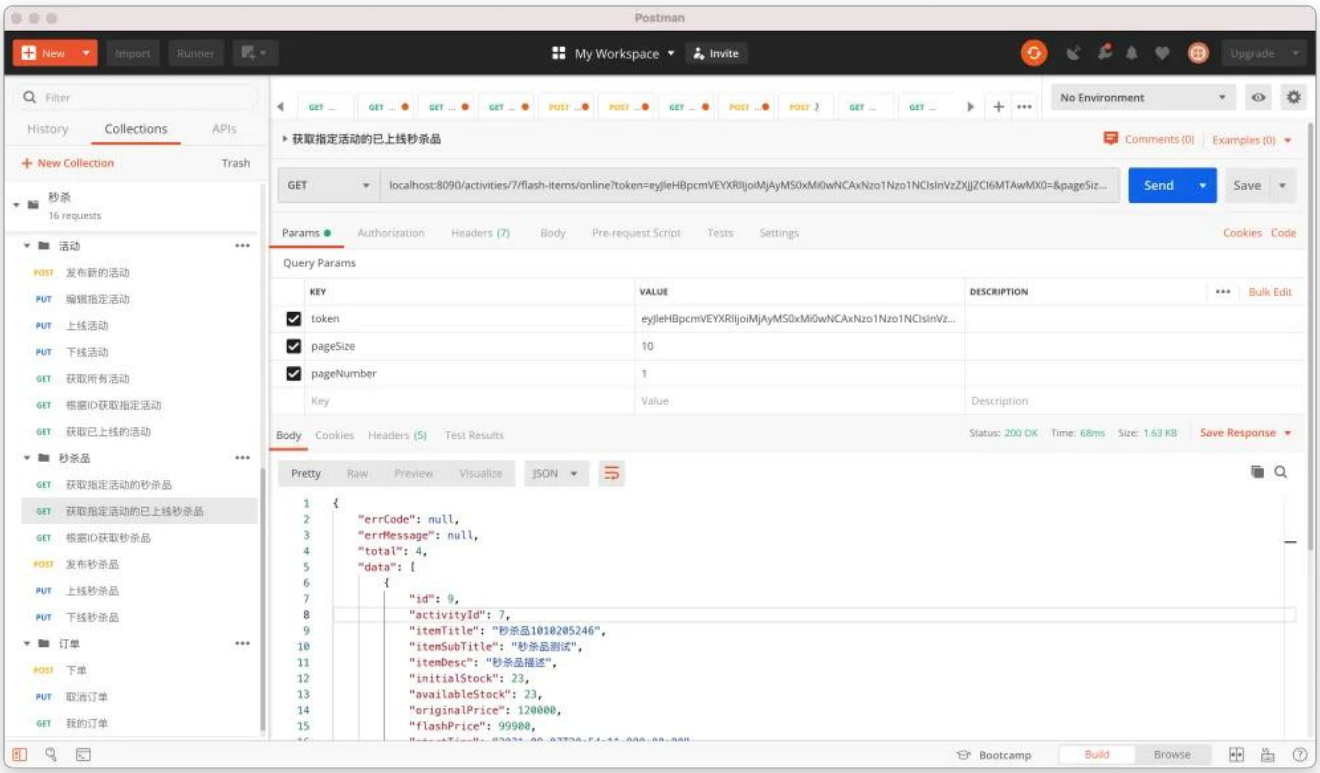
在完成中间件的安装和初始化，并启动应用后，接下来我们可以试着测试接口，来判断中间件和应用是否已经就绪并工作正常。

同样的，我们不会让读者自己创建脚本和准备测试数据，毕竟这不符合我们**读者至上**和**开箱即用**的原则。为此，在项目的根目录下，你会看到我们提供的`postman`目录，它是Postman的测试脚本，包含了接口定义和测试数据，你可以直接选择某个接口点击测试即可。

Postman的脚本位置如下所示：

```
├── environment
│   ├── config
│   ├── data
│   ├── docker-compose.yml
│   └── grafana-storage
└── postman
    └── flash-sale-postman.json # 测试脚本
```

FlashSale的测试脚本如下图所示。在脚本中，我们将脚本分为了单机测试和集群测试两部分，在本地开发模式下你可以选择单机测试API集合。另外，关于压测内容我们在此暂不详述，它相对独立且更为复杂，我们会开辟单独章节讲述。



六、如何获取高清图片

对于小册中的时序图、类图等图片，如果你在阅读时觉得不够清晰地话，你可以选择两种方案：

1. **插入的图片都是高清，可以在图片上点击右键到新窗口中打开图片；**
2. **时序图等图片由PlantUML生成，相关脚本已经加入到源码的diagram目录下，IDE安装PlantUML插件即可随时查看。**

需要注意的是，查看PlantUML插件使用时需要安装graphviz，如果你的电脑未安装，可以参考这篇文章。

小结

以上就是本文的全部内容。在本文中，我们主要帮助你获取源码并完成中间件以及如何启动系统并完成基本调试，如果有细节需要我们补充以及其他建议，欢迎在评论区指出，我会持续优化相关内容。

在下一章节中，我们将讲解秒杀活动和秒杀品相关的关键设计与源码实现。

本章节到此结束，欢迎在评论区留下你的收获和疑问。对于本文未提及或未讲清楚的地方，也欢迎评论区批评指正，我会及时调整。如果你对小册有内容建议，请移步填写表单。

第06章—领域驱动：核心领域设计

在上一章节中，我们介绍了如何构建FlashSale的系统环境并启动应用以完成基本测试，相信你对系统已经有了整体印象。从本文开始，我们将对系统的关键模块进行细致的讲解，首先讲解的是秒杀活动和秒杀品相关的设计和实现。

对于秒杀系统而言，秒杀活动和秒杀品的设计是不可或缺的重要基础设计。对于这两部分，我们将从用例出发，逐步介绍包括领域设计、代码分层设计、数据库设计、接口设计和其他设计等相关内容。

✨*__阅读提示__文章篇幅较长且大图较多，阅读时可以浏览器右键新窗口查看高清大图，或在IDE中查看图片脚本及预览；此外，还可以安装Smart TOC浏览器插件完美体验可拖拽目录阅读，源码下载请参考第五章节。*

一、用例梳理

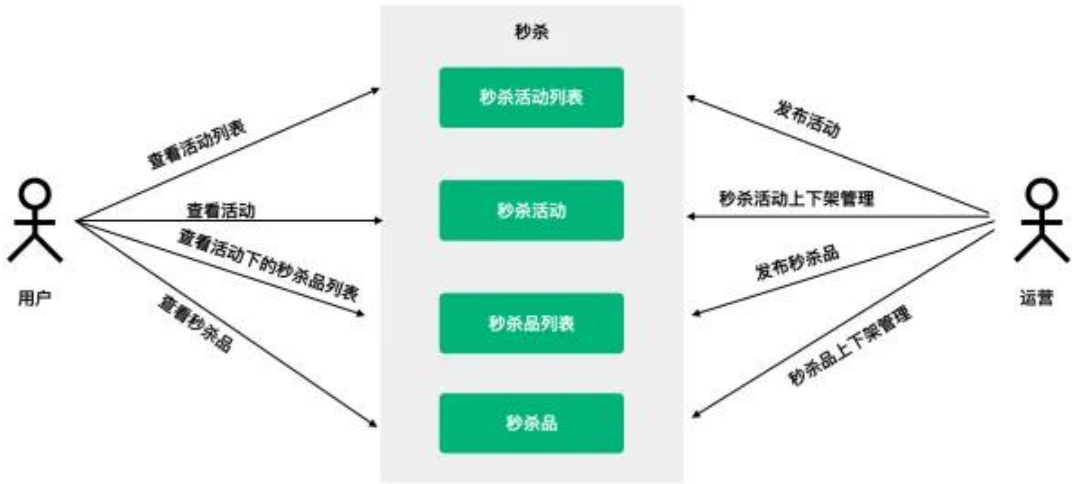
在FlashSale中，我们提取了秒杀活动和秒杀品相关的几个重要的用例。当然，这些用例并不是全部的用例，在真实的系统中，我们可能还需要更丰富的其他用例，比如上下架审核流程和相关的规则配置。这些用例如下图所示，这些用例包括：

运营侧

- 发布活动
- 秒杀活动上下架管理
- 发布秒杀品
- 秒杀品上下架管理

用户侧

- 查看已上线的活动列表
- 查看活动
- 查看活动下的秒杀品列表
- 查看秒杀品



二、领域设计

在领域设计中，我们将定义秒杀活动和秒杀品的模型、领域服务以及领域事件。**领域设计是后续设计的基础，领域设计中定义的通用语言应当作为系统的指导设计语言，对于同一概念而言，应当使用本设计中定义的名称，不要创建其他名称。**

此外，领域设计中所定义的模型和领域服务应具有较高的抽象性和稳定性，它们面向领域而设计并定义，不会随着用例的变化而变化。

（一）秒杀活动领域设计

秒杀活动

秒杀活动是秒杀品的载体，一个秒杀活动可以包含多个秒杀品，并且多个秒杀活动可以同时存在。

领域模型

- 活动名称：通用名称：activityName. 比如“圣诞节限时秒杀”；

- 活动开始时间和结束时间：通用名称：startTime与endTime. 在活动期间内，用户才可以进行秒杀，用户下单时将进行活动时间校验；
- 活动状态：通用名称：status. 分为发布、上线和下线等。只有处于上线状态的活动才对C端用户透出，也只有上线状态的活动才能进行秒杀下单；
- 活动描述：通用名称：activityDesc. 可选，可为活动添加丰富的图文描述。

领域服务

- 发布秒杀活动：当运营侧需要新的秒杀活动时，可以发布新的秒杀活动。秒杀活动发布后，将不对C端用户透出；
- 上线活动：当运营侧已确认活动可以上线时，可以对活动执行上线操作。活动上线后，将对C端透出；
- 下线活动：当活动不需要继续进行时，可以对活动执行下线操作。活动下线后，C端将不可见，同时将不再接收新的秒杀订单；
- 秒杀活动查询和获取：运营侧和C端用户侧均可以获取秒杀活动集合。不同的是，用户侧仅能看见已上线的活动；
- 秒杀活动下单条件检查：当用户执行秒杀下单时，秒杀活动可以根据此前配置的规则执行下单前置校验，已确认当前活动是否允许下单。

领域事件

- 秒杀活动上线：当秒杀活动被执行上线操作时，将发出对应的上线事件，以通知相关订阅者处理；
- 秒杀活动下线：当秒杀活动被执行下线操作时，将发出对应的下线事件，以通知相关订阅者处理。

（二）秒杀品领域设计

秒杀品：可上架进行售卖和预定的商品或其他物品。

领域模型

- 所属活动：通用名称：activityId. 秒杀品不可以独立存在，需要关联到秒杀活动；
- 秒杀品标题：通用名称：itemTitle. 秒杀品的主标题；
- 秒杀品副标题：通用名称：itemSubTitle. 可选，可用于对主标题的补充；
- 初始库存：通用名称：initialStock. 秒杀品的原始库存，该值在秒杀活动期间不会发生变化；
- 可用库存：通用名称：availableStock. 秒杀品当前可用库存，该值在初始时与”初始库存“的数值一致，库存扣减将发生该值上。因此，该值会随着用户下单二不断变化，并且是高并发的主要发生字段；
- 原价：通用名称：originalPrice. 秒杀品在其他渠道的售卖价格，该值在活动期间不会发生变化，仅用于信息展示；
- 秒杀价：通用名称：flashPrice. 秒杀品的秒杀价格，该值与”原价“可能不同，并参与到价格计算中；
- 秒杀开始和结束时间：通用名称：startTime与endTime. 当系统时间出于startTime和endTime之间时，秒杀品出于活动中状态，可进行秒杀下单。超过该时间范围后，将不再接收用户下单；
- 秒杀品状态：通用名称：status. 当前秒杀品的状态，存在发布、上线和下线等不同状态，仅处于”上线“状态的秒杀品才会对用户侧透出并接收下单。

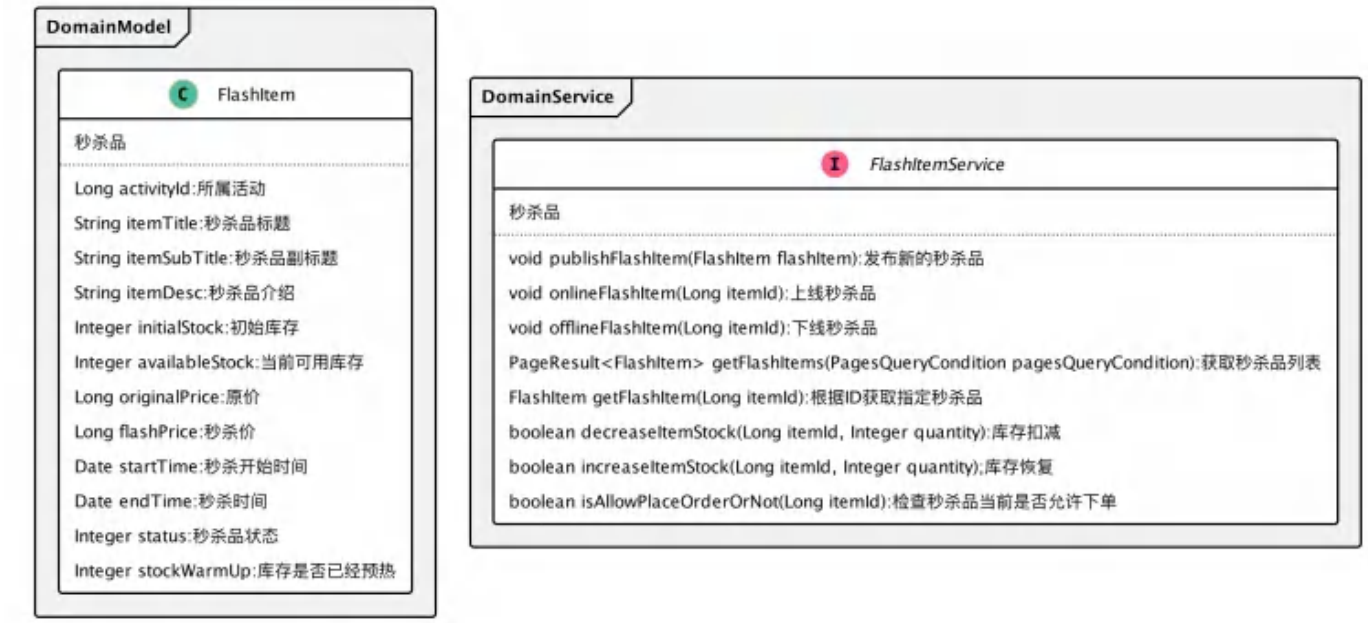
领域服务

- 发布秒杀品：当需要发布新的秒杀品，可以执行秒杀品发布操作；
- 上线秒杀品：当确认秒杀品可以上线时，可对其执行上线操作，并在上线后对用户侧透出；
- 下线秒杀品：当秒杀品不再需要继续上线时，可以对其执行下线操作；
- 获取秒杀品及列表：对运营侧和用户侧提供秒杀品获取能力，用户侧仅能获取已上线的秒杀品和秒杀活动；
- 库存扣减：当用户执行下单订单操作时，将进行库存扣减，库存扣减失败时将执行事务回滚；
- 库存恢复：当用户执行取消订单等相关操作时，将进行库存恢复，只有库存恢复成功时，其他操作才能认定成功或继续操作，否则将执行事务回滚；
- 秒杀品下单条件检查：当用户执行秒杀下单时，秒杀品可以根据此前配置的规则执行下单前置校验，已确认当前秒杀品是否允许下单。比如，业务规定同一用户仅能下单一次，那么当用户

再次下单时即可拒绝；

领域事件

- 上线秒杀品：当秒杀品被执行上线操作时，将发出对应的上线事件，以通知相关订阅者处理；
- 下线秒杀品：当秒杀品被执行下线操作时，将发出对应的下线事件，以通知相关订阅者处理。



三、代码分层设计

如前序章节所述，FlashSale的源码在分层具有以下几层，具体各层职责可参考第3章节：

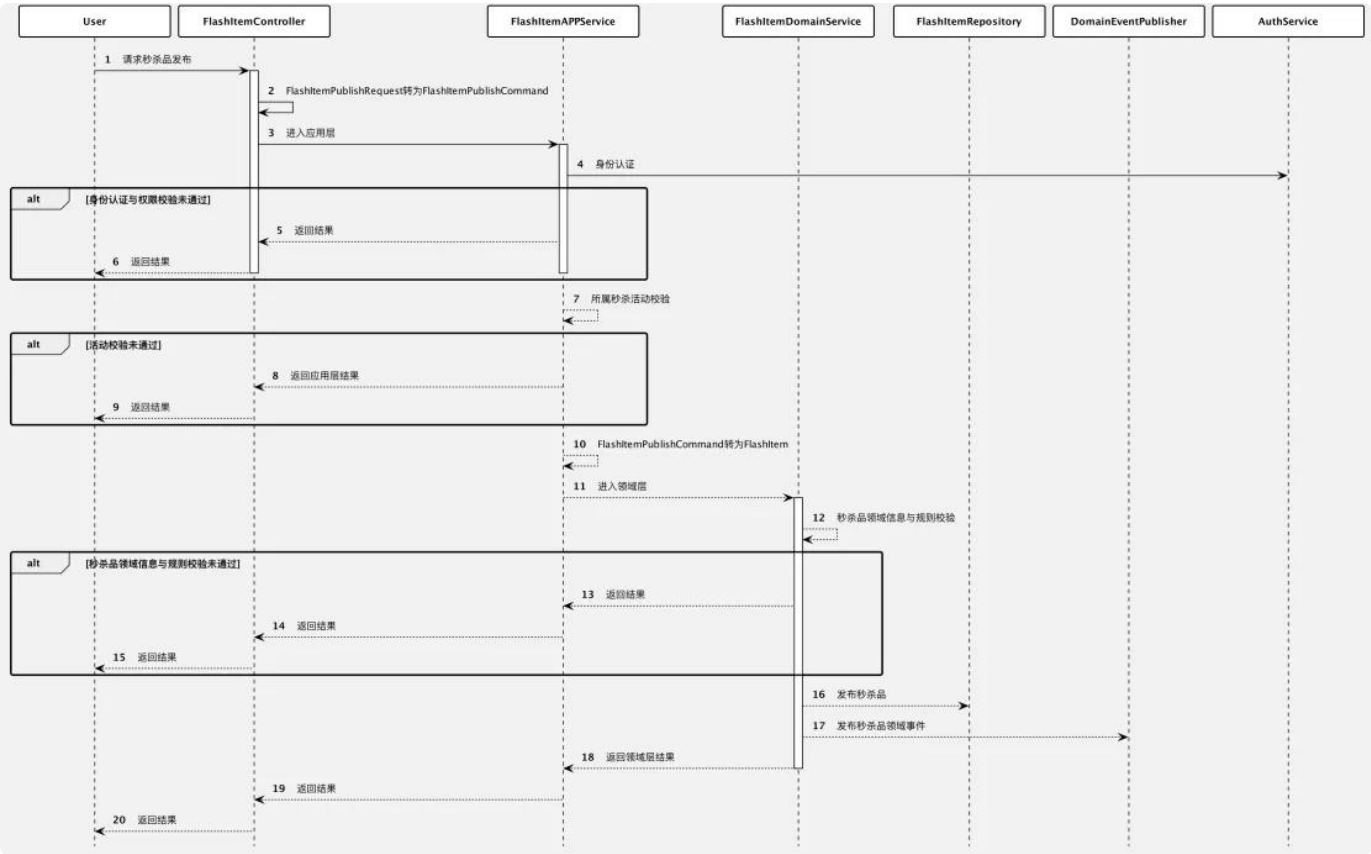
- flash-sale-controller
- flash-sale-app
- flash-sale-domain
- flash-sale-infrastructure

这里需要说明的是，在FlashSale中我们将严格遵循分层职责定义组织代码逻辑，对代码中的职责范围和对象使用都有明确的范围界定。换句话说，我们将通过**限界上下文**的方式组织代码，以保证各模块的纯粹和干净。比如，FlashItemPublishRequest对象只能在Controller层使用，需要传输数据到其他层时，则需要将数据转换为目标层所提供的对象。另外，诸如缓存处理、事件处理和调度等，我们将严格限制在应用层，而不会让它们蔓延或扩散到其他层。

对于FlashSale中的秒杀活动和秒杀品的代码分层设计和时序图，我们在这里不会一一列举，那样篇幅会比较大，所以我们选取了**发布秒杀品**、**上线秒杀品**和**获取秒杀品**作为典型样例供你参考领

会。更多细节和代码，请下载源码阅读。

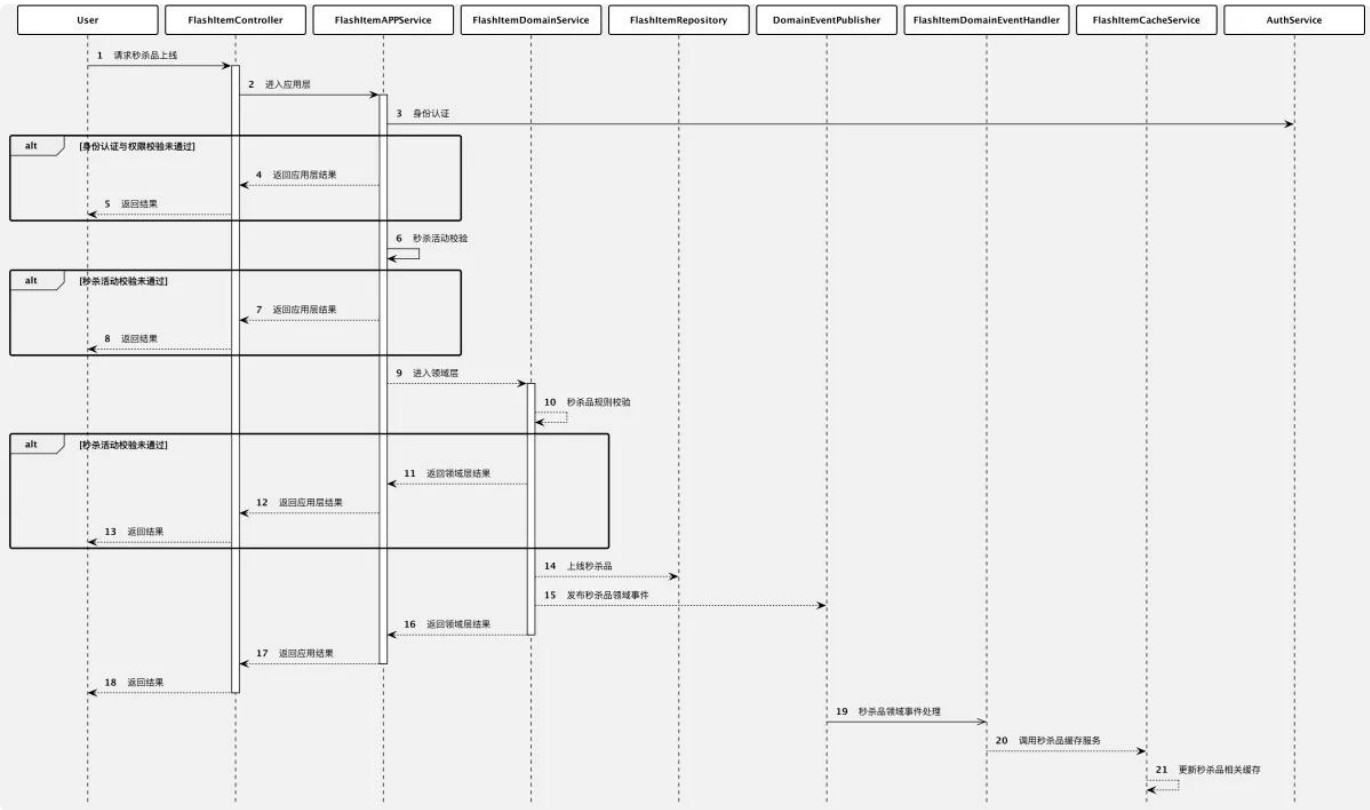
（一）发布秒杀品



应用层源码如下：

```
@Override
public AppResult publishFlashItem(Long userId, Long activityId, FlashItemPublishCommand itemPublishCommand) {
    logger.info("itemPublish|发布秒杀品|{},{},{}", userId, activityId, JSON.toJSON(itemPublishCommand));
    if (userId == null || activityId == null || itemPublishCommand == null || !itemPublishCommand.validate()) {
        throw new BizException(INVALID_PARAMS);
    }
    AuthResult authResult = authorizationService.auth(userId, FLASH_ITEM_CREATE);
    if (!authResult.isSuccess()) {
        throw new AuthException(UNAUTHORIZED_ACCESS);
    }
    DistributedLock itemCreateLock = lockFactoryService.getDistributedLock(getItemCreateLockKey(userId));
    try {
        boolean isLockSuccess = itemCreateLock.tryLock(500, 1000, TimeUnit.MILLISECONDS);
        if (!isLockSuccess) {
            throw new BizException(FREQUENTLY_ERROR);
        }
        FlashActivity flashActivity = flashActivityDomainService.getFlashActivity(activityId);
        if (flashActivity == null) {
            throw new BizException(ACTIVITY_NOT_FOUND);
        }
        FlashItem flashItem = toDomain(itemPublishCommand);
        flashItem.setActivityId(activityId);
        flashItem.setStockWarmUp(0);
        flashItemDomainService.publishFlashItem(flashItem);
        logger.info("itemPublish|秒杀品已发布");
        return AppResult.buildSuccess();
    } catch (Exception e) {
        logger.error("itemPublish|秒杀品发布失败|{}", userId, e);
        throw new BizException("秒杀品发布失败");
    } finally {
        itemCreateLock.unlock();
    }
}
```

（二）上线秒杀品

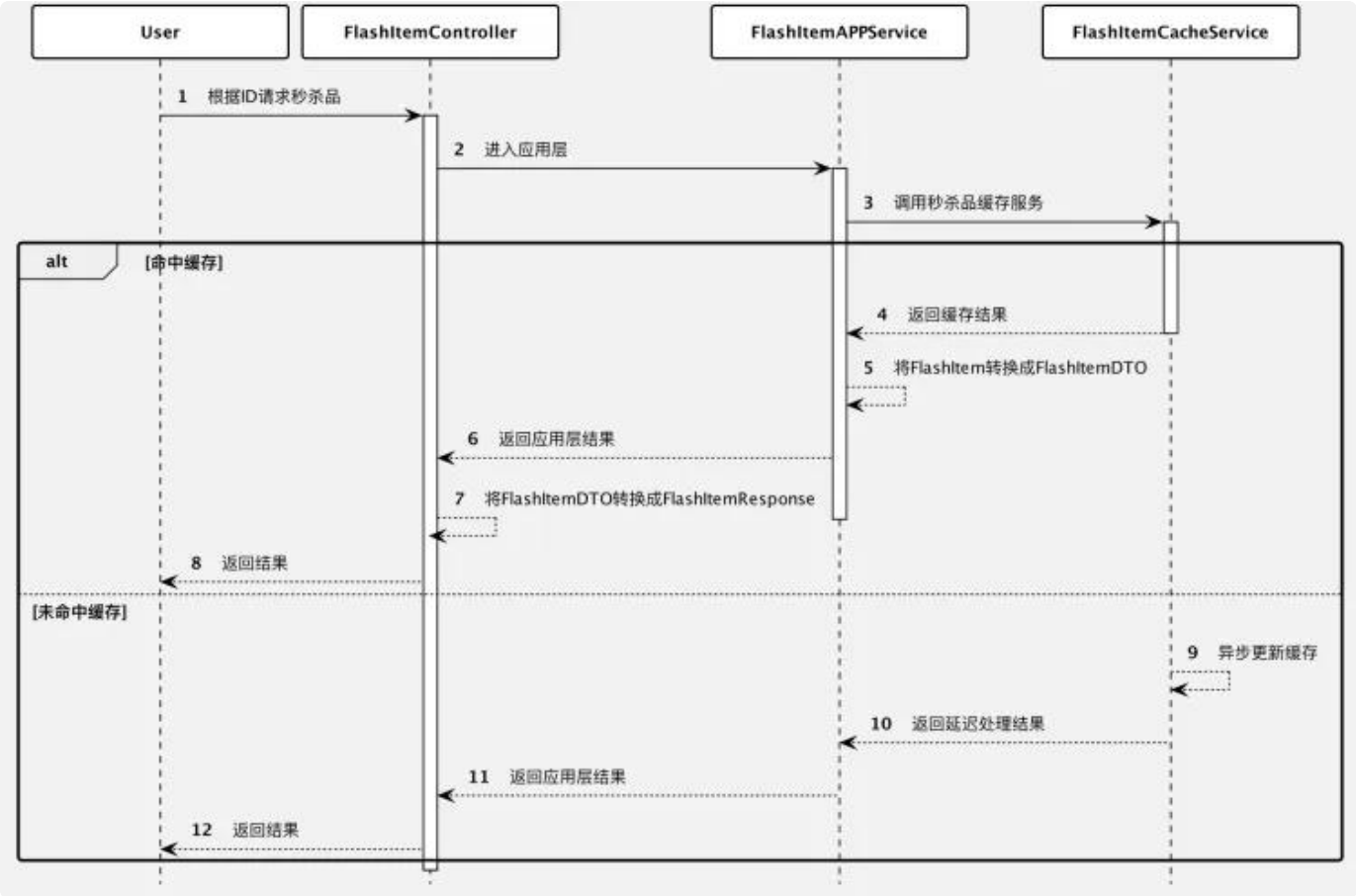

User
FlashItemController
FlashItemAPPService
FlashItemDomainService
FlashItemRepository
DomainEventPublisher
FlashItemDomainEventHandler
FlashItemCacheService
AuthService
1 请求秒杀品上线
2 进入应用层
3 身份认证
alt [身份认证与权限校验未通过]
4 返回应用层结果
5 返回结果
6 秒杀活动校验
alt [秒杀活动校验未通过]
7 返回应用层结果
8 返回结果
9 进入领域层
10 秒杀品规则校验
alt [秒杀活动校验未通过]
11 返回领域层结果
12 返回应用层结果
13 返回结果
14 上线秒杀品
15 发布秒杀品领域事件
16 返回领域层结果
17 返回应用结果
18 返回结果
19 秒杀品领域事件处理
20 调用秒杀品缓存服务
21 更新秒杀品相关缓存

应用层源码如下：

Plain Text

```
@Override
public AppResult onlineFlashItem(Long userId, Long activityId, Long itemId) {
    logger.info("itemOnline|上线秒杀品|{},{},{}", userId, activityId, itemId);
    if (userId == null || activityId == null || itemId == null) {
        throw new BizException(INVALID_PARAMS);
    }
    AuthResult authResult = authorizationService.auth(userId, FLASH_ITEM_MODIFICATION);
    if (!authResult.isSuccess()) {
        throw new AuthException(UNAUTHORIZED_ACCESS);
    }
    DistributedLock itemModificationLock = lockFactoryService.getDistributedLock(getItemModificationLockKey(userId));
    try {
        boolean isLockSuccess = itemModificationLock.tryLock(500, 1000, TimeUnit.MILLISECONDS);
        if (!isLockSuccess) {
            throw new BizException(LOCK_FAILED_ERROR);
        }
        flashItemDomainService.onlineFlashItem(itemId);
        logger.info("itemOnline|秒杀品已上线");
        return AppResult.buildSuccess();
    } catch (Exception e) {
        logger.error("itemOnline|秒杀品已上线失败|{}", userId, e);
        throw new BizException("秒杀品已上线失败");
    } finally {
        itemModificationLock.unlock();
    }
}
```

（三） 获取秒杀品


User
FlashItemController
FlashItemAPPService
FlashItemCacheService
1 根据ID请求秒杀品
2 进入应用层
3 调用秒杀品缓存服务
alt
[命中缓存]
4 返回缓存结果
5 将FlashItem转换成FlashItemDTO
6 返回应用层结果
7 将FlashItemDTO转换成FlashItemResponse
8 返回结果
[未命中缓存]
9 异步更新缓存
10 返回延迟处理结果
11 返回应用层结果
12 返回结果

应用层源码如下：

```
Plain Text
```

```
@Override
public AppSingleResult < FlashItemDTO > getFlashItem(String token, Long activityId, Long itemId, Long version) {
    AuthResult authResult = authorizationService.auth(token, FLASH_ITEMS_GET);
    if (!authResult.isSuccess()) {
        throw new AuthException(INVALID_TOKEN);
    }

    FlashItemCache flashItemCache = flashItemCacheService.getCachedItem(itemId, version);
    if (!flashItemCache.isExist()) {
        throw new BizException(ACTIVITY_NOT_FOUND.getErrDesc());
    }
    if (flashItemCache.isLater()) {
        return AppSingleResult.tryLater();
    }
    updateLatestItemStock(flashItemCache.getFlashItem());
    FlashItemDTO flashItemDTO = FlashItemAppBuilder.toFlashItemDTO(flashItemCache.getFlashItem());
    flashItemDTO.setVersion(flashItemCache.getVersion());
    return AppSingleResult.of(flashItemDTO);
}
```

在获取秒杀品的源码和时序图中，你可能会发现请求只会到达应用层。没错，对于高并发的读取请求，我们会将其阻止在应用层，并在应用通过缓存等方式实现数据获取，而不会让请求进入领域层。

四、数据库模型设计

秒杀活动和秒杀品的数据库设计，在前序章节中我们已经提过，为了方便你沉浸式阅读，我们在此再次提供。

- 秒杀活动表设计

Plain Text

```
CREATE TABLE IF NOT EXISTS flash_sale.`flash_activity` (
  `id`            bigint(20)  NOT NULL AUTO_INCREMENT
  COMMENT '主键',
  `activity_name` varchar(50) NOT NULL
  COMMENT '秒杀活动名称',
  `activity_desc` text COMMENT '秒杀活动描述',
  `start_time`    datetime    NOT NULL
  COMMENT '秒杀活动开始时间',
  `end_time`      datetime    NOT NULL
  COMMENT '秒杀活动结束时间',
  `status`        int(11)     NOT NULL DEFAULT '0'
  COMMENT '秒杀活动状态',
  `modified_time` datetime    NOT NULL DEFAULT CURRENT_TIMESTAMP
  COMMENT '更新时间',
  `create_time`   datetime    NOT NULL DEFAULT CURRENT_TIMESTAMP
  COMMENT '创建时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `flash_activity_id_uk` (`id`),
  KEY `flash_activity_end_time_idx` (`end_time`),
  KEY `flash_activity_start_time_idx` (`start_time`),
  KEY `flash_activity_status_idx` (`status`)
)
  ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4
  COMMENT = '秒杀活动表';
```

- 秒杀品表设计

```
CREATE TABLE IF NOT EXISTS flash_sale.`flash_item` (
  `id`              bigint(20)  NOT NULL AUTO_INCREMENT
  COMMENT '主键',
  `item_title`      varchar(50) NOT NULL
  COMMENT '秒杀品名称标题',
  `item_sub_title`  varchar(50) NULL
  COMMENT '秒杀品副标题',
  `item_desc`       text COMMENT '秒杀品介绍富文本文案',
  `initial_stock`   int(11)     NOT NULL DEFAULT '0'
  COMMENT '秒杀品初始库存',
  `available_stock` int(11)     NOT NULL DEFAULT '0'
  COMMENT '秒杀品可用库存',
  `stock_warm_up`    int(11)     NOT NULL DEFAULT '0'
  COMMENT '秒杀品库存是否已经预热',
  `original_price`  bigint(20)  NOT NULL
  COMMENT '秒杀品原价',
  `flash_price`     bigint(20)  NOT NULL
  COMMENT '秒杀价',
  `start_time`      datetime    NOT NULL
  COMMENT '秒杀开始时间',
  `end_time`        datetime    NOT NULL
  COMMENT '秒杀结束时间',
  `rules`           text COMMENT '秒杀可配规则，JSON格式',
  `status`          int(11)     NOT NULL DEFAULT '0'
  COMMENT '秒杀品状态',
  `activity_id`     bigint(20)  NOT NULL
  COMMENT '所属活动id',
  `modified_time`   datetime    NOT NULL DEFAULT CURRENT_TIMESTAMP
  COMMENT '更新时间',
  `create_time`     datetime    NOT NULL DEFAULT CURRENT_TIMESTAMP
  COMMENT '创建时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `flash_item_id_uk` (`id`),
  KEY `flash_item_end_time_idx` (`end_time`),
  KEY `flash_item_start_time_idx` (`start_time`),
  KEY `flash_item_status_idx` (`status`)
)
  ENGINE = InnoDB
  DEFAULT CHARSET = utf8mb4
  COMMENT = '秒杀品';
```

五、接口定义与设计

在接口定义与设计时，我们遵循了RESTful的设计风格。通过领域概念提取出相应的**资源**，而在对资源操作时也遵循了HTTP的请求方法，比如读取资源时使用GET，创建资源时使用POST，而在销毁资源时则使用DELETE. 在URL设计上，也尽量使用资源相关的名词或动名词，并尽可能体验资源的嵌套和从属关系。

遵循良好的RESTful的设计风格，可以让我们的接口更为优雅直观，且具有更为宽泛的可理解性。

在本文所呈现的接口定义中，我们尽可能提供全面、完整的信息，并提供了Controller层的源码，希望可以让你有一目了然的理解。此外，相关接口的测试可以参考第4章节。

（一） 秒杀活动接口设计

1. 上线活动

请求路径:

```
Plain Text
Method: PUT
URL: localhost:8090/flash-activities/1/online
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

源码实现:

```
Plain Text
@PutMapping(value = "/flash-activities/{activityId}/online")
public Response onlineFlashActivity(@RequestParam String token, @PathVariable Long activityId) {
    AppResult appResult = flashActivityAppService.onlineFlashActivity(token, activityId);
    return ResponseBuilder.with(appResult);
}
```

2. 下线活动

请求路径:

```
Method: PUT
URL: localhost:8090/flash-activities/1/offline
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

源码实现:

```
@PutMapping(value = "/flash-activities/{activityId}/offline")
public Response offlineFlashActivity(@RequestParam String token, @PathVariable Long activityId) {
    AppResult appResult = flashActivityAppService.offlineFlashActivity(token, activityId);
    return ResponseBuilder.with(appResult);
}
```

3. 发布新的活动

请求路径:

```
Method: POST
URL: localhost:8090/flash-activities
```

请求头设置:

KEY	VALUE	DESCRIPTION
Content-Type	application/json	

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

请求体设置:

Plain Text

```
{
    "activityName": "示例活动{{currentTime}}",
    "activityDesc": "国庆节活动描述。。。",
    "startTime": "2021-09-07 20:54:11",
    "endTime": "2021-12-09 20:54:21"
}
```

源码实现:

Plain Text

```
@PostMapping(value = "/flash-activities")
public Response publishFlashActivity(@RequestParam String token, @RequestBody FlashActivityPublishRequest flashActivityPublishRequest) {
    FlashActivityPublishCommand activityPublishCommand = FlashActivityBuilder.toCommand(flashActivityPublishRequest);
    AppResult appResult = flashActivityAppService.publishFlashActivity(token, activityPublishCommand);
    return ResponseBuilder.with(appResult);
}
```

4. 根据ID获取指定活动

请求路径:

Plain Text

```
Method: GET
URL: localhost:8090/flash-activities/1
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

5. 编辑指定活动

请求路径:

Plain Text

```
Method: PUT
URL: localhost:8090/flash-activities/1
```

请求头设置:

KEY	VALUE	DESCRIPTION
Content-Type	application/json	

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

请求体设置:

```
{
    "activityName": "示例活动{{currentTime}}",
    "activityDesc": "国庆节活动描述。。。",
    "startTime": "2021-09-07 20:54:11",
    "endTime": "2021-12-09 20:54:21"
}
```

源码实现:

```
@PutMapping(value = "/flash-activities/{activityId}")
public Response modifyFlashActivity(@RequestParam String token, @PathVariable Long activityId, @RequestBody FlashActivityPublishRequest flashActivityPublishRequest) {
    FlashActivityPublishCommand activityPublishCommand = FlashActivityBuilder.toCommand(flashActivityPublishRequest);
    AppResult appResult = flashActivityAppService.modifyFlashActivity(token, activityId, activityPublishCommand);
    return ResponseBuilder.with(appResult);
}
```

6. 获取已上线的活动

请求路径:

```
Method: GET
URL: localhost:8090/flash-activities/online
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	
pageSize	10	

pageNumber	1	

源码实现:

```
Plain Text
```

```
@GetMapping(value = "/flash-activities/online")
@SentinelResource("GetOnlineActivitiesResource")
public MultiResponse < FlashActivityResponse > getOnlineFlashActivities(@RequestParam String token,
    @RequestParam Integer pageSize,
    @RequestParam Integer pageNumber,
    @RequestParam(required = false) String keyword) {
    FlashActivitiesQuery flashActivitiesQuery = new FlashActivitiesQuery()
        .setKeyword(keyword)
        .setPageSize(pageSize)
        .setPageNumber(pageNumber)
        .setStatus(FlashActivityStatus.ONLINE.getCode());

    AppMultiResult < FlashActivityDTO > flashActivitiesResult = flashActivityAppService.getFlashActivities(token, flashActivitiesQuery);
    if (!flashActivitiesResult.isSuccess() || flashActivitiesResult.getData() == null) {
        return ResponseBuilder.withMulti(flashActivitiesResult);
    }
    return MultiResponse.of(toFlashActivitiesResponse(flashActivitiesResult.getData()), flashActivitiesResult.getTotal());
}
```

7. 获取所有活动

请求路径:

```
Plain Text
```

```
Method: GET
URL: localhost:8090/flash-activities
```

请求参数:

KEY	VALUE	DESCRIPTION

token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	
pageSize	10	
pageNumber	1	

源码实现:

```
Plain Text
```

```
@GetMapping(value = "/flash-activities")
@SentinelResource("GetActivitiesResource")
public MultiResponse < FlashActivityResponse > getFlashActivities(@RequestParam String token,
    @RequestParam Integer pageSize,
    @RequestParam Integer pageNumber,
    @RequestParam(required = false) String keyword) {
    FlashActivitiesQuery flashActivitiesQuery = new FlashActivitiesQuery()
        .setKeyword(keyword)
        .setPageSize(pageSize)
        .setPageNumber(pageNumber);

    AppMultiResult < FlashActivityDTO > flashActivitiesResult = flashActivityAppService.getFlashActivities(token, flashActivitiesQuery);
    return ResponseBuilder.withMulti(flashActivitiesResult);
}
```

（二） 秒杀品接口设计

1. 上线秒杀品

请求路径:

```
Plain Text
```

```
Method: PUT
URL: localhost:8090/activities/1/flash-items/9/online
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

源码实现:

```
Plain Text
@PutMapping(value = "/activities/{activityId}/flash-items/{itemId}/online")
public Response onlineFlashItem(@RequestParam String token, @PathVariable Long activityId, @PathVariable Long itemId) {
    AppResult onlineResult = flashItemAppService.onlineFlashItem(token, activityId, itemId);
    return ResponseBuilder.with(onlineResult);
}
```

2. 下线秒杀品

请求路径:

```
Plain Text
Method: PUT
URL: localhost:8090/activities/1/flash-items/2/offline
```

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

源码实现:

```
Plain Text
```

```
@PutMapping(value = "/activities/{activityId}/flash-items/{itemId}/offline")
public Response offlineFlashItem(@RequestParam String token, @PathVariable Long activityId, @PathVariable Long itemId) {
    AppResult onlineResult = flashItemAppService.onlineFlashItem(token, activityId, itemId);
    return ResponseBuilder.with(onlineResult);
}
```

3. 发布秒杀品

请求路径:

```
Plain Text
```

```
Method: POST
URL: localhost:8090/activities/1/flash-items
```

请求头设置:

KEY	VALUE	DESCRIPTION
Content-Type	application/json	

请求参数:

KEY	VALUE	DESCRIPTION
token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

请求体设置:

```
{
    "itemTitle": "秒杀品{{currentTime}}",
    "itemSubTitle": "秒杀品测试",
    "itemDesc": "秒杀品描述",
    "itemPrice": 1999,
    "initialStock": 23,
    "availableStock":23,
    "originalPrice":120000,
    "flashPrice":99900,
    "startTime": "2021-09-07 20:54:11",
    "endTime": "2021-12-09 20:54:21"
}
```

源码实现:

```
@PostMapping(value = "/activities/{activityId}/flash-items")
public Response publishFlashItem(@RequestParam String token, @PathVariable Long activityId, @RequestBody FlashItemPublishRequest flashItemPublishRequest) {
    AppResult publishResult = flashItemAppService.publishFlashItem(token, activityId, toCommand(flashItemPublishRequest));
    return ResponseBuilder.with(publishResult);
}
```

4. 根据ID获取秒杀品

请求路径:

```
Method: GET
URL: localhost:8090/activities/1/flash-items/3
```

请求参数:

KEY	VALUE	DESCRIPTION

token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	

源码实现:

```
Plain Text
```

```
@GetMapping(value = "/activities/{activityId}/flash-items/{itemId}")
@SentinelResource("GetFlashItem")
public SingleResponse < FlashItemResponse > getFlashItem(@RequestParam String token,
    @PathVariable Long activityId,
    @PathVariable Long itemId,
    @RequestParam(required = false) Long version) {
    AppSingleResult < FlashItemDTO > flashItemResult = flashItemAppService.getFlashItem(token, activityId, itemId, version);
    if (!flashItemResult.isSuccess() || flashItemResult.getData() == null) {
        return ResponseBuilder.withSingle(flashItemResult);
    }
    return SingleResponse.of(toFlashItemResponse(flashItemResult.getData()));
}
```

5. 获取指定活动的已上线秒杀品

请求路径:

```
Plain Text
```

```
Method: GET
URL: localhost:8090/activities/1/flash-items/online
```

请求参数:

KEY	VALUE	DESCRIPTION

token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	
pageSize	10	
pageNumber	1	

源码实现:

Plain Text

```
@GetMapping(value = "/activities/{activityId}/flash-items")
@SentinelResource("GetFlashItems")
public MultiResponse < FlashItemDTO > getFlashItems(@RequestParam String token,
    @PathVariable Long activityId,
    @RequestParam Integer pageSize,
    @RequestParam Integer pageNumber,
    @RequestParam(required = false) String keyword) {
    FlashItemsQuery flashItemsQuery = new FlashItemsQuery()
        .setKeyword(keyword)
        .setPageSize(pageSize)
        .setPageNumber(pageNumber);
    AppMultiResult < FlashItemDTO > flashItemsResult = flashItemAppService.getFlashItems(token, activityId, flashItemsQuery);
    return ResponseBuilder.withMulti(flashItemsResult);
}
```

6. 获取指定活动的秒杀品

请求路径:

Plain Text

```
Method: GET
URL: localhost:8090/activities/1/flash-items
```

请求参数:

KEY	VALUE	DESCRIPTION

token	eyJleHBpcmVEYXRlIjoiMjAyMS0xMi0wNCAxNzo1Nzo1NCIsInVzZXJJZCI6MTAwMX0=	
pageSize	10	
pageNumber	1	

源码实现:

```
Plain Text
```

```
@GetMapping(value = "/activities/{activityId}/flash-items/online")
@SentinelResource("GetOnlineFlashItems")
public MultiResponse < FlashItemResponse > getOnlineFlashItems(@RequestParam String token,
    @PathVariable Long activityId,
    @RequestParam Integer pageSize,
    @RequestParam Integer pageNumber,
    @RequestParam(required = false) String keyword) {
    FlashItemsQuery flashItemsQuery = new FlashItemsQuery()
        .setKeyword(keyword)
        .setPageSize(pageSize)
        .setPageNumber(pageNumber)
        .setStatus(FlashItemStatus.ONLINE.getCode());
    AppMultiResult < FlashItemDTO > flashItemsResult = flashItemAppService.getFlashItems(token, activityId, flashItemsQuery);
    if (!flashItemsResult.isSuccess() || flashItemsResult.getData() == null) {
        return ResponseBuilder.withMulti(flashItemsResult);
    }
    return MultiResponse.of(toFlashItemsResponse(flashItemsResult.getData()), flashItemsResult.getTotal());
}
```

六、预热与客户端倒计时控制

在这部分，我们将讨论秒杀品预热和客户端倒计时的控制问题。

在秒杀过程中，我们需要提前将秒杀品库存等相关的数据设置到分布式缓存或本地缓存中，这个过程即秒杀品预热。通过预热，我们可以让大量请求在瞬时即可命中缓存。

（一）秒杀品预热

在FlashSale中，我们在应用层的调度模块中处理秒杀品的相关逻辑，相关源代码如下所示。

通过定时调度，每5秒（此处数值应可配）对数据库中的秒杀品进行一次检测，对**已上线但未预热**的秒杀品进行预热，在缓存中设置好库存。FlashSale使用的Spring的调度框架，并通过SchedulerLock设定调度锁，避免分布式部署时的并发执行问题。当然，对于有条件的组织来说，这里最好通过统一的调度平台进行调度。

下面有两处可优化点：

- 预热时可以进一步根据秒杀品的上线时间进行预热，只预热临近开始的秒杀品；
- 预热过程中，应对已预热但即将开始的活动进行预热复查，防止数据库中显示已预热，但缓存中的数据已经丢失。

Plain Text

```
@Scheduled(cron = "*/5 * * * * ?")
@BetaTrace
public void warmUpFlashItemTask() {
    logger.info("warmUpFlashItemTask|秒杀品预热调度");
    PagesQueryCondition pagesQueryCondition = new PagesQueryCondition();
    pagesQueryCondition.setStockWarmUp(0);
    PageResult < FlashItem > pageResult = flashItemDomainService.getFlashItems(pagesQueryCondition);
    pageResult.getData().forEach(flashItem -> {
        boolean initSuccess = itemStockCacheService.alignItemStocks(flashItem.getId());
        if (!initSuccess) {
            logger.info("warmUpFlashItemTask|秒杀品库存已经初始化预热失败", flashItem.getId());
            return;
        }
        flashItem.setStockWarmUp(1);
        flashItemDomainService.publishFlashItem(flashItem);
        logger.info("warmUpFlashItemTask|秒杀品库存已经初始化预热成功", flashItem.getId());
    });
}
```

（二）客户端倒计时控制

在秒杀活动中，页面往往需要显示秒杀倒计时。倒计时未结束时，按钮无法点击。如此，就会涉及到倒计时时钟的统一问题。倒计时的时钟绝对不能依赖于客户端，因为当用户的设备时钟不准确或

用户处于非当前时钟时，那么倒计时就会出错。所以，倒计时的时间需要服务端统一返回，客户端根据服务端的时间进行倒计时数秒，并根据服务端的状态控制按钮。

对于服务端如何提供统一的时间，有不同的方案。讲究点的可以提供统一的时间接口，这个时间是绝对准确的，不会依赖于服务器主机。当然，不讲究的可以使用下面这种方式，直接在返回秒杀品详情数据的时候返回时间和状态。在很多组织中，对于分布式的服务器都会进行统一的时间管理，不允许出现服务器之间的时钟差异。

Plain Text

```
@Data
public class FlashItemResponse {
    ...

    /**
     * 当前服务器时间
     */
    private long serverTimeMills = System.currentTimeMillis();

    /**
     * 当前秒杀品秒杀是否开始
     */
    public boolean isStarted() {
        if (!FlashItemStatus.isOnline(status)) {
            return false;
        }
        if (startTime == null || endTime == null) {
            return false;
        }
        Date now = new Date();
        return (startTime.equals(now) || startTime.before(now)) && endTim
e.after(now);
    }
}
```

小结

以上就是关于秒杀活动和秒杀品相关的全部内容，本章节到此为止。在本章节中，我们讲述了秒杀活动和秒杀品的模型设计、分层代码设计、接口设计以及秒杀品预热和客户端的时钟控制，希望对你有所帮助。对于本文未提及或未讲清楚的地方，欢迎评论区批评指正。

此外，利析秋毫的你可能已经发现本文似乎少了些什么，秒杀品活动和秒杀品在读取时的缓存部分哪里去了？没错，对于缓存的设计部分，我们将在下一章节详细讲述。

本章节到此结束，欢迎在评论区留下你的收获和疑问。对于本文未提及或未讲清楚的地方，也欢迎评论区批评指正，我会及时调整。如果你对小册有内容建议，请移步填写表单。